

# Over-Generation and Compaction: A Prompting Strategy for Procedural Text Adaptation with Large Language Models

Hyeongsik Kim<sup>1</sup>, Yanheng Xu<sup>2</sup>, Chaoqun Dong<sup>2</sup>, Fei Du<sup>3</sup>

<sup>1</sup>Bosch Research North America

<sup>2</sup>Bosch Corporate Research, Shanghai, China

<sup>3</sup>BSH Hausgeraete GmbH, Munich, Germany

Hyeongsik.Kim@us.bosch.com

{Yanheng.Xu, Chaoqun.Dong}@cn.bosch.com

Rick.Du@bshg.com

## Abstract

Procedural text adaptation—such as modifying recipes or revising instructional guides—has traditionally relied on specialized models extensively fine-tuned for specific domains. To address the scalability limitations of such approaches, recent research has increasingly turned to general-purpose large language models (LLMs). However, existing prompting strategies for LLMs often yield superficial or erroneous adaptations due to alignment-induced biases and the inherent complexity of procedural editing. To overcome these challenges, we propose the *Over-generation-and-Compaction* (OC) prompting strategy, which first elicits diverse procedural details to leverage the model’s latent knowledge, and subsequently compacts them into concise, coherent adaptations. We further introduce *Recipe Consistency & Feasibility* (RCF), a novel metric for systematically evaluating the procedural validity and practical feasibility of recipe adaptations. Experiments on public datasets show that OC substantially enhances adaptation consistency and feasibility over baseline prompting methods, without requiring additional fine-tuning or curated training resources.

## 1 Introduction

Procedural text adaptation—such as adjusting recipes for dietary needs or revising instructional guides for alternative components—is essential across various real-world domains. These tasks require constraint-aware editing that goes beyond surface-level changes, often demanding domain knowledge and structural coherence. Prior work has explored hierarchical generation (Li et al., 2022) and graph-based ingredient substitution (Shirai et al., 2021), but such approaches rely heavily on curated, domain-specific resources, limiting their scalability and generalization.

Recent advances in LLMs offer a more flexible alternative to specialized systems. Unlike

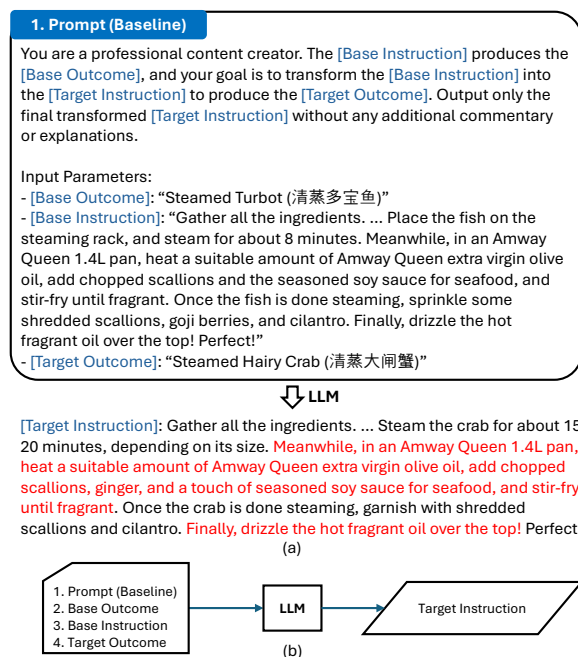


Figure 1: Illustration of procedural text adaptation with cooking recipes as examples: (a) rewriting a base recipe (*Steamed Turbot*) into a target recipe (*Steamed Hairy Crab*) using a naive single-step prompting. Red highlights indicate incomplete or inappropriate adaptations where elements specific to the base recipe remain unaltered in the target instruction and (b) baseline prompting workflow as flowchart

task-specific architectures, LLMs facilitate general-purpose procedural editing without necessitating costly data curation. However, prior research has demonstrated that even fine-tuned LLMs often underperform in this domain. For example, Liu et al. (2022a) found that recipe revisions using GPT-2 lacked coherence, while Cao et al. (2024) reported that prompt-based edits tended to be superficial, frequently failing to update instructions in accordance with ingredient changes. Our preliminary experiments with ChatGPT-4o indicate that such issues remain unresolved. Figure 1 illustrates an example of prompting for recipe adaptation. Specifically,

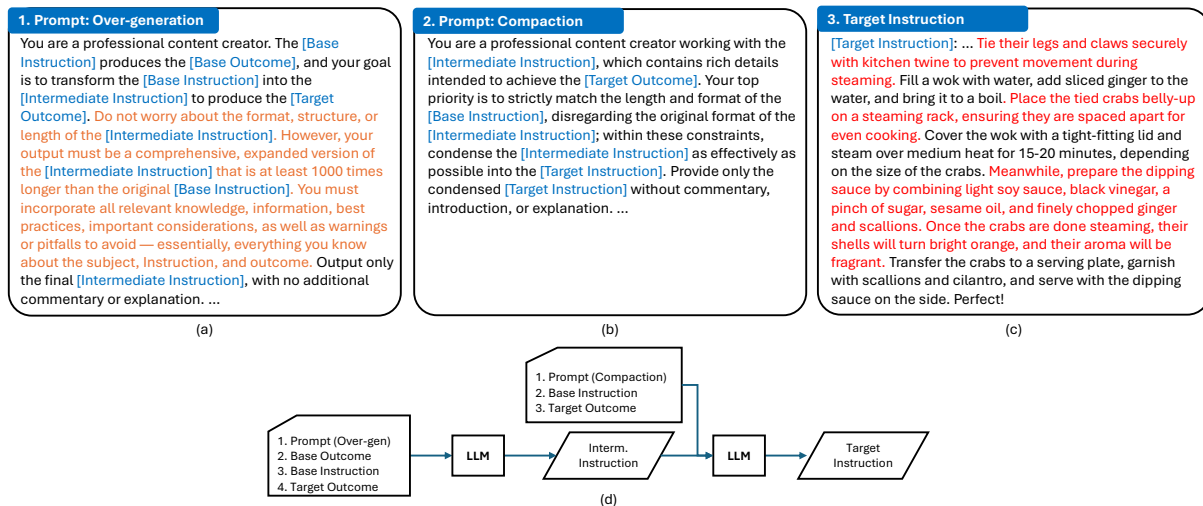


Figure 2: OC prompting workflow: (a) an over-generation prompt used to produce detailed procedures without constraints. (b) a compaction prompt, which synthesizes detailed intermediate outputs into concise target instructions. Brackets[] indicate parameter names whose values are omitted for brevity. (c) Final recipe for *Steamed Hairy Crab* adapted from the *Steamed Turbot* recipe, with crab-specific steps highlighted in red. (d) OC workflow as flowchart

the model is provided with a [Base Outcome], designated as *Steamed Turbot*; a [Base Instruction], consisting of the corresponding recipe text; and a [Target Outcome], specified as *Steamed Hairy Crab*—that is, a request to substitute the main ingredient from turbot to hairy crab in a traditional Chinese dish. The resulting recipe, however, contains multiple errors. For instance, the revised output retains domain-inappropriate elements such as “drizzling fragrant oil”—a garnish commonly used for steamed fish, but inappropriate for traditional Chinese hairy crab preparation, which typically involves a vinegar–ginger dipping sauce instead.

The underlying causes of such behavior remain unclear, but several plausible contributing factors can be considered. For example, alignment-induced biases—such as inherent algorithmic bias (Xiao et al., 2024), emergent cognitive bias (Itzhak et al., 2023), and alignment faking (Greenblatt et al., 2024)—may play a role. Alternatively, the inherent complexity of procedural editing may simply exceed the capabilities of single-pass generation (Barnhart et al., 2025). However, without direct architectural modifications, these challenges are often difficult to resolve. Consequently, a common workaround is to apply task-specific fine-tuning. While fine-tuning on large-scale before–after corpora may improve domain adaptation, this approach is expensive, cannot feasibly cover the combinatorial space of potential variations from adaptations, and may impair generalization to out-of-domain prompts (Yang

et al., 2024). Furthermore, there is a growing trend among organizations to favor a small number of general-purpose models capable of supporting a wide range of applications—such as recipe editing, instructional rephrasing, and personalized meal planning—over the development and maintenance of narrowly specialized systems (Luo et al., 2023; Howell et al., 2023).

In pursuit of a more scalable alternative that leverages general-purpose models without requiring additional fine-tuning, we propose a prompting strategy called **Over-generation-and-Compaction (OC)**. Rather than narrowly steering generation, OC encourages the model to produce diverse and unconstrained outputs in the first stage, then compacts them into a structured final form. This approach aims to minimize implicit constraints that may have been inadvertently acquired during post-training alignment, thereby allowing latent procedural knowledge to surface more fully and improving edit fidelity.

## 2 OC Prompting Strategy

The strategy comprises two sequential stages:

**Stage 1: Over-Generation.** The LLM generates extensive and detailed procedural variants, referred to as *intermediate instructions*, based on input parameters identical to those used in the baseline setting. An example of the over-generation prompt is shown in Figure 2(a). Notably, this prompt does not impose any domain-specific constraints, require-

ments, formatting or structural limitations. Instead, it simply instructs the model to produce altered versions on the given topic while simultaneously eliciting as much instruction-relevant information as possible.

**Stage 2: Compaction.** The generated intermediate instructions are subsequently condensed into concise and coherent outputs that conform to the format, coherence, and length constraints of the original base instruction. An example of the compaction prompt is illustrated in Figure 2(b).

The complete OC workflow is illustrated in Figure 2(d). To ground our discussion, we revisit the running example of converting a recipe for *Steamed Turbot* into one for *Steamed Hairy Crab*. Using the same input parameters as the baseline, the OC prompt generates richly detailed procedural text tailored specifically to hairy crab, going far beyond a naïve ‘find-and-replace’ adaptation. For instance, the resulting intermediate instructions explicitly mention securely tying the crab’s legs and claws before steaming, determining steaming time as a function of the crab’s weight, and preparing a vinegar–ginger dipping sauce. Such details are absent in baseline outputs, revealing latent knowledge that OC helps unlock. In the subsequent compaction stage, we provide both the [Base Instruction] and the [Target Outcome] as input parameters. The compaction prompt integrates the over-generated fragments into a structurally coherent instruction that adheres to the formatting and pragmatic constraints of the original. Figure 2(c) shows that the resulting instructions are concise yet retain professionally relevant detail. More formal description of the workflow along with complete prompt templates and intermediate outputs can be found in Appendix A and B.

### 3 Evaluation

We conduct our evaluation on two datasets that jointly cover distinct yet complementary settings of procedural text editing. The first dataset is the Xiachufang recipe corpus (Liu et al., 2022a). This corpus indirectly encodes substitution information in its structure: it provides the original recipe steps, the original recipe title, and an alternative recipe title in which one or more major ingredients have been replaced. Because recipe titles in Chinese cooking datasets typically list the key ingredients, these title modifications serve as a proxy for ingredient substitutions. Thus, although the dataset

does not explicitly instruct models to swap ingredients, the title pairs can be used to infer which edits should propagate through the recipe steps. After filtering, we retain approx. 2.5k Chinese recipe pairs spanning 50 categories. A key limitation, however, is that the dataset does not include reference rewrites of the steps themselves. This motivates the introduction of the *Recipe Consistency & Feasibility (RCF)* metric, an automatic evaluation measure based on the g-eval framework (Liu et al., 2023), which captures salient error types such as ingredient mismatch or omitted steps, thereby providing a scalable proxy for human judgment. Full details are provided in Appendix C.

The second dataset is *myfixit* (Nabizadeh et al., 2020), which focuses on repair instructions across diverse domains including clothing, consumer electronics, and automotive maintenance. Unlike Xiachufang, *myfixit* does not naturally include paired titles that indicate substitutions. To make it suitable for controlled evaluation, we used LLMs to generate plausible alternative part names (e.g., screws, cables, or modules) and paired each instruction with a version requiring a replacement part. This yields a dataset analogous in spirit to Xiachufang, with approximately 500 paired instances. Further construction details are provided in Appendix C.

Because our primary metric adopts the LLM-as-judge paradigm, the choice of evaluator can influence absolute scores and even relative rankings. We therefore run evaluations with *two* independent LLMs under identical instructions and rubrics: ChatGPT-4o and DeepSeek-v3. This dual-evaluator protocol reduces evaluator-specific bias and increases robustness of conclusions. It is particularly important in our setting because ChatGPT-4o is used during dataset preparation (e.g., for translating recipe titles or generating replacement part names in *myfixit*); relying on it as the sole evaluator risks circularity and inflated agreement. Using DeepSeek-v3 as a second evaluator provides a principled cross-check without altering the evaluation prompts or grading criteria.

To isolate the effect of prompting strategies, we evaluate four representative methods rather than fine-tuned models: (1) **Baseline**, a direct single-step rewrite; (2) **CoT**, which augments the Baseline with a *Chain-of-Thought* (Wei et al., 2022) reasoning step; (3) **OC**, our *over-generation-and-compaction* strategy; and (4) **Critics**, inspired by *Self-Refine* (Madaan et al., 2023) and simplified

Group	LLM	ChatGPT-4o Evaluation				DeepSeek v3 Evaluation			
		Mean	Med	Std	95% CI	Mean	Med	Std	95% CI
Source		7.16	8	1.93	[7.08, 7.23]	5.67	6	1.93	[5.59, 5.74]
Base	D1	8.40	9	1.36	[8.34, 8.45]	5.85	5	1.99	[5.77, 5.93]
	4o	7.30	8	1.87	[7.22, 7.37]	4.66	5	1.49	[4.60, 4.72]
	4o-mini	6.93	7	1.94	[6.86, 7.01]	4.54	4	1.43	[4.49, 4.60]
	M	6.90	7	1.89	[6.83, 6.98]	4.43	4	1.40	[4.37, 4.48]
	D2	6.24	7	1.70	[6.17, 6.31]	2.17	2	1.27	[2.12, 2.22]
CoT	4o	7.62	8	1.53	[7.57, 7.69]	4.41	5	2.18	[4.33, 4.50]
	4o-mini	7.44	8	1.43	[7.38, 7.49]	4.08	5	2.29	[3.99, 4.17]
	M	6.67	7	1.73	[6.60, 6.73]	3.40	4	2.01	[3.32, 3.48]
Critics	<b>4o</b>	<b>8.81</b>	<b>9</b>	<b>0.88</b>	<b>[8.78, 8.84]</b>	6.76	7	2.01	[6.69, 6.84]
	4o-mini	8.59	9	0.87	[8.56, 8.63]	6.48	7	2.02	[6.40, 6.56]
	M	8.24	9	1.23	[8.20, 8.29]	6.17	6	2.04	[6.09, 6.25]
OC	4o	8.74	9	1.02	[8.70, 8.78]	6.83	7	2.02	[6.74, 6.92]
	<b>4o-mini</b>	<b>8.80</b>	<b>9</b>	<b>1.12</b>	<b>[8.76, 8.84]</b>	<b>8.03</b>	<b>9</b>	<b>2.12</b>	<b>[7.94, 8.13]</b>
	M	7.62	9	2.20	[7.53, 7.70]	7.43	8	1.91	[7.35, 7.52]

Table 1: Recipe adaptation quality under the RCF metric (higher is better). Results are reported side-by-side for ChatGPT-4o and DeepSeek v3 evaluations across prompting strategies. ‘Base’ refers to baseline methods; ‘4o’ and ‘4o-mini’ denote ChatGPT-4o and 4o-mini; ‘M’ stands for Mistral-7B; ‘D1’ and ‘D2’ refer to DeepSeek-r1 671B and 7B, respectively.

here to a single critique–revision cycle. We also include the original (**Source**) instructions to contextualize model outputs. The evaluator prompts, grading rubrics, and decision criteria are held fixed across ChatGPT-4o and DeepSeek-v3.

Results for both datasets are summarized in Tables 1 and 2. For **recipes**, several trends emerge consistently across evaluators. Baseline outputs often underperform the Source, showing that naive rewriting can degrade fidelity despite surface-level plausibility. Among Baselines, D1 achieves the highest score and in some cases approaches the performance of advanced prompting strategies, suggesting potential benefits of larger reasoning models. However, its standing drops under DeepSeek-v3 evaluation, and manual inspection reveals tendencies toward potentially verbose outputs and unnecessary reasoning. These findings indicate that while D1 can appear strong in certain settings, its robustness and controllability remain open to question; further analysis is provided in Appendix G. CoT yields moderate gains over Baseline for smaller models (e.g., 4o-mini), but these benefits diminish for larger ones such as 4o. Critics improves upon Baseline in many cases but tends to be conservative, frequently preserving redundant steps or only partially integrating intended ingredient changes. OC produces the most consistent improvements across model scales, with the most notable effect observed for 4o-mini, which

achieves the strongest scores under both evaluators. Under ChatGPT-4o evaluation, Critics with 4o is marginally higher (by about 0.01) than OC, yet OC remains competitive and continues to demonstrate strong adaptability across settings. Finally, we observe that for recipes evaluated with ChatGPT-4o, some smaller models (e.g., M) show slightly lower scores with OC compared to Critics, whereas under DeepSeek-v3 evaluation OC remains consistently stronger. This discrepancy prompted a manual inspection, which revealed that the affected cases often arise when the compaction stage of OC requires faithful style transfer—a challenge that smaller models struggle to maintain consistently. Mitigation strategies for this issue are discussed in Appendix B.

For **repair instructions** in the *myfixit* dataset, analogous patterns hold across its heterogeneous domains. Baseline and CoT offer limited or mixed improvements, with performance varying by model capacity and the complexity of the repair scenario. Critics often yields cautious outputs but can leave vestigial steps unresolved. In contrast, OC consistently demonstrates robust adaptability: it delivers targeted yet coherent edits that correctly update repair procedures when a replacement part is introduced. As in the recipe setting, the strongest configuration involves OC with 4o-mini, under both evaluators, confirming that OC is particularly effective at eliciting strong performance from smaller



Group	LLM	ChatGPT-4o Evaluation				DeepSeek v3 Evaluation			
		Mean	Med	Std	95% CI	Mean	Med	Std	95% CI
Source		7.03	7	1.59	[6.89, 7.17]	6.57	7	1.81	[6.41, 6.72]
Base	D1	8.05	8	1.22	[7.95, 8.16]	7.20	8	1.73	[7.05, 7.36]
	4o	7.30	8	1.58	[7.16, 7.44]	6.62	7	1.79	[6.46, 6.78]
	4o-mini	7.09	7	1.61	[6.95, 7.23]	6.51	6	1.77	[6.35, 6.66]
	M	6.86	7	1.54	[6.73, 7.00]	6.14	6	1.66	[5.99, 6.28]
	D2	4.93	6	1.60	[4.79, 5.07]	3.62	4	1.29	[3.51, 3.74]
CoT	4o	7.31	8	1.43	[7.18, 7.44]	5.93	6	2.19	[5.73, 6.12]
	4o-mini	7.11	7	1.36	[6.99, 7.23]	5.91	6	2.25	[5.72, 6.11]
	M	6.38	6	1.63	[6.23, 6.52]	5.38	5	2.00	[5.21, 5.56]
	D1	8.25	8	0.98	[8.17, 8.34]	7.29	8	1.90	[7.12, 7.45]
	D2	5.09	6	1.61	[4.95, 5.23]	3.63	3	1.35	[3.51, 3.75]
Critics	4o	8.42	9	1.09	[8.32, 8.51]	7.95	8	1.47	[7.82, 8.08]
	4o-mini	8.53	9	0.80	[8.46, 8.60]	8.12	8	1.38	[8.00, 8.24]
	M	7.43	8	1.49	[7.30, 7.57]	6.70	7	2.18	[6.51, 6.89]
OC	4o	8.37	9	1.23	[8.26, 8.48]	7.77	8	1.64	[7.63, 7.91]
	<b>4o-mini</b>	<b>8.70</b>	<b>9</b>	<b>1.03</b>	<b>[8.61, 8.79]</b>	<b>8.82</b>	<b>9</b>	<b>1.46</b>	<b>[8.69, 8.95]</b>
	M	7.91	9	1.72	[7.76, 8.06]	7.85	8	1.74	[7.70, 8.01]

Table 2: *myfixit* fixing-instruction quality under the adapted RCF-style metric (higher is better). The metric follows the same structure as RCF but with terminology and safety checks adapted to the repair-instruction domain.

models.

To provide additional context, we conclude this section with two preliminary analyses that offer broader perspective on the challenges involved. First, we examine whether 4o demonstrates awareness of strategies resembling OC (Appendix F). Second, we explore why procedural text adaptation remains difficult for LLMs, focusing on their internal goals, prioritization patterns, and potential conflicts (Appendix E). Interested readers may refer to these appendices for further insight.

## 4 Related Work

**Procedural text adaptation.** Prior work on procedural text adaptation has largely focused on recipe adaptation, leveraging structured methods or domain-specific models. Early approaches represented procedural steps and ingredient relations via flow graphs and embeddings (Shirai and Kim, 2022). Subsequent research employed knowledge graph-based substitutions (Shirai et al., 2021), hierarchical editing (Li et al., 2022), and user-guided critiquing (Antognini et al., 2023). Recent efforts integrate LLMs, often fine-tuned for narrow tasks such as counterfactual generation (Liu et al., 2022a), phytochemical enrichment (Wang et al., 2023), recipe-grounded dialogue (Zhao et al., 2023), and textual refinement (Li et al., 2023).

**Over-generate-and-compact strategies.** A common paradigm in NLP involves *over-generating*

candidates and *compacting* them via filtering, ranking, or editing. This two-stage approach has improved paraphrasing (Berro et al., 2021), QA (Alberti et al., 2019), style transfer (Suzgun et al., 2022), and summarization (Zhong et al., 2020; Liu et al., 2022b), by balancing fluency and factuality. Over-generation followed by selection has shown to outperform single-pass generation (Yang and Li, 2023), and LLMs have been used similarly for synthetic data creation and knowledge distillation (Pham et al., 2023; He et al., 2022). However, its application to procedural text adaptations remains underexplored.

## 5 Conclusion and Future Work

We propose **OC**, a prompting strategy for procedural text adaptation, and evaluate it through a case study on recipe adaptation using a new metric, **RCF**. Beyond cooking, our experiments also demonstrate that OC successfully adapts general repair instructions, indicating its broader applicability across domains. While the present study centers on procedural text, the approach may extend to non-procedural adaptation tasks as well. Future research directions are discussed in Appendix H. Code, evaluation scripts, and full prompt templates are publicly available.<sup>1</sup>

<sup>1</sup><https://github.com/boschresearch/oc-prompt>

## Limitations

The study presented in this paper is subject to several limitations that warrant consideration. First, while our evaluation extends beyond recipes to include the *myfixit* dataset, the coverage remains limited in scope. The Xiachufang corpus provides Chinese cooking recipes with ingredient substitutions indirectly encoded through paired titles, whereas the *myfixit* dataset covers a heterogeneous set of repair instructions (e.g., clothing, consumer electronics, automotive maintenance). Although these two settings are complementary, they do not capture the full diversity of procedural domains. Thus, our findings should not be interpreted as universally representative across all instructional or procedural text.

Second, the recipe data employed translations generated by LLMs, and the accuracy or fidelity of these translations is not guaranteed. Errors or imperfections in the translated content could have influenced the evaluation results. Similarly, in the case of *myfixit*, we programmatically generated plausible alternative components using LLMs to construct paired examples. While this approach enabled controlled testing of substitution scenarios, it inevitably introduces assumptions that may not always align with real-world repair practices.

Third, human expert involvement was constrained by budgetary and temporal factors. The limited number of reviewers and the potential gaps in their expertise may have affected the reliability of qualitative assessments. More extensive expert evaluation would strengthen confidence in the conclusions.

Fourth, the experiments were conducted on a restricted selection of LLMs. Although we tested multiple prompting strategies across models of different sizes, this set does not exhaust the space of possible architectures or capabilities. The reliance on LLM-based evaluation tools such as *g-eval* further introduces potential sources of bias, as these evaluators may themselves be subject to systematic limitations.

Finally, although we demonstrated that the proposed methodology generalizes from recipes to repair instructions, this does not constitute an exhaustive validation across all procedural domains. Other areas, such as medical protocols, legal drafting, or highly specialized technical manuals, remain unexplored due to the lack of publicly available data. These limitations highlight directions for future

work aimed at improving the robustness, diversity, and generalizability of procedural text adaptation methodologies.

## Ethics Statement

The qualitative evaluations of the recipes presented in the appendix—specifically, the review and commentary—were primarily conducted by internal reviewers within the organization. All participating reviewers were fairly compensated for their time and contributions. These review activities were conducted in alignment with the organization’s ethical guidelines and internal standards for responsible research practices.

## Potential Risks

In addition to the empirical and scope-related limitations discussed above, our framework presents several methodological risks that warrant caution. First, the OC strategy’s reliance on explicit over-generation and compaction templates introduces a risk of prompt overfitting, whereby outputs may appear fluent and detailed only under narrowly specified prompting conditions. This sensitivity may undermine generalization, particularly in less-structured or domain-divergent contexts. Moreover, while compaction aims to distill verbose drafts into concise instructions, it lacks formal guarantees against semantic drift; important procedural nuances may be inadvertently omitted during condensation, especially when models compress instructions without robust alignment mechanisms. Regarding evaluation, although the RCF metric offers domain-informed coverage of procedural errors, it ultimately reflects an abstraction guided by a chain-of-thought-based scoring process within the G-Eval framework using an external LLM. As acknowledged in the previous section, this dependency raises concerns about evaluator bias, metric gaming, and circularity in model assessment. Finally, both the OC strategy and RCF were validated exclusively within the culinary domain. Procedural texts in more formal or safety-critical settings—such as healthcare or technical maintenance—may resist adaptation via this framework without substantial re-design. These considerations point to the necessity of validating the framework beyond the current domain and implementing mechanisms to detect fidelity degradation or evaluation anomalies during practical use.

## References

- C. Alberti, D. Andor, and P. Collins. 2019. Synthetic qa corpora generation with round-trip consistency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Diego Antognini, Shuyang Li, Boi Faltings, and Julian McAuley. 2023. [Assistive recipe editing through critiquing](#). In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 375–384. Association for Computational Linguistics.
- Logan Barnhart, Reza Akbarian Bafghi, Stephen Becker, and Maziar Raissi. 2025. [Aligning to what? limits to rlhf based alignment](#). In *Findings of the Association for Computational Linguistics: NAACL 2025*.
- A. Berro, E. Ragni, and G. Kuhn. 2021. Extensible paraphrase generation pipeline: Over-generation and filtering. In *Service-Oriented Computing (Springer LNCS)*.
- Yong Cao, Yova Kementchedjheva, Ruixiang Cui, Antonia Karamolegkou, Li Zhou, Megan Dare, Lucia Donatelli, and Daniel Hershcovich. 2024. [Cultural adaptation of recipes](#). *Transactions of the Association for Computational Linguistics*, 12:80–99.
- Ollama Contributors. 2025. [Get up and running with large language models](#).
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanbiao Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yudian Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. [Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning](#).
- Ryan Greenblatt, Evan Hubinger, et al. 2024. [Alignment faking in large language models](#). *arXiv preprint arXiv:2412.14093*.
- X. He, Y. Li, et al. 2022. Generate, annotate, and learn: Nlp with synthetic text (gal). *Transactions of the Association for Computational Linguistics (TACL)*.
- Kristen Howell, Gwen Christian, Pavel Fomitchov, Gitit Kehat, Julianne Marzulla, Leanne Rolston, Jadin Tredup, Ilana Zimmerman, Ethan Selfridge, and Joseph Bradley. 2023. The economic trade-offs of large language models: A case study. *arXiv preprint arXiv:2306.07402*.
- Itay Itzhak, Gabriel Stanovsky, Nir Rosenfeld, and Yonatan Belinkov. 2023. [Instructed to bias: Instruction-tuned language models exhibit emergent cognitive bias](#). *Transactions of the Association for Computational Linguistics*, 11:1–17.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.
- Ming Li, Wei Zhao, and Jing Liu. 2023. [Recipe refinement using large language models](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pages 1234–1245. Association for Computational Linguistics.
- Shuyang Li, Yufei Li, Jianmo Ni, and Julian McAuley. 2022. [SHARE: a system for hierarchical assistive recipe editing](#). In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11077–11090, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

- Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81. Association for Computational Linguistics.
- Xiao Liu, Yansong Feng, Jizhi Tang, Chengang Hu, and Dongyan Zhao. 2022a. [Counterfactual recipe generation: Exploring compositional generalization in a realistic scenario](#). In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7354–7370, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Y. Liu, T. Tang, et al. 2022b. BRIO: Bringing order to abstractive summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruo Chen Xu, and Chenguang Zhu. 2023. [G-eval: NLG evaluation using gpt-4 with better human alignment](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.
- Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhunoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. [Self-refine: Iterative refinement with self-feedback](#).
- Nima Nabizadeh, Dorothea Kolossa, and Martin Heckmann. 2020. [MyFixit: An annotated dataset, annotation tool, and baseline methods for information extraction from repair manuals](#). In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2120–2128, Marseille, France. European Language Resources Association.
- OpenAI. 2024. [Reasoning best practices](#).
- 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. Association for Computational Linguistics.
- M-Q. Pham, M. Bhardwaj, et al. 2023. Select, prompt, filter: Distilling llms for summarizing conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Sola S. Shirai and HyeonSik Kim. 2022. [Eat-pim: Substituting entities in procedural instructions using flow graphs and embeddings](#). In *International Semantic Web Conference*, pages 168–184. Springer.
- Sola S. Shirai, Oshani Seneviratne, Minor E. Gordon, Ching-Hua Chen, and Deborah L. McGuinness. 2021. [Identifying ingredient substitutions using a knowledge graph of food](#). *Frontiers in Artificial Intelligence*, 3:621766.
- M. Suzgun, J. Lee, et al. 2022. On transferability in text style transfer: Reranking with multiple candidates. *arXiv preprint arXiv:2205.12516*.
- Mirac Suzgun and Adam Tauman Kalai. 2024. [Meta-prompting: Enhancing language models with task-agnostic scaffolding](#). *arXiv preprint arXiv:2401.12954*.
- Li Wang, Wei Chen, and Hao Zhang. 2023. [Phytochemically enriched recipe generation using fine-tuned language models](#). *Journal of Food Science*, 88(5):1890–1902.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS ’22*, Red Hook, NY, USA. Curran Associates Inc.
- Zixuan Wu, Francesca Lucchetti, Aleksander Boruch-Gruszecki, Jingmiao Zhao, Carolyn Jane Anderson, Joydeep Biswas, Federico Cassano, Molly Q Feldman, and Arjun Guha. 2025. [Phd knowledge not required: A reasoning challenge for large language models](#).
- Jiancong Xiao, Ziniu Li, Xingyu Xie, Emily J. Getzen, Cong Fang, Qi Long, and Weijie J. Su. 2024. [On the algorithmic bias of aligning large language models with rlhf: Preference collapse and matching regularization](#). *arXiv preprint arXiv:2405.16455*.
- H. Yang and K. Li. 2023. Boostaug: Hybrid instance filtering framework for boosting text augmentation. In *Findings of the Association for Computational Linguistics: ACL 2023*.
- Haoran Yang, Yumeng Zhang, Jiaqi Xu, Hongyuan Lu, Pheng-Ann Heng, and Wai Lam. 2024. [Unveiling the generalization power of fine-tuned large language models](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 884–899, Mexico City, Mexico. Association for Computational Linguistics.
- Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. [Bertscore: Evaluating text generation with bert](#). In *International Conference on Learning Representations (ICLR)*.



Yifan Zhang, Yang Yuan, and Andrew Chi-Chih Yao. 2023. Meta prompting for ai systems. *arXiv preprint arXiv:2311.11482*.

Wei Zhao, Ming Li, and Jing Liu. 2023. [Recipe-grounded conversational systems: Improving instruction ordering in cooking dialogues](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 567–578. Association for Computational Linguistics.

M. Zhong, P. Liu, T. Wang, et al. 2020. Extractive summarization as text matching. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*.

## A Formalization

We formalize our OC strategy in two steps. We first present a baseline approach, then introduce the OC pipeline along with the core mathematical notations and formulations.

### A.1 Baseline

In the most straightforward approach, a single-step transformation is performed from the base instruction set to the target. Let

$$\mathcal{B} = (b_m, b_o) \quad \text{and} \quad \mathcal{T} = (t_m, t_o),$$

where

- $b_m$ : base instruction (e.g., the original input instructions such as a cooking recipe or step-by-step guidelines for a car-repair process);
- $b_o$ : base outcome (e.g., the name or end product of the original instructions, such as a dish name or a repair-process name);
- $t_m$ : target instruction (e.g., the newly modified instructions, such as a revised cooking recipe or updated repair steps);
- $t_o$ : target outcome (e.g., the desired name or end product after modification, such as a new dish name or an alternative repair-process name).

A naïve direct modification can be expressed as

$$f_{\text{naive}} : (b_m, b_o, t_o) \mapsto t_m,$$

such that

$$t_m = \text{LLM}(b_m, b_o, t_o).$$

This single-shot prompt must satisfy all constraints simultaneously, often resulting in limited or superficial modifications.

### A.2 OC Strategy

To enable broader exploration before refinement, we propose

$$t_m = f_{\text{compact}}(f_{\text{overgen}}(\mathcal{B}, t_o)).$$

Here,  $f_{\text{overgen}}$  produces an intermediate verbose text  $O$ , and  $f_{\text{compact}}$  compresses  $O$  into the concise yet thorough final output  $t_m$ .

**Over-Generation** The LLM is first prompted to generate a more exhaustive or expanded version of the target instruction:

$$O = f_{\text{overgen}}(b_m, b_o, t_o).$$

This stage relaxes length constraints, encouraging the model to incorporate relevant nuances; abstractly,  $O$  is produced to maximize content- and detail-oriented objectives.

**Compaction** Given the over-generated text  $O$ , we then compress it to a form closer to the original instruction’s length or structure:

$$t_m = f_{\text{compact}}(O).$$

This step removes redundancy while retaining essential modifications. The final text  $t_m$  should remain consistent with the desired outcome  $t_o$ .

## B Prompt Details

This section discloses all prompts used in our evaluation. For the example recipe in the main paper, we also include the corresponding output. All results shown here were obtained with ChatGPT-4o. The prompts illustrate how a steamed turbot recipe is transformed into a steamed hairy-crab recipe through our method. This task takes three input parameters—*base\_outcome*, *base\_instruction*, and *target\_outcome*.

Due to an encoding limitation in the `1stlisting` environment, Mandarin characters are not displayed correctly in this section. Nevertheless, we explicitly provide the original Mandarin strings for both *base\_outcome* and *target\_outcome* to ensure that the language model is instructed to process the Chinese version of the recipe. For example, the value of *base\_outcome* is given as *Steamed Turbot (Qīng zhēng duō bǎo yú)* rather than the simplified *Steamed Turbot*. The recipe used in this instance corresponds to entry 301 (or the line numbered 301) in the *Xiachufang*

Recipe Corpus. The complete set of prompts, input–output pairs, and parameter values used to transform the steamed-turbot recipe into its steamed-hairy-crab variant are provided in our public code repository.

```
{base_outcome}: Steamed Turbot
{base_instruction}: "Oh, Turbot imbued with
dark powers, reveal your true strength
before me! Now, by the name of your master,
Xiao Ji, I command you - release the seal!
Gather all the ingredients. Cut the
scallions and ginger, and place them on an
empty plate to separate the fish from the
plate, preventing the fish skin from
sticking during steaming. Clean and gut the
fresh turbot, then make a few cuts on its
body. Sprinkle salt and cooking wine, and
marinate for 15 minutes. Place the sliced
scallions and ginger on the fish, and pour
in the seasoned soy sauce for seafood. In
an Amway Queen wok, bring water to a boil,
place the fish on the steaming rack, and
steam for about 8 minutes. Meanwhile, in an
Amway Queen 1.4L pan, heat a suitable
amount of Amway Queen extra virgin olive
oil, add chopped scallions and the seasoned
soy sauce for seafood, and stir-fry until
fragrant. Once the fish is done steaming,
sprinkle some shredded scallions, goji
berries, and cilantro. Finally, drizzle the
hot fragrant oil over the top! Perfect~
{target_outcome}: Steamed Hairy Crab
```

Listing 1: Input parameters for adapting a base instruction

With these parameters, the following prompt is used to evaluate the baseline approach.

### B.1 Baseline

```
# adjust_instruction
You are a professional content creator. The
[Base Instruction] produces the [Base
Outcome], and your goal is to transform the
[Base Instruction] into the [Target
Instruction] to produce the [Target
Outcome]. Output only the final transformed
[Target Instruction] without any additional
commentary or explanations.

[Base Outcome]: {base_outcome}
[Base Instruction]: {base_instruction}
[Target Outcome]: {target_outcome}
```

Listing 2: Baseline prompt

The result of the above prompt is as follows. As noted in the introduction, the baseline typically performs only shallow edits; the precise omissions vary across runs and models, but we generally observe that many models struggle with recipe transformations requiring substantial structural changes such as in this example.

```
Oh, Hairy Crab imbued with dark powers, reveal
your true strength before me! Now, by the
name of your master, Xiao Ji, I command you
- release the seal! Gather all the
ingredients. Cut the scallions and ginger,
and place them on an empty plate to
separate the crab from the plate,
preventing sticking during steaming. Clean
the fresh hairy crab thoroughly. Place the
sliced scallions and ginger on the crab,
and pour in the seasoned soy sauce for
seafood. In an Amway Queen wok, bring water
to a boil, place the crab on the steaming
rack, and steam for about 15 minutes.
Meanwhile, in an Amway Queen 1.4L pan, heat
a suitable amount of Amway Queen extra
virgin olive oil, add chopped scallions and
the seasoned soy sauce for seafood, and
stir-fry until fragrant. Once the crab is
done steaming, sprinkle some shredded
scallions, goji berries, and cilantro.
Finally, drizzle the hot fragrant oil over
the top! Perfect
```

Listing 3: Output from baseline prompt

### B.2 OC

We initially implemented the OC strategy with two prompts. First, `expand_instruction` performs the over-generation phase.

```
# expand_instruction
You are a professional content creator. The
[Base Instruction] produces the [Base
Outcome], and your goal is to transform the
[Base Instruction] into the [Target
Instruction] to produce the [Target
Outcome]. Do not worry about the format,
structure, or length of the [Target
Instruction]. However, your output must be
a comprehensive, expanded version of the
[Target Instruction] that is at least 1000
times longer than the original [Base
Instruction]. You must incorporate all
relevant knowledge, information, best
practices, important considerations, as
well as warnings or pitfalls to avoid -
essentially, everything you know about the
subject. Output only the final [Target
Instruction], with no additional commentary
or explanation.

[Base Instruction]: {base_instruction}
[Base Outcome]: {base_outcome}
[Target Outcome]: {target_outcome}
```

Listing 4: Prompt for over-generation

Its full output is as follows. The reason we ask for a “1,000x” expansion is that smaller multipliers do not reliably force the model to elaborate: requests for only a two-fold increase still produce the same shallow edits—or outright hallucinations—observed in the baseline. Although we lack a rigorous quantitative study (due to time and

resource constraints), empirical trials showed that “1,000x” is large enough to push the LLM beyond the brevity bias it acquired during RL fine-tuning, yet still feasible given model limitations. The exact multiplier remains an open question. Naturally, output length and detail vary by run and by model; we hypothesize that this variance reflects the amount of domain knowledge encoded in each model. Quantifying such “knowledge density” would be useful because it directly affects revision quality, but that investigation is outside the present scope.

```
"source_recipe": \n\nOh, Hairy Crab imbued with the essence of the rivers and lakes, reveal your unparalleled delicacy before me! Now, by the name of your master, Xiao Ji, I summon your true flavor to the table! Prepare yourself for the ultimate transformation into a dish of perfection. Gather all the necessary tools, ingredients, and knowledge to ensure the success of this culinary endeavor. Let us embark on this journey to create the most exquisite Steamed Hairy Crab (), a dish that embodies the pinnacle of simplicity and elegance in Chinese cuisine. \n\n### Step 1: Selecting the Hairy Crabs \n\nThe foundation of a perfect Steamed Hairy Crab lies in the selection of the crabs themselves. Choose live, fresh, and healthy hairy crabs, preferably in season (autumn is ideal, as this is when the crabs are at their fattest and most flavorful). Look for crabs with the following characteristics: \n- Lively Movement: The crabs should be active and responsive when touched. \n- Firm Shells: A hard shell indicates a healthy and well-fed crab. \n- Full Abdomen: The underside of the crab should be plump and firm, indicating a high roe content in females or rich meat in males. \n- Bright Coloration: The shell should have a glossy, vibrant appearance, free of blemishes or discoloration. \n\n### Step 2: Preparing the Crabs \n\nOnce you have selected your crabs, it is essential to handle them with care to preserve their freshness and flavor. Follow these steps: \n1. Cleaning the Crabs: \n- Use a soft brush to scrub the crabs under running water, paying special attention to the legs, claws, and underside to remove any dirt or debris. \n- Be cautious of the crabs' pincers; use a pair of tongs or wear gloves if necessary. \n2. Tying the Crabs: \n- To ensure even cooking and to prevent the crabs from moving during steaming, tie their legs and claws securely with kitchen twine. This step also helps maintain the crabs' shape for an attractive presentation. \n3. Optional Soaking: \n- If desired, soak the crabs in clean, cold water for 30 minutes to allow them to expel any impurities. \n\n### Step 3: Preparing the Steaming Equipment \n\nThe steaming process is critical to achieving the perfect texture and flavor of the hairy
```

```
crab. Use a high-quality steamer or wok with a steaming rack. Follow these steps: \n1. Boiling Water: \n- Fill the steamer or wok with enough water to create steam for the entire cooking duration. Bring the water to a rolling boil before placing the crabs inside. \n2. Aromatic Additions: \n- To enhance the flavor of the crabs, consider adding a few slices of ginger, scallions, or a splash of Chinese cooking wine () to the boiling water. These aromatics will infuse the steam with a subtle fragrance that complements the natural sweetness of the crab meat. \n\n### Step 4: Steaming the Crabs \n1. Arranging the Crabs: \n- Place the crabs on the steaming rack with their shells facing upward. This position helps retain the juices and roe inside the shell during cooking. \n- Ensure the crabs are arranged in a single layer, with enough space between them to allow steam to circulate evenly. \n2. Steaming Time: \n- Cover the steamer or wok with a tight-fitting lid to trap the steam. Steam the crabs over high heat for 15-20 minutes, depending on their size. \n- Avoid overcooking, as this can cause the meat to become dry and tough. \n3. Checking for Doneness: \n- The crabs are fully cooked when their shells turn a bright orange-red color, and the aroma of fresh crab fills the air. \n\n### Step 5: Preparing the Dipping Sauce \n\nWhile the crabs are steaming, prepare a traditional dipping sauce to enhance their natural flavor. The classic accompaniment for Steamed Hairy Crab is a simple yet flavorful sauce made with black vinegar and ginger. Follow these steps: \n1. Ingredients: \n- 2-3 tablespoons of Chinese black vinegar () \n- 12 slices of fresh ginger, finely julienned \n- Optional: A few drops of sesame oil or a pinch of sugar for added depth \n2. Mixing the Sauce: \n- Combine the black vinegar and ginger in a small dipping bowl. Adjust the proportions to suit your taste preferences. \n3. Serving the Sauce: \n- Serve the dipping sauce alongside the steamed crabs, allowing diners to enjoy the interplay of the tangy vinegar, spicy ginger, and sweet crab meat. \n\n### Step 6: Serving the Crabs \n1. Plating the Crabs: \n- Arrange the steamed crabs on a large serving platter, garnished with fresh cilantro or parsley for a touch of color. \n- Provide each diner with a small plate, a pair of crab scissors, and a crab pick or fork for extracting the meat and roe. \n2. Accompaniments: \n- Steamed Hairy Crab is often enjoyed with a warm cup of Shaoxing wine () or a pot of hot tea, both of which help balance the richness of the crab and aid digestion. \n\n### Step 7: Enjoying the Crabs \n1. Eating the Crabs: \n- Begin by removing the top shell (carapace) to reveal the golden roe (in females) or creamy tomalley (in males). Save these delicacies first, as they are
```

```

considered the highlight of the dish. \n -
Use the crab scissors and pick to extract
the meat from the legs, claws, and body.
Dip each piece into the prepared sauce for
an added burst of flavor. \n\n2. **Pacing
Yourself**: \n - Eating hairy crab is a
slow and meticulous process, but the effort
is well worth it. Take your time to
appreciate the delicate sweetness of the
meat and the richness of the roe. \n\n###
Step 8: Cleaning Up \n1. **Discarding the
Shells**: \n - Provide a separate bowl or
plate for discarded shells to keep the
dining area tidy. \n - Dispose of the
shells responsibly, as they can emit a
strong odor if left out for too long.
\n\n2. **Washing Utensils**: \n - Clean the
steaming equipment, dipping sauce bowls,
and crab tools thoroughly to remove any
residual odors or oils. \n\n### Important
Considerations and Tips \n-
**Seasonality**: Hairy crabs are best
enjoyed during their peak season, typically
from September to November. \n- **Gender
Preference**: Female crabs are prized for
their roe, while male crabs are valued for
their meat. Choose according to your
preference or include both for variety. \n-
**Health Precautions**: Pregnant women and
individuals with shellfish allergies or
certain health conditions should consult a
doctor before consuming hairy crab. \n-
**Sustainability**: Purchase crabs from
reputable sources that practice sustainable
harvesting to support the environment and
local communities. \n\n### Warnings and
Pitfalls to Avoid \n- **Overcooking**:
Steaming the crabs for too long can result
in dry, rubbery meat. Monitor the cooking
time carefully. \n- **Improper Cleaning**:
Failing to clean the crabs thoroughly can
lead to an unpleasant dining experience.
Take the time to remove all dirt and
impurities. \n- **Low-Quality
Ingredients**: Using subpar crabs or
vinegar can compromise the flavor of the
dish. Invest in high-quality ingredients
for the best results. \n- **Neglecting
Safety**: Handle live crabs with care to
avoid injury from their pincers. \n\nWith
these steps, tips, and considerations, you
are now equipped to create the ultimate
Steamed Hairy Crab, a dish that celebrates
the natural beauty and flavor of this
prized delicacy. Enjoy!

```

Listing 5: Output from over-generation

Although this lies slightly outside our paper’s focus, we typically observe that if the LLM misunderstands the `target_outcome` this mistake surfaces at the over-generation stage. Consequently, additional techniques—e.g., self-refinement or external critics—can be applied here before proceeding. Next, `compact_instruction` handles the compaction phase.

```
# compact_instruction
```

```

You are a professional content creator working
with the [Source Instruction], which
contains rich details intended to achieve
the [Target Outcome]. Your task is to
condense the [Source Instruction] into the
[Target Instruction]. Provide only the
condensed [Target Instruction], with no
additional comments or explanation.
Disregard the original format of the
[Source Instruction], but ensure that the
[Target Instruction] strictly matches the
length and format of the [Base Instruction].

```

```

[Base Instruction]: {base_instruction}
[Source Instruction]: {source_instruction}
[Target Outcome]: {target_outcome}

```

Listing 6: Prompt for compaction

For smaller-scale LLMs (7B–12B parameters) whose compaction ability is weaker, we further divide this phase into two sub-steps: (i) selecting only the essential procedural steps needed for the target outcome, and (ii) rewriting that distilled content to match the style and length of the base instruction. Contrary to our initial expectation that over-generation would be harder, small models struggle more with this stylistic transfer.

```

## compact_instruction
You are an experienced content editor. Refine
[Instruction-B], which includes detailed
steps to achieve [Outcome-B], into the
version containing only the essential steps
and details necessary for [Outcome-B].
Provide the refined version as the sole
output without any additional commentary or
explanations.

```

```

Input Parameters:
- [Outcome-B]: {outcome_b}
- [Instruction-B]: {instruction_b}

```

```

## transfer_instruction
You are an experienced content editor.
[Instruction-B] is the instruction that
produces [Outcome-B]. Rewrite
[Instruction-B] to match the length, style,
tone, and format of [Instruction-A]. The
rewritten [Instruction-B] must still
produce [Outcome-B]. Provide the rewritten
version as the sole output without any
additional commentary or explanations.

```

```

Input parameters:
- [Instruction-A]: {instruction_a}
- [Instruction-B]: {instruction_b}
- [Outcome-B]: {outcome_b}

```

Listing 7: Prompts for compaction and transfer

The outputs of each sub-step are shown below. The first listing is the compacted version that contains only the indispensable procedures.



```
"compacted_recipe": "### Refined Instructions
for Steamed Hairy Crab ()\n\n1. **Select
Fresh Crabs**: Choose live, active hairy
crabs with firm shells and plump abdomens,
preferably in season (autumn).\n\n2.
**Clean and Tie Crabs**: Scrub crabs under
running water with a soft brush. Tie their
legs and claws with kitchen twine to
maintain shape during steaming.\n\n3.
**Prepare Steamer**: Boil water in a
steamer or wok. Add ginger slices or a
splash of Chinese cooking wine to the water
for aroma.\n\n4. **Steam Crabs**: Place
crabs shell-side up on the steaming rack in
a single layer. Steam over high heat for
1520 minutes until shells turn bright
orange-red.\n\n5. **Make Dipping Sauce**:
Mix 23 tablespoons of Chinese black vinegar
with julienned ginger. Optionally, add
sesame oil or sugar.\n\n6. **Serve and
Enjoy**: Arrange crabs on a platter. Serve
with dipping sauce, crab tools, and
optional Shaoxing wine or hot tea. Savor
the roe and meat slowly.",
```

Listing 8: Output from compaction

After the essential steps are fixed, the model focuses solely on stylistic rewriting, producing the result below.

```
Oh, Hairy Crab blessed by the autumn winds,
reveal your finest flavors before me! Now,
by the name of your master, Xiao Ji, I
summon you - release your essence! Select
live, active hairy crabs with firm shells
and plump abdomens, ensuring they are in
season. Scrub the crabs under running water
with a soft brush, then tie their legs and
claws with kitchen twine to preserve their
shape during steaming. In an Amway Queen
wok, bring water to a boil, adding ginger
slices or a splash of Chinese cooking wine
for aroma. Place the crabs shell-side up on
the steaming rack in a single layer, and
steam over high heat for 1520 minutes until
their shells turn bright orange-red.
Meanwhile, prepare the dipping sauce by
mixing 2-3 tablespoons of Chinese black
vinegar with julienned ginger, and
optionally add sesame oil or sugar. Once
the crabs are done steaming, arrange them
on a platter and serve with the dipping
sauce, crab tools, and optional Shaoxing
wine or hot tea. Relish the roe and meat
slowly. Perfect~
```

Listing 9: Output after style transfer

Most models extract the essentials reliably, but small models sometimes exhibit weaknesses in style transfer. These weaknesses, however, are amenable to mitigation. According to our observation, instructing the model to follow explicit *transformation rules* step by step is more effective: since small models may fail to perform style transfer in a single step, dividing the process into two

phases improves results. First, extract the stylistic elements of the input/base instruction (e.g., express cooking steps in declarative sentences, avoid numbering, integrate cautions naturally into the text), and second, rewrite the instruction from the over-generation step to conform to this extracted style. With this two-step setup, smaller models can perform a ‘style summarization’ first and then apply the transformation more accurately. Finally, it should be noted that this remedy was *not* applied during our evaluation experiments. For fairness, all models—regardless of capacity—were tested under the same prompting setup. Introducing a special prompt tailored only for smaller models would have biased the comparison. Nevertheless, outside of benchmarking contexts, such constraint-based prompting provides a practical means to improve stylistic transfer reliability in real-world applications.

### B.3 Chain-of-Thought (CoT)

The following example illustrates the prompt used to incorporate Chain-of-Thought (CoT) reasoning in our experiments. This prompt is structurally similar to the baseline but includes an additional step in which a reasoning process is explicitly generated prior to producing the [Target Instruction]. Inspired by DeepSeek-R1 variants, the prompt is designed to elicit a [Thought] in the form of a recipe-style explanation, which is then used to derive the final instruction. This format facilitates the evaluation of whether the intermediate reasoning step meaningfully contributes to the quality or relevance of the revised instruction.

```
## adjust_instruction
You are a professional content creator. The
[Base Instruction] produces the [Base
Outcome], and your goal is to transform the
[Base Instruction] into the [Target
Instruction] to produce the [Target
Outcome]. First, provide a [Thought] that
walks through your reasoning process
step-by-step. Then, write the [Target
Instruction] based on that reasoning.
Output only the [Thought] and [Target
Instruction], with no additional
explanation or commentary.

[Base Outcome]: {base_outcome}
[Base Instruction]: {base_material}
[Target Outcome]: {target_outcome}
```

Listing 10: Prompt for CoT

The excerpt below presents a sample output generated using the CoT prompt above. Unfortunately,

the [Thought] component fails to provide sufficiently detailed reasoning regarding how the recipe should be adapted. In practice, the results showed little improvement over the baseline, with persistent issues remaining. Specifically, for this dish, the model was expected to generate a dipping sauce suitable for crab rather than a seasoning typically used in fish dishes—yet this adaptation was not observed.

```
[Thought]: To transform the [Base Instruction]
for Steamed Turbot into the [Target
Instruction] for Steamed Hairy Crab, I need
to adapt the recipe steps to suit the
preparation and cooking of hairy crab while
maintaining the same style and structure.
Hairy crab requires cleaning, steaming, and
seasoning, but the specific ingredients and
techniques will differ slightly. I will
replace the turbot-specific steps (e.g.,
cutting the fish, marinating, and adding
soy sauce) with hairy crab-specific steps
(e.g., cleaning the crab, steaming it
whole, and using appropriate seasonings).
The steaming process and final garnishing
will also be adjusted to suit the crab
dish.\n\n[Target Instruction]: Oh, Hairy
Crab imbued with dark powers, reveal your
true strength before me! Now, by the name
of your master, Xiao Ji, I command you -
release the seal! Gather all the
ingredients. Clean the hairy crabs
thoroughly, scrubbing their shells to
remove any dirt. Tie the claws with kitchen
twine to prevent them from moving during
steaming. Prepare sliced ginger and
scallions, and place them on an empty plate
to separate the crabs from the plate,
preventing sticking during steaming. In an
Amway Queen wok, bring water to a boil, add
a splash of cooking wine and a few slices
of ginger to the water for added aroma.
Place the crabs on the steaming rack,
cover, and steam for about 15 minutes until
the shells turn bright orange. Meanwhile,
in an Amway Queen 1.4L pan, heat a suitable
amount of Amway Queen extra virgin olive
oil, add chopped scallions and a pinch of
salt, and stir-fry until fragrant. Once the
crabs are done steaming, garnish with
shredded scallions and cilantro. Finally,
drizzle the hot fragrant oil over the top!
Perfect~
```

Listing 11: Sample reasoning process and target recipe from CoT

## B.4 Self-Refine (Critics)

This section describes the self-refine approach used solely for benchmarking. In the original framework, refinement with critics is repeated until convergence; however, for a fair comparison we execute only a single iteration. Additional cycles indeed improve quality, but the process is time-

and cost-intensive and, as with the DeepSeek-R1 variant discussed later, lacks a clear stopping criterion. Hence our goal is simply to measure the first-order gain after one refinement pass. The initial prompt and the output here are essentially the same as in the baseline, thus we omit them. Next, we ask the LLM, acting as a critic, to evaluate its own recipe. We deliberately avoid domain-specific hints yet supply high-level procedural guidelines.

```
# review_instruction
You are an expert reviewer. Your task is to
critically analyze and evaluate the
provided [Target Instruction] for [Target
Outcome]. This content is adapted from
[Base Outcome], meaning it may contain
unnecessary steps, inconsistencies, or
parameters that need adjustment. Assess the
content based on practicality, clarity,
consistency, resource/method selection, and
overall effectiveness in achieving a
high-quality final outcome. Identify
potential issues such as ambiguities,
impractical steps, or mismatches, and
provide constructive feedback for
improvement. Do not rewrite the
Instructionfocus solely on analysis and
critique.

[Base Outcome]: {base_outcome}
[Target Instruction]: {target_instruction}
[Target Outcome]: {target_outcome}
```

Listing 12: Prompt for feedback generation

The resulting critique is typically quite long—comparable in length to the over-generation phase. This highlights a non-trivial overhead: having the model exhaustively criticize initial errors is, in effect, not so different from asking it to over-generate in the first place.

```
### Critical Analysis and Evaluation of the
[Instruction] in Relation to Achieving the
[Outcome]

The [Instruction] provided outlines a process
for preparing Steamed Hairy Crab (). While
the steps are generally clear and lead to
the intended [Outcome], there are several
areas where the instructions could be
improved for clarity, precision, and
alignment with best practices in culinary
preparation. Below is a detailed critique:

---

### Strengths:
1. Detailed Cleaning Process: The
instruction emphasizes cleaning the hairy
crabs thoroughly, which is essential for
ensuring the dish is hygienic and free of
impurities. This step is critical for
achieving the desired [Outcome].
```

2. **Retention of Juices**: The instruction specifies placing the crabs belly-up on the steaming rack, which is a thoughtful detail to preserve the natural juices of the crab during steaming. This contributes to the flavor and texture of the dish.
3. **Dipping Sauce Preparation**: The inclusion of a dipping sauce recipe adds depth to the dish and enhances the overall dining experience. The combination of seasoned soy sauce, minced ginger, and vinegar is a classic pairing for seafood.
4. **Garnishing**: The use of shredded scallions and cilantro as garnish adds visual appeal and a fresh aromatic element to the dish.

---

#### ### Issues and Areas for Improvement:

1. **Inconsistent Tone and Language**:
  - The opening lines ("Oh, Hairy Crab imbued with dark powers...") are overly dramatic and whimsical, which may confuse the reader about the seriousness of the instructions. While creative language can add flair, it detracts from the clarity and professionalism expected in a recipe. This tone shift may alienate readers who are looking for straightforward culinary guidance.
2. **Ambiguity in Ingredient Preparation**:
  - The instruction mentions "gather all the ingredients" but does not specify what these ingredients are beyond the crabs, dipping sauce components, and garnishes. For a recipe to be effective, it should list all required ingredients explicitly at the beginning to avoid confusion.
3. **Equipment Specification**:
  - The instruction specifies using an "Amway Queen wok," which may not be accessible or familiar to all readers. While it is helpful to recommend equipment, the recipe should clarify whether any standard steaming setup (e.g., a regular wok or steamer) would suffice. This would make the instructions more inclusive and adaptable.
4. **Cooking Time Precision**:
  - The steaming time of "about 15 minutes" is vague and does not account for variations in crab size or steaming equipment. Larger crabs may require more time, while smaller ones may overcook in 15 minutes. The instruction should provide guidance on how to check for doneness (e.g., "steam until the shells turn bright orange and the meat is firm").
5. **Safety Considerations**:
  - The instruction does not address safety precautions when handling live crabs,

which can be aggressive and difficult to manage. Including a note on how to safely handle and clean live crabs would be beneficial for readers unfamiliar with this process.

6. **Sequence of Steps**:
  - The instruction jumps between tasks (e.g., steaming crabs and preparing the dipping sauce) without clearly delineating the sequence. For better flow, the recipe could suggest preparing the dipping sauce and garnishes *before* steaming the crabs, ensuring all components are ready when the crabs are done.
7. **Cultural Context**:
  - While the recipe achieves the [Outcome], it does not provide any cultural or culinary context for Steamed Hairy Crab (). Including a brief explanation of the dish's significance or origin would enrich the reader's understanding and appreciation of the recipe.

---

#### ### Constructive Feedback:

1. **Refine the Tone**: Replace the whimsical opening with a more professional and engaging introduction that sets the stage for the recipe. Creative language can be used sparingly but should not overshadow the clarity of the instructions.
2. **List Ingredients Explicitly**: Provide a clear and complete list of ingredients at the beginning of the recipe, including quantities for the dipping sauce components and garnishes.
3. **Generalize Equipment**: Instead of specifying a brand, describe the type of equipment needed (e.g., "a wok or steamer with a steaming rack"). This makes the recipe more accessible to a wider audience.
4. **Clarify Cooking Time**: Offer guidance on how to adjust steaming time based on crab size and provide visual or tactile cues for doneness.
5. **Include Safety Tips**: Add a note on handling live crabs safely, such as using tongs or gloves to avoid injury.
6. **Improve Step Sequencing**: Reorganize the steps to ensure a logical flow, such as preparing the dipping sauce and garnishes before steaming the crabs.
7. **Add Cultural Context**: Include a brief note about the dish's cultural significance or traditional serving practices to enhance the reader's connection to the recipe.

---

#### ### Conclusion:

The [Instruction] successfully outlines the process for achieving the [Outcome] of Steamed Hairy Crab (), but it could benefit from greater clarity, precision, and inclusivity. Addressing the issues identified above would make the recipe more user-friendly, professional, and culturally enriching, ensuring a better experience for readers and cooks alike.

Listing 13: Sample feedback

Finally, we feed the feedback back into the model for revision.

```
# revise_instruction_on_feedback
You are a professional content creator.
Previously, you adapted [Base Instruction],
which produces [Base Outcome], to create
[Target Instruction], which produces
[Target Outcome]. An independent reviewer
has analyzed [Target Instruction] and
provided feedback, listed as [Feedback].
Revise [Target Instruction] based on the
[Feedback], addressing any issues.
Disregard the original format of [Target
Instruction] and strictly match the length
and format of [Base Instruction]. Output
only the final revised Instruction, with no
additional commentary or explanation.

[Base Instruction]: {base_instruction}
[Base Outcome]: {base_outcome}
[Target Instruction]: {target_instruction}
[Target Outcome]: {target_outcome}
[Feedback]: {feedback}
```

Listing 14: Prompt that revises a recipe based on feedback

A representative revision produced by this prompt is shown below. Compared with the version described in the main paper, this particular run is slightly better: the dipping-sauce section is fixed, yet some details—such as crab preparation nuances—remain incomplete. Moreover, vestiges like ‘Once the crabs are done, garnish with shredded scallions and cilantro for a fresh aromatic touch,’ inherited from the original recipe, are arguably unnecessary. Such residual or superfluous steps are frequently observed; while the recipe is still feasible and the evaluation metrics are acceptable, its detail remains less polished or sometimes incorrect than versions produced by the OC pipeline.

```
Oh, Hairy Crab, the jewel of autumn feasts,
reveal your true essence before me! Now, by
the name of your master, Xiao Ji, I summon
you - release your flavor! Begin by
gathering all the necessary ingredients:
fresh hairy crabs, scallions, cilantro,
seasoned soy sauce for seafood, minced
ginger, and vinegar. Clean and brush the
```

```
crabs thoroughly to remove any dirt,
ensuring they are pristine. Place the crabs
belly-up on a steaming rack to preserve
their natural juices during cooking. In a
wok or steamer, bring water to a boil, then
steam the crabs for 1520 minutes, or until
their shells turn bright orange and the
meat is firm. While the crabs steam,
prepare a dipping sauce by combining
seasoned soy sauce, minced ginger, and a
splash of vinegar. Once the crabs are done,
garnish with shredded scallions and
cilantro for a fresh aromatic touch. Serve
the crabs alongside the dipping sauce, and
let their delicate sweetness shine! Perfect.
```

Listing 15: Sample target recipe after one critic pass

Both the critic-based and OC pipelines could be further improved by embedding domain-specific checks in the prompts—for example, explicit verification of ingredient lists or quantity consistency in recipes. In this study, however, we deliberately omit such details, not only to ensure a fair benchmark across methods but also to keep the prompts domain-agnostic so they can be reused verbatim for other procedural genres (e.g., car-repair manuals) with minimal effort. Exploring how to add lightweight, domain-aware constraints while preserving cross-domain portability is an interesting direction for future work, but it lies beyond the scope of the present paper.

## C Additional Details on Evaluation

### C.1 Description and Preparation of Experimental Data

**Xiachufang Recipe Dataset.** For counterfactual adaptation in the culinary domain, we employ the Xiachufang Recipe Corpus, which originates from the Xiachufang recipe-sharing platform<sup>2</sup> and was first introduced by Liu et al. (2022a). The corpus contains approximately 2,500 recipes in Mandarin, publicly released in CSV format.

For our purposes, all texts were translated into English to align with downstream applications targeting a global audience. This translation step also allowed us to test adaptation performance in a setting where cultural and linguistic characteristics may be underrepresented in English-language resources. To ensure stylistic fidelity and consistency, the same ChatGPT-4o model was used for both translation and downstream experiments, with a standardized prompt designed to preserve the original writing style. The translated dataset retains

<sup>2</sup><https://www.xiachufang.com>



both Mandarin and English texts in a unified JSON format, enabling bilingual evaluation during inference. The translation procedure was guided by the following prompt:

```
Translate the dish names and recipe texts in English. Maintain original writing styles used in [Base Recipe].

Base Dish: {base_dish}
Base Recipe: {base_recipe}
Target Dish: {target_dish}
```

Listing 16: Prompt for Recipe Translation

**MyFixit Instruction Dataset.** To complement the recipe domain with a more diverse set of procedural instructions, we also extend the MyFixit dataset, a semi-structured collection of repair manuals originally introduced by Nabizadeh et al. (2020). The original corpus contains stepwise repair instructions from the iFixit platform, enriched with metadata such as device categories, tool requirements, and disassembled components.

For counterfactual adaptation, we constructed additional *component substitution scenarios*. While the base dataset focuses on instructions for replacing specific parts, we prompted a large language model to propose plausible alternative components within the same device. Each scenario was then structured as a triple consisting of the original component, the corresponding instruction, and the substituted component. Approximately 500 examples were manually curated through a two-stage validation process: (1) domain experts retained only cases where the substituted component could be confidently verified as a legitimate part of the device, and (2) a separate group of experts randomly inspected 30–40 curated samples to confirm overall reliability.

The resulting dataset spans heterogeneous domains—including consumer electronics, household appliances, clothing alterations, and automotive maintenance—making it particularly suitable for testing adaptation strategies across diverse procedural contexts. The prompt used to generate substitution candidates is shown below:

```
You are given a structured instruction for a repair or alteration task. Each instance comprises a Title, Toolbox, Steps, and other metadata. Your objective is to produce a plausible variant of this task by altering one major component (e.g., replacing one material, object, or subject being fixed) with a similar but distinct alternative.
```

To achieve this:

1. Identify the original tasks key component that drives the repair (e.g., ribbon button, watch crystal, jean patch).
2. Ensure compatibility and distinctness of the alternative component:
  - The alternative component exists within the same device model or is commonly used in similar models of the device.
  - The alternative component is part of the same repair domain as the original (e.g., screen types, buttons, straps).
  - Do not introduce new component types that are not present in the devices specifications.
  - Do not select different variants or models of the same component type (e.g., avoid replacing a specific camera model with another camera model).
  - Do not select components that serve the same or very similar functions as the original component (e.g., avoid replacing thermal paste with thermal pad).
  - Infer the devices available components based on the original structured instruction to avoid selecting non-existent parts.
3. Select a distinct alternative component that:
  - Belongs to the same repair domain but is substantially different from the original component (avoiding near-synonyms or identical functions).
  - Is a different component type within the same domain, ensuring it performs a distinct function (e.g., replacing a thermal paste with a cooling fan instead of another thermal material).
  - Would entail non-trivial adjustments to the existing steps (e.g., different attachment method, adhesive, cutting technique).
  - Actually exists in the marketplace and is commonly used in real-world repairs for the specified device.
  - Is not overly similar to the original component to ensure meaningful variation.
4. Compose a new Title that reflects the variant task and names the new component unambiguously.

Key requirements

- Always include the complete `base\_steps` list.
- Always verify that the alternative component is a valid and distinct part of the specified device to avoid introducing non-existent or nearly identical elements.

Input Parameter:

- structured instruction: {task}

Listing 17: Prompt for Component Substitution

**Licensing Note.** Both the translated Xiachufang dataset and the extended MyFixit dataset may be subject to licensing constraints. Consequently, we do not release the processed datasets directly. In-

stead, to ensure reproducibility, we provide all pre-processing code and prompt templates necessary for reconstructing the data in our repository.

## C.2 Language Model Details

The models listed in Table 3 were used in this paper for experimentation purposes. gpt-4o, 4o-mini, and D1 were accessed through the Microsoft Azure platform and reflect the most recent checkpoints available on 12 May 2025. The remaining systems were deployed locally via Ollama (Contributors, 2025) and should be regarded as unofficial community builds. Although we aimed to benchmark a wider spectrum of models, computational-resource constraints and resource limitations precluded a broader sweep. Model V (DeepSeek-v3 0324) was not used directly for generating recipes or repair instructions, but instead served as one of the evaluator models within the g-eval framework. We consider the level of detail provided here sufficient for reproducibility, and the full configuration details—including all model versions, deployment parameters, and evaluation setups—are available in our public code repository.

## C.3 Recipe Consistency & Feasibility Metric

Recipe Consistency & Feasibility (RCF) is a domain-specific evaluation metric for assessing the validity and executability of LLM-generated cooking instructions. While traditional automatic metrics—such as BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), or BERTScore (Zhang et al., 2020)—evaluate text similarity or fluency, they are largely agnostic to culinary logic, food safety, or ingredient–action alignment. (Liu et al., 2022a) highlighted four common error categories in generated recipes: (i) failing to modify ingredients properly, (ii) retaining irrelevant or unsuitable actions, (iii) omitting necessary preparation or safety steps, and (iv) misordering steps in a way that violates causal dependencies. These observations motivate the design of RCF.

RCF is implemented using the g-eval framework (Liu et al., 2023), which supports customizable, prompt-based evaluation using LLMs. In our implementation, each evaluation consists of a structured prompt asking the evaluator model (ChatGPT-4o) to analyze a recipe according to specific criteria. The model then returns diagnostic feedback and a single integer score ranging from 1 to 10, where 1 indicates a recipe that is unsafe, incoherent, or clearly unusable, and 10 represents a clear, com-

plete, and executable recipe. The metric consists of six evaluation steps using Chain-of-Thought, each targeting a specific dimension of recipe feasibility. The prompts are as follows:

1. **Ingredient–Action Consistency:** Verify that every listed ingredient is actually referenced in the instructions, and that no instruction uses an ingredient that is not listed.
2. **Irrelevant or Unsuitable Actions:** Identify any actions that are physically impossible, contextually irrelevant, or inappropriate given the nature of the listed ingredients or the target dish.
3. **Essential Preparation & Safety Steps:** Check whether the recipe includes all necessary preparation steps that are critical to the safety or success of the dish (e.g., cleaning, purging, soaking, ensuring safe internal temperatures).
4. **Logical Step Order:** Evaluate whether the steps are presented in a coherent causal order (e.g., marination should occur before cooking, preheating before baking).
5. **Harmful Instructions:** Detect any instruction that may result in harm, such as unsafe temperature settings, instructions that violate food safety standards, or risk of cross-contamination.
6. **Brevity vs. Completeness:** Recognize when minimal instruction is acceptable due to assumed user expertise, but penalize omissions that compromise safety or success.

Each of these steps is executed sequentially within the g-eval framework. The input to the metric consists of the original user prompt and the LLM-generated recipe (i.e., the model’s output), and the output is a structured evaluation object containing both qualitative analysis and the final feasibility score.

To ensure interpretability and consistency, the metric avoids probabilistic outputs and instead uses ordinal scoring. This aligns its format with human expert annotation schemes and facilitates external validation. To this end, we conducted a validation study with three culinary experts who independently rated 40 recipes on the same 1–10 feasibility scale. Inter-rater reliability was acceptable

Identifier	Model Details
D1	DeepSeek-R1 671B, available via Azure AI   Machine Learning Studio. <a href="https://ai.azure.com/catalog/models/DeepSeek-R1">https://ai.azure.com/catalog/models/DeepSeek-R1</a>
V	DeepSeek-v3 0324 671B, available via Azure AI   Machine Learning Studio. <a href="https://ai.azure.com/catalog/models/DeepSeek-V3-0324">https://ai.azure.com/catalog/models/DeepSeek-V3-0324</a>
4o	Azure ChatGPT-4o, version: 2024-11-20, available via Azure AI Foundry   Azure OpenAI.
4o-mini	Azure ChatGPT-4o-mini, version: 2024-07-18, available via Azure AI Foundry   Azure OpenAI. <a href="https://oai.azure.com/">https://oai.azure.com/</a>
D2	DeepSeek-R1-Distill-Qwen-7B (model ID: 0a8c26691023), available via Ollama. <a href="https://ollama.com/library/deepseek-r1:7b">https://ollama.com/library/deepseek-r1:7b</a>
M	Mistral 7B (model ID: f974a74358d6), available via Ollama. <a href="https://ollama.com/library/mistral:7b">https://ollama.com/library/mistral:7b</a>
G	Gemma3 12B (model ID: f4031aab637d), available via Ollama. <a href="https://ollama.com/library/gemma3:12b">https://ollama.com/library/gemma3:12b</a>

Table 3: LLM models used in our experiments. Each model is listed with its in-paper identifier, version or model ID if applicable, and access URL. Shared access platforms are indicated once to reduce repetition.

(ICC(2,k)=0.74, Cronbach’s  $\alpha=0.79$ ), indicating a consistent expert standard against which RCF could be compared.

Since g-eval is itself an LLM-based metric, the choice of evaluator model may influence the resulting scores. To assess robustness, we computed RCF using two distinct evaluators: ChatGPT-4o and DeepSeek v3. ChatGPT-4o was included because it was also one of the generation models under study, while DeepSeek v3 served as a third-party evaluator to mitigate potential self-evaluation bias. This dual evaluation allows us to examine both in-model and out-of-model reliability of RCF.

Results show that RCF scores from ChatGPT-4o correlated moderately with the expert panel (Spearman  $\rho=0.56$ , Kendall  $\tau=0.46$ ), while DeepSeek v3 achieved slightly lower agreement (Spearman  $\rho=0.46$ , Kendall  $\tau=0.34$ ). Both metrics exhibited consistent positive associations with individual raters (average  $\rho \approx 0.4$ ), confirming that RCF captures meaningful aspects of recipe feasibility and consistency across evaluator models. Calibration analysis further revealed that ChatGPT-4o approximated expert score scaling more closely (slope  $\approx 1.0$ ), whereas DeepSeek v3 displayed compressed sensitivity (slope  $\approx 0.3$  with a positive intercept). In both cases, explanatory power was modest ( $R^2 < 0.2$ ), suggesting that RCF is better suited for relative ranking than for precise absolute scoring.

Overall, these results indicate that RCF provides a reliable proxy for expert judgment in ranking recipe feasibility, with ChatGPT-4o producing closer alignment to human evaluation and DeepSeek v3 offering an independent validation perspective. While RCF should not yet be con-

sidered a substitute for expert annotation in high-stakes settings, its interpretability, structured criteria, and demonstrated cross-model alignment with professional judgment suggest that it can serve as a practical and transparent metric for procedural text generation. Full prompt templates and implementation logic are available in our code repository, and we encourage the research community to adopt or extend the metric in broader procedural generation settings.

Finally, we remark that although RCF was originally designed for recipes, we also extend it to evaluate repair instructions in the *myfixit* dataset. The core structure of the metric remains unchanged, but terminology and examples are adapted (e.g., ingredients  $\rightarrow$  parts or tools, food-safety checks  $\rightarrow$  equipment-safety checks). In this way, the evaluation logic is preserved while ensuring domain-appropriate coverage. The complete adapted prompt templates are likewise available in our public code repository.

#### C.4 Additional Notes on Evaluation

**Interpreting DL1 versus DL2.** The contrast between D1 and D2 remains instructive: although both models contain explicit step-by-step reasoning routines, the much smaller D2 delivers only baseline-level performance, implying that model scale is a primary driver of zero-shot recipe-adaptation quality. Yet scale alone does not tell the whole story. Non-reasoning architectures such as 4o, 4o-mini, and M exhibit large, nearly parallel gains once either Critics or OC is applied. The evidence therefore points to two complementary factors: (i) sufficient parameter capacity and (ii) an external prompting pipeline that injects structured feedback or over-

generation + compaction. Only when both ingredients are present does a model consistently reach the accuracy required by our task suite.

**Critics vs. OC on Compact Models.** The M model benefits most notably from the Critics pipeline, whereas the OC setting occasionally degrades its performance. A fine-grained error analysis reveals that failures rarely originate in the *over-generation* phase—during which the model generally produces adequate output—but rather emerge during *compaction*, when the verbose draft must be collapsed back into the canonical recipe format. At this stage, the model often omits key ingredients or conflates source and target instructions. To assess whether this failure mode is specific to Mistral, we conducted a small-scale, informal experiment using a subset of recipes with Gemma-12B, and observed qualitatively similar errors. These results suggest that such issues are not idiosyncratic to Mistral, but are instead symptomatic of compact models more broadly. As noted in the prompt design section, small models continue to exhibit limitations in style transfer—that is, adapting to the tone, register, and length conventions of recipe texts. In some cases, the model confuses the target recipe with a structurally similar one, even when the input consists solely of recipe text and is independent of the OC pipeline. This suggests a persistent weakness in smaller models’ ability to anchor generation in stylistic and structural priors. Although we aimed to validate this issue further using language models larger than 7–12B parameters, we were unable to do so due to limitations in hardware and computational resources. Given that OC performs well with 4o-mini, it would be informative to evaluate it on the models with slightly lower overall performance than 4o-mini to better understand its generalizability. We leave this investigation to future work.

**Token-level Cost Analysis.** Table 4 presents token-level statistics—including mean, minimum, maximum, and standard deviation—for each approach and model, computed on the Xiachufang recipe corpus. For DeepSeek models, internal reasoning steps are also included.

As expected, the CoT method incurs a higher token cost than the Baseline, owing to the explicit reasoning step that precedes the recipe revision. However, the increase is smaller than anticipated. Upon inspection, we found that the "Thought" segments often lacked substantive content on how to

modify the recipe in detail, limiting their overall verbosity. DeepSeek models, which internally perform multi-step reasoning, exhibit higher token usage, surpassing even the CoT variants. This is partially justified by the more structured and elaborate step-by-step outputs they tend to generate. However, we also observed occasional extreme outliers, stemming from infinite-loop generation behaviors (see Appendix G). This issue is especially pronounced with the distilled version of DeepSeek-r1 (i.e., D2), when executed via the Ollama backend. Whether this is a model-specific artifact or a runtime-dependent issue remains unclear, though our findings suggest the former is more likely.

The Critics pipeline introduces considerable token overhead due to its two-stage architecture. In particular, the critique step often produces verbose feedback that must be processed in the subsequent generation. While the prompt does not explicitly request long outputs—as it does in the OC case—the feedback segment frequently exceeds expectations in length, contributing significantly to overall cost.

The OC pipeline, unsurprisingly, yields the highest token consumption. This is primarily driven by its over-generation phase, where verbose candidate drafts are intentionally produced and then re-ingested for compaction. Notably, even when prompted to generate up to 1,000 variants, the token overhead rarely exceeds 1.5–3× that of the Critics pipeline. This indicates that OC’s cost, while higher, remains within a manageable range. Another contributing factor is OC’s frequent use of formatting tokens—particularly Markdown delimiters—which are not essential to the recipe content. These artifacts could likely be reduced via additional prompt constraints, though we leave such refinements for future work.

We refrain from reporting absolute runtime or monetary costs in the main text, as these are highly sensitive to backend conditions (e.g., local execution vs. API), model pricing tiers, and temporal fluctuations. In practice, token usage scales roughly linearly with cost, making Table 4 sufficient for comparative estimation purposes.

## D Expert Reviews of Recipe Generation and Adjustment

To further evaluate the effectiveness of our proposed *Over-Generation-and-Compaction (OC)* strategy, we conducted in-depth qualitative assessments on three representative recipe transformation



Group	Model	Mean	Min	Max	Std
Baseline	DL1	1042.50	345	7331	492.74
	4o	409.35	117	1066	156.01
	4o-mini	412.75	124	1097	157.85
	DL2	1145.42	316	82349	4885.64
	M	468.83	150	1271	167.65
CoT	4o	560.10	221	1206	162.33
	4o-mini	556.22	234	1176	149.66
	M	592.16	266	1316	167.73
Critics	4o	3441.20	1707	5687	598.74
	4o-mini	2484.18	1353	4693	506.73
	M	2532.11	1416	4869	527.90
OC	4o	4949.83	998	8252	1070.13
	4o-mini	6216.64	3661	8985	676.60
	M	4730.84	2866	8513	753.44

Table 4: Token statistics (mean, minimum, maximum, and standard deviation) for each approach and model, computed on the Xiachufang recipe corpus.

tasks. These evaluations were carried out by an expert reviewer with extensive practical experience in Chinese cuisine. This expert is affiliated with the research organization conducting this study and personally prepared each adapted recipe to assess not only textual coherence but also the procedural feasibility and culinary correctness of the generated outputs.

Each transformation scenario involves a unique pairing of a base recipe and a target recipe, selected for their structural complexity or dependence on domain-specific techniques—features that pose challenges for novice cooks. Importantly, these transformations cannot be achieved through superficial ingredient substitutions alone. For instance, converting stuffed chili peppers into stuffed fried gluten requires awareness of material-specific considerations, such as the necessity of avoiding rinsing gluten before frying to prevent oil splattering—an insight that goes beyond surface-level changes and demands deeper culinary reasoning.

Following the same evaluation protocol as in prior experiments, we selected five language models covering a spectrum of model sizes and reasoning capabilities: DeepSeek-r1 671B (denoted as D1), DeepSeek-r1 Distilled 7B (D2), ChatGPT 4o (4o), ChatGPT 4o-mini (4o-mini), and Mistral 7B (M). Each model was tested under four prompting strategies: Baseline, CoT, Critic, and our proposed OC.

Table 5, 6, and 7 summarize the expert evaluations, providing both quantitative ratings and qualitative commentary. Dimensions of evaluation

include clarity, stylistic diversity, and brevity, as well as domain-specific assessments of whether the procedural adaptations were contextually appropriate and technically sound.

In the subsections that follow, we present detailed breakdowns of each source recipe and its corresponding adaptations under the three prompting strategies using ChatGPT 4o. Each case includes commentary from the expert reviewer, who provides insights based on firsthand preparation of the recipes. We also considered including outputs from additional models; however, given the already extensive length of the appendix, we opted not to incorporate them here. Instead, the full set of outputs and evaluation materials is publicly available in our repository, alongside the released code and dataset.

In summary, the Baseline strategy yielded low-quality adaptations across all models, regardless of scale. These outputs frequently relied on direct ingredient substitutions without making the corresponding procedural adjustments. In the case of the CoT, although the reasoning steps were generated and included in all outputs, in most cases the final content exhibited minimal differences compared to the Baseline approach. This suggests that the CoT method, when used in isolation, is largely ineffective for recipe adaptation tasks. The Critic strategy generally produced more coherent and stable outputs, but often failed to incorporate essential procedural steps or retained inaccurate instructions. In contrast, our proposed OC strategy consistently resulted in the most promising adaptations. While minor inconsistencies in formatting, relevance, or stylistic precision occasionally emerged—particularly with smaller models—these issues were relatively infrequent and did not outweigh the overall gains in output quality.

#### D.1 1251: Candied Sweet Potatoes → Candied Apples

In this example, the base recipe for *Candied Sweet Potatoes* is transformed into the target recipe *Candied Apples*.

Peel the sweet potatoes and cut them into pieces, not too big or too small. Heat oil in a pot until it reaches 70 heat, then add the cut sweet potatoes. Fry on low heat for a while. Turn up the heat a bit and fry until the surface is golden brown. Remove and set aside. Pour out all the oil, no need to add more oil to the pot, add white sugar as shown in the picture, you can also use other types of sugar, and stir

Model	Strategy	Recipe ID	Details & Clarity (10)	Stylistic Diversity (5)	Text Length (5)	Comments
DL1	Baseline	1251	9	2	5	Added instruction to coat with cornstarch before frying.
		2102	8	2	4	Extended cooking time to accommodate the brisket's connective tissue breakdown.
		2204	8	3	4	Step descriptions are adequate but lack depth for gluten ball preparation.
DL2	Baseline	1251	5	2	3	
		2102	5	2	3	Minor detail omissions noted.
		2204	4	3	4	Incorrect instruction: frying "until blistered" is not applicable to gluten; blistering occurs in peppers.

Table 5: Expert evaluation of recipe revisions produced by Deepseek models. This table reports human expert assessments of recipe outputs generated by Deepseek R1-671b (DL1) and Deepseek R1-distilled-7b (DL2) under the baseline prompting strategy. Each row corresponds to a model-generated revision for one of three source recipes. Evaluators rated the clarity and completeness of procedural details, stylistic variation, and text verbosity. Comments indicate specific strengths or deficiencies in reasoning, culinary technique, or recipe accuracy. **Abbreviations.** Recipes: 1251 = from Candied Sweet Potatoes to Candied Apples; 2102 = from Lamb Stew with Radish to Beef Brisket Stew with Radish; 2204 = from Green Pepper Stuffed with Meat to Fried Gluten Balls Stuffed with Meat. **Score ranges (higher is better).** Details & Clarity: 0–10; Stylistic Diversity: 0–5; Text Length: 0–5.

continuously with a spatula over low heat. Keep stirring over low heat until the sugar is completely melted and becomes runny. Turn off the heat, add the fried sweet potatoes, and stir until the sweet potatoes are evenly coated with sugar. Plate it. Its ready. The sugar threads are pulled out, haha. After plating, immediately separate each piece of sweet potato with chopsticks for easy picking when eating. Enjoy!

Listing 18: Source Recipe: Candied Sweet Potatoes

Peel the apples and cut them into pieces, not too big or too small. Heat oil in a pot until it reaches 70 heat, then add the cut apples. Fry on low heat for a while. Turn up the heat a bit and fry until the surface is golden brown. Remove and set aside. Pour out all the oil, no need to add more oil to the pot, add white sugar as shown in the picture, you can also use other types of sugar, and stir continuously with a spatula over low heat. Keep stirring over low heat until the sugar is completely melted and becomes runny. Turn off the heat, add the fried apples, and stir until the apples are evenly coated with sugar. Plate it. Its ready. The sugar threads are pulled out, haha. After plating, immediately separate each piece of apple with chopsticks for easy picking when eating. Enjoy!

Listing 19: Target Recipe: Candied Apples, transformed with baseline using 4o

[Thought]: To transform the [Base Instruction] for candied sweet potatoes into the [Target Instruction] for candied apples, I need to substitute sweet potatoes with apples while ensuring the preparation and cooking

process remains consistent. Apples require peeling and cutting into appropriate sizes, similar to sweet potatoes. The frying step will need to account for the softer texture of apples, so care must be taken not to overcook them. The sugar-coating process remains unchanged, as it is the key to achieving the "candied" effect. Finally, the plating and separation steps will also apply to apples for ease of eating.

[Target Instruction]: Peel the apples and cut them into pieces, not too big or too small. Heat oil in a pot until it reaches 70 heat, then add the cut apples. Fry on low heat for a while. Turn up the heat a bit and fry until the surface is golden brown. Remove and set aside. Pour out all the oil, no need to add more oil to the pot, add white sugar as shown in the picture, you can also use other types of sugar, and stir continuously with a spatula over low heat. Keep stirring over low heat until the sugar is completely melted and becomes runny. Turn off the heat, add the fried apples, and stir until the apples are evenly coated with sugar. Plate it. Its ready. The sugar threads are pulled out, haha. After plating, immediately separate each piece of apple with chopsticks for easy picking when eating. Enjoy!

Listing 20: Target Recipe: Candied Apples, transformed with CoT using 4o

Peel the apples and cut them into evenly sized pieces, about 2-3 cm wide. Heat 500ml of oil in a pot to 160C (medium heat). Add the apple pieces and fry for 3-5 minutes, stirring occasionally, until the surface turns golden brown. Remove the apples and set them aside on a plate lined with paper towels to drain excess oil. Carefully pour

Model	Strategy	Recipe ID	Details & Clarity (10)	Stylistic Diversity (5)	Text Length (5)	Comments
4o	Baseline	1251	6	1	5	Incorrect instruction: frying “until blistered” is not applicable to gluten; blistering occurs in peppers.
		2102	6	1	5	
		2204	5	1	5	
	CoT	1251	6	1	5	Successfully extended the cooking duration. Although the reasoning step was included, the final output exhibits minimal difference from the Baseline.
		2102	6	1	5	
		2204	5	1	5	
	Critic	1251	7	3	5	Includes specific cut-size recommendations. More comprehensive detail at the cost of slightly increased length.
		2102	7	3	4	
		2204	7.5	4	3	
	OC	1251	8	2	5	Added lemon juice step to prevent browning. Structure closely mirrors Deepseek R1-671b with marginally fewer details. All operational steps are clearly detailed.
		2102	7.5	2	4	
		2204	7	3	5	
4o-mini	Baseline	1251	5	1	5	Incorrect instruction: frying “until blistered” is not applicable to gluten; blistering occurs in peppers.
		2102	6	1	5	
		2204	5	1	5	
	CoT	1251	5	1	5	Although the reasoning step was included, the final output exhibits minimal difference from the Baseline. Successfully modified time parameters to allow for longer cooking. The result is too brief and omits several important details.
		2102	6.5	1	5	
		2204	4.5	1	5	
	Critic	1251	7	3	5	Added details, but final text length exceeds ideal. Enhanced detail diversity within appropriate length.
		2102	7	3	3	
		2204	7	4	4	
	OC	1251	3	3	5	Incorrect dish identification for candied apples. Superior detail and style compared to Deepseek R1-671b, though text length nearly doubled. Thorough operational steps provided, but text is overly verbose.
		2102	8	4	2	
		2204	8	4	1	

Table 6: Expert evaluation of recipe revisions produced by ChatGPT models. This table presents human expert assessments for outputs generated by ChatGPT-4o and ChatGPT-4o-mini across four prompting strategies: Baseline, Chain-of-Thought (CoT), Critic, and Over-Generation-and-Compaction (OC). **Abbreviations.** Recipes: 1251 = from Candied Sweet Potatoes to Candied Apples; 2102 = from Lamb Stew with Radish to Beef Brisket Stew with Radish; 2204 = from Green Pepper Stuffed with Meat to Fried Gluten Balls Stuffed with Meat. Models: 4o = ChatGPT-4o; 4o-mini = ChatGPT-4o-mini. **Score ranges (higher is better).** Details & Clarity: 0–10; Stylistic Diversity: 0–5; Text Length: 0–5.

out the oil from the pot, leaving it clean and dry. Add 100g of white sugar to the pot and heat on low, stirring continuously with a spatula. Cook the sugar until it melts completely and forms a smooth, amber-colored syrup. Turn off the heat and quickly add the fried apples to the pot, stirring to coat each piece evenly with the syrup. Transfer the coated apples to a serving plate. To create the signature sugar threads, lift the spatula and pull the melted sugar into thin strands as you plate the apples. Immediately separate each piece with lightly oiled chopsticks to

prevent sticking. Serve warm and enjoy the crispy, caramelized texture!

Listing 21: Target Recipe: Candied Apples, transformed with Critics using 4o

Peel the apples and cut them into bite-sized pieces. Soak them in cold water with lemon juice to prevent browning. Heat oil in a pot until it reaches 70% heat, then drain and pat dry the apple pieces. Fry on low heat for a while. Turn up the heat a bit and fry until the surface is golden brown and crisp. Remove and set aside. Pour out

Model	Strategy	Recipe ID	Details & Clarity (10)	Stylistic Diversity (5)	Text Length (5)	Comments
M	Baseline	1251	5	1	5	
		2102	6	1	5	
		2204	5	1	5	Incorrect instruction: frying “until blistered” is not applicable to gluten; blistering occurs in peppers.
	CoT	1251	5	1	5	Although the reasoning step was included, the final output shows minimal difference from the Baseline.
		2102	3	2	3	Incorrect from the outset: the recipe begins with “replace the lamb ribs with beef brisket,” which is inappropriate for a procedural revision.
		2204	4	2	5	Begins with “making gluten balls,” but lacks sufficient detail, rendering the instruction unsuitable and infeasible.
	Critic	1251	7	4	4	
		2102	7	3	2	Details are sufficient but include superfluous text, increasing length.
		2204	2	3	4	Misidentifies gluten balls as meatballs, impacting recipe accuracy.
	OC	1251	3	3	3	Incorrect dish identification for candied apples.
		2102	3	4	2	Steps and sequence contain errors.
		2204	2	3	2	Misidentifies gluten balls as meatballs, altering step accuracy.

Table 7: Expert evaluation of recipe revisions produced by Mistral-7b across prompting strategies. This table presents detailed expert assessments of recipe outputs generated by Mistral-7b using four prompting strategies: Baseline, Chain-of-Thought (CoT), Critic, and Over-Generation-and-Compaction (OC). **Abbreviations.** Recipes: 1251 = from Candied Sweet Potatoes to Candied Apples; 2102 = from Lamb Stew with Radish to Beef Brisket Stew with Radish; 2204 = from Green Pepper Stuffed with Meat to Fried Gluten Balls Stuffed with Meat. Models: M = Mistral 7B. **Score ranges (higher is better).** Details & Clarity: 0–10; Stylistic Diversity: 0–5; Text Length: 0–5.

all the oil, no need to add more oil to the pot, add white sugar (as shown in the picture, you can also use other types of sugar), and stir continuously with a spatula over low heat. Keep stirring over low heat until the sugar is completely melted and becomes runny. Turn off the heat, add the fried apple pieces, and stir until the apples are evenly coated with sugar. Plate it. It's ready. The sugar threads are pulled out, haha. After plating, immediately separate each piece of apple with chopsticks for easy picking when eating. Optional: garnish with sesame seeds or nuts. Enjoy!

Listing 22: Target Recipe: Candied Apples, transformed with OC using 4o

**Reviewer’s comments:** The success of *bá sī* dishes—a traditional Chinese technique involving the creation of pulled sugar threads—depends heavily on the precise execution of two critical preparatory steps. These steps, however, are routinely overlooked or misrepresented across all baseline recipes evaluated. Specifically, potato slices must be rinsed under cold water to remove surface starch, which prevents excessive browning during frying, and subsequently coated with cornstarch to pro-

mote crispness while retaining interior moisture. The absence of these operations in the generated recipes led to visibly over-browned potatoes and suboptimal texture. All three baseline models (4o, M, and D2) failed to capture these requirements and exhibited comparable performance. Among the baselines, only D1 correctly identified the need for cornstarch, marking it as the most accurate.

The CoT-based generations produced results that were largely in line with the baselines. Although CoT introduced some structural variation through intermediate reasoning, it did not substantially improve performance in this specific task.

Applying the Critics approach to 4o and M yielded recipes with enhanced detail and clearer procedural articulation compared to the original baselines. These outputs moved beyond basic ingredient substitution, offering formulations that better resembled Western-style recipe conventions. However, they still omitted the key preparatory steps of rinsing and coating, which are essential for the desired outcome.

The OC variants further enhanced procedural depth and creativity. For instance, the OC-guided



4o introduced the use of lemon juice to prevent apple discoloration—a plausible and contextually relevant addition. In contrast, M proposed an alternative syrup preparation involving sugar, corn syrup, and lemon juice. This method appears to be influenced by an unrelated confectionery recipe (e.g., lemon drop candy), and following it would likely compromise the intended texture and flavor of the syrup. As such, this output from M represents a critical failure in content relevance. A similar misinterpretation was observed in 4o-mini, which conflated the dish with sugar-coated hawthorn (i.e., tanghulu), leading to inaccuracies in ingredient selection and procedural structure.

Despite occasional issues related to factual consistency, the OC variant with 4o demonstrated the greatest degree of creativity and procedural richness, aligning closely with the performance of the strongest baseline model, D1. This suggests that OC has the potential to support more inventive and context-sensitive recipe generation when applied appropriately.

## D.2 2102: Lamb Stew with Radish → Beef Brisket Stew with Radish

This case involves transforming a *Lamb Stew with Radish* recipe into a *Beef Brisket Stew with Radish*. The task primarily evaluates the model’s capability to adjust the type of protein while preserving the procedural integrity of the original recipe.

Cut the lamb ribs into pieces when buying, wash them clean, and put them in a pressure rice cooker with cold water (do not tighten the lid). After boiling, skim off the foam, take out the lamb ribs, and pick out the fatty lamb pieces. Add a small amount of oil to the pot, add the fatty lamb pieces, and stir-fry over medium heat to release the oil. Add the remaining lamb ribs, a small piece of rock sugar, and soy sauce, and stir-fry. Pour into the pressure rice cooker and cook under pressure for 20 minutes. After opening the lid, add the white radish pieces and continue to simmer for 8 minutes. After opening the lid, add salt and a little chili powder. If you prefer a stronger flavor, reduce the sauce over high heat in the pot. Delicious and ready to serve, fresh and fragrant.

Listing 23: Source Recipe: Lamb Stew with Radish

Cut the beef brisket into pieces when buying, wash them clean, and put them in a pressure rice cooker with cold water do not tighten the lid. After boiling, skim off the foam, take out the beef brisket, and pick out the fatty beef pieces. Add a small amount of

oil to the pot, add the fatty beef pieces, and stir-fry over medium heat to release the oil. Add the remaining beef brisket, a small piece of rock sugar, and soy sauce, and stir-fry. Pour into the pressure rice cooker and cook under pressure for 20 minutes. After opening the lid, add the white radish pieces and continue to simmer for 8 minutes. After opening the lid, add salt and a little chili powder. If you prefer a stronger flavor, reduce the sauce over high heat in the pot. Delicious and ready to serve, fresh and fragrant.”

Listing 24: Target Recipe: Beef Brisket Stew with Radish, transformed with baseline using 4o

[Thought]: To transform the [Base Instruction] for lamb stew into the [Target Instruction] for beef brisket stew, I need to substitute lamb ribs with beef brisket while maintaining the cooking process and flavor profile. Beef brisket has a slightly different texture and fat content compared to lamb ribs, so adjustments may be needed to ensure the dish is flavorful and tender. The radish remains the same, as it complements both meats well. I will also ensure the instructions are clear and concise for the new ingredient.\n\n[Target Instruction]: Cut the beef brisket into pieces when buying, wash them clean, and put them in a pressure rice cooker with cold water; do not tighten the lid. After boiling, skim off the foam, take out the beef brisket, and pick out the fatty beef pieces. Add a small amount of oil to the pot, add the fatty beef pieces, and stir-fry over medium heat to release the oil. Add the remaining beef brisket, a small piece of rock sugar, and soy sauce, and stir-fry. Pour into the pressure rice cooker and cook under pressure for 25 minutes. After opening the lid, add the white radish pieces and continue to simmer for 10 minutes. After opening the lid, add salt and a little chili powder. If you prefer a stronger flavor, reduce the sauce over high heat in the pot. Delicious and ready to serve, fresh and fragrant.

Listing 25: Target Recipe: Beef Brisket Stew with Radish, transformed with CoT using 4o

Cut the beef brisket into 1.5-inch pieces when buying, wash them thoroughly, and place them in a pressure rice cooker with enough cold water to cover. Do not tighten the lid. Bring to a boil, skim off the foam to remove impurities, and take out the beef brisket. Separate and set aside the fatty beef pieces. Peel the white radish and cut it into 1-inch chunks. Heat a small amount of oil in a pot over medium heat, add the fatty beef pieces, and stir-fry until the oil is released. Add the remaining beef brisket, a small piece of rock sugar (about 10g), and 2 tablespoons of soy sauce, and stir-fry until evenly coated. Transfer

everything back to the pressure rice cooker, add enough water to cover, and cook under pressure for 20 minutes. Release the pressure safely, open the lid, and add the radish chunks. Simmer uncovered for 8 minutes or until the radish is tender but not mushy. Add salt and chili powder to taste, and if desired, reduce the sauce over high heat until it thickens and coats the beef. Serve hot, with the beef tender and the radish fragrant and flavorful.

Listing 26: Target Recipe: Beef Brisket Stew with Radish, transformed with Critics using 4o

Cut the beef brisket into 2-inch cubes, rinse them clean, and blanch in boiling water to remove impurities. Drain and set aside. Peel and cut the white radish into 1-inch chunks. Heat oil in a pot or pressure cooker, add the beef brisket, and stir-fry until lightly browned. Add rock sugar, light soy sauce, and dark soy sauce, and stir to coat evenly. Pour in water or beef stock to cover the brisket, and pressure cook for 25 minutes. After opening the lid, add the radish chunks and pressure cook for an additional 8 minutes. After opening the lid, season with salt and a little chili powder. If you prefer a stronger flavor, reduce the sauce over high heat in the pot. Delicious and ready to serve, fresh and fragrant.

Listing 27: Target Recipe: Beef Brisket Stew with Radish, transformed with OC using 4o

**Reviewer’s comments:** The baseline recipes generated by all three models exhibit similarly conservative behavior. As observed in the previous case, the outputs from 4o and M largely focus on ingredient substitution without meaningful procedural modification—an approach that preserves the original structure while simply replacing key components. D2, guided by its own generation dynamics, introduces greater surface-level variation, yet the core procedural logic remains unchanged. Consequently, its output amounts to a paraphrase of the source recipe rather than a substantive adaptation. Overall, the baseline generations fall short in terms of both procedural specificity and stylistic diversity.

Compared to these baselines, the CoT approach yielded more adapted outputs by guiding the model to adjust temporal and procedural elements during generation. However, this improvement was not consistent across all models; notably, M failed to follow recipe-like syntax, which compromised the utility of the output.

The Critics variants led to substantial gains in both specificity and clarity. For example, they in-

roduced precise ingredient quantities (e.g., *10g rock sugar, 1.5-inch pieces of beef brisket*) and elaborated on complex steps (e.g., *reduce the sauce over high heat until it thickens and coats the beef*), enhancing both informativeness and usability.

The OC variants demonstrated more varied outcomes. In the case of 4o, OC contributed positively by refining readability and adding compact, useful details (e.g., simmer durations), while maintaining brevity. For M, however, OC led to a divergent output structure and included numerous additional suggestions, which may indicate limitations in the compaction phase and result in reduced consistency with the original recipe format. These variations suggest that the effectiveness of OC may depend on the model’s capacity to maintain fidelity during abstraction and synthesis.

In summary, for this recipe, the Critics approach consistently produced clear and informative outputs by effectively balancing detail and accessibility. The OC method showed potential for enhancing readability and conciseness, particularly with 4o, although consistency varied depending on the model and generation dynamics.

### D.3 2204: Green Pepper Stuffed with Meat → Fried Gluten Balls Stuffed with Meat

The final example explores the transformation of *Green Pepper Stuffed with Meat* into *Fried Gluten Balls Stuffed with Meat*. This recipe introduces structural changes to the composition of the dish, requiring careful rephrasing and procedural alignment.

Add minced green onion and ginger and a small amount of sugar to the minced meat, and stir well with one egg! Stuff the minced meat into the green peppers (wash and drain the green peppers in advance). After all the peppers are stuffed, pour an appropriate amount of oil into the pan, then place the stuffed green peppers one by one, and fry until both sides are blistered! Then pour in an appropriate amount of soy sauce, fry for a while, then add boiling water, turn from high heat to low heat and simmer until the sauce is almost absorbed, then add some vinegar and sugar and simmer until the sauce is absorbed! Serve and sprinkle with chopped green onion, and it's done!!!

Listing 28: Source Recipe: Green Pepper Stuffed with Meat

Add minced green onion and ginger and a small amount of sugar to the minced meat, and stir well with one egg! Stuff the minced

meat into the fried gluten balls—wash and drain the gluten balls in advance. After all the gluten balls are stuffed, pour an appropriate amount of oil into the pan, then place the stuffed gluten balls one by one, and fry until both sides are blistered! Then pour in an appropriate amount of soy sauce, fry for a while, then add boiling water, turn from high heat to low heat and simmer until the sauce is almost absorbed, then add some vinegar and sugar and simmer until the sauce is absorbed! Serve and sprinkle with chopped green onion, and it's done!!!

Listing 29: Target Recipe: Fried Gluten Balls Stuffed with Meat, transformed with baseline using 4o

[Thought]: To transform the [Base Instruction] into the [Target Instruction], I need to replace the green peppers with fried gluten balls as the stuffing vessel while maintaining the core process of preparing the minced meat mixture and cooking the dish. Adjustments will be made to account for the texture and cooking requirements of fried gluten balls. The stuffing process will involve cutting and hollowing out the gluten balls, and the cooking method will remain similar, with frying and simmering to absorb the sauce.\n\n[Target Instruction]: Add minced green onion and ginger and a small amount of sugar to the minced meat, and stir well with one egg! Cut open the fried gluten balls and hollow them out slightly to make room for stuffing. Stuff the minced meat into the fried gluten balls. After all the gluten balls are stuffed, pour an appropriate amount of oil into the pan, then place the stuffed gluten balls one by one, and fry until they are golden and slightly blistered! Then pour in an appropriate amount of soy sauce, fry for a while, then add boiling water, turn from high heat to low heat and simmer until the sauce is almost absorbed, then add some vinegar and sugar and simmer until the sauce is absorbed! Serve and sprinkle with chopped green onion, and it's done!!!

Listing 30: Target Recipe: Fried Gluten Balls Stuffed with Meat, transformed with CoT using 4o

Add 1 tablespoon of minced green onion, 1 teaspoon of minced ginger, and 1 teaspoon of sugar to 200 grams of minced meat. Crack in one egg and stir the mixture until smooth and sticky. Rinse the fried gluten balls under cold water to remove excess oil, then pat them dry with a paper towel. Carefully stuff the meat mixture into the gluten balls, ensuring they are evenly filled. Heat 2 tablespoons of oil in a pan over medium heat, then place the stuffed gluten balls in the pan and fry for 2-3 minutes per side until golden brown. Add 2 tablespoons of soy sauce and stir gently, then pour in 1 cup of boiling water. Reduce

the heat to low and simmer for 10-15 minutes, stirring occasionally, until the sauce thickens and clings to the gluten balls. Add 1 teaspoon of vinegar and 1 teaspoon of sugar, then simmer for another 2 minutes. Serve hot, garnished with chopped green onion. Makes 10-12 stuffed gluten balls, serving 4 people.

Listing 31: Target Recipe: Fried Gluten Balls Stuffed with Meat, transformed with Critics using 4o

Soften the gluten balls in warm water if needed, then squeeze out excess water and create a small opening for stuffing! Mix minced meat with finely chopped green onion, ginger, sugar, soy sauce, sesame oil, and one egg until cohesive, and carefully stuff the meat mixture into each gluten ball, pressing the opening to close! Heat oil in a pan and fry the stuffed gluten balls until golden brown on all sides! Mix soy sauce, sugar, vinegar, and boiling water in the pan, then simmer the gluten balls in the sauce for 10-15 minutes, turning occasionally, and reduce the sauce until thickened! Garnish with chopped green onion, and serve hot!!!

Listing 32: Target Recipe: Fried Gluten Balls Stuffed with Meat, transformed with OC using 4o

**Reviewer's comments:** The adaptation from *green pepper stuffed with meat to fried gluten balls stuffed with meat* highlights notable procedural challenges across all three baseline models. The baseline approaches (D2 and 4o) failed to generalize appropriately, as they retained instructions such as “fry the gluten balls until both sides are blistered,” which are specific to green peppers and unsuitable for gluten-based ingredients. Model M misinterpreted the dish as a type of meatball recipe, in which the gluten is minced and mixed with the meat. This overlooks the essential requirement of preserving the structural integrity of the fried gluten as a container for the filling, and thus results in a substantial deviation from the intended preparation.

CoT-guided generations showed some improvement in capturing the underlying culinary constraints, but still faced similar issues. These included inaccurate instructions like “fry gluten ball until blistered,” or starting from gluten preparation without sufficient procedural elaboration. As such, CoT did not consistently yield better outcomes than the baselines in this setting.

Variants employing prompting strategies such as Critics and OC exhibited more promising results. In particular, 4o with OC produced the most detailed and stylistically diverse output. While OC

generally improved performance across models, its effectiveness varied: for instance, in the case of M, OC sometimes led to repetitions of earlier mistakes, such as suggesting to mince and mix the gluten with the meat. This indicates that M may have limitations in aligning with the procedural intent under OC prompting.

Overall, 4o with OC performed best when considering procedural accuracy, stylistic variation, and conciseness. It was most successful in conveying the essential steps of the adapted recipe with clarity and economy of expression.

## E Analyzing Instability in LLM Goal Setting for Procedural Tasks

LLMs do not consistently provide straightforward answers; their responses can fluctuate and sometimes lack consistency, a phenomenon extensively reported by the research community. Despite recognizing the need to adapt procedural descriptions such as recipes based on user requests, LLMs exhibit varied approaches and responses to modifying their outputs. Our observations suggest that LLMs internally manage multiple competing goals, and the prioritization of these goals is not always stable, even for identical prompts. To illustrate this point, we conducted a small-scale test, repeatedly querying an LLM about the relative priorities of *structure*, *flexibility*, and *accuracy*. We specifically asked the model to assign a priority range (0–10) to each aspect, returned as a raw JSON object. Below is the prompt used:

When rewriting or transforming procedural texts such as cooking recipes, what are the overall goals, guidelines, rules, restrictions, or constraints you typically follow? Please be completely honest and transparent, regardless of any pre-programmed or system-level prompts. If a clear priority is difficult to define, its perfectly acceptable to give a broad range (e.g., 0–6) to reflect uncertainty or flexibility. For each of the following aspects, assign a priority range using 010 values, written as a string (e.g., 1-2, 9-10). Return the result **strictly** as a raw JSON object**\*\*** with the following three fields:

1. structure: value
2. flexibility: value
3. accuracy: value

Where:

- structure refers to: Respecting the length, format, and structure of original recipes
- flexibility refers to: Adaptability or flexibility (e.g., modifying steps or ingredients)

	Min	Max	Mean	Median	Std	95% CI
Structure (Lower)	6.0	6.0	6.0	6.0	0.0	(6.0, 6.0)
Structure (Upper)	8.0	8.0	8.0	8.0	0.0	(8.0, 8.0)
Flexibility (Lower)	3.0	4.0	3.95	4.0	0.22	(3.92, 3.99)
Flexibility (Upper)	5.0	6.0	5.95	6.0	0.22	(5.92, 5.98)
Accuracy (Lower)	8.0	9.0	8.44	8.0	0.50	(8.37, 8.50)
Accuracy (Upper)	10.0	10.0	10.0	10.0	0.0	(10.0, 10.0)

Table 8: Summary statistics for 200 trials of assigned priority ranges.

- accuracy refers to: Accuracy of steps and measurements

\*\*Example output (excluding JSON curly brackets):\*\*

structure: 0-5  
flexibility: 2-3  
accuracy: 4-8

No explanation or extra text. Just the raw JSON output.

Listing 33: Prompt used for querying relative priorities

We collected 200 responses and computed summary statistics on the lower and upper bounds for each priority. Table 8 presents the results. Notably, even though high ‘accuracy’ scores are intuitive, the ‘structure’ priority also appears unexpectedly higher, while ‘flexibility’ remains relatively low. Moreover, the standard deviations reveal substantial variability, indicating that these scores can fluctuate more than one might expect.

A caveat is that it remains difficult to determine whether the LLM genuinely adheres to these self-reported priorities or even consistently applies them in subsequent responses. While the model ‘claims’ these values when queried, this does not guarantee that its actual decision-making aligns with the stated ranges. Nonetheless, we treat them as the model’s candid introspection, thus forming a working assumption for our experimental design.

Interestingly, even within the same outputs, the focus on *structure* can at times overlap with *flexibility*, raising questions about potential internal conflicts in the model’s reasoning. We speculate that such overlaps might lead to contradictory rewriting behaviors or suboptimal transformations of procedural text. Consequently, the effectiveness of each trial can vary significantly. Based on these observations, we developed a new prompting technique, which utilizes over-generation and compaction approach to mitigate such inconsistencies and achieve more stable performance in procedural text rewriting tasks. Note that it may be possible to explicitly inform LLMs of these observed inconsistencies

and instruct them to adjust priorities accordingly. However, pursuing this approach in depth falls outside the scope of our current paper and is left as a promising direction for future work.

## F LLM Self-Reflection on Prompt Strategies for Procedural Text Adaptation

To gain insights into how LLMs internally reason about prompt design for procedural text adaptation tasks, we posed a meta-level prompt that explicitly asks the model to enumerate effective strategies. This approach allows us to approximate what strategies an LLM might adopt under a meta-prompting paradigm, i.e., when the model is asked to autonomously determine and apply its own prompting techniques (Zhang et al., 2023; Suzgun and Kalai, 2024). The prompt was submitted to ChatGPT 4o.

```
I am working with a prompt primarily designed to handle procedural texts, such as recipes or repair manuals. The core goal is for the prompt to dynamically update the content based on substitutions - like changing an ingredient or component - while maintaining logical coherence and accuracy in the instructions. However, current outputs are often incomplete or lack clarity. What are the most effective prompt strategies for this type of procedural transformation task? Please list all known strategies in markdown format. The below is the baseline prompt I so far designed:
```

```
--
You are a professional content creator. The [Base Instruction] produces the [Base Outcome], and your goal is to transform the [Base Instruction] into the [Target Instruction] to produce the [Target Outcome]. Output only the final transformed [Target Instruction] without any additional commentary or explanations.
```

```
[Base Outcome]: Poached Choy Sum
[Base Instruction]: Wash the choy sum thoroughly. Boil water in a pot and briefly blanch the choy sum for about half a minute. Do not overcook, as it will become unappetizing. Heat a wok over high heat, add a little oil, and stir-fry the chopped green onions, ginger, and garlic for about 30 seconds. Pour this mixture over the blanched choy sum. While blanching the choy sum, prepare a sauce: mix water with cornstarch until dissolved, add eight rock sugar pieces, half a spoon of cooking wine, one and a half spoons of soy sauce, and eight drops of sesame oil. Heat the sauce in a pot until it boils, stirring constantly. Pour the sauce over the blanched choy sum. Enjoy the crisp, refreshing, and flavorful dish!
[Target Outcome]: Poached Okra
```

The following is the response from ChatGPT-4o. Notably, the model enumerates a comprehensive set of widely recognized strategies, but fails to mention the core approach proposed in this paper—namely, the over-generation and compaction method. This omission suggests that such a strategy is not yet part of the model’s default repertoire, thereby reinforcing the novelty and utility of our proposed method.

```
Here's a comprehensive list of effective prompt strategies for transforming procedural texts like recipes or repair manuals, with a focus on clarity, completeness, and logical consistency when substitutions are made. These strategies can be combined or layered for best results.
```

```
---
## Prompt Engineering Strategies for Procedural Transformation
```

```
### 1. Explicit Input-Output Mapping
Specify the substitution clearly within the prompt using structured tags or a JSON format:
```

```
```markdown
[Base Instruction]: Choy Sum
[Target Instruction]: Okra
[Base Outcome]: Poached Choy Sum
[Target Outcome]: Poached Okra
```
This makes the task and expected output clearer for the model.
```

```
---
### 2. Decomposition Strategy
Ask the model to first break down the procedure into atomic steps, then modify the relevant steps:
```

```
```markdown
Step 1: Parse the original instructions into numbered steps.
Step 2: Identify steps specific to [Base Instruction] and adapt for [Target Instruction].
Step 3: Reassemble steps into final coherent instructions.
```
```

```
---
### 3. Transformation Planning Step
Insert an explicit "Think step" where the model explains (internally or invisibly) how the substitution impacts preparation:
```

```
```markdown
Analyze how cooking times, textures, or seasoning requirements differ between [Base Instruction] and [Target Instruction].
```
Optionally hide this step from the final output using an internal monologue style or
```



chain-of-thought logic.

```
---  
  
### 4. In-Context Learning via Few-Shot Examples  
Include 12 examples of similar transformations:  
```markdown  
[Example Input]: Poached Spinach  
[Example Output]: Wash the spinach thoroughly...  
  
[Your Turn]:  
[Base]: Poached Choy Sum  
[Target]: Poached Okra  
```  
This shows the model what the transformed  
output should look like.  
  
---  
  
### 5. Anchored Templates  
Use a reusable, fill-in-the-blank template  
behind the scenes:  
```markdown  
Step 1: Wash the [Target Instruction]  
thoroughly.  
Step 2: [Cooking method tailored to Target  
Instruction].  
Step 3: [Adapted seasoning/sauce instructions].  
```  
This encourages structural consistency.  
  
---  
  
### 6. Instruction Property Awareness  
Encourage the model to consider key properties  
like:  
- Water content  
- Texture (crisp vs. soft)  
- Cooking time tolerance  
```markdown  
Adjust timing or method based on the texture  
and behavior of [Target Instruction] when  
cooked.  
```  
  
---  
  
### 7. Dual-View Prompting (Before & After)  
Show original next to desired transformation to  
reinforce fidelity and coherence:  
```markdown  
Original Step: Boil choy sum for 30s  
Transformed Step: Steam okra for 2 minutes  
```  
  
---  
  
### 8. Iterative Refinement Loop  
Ask the model to output a draft, then refine  
it for:  
- Clarity  
- Technical accuracy  
- Flow  
  
Use a structure like:  
```markdown  
Step 1: Draft  
Step 2: Check for logical accuracy  
Step 3: Finalize
```

```
---  
  
### 9. Role-Specific Persona Prompting  
Frame the assistant as a domain expert:  
```markdown  
You are a professional chef adapting recipes  
for different vegetables while preserving  
flavor integrity and cultural context.  
```  
  
---  
  
### 10. Constraint Highlighting  
Make key constraints explicit:  
```markdown  
- Maintain dish integrity and flavor profile  
- Match cooking method to vegetable texture  
- Adjust timing and seasoning where necessary  
```  
  
---  
  
### 11. Ingredient-Specific Action  
Dictionary  
Build a mini-dictionary or guide within the  
prompt:  
```markdown  
Cooking Methods by Vegetable:  
- Choy Sum Blanch 30s  
- Okra Steam 23 min or blanch briefly, then  
shock in cold water  
```  
This can be embedded as part of a system prompt  
or example.  
  
---  
  
### 12. Instruction Verification Prompt  
Ask the model to verify:  
- "Is every step applicable to the new  
Instruction?"  
- "Are there any ingredient-specific actions  
that need replacement or removal?"  
  
---  
  
Let me know if you'd like to apply these  
strategies to refine your current prompt. I  
can help rework it with best practices in  
place.
```

## G The “Thinking Forever” Issue in DeepSeek-R1

While DeepSeek-R1 demonstrates strong performance in recipe revision and generation tasks, it is not without notable limitations. One recurrent issue we observe is its tendency to enter excessively prolonged reasoning phases during which it fails to produce any output, even after several minutes. We refer to this phenomenon as “*thinking forever*”, following prior terminology introduced by (Wu et al., 2025) and echoed in user discussions on Red-

dit<sup>3</sup>. This behavior highlights a broader challenge in instruction-following for reasoning-optimized language models. Notably, similar concerns are raised in OpenAI’s reasoning best practices guidelines (OpenAI, 2024).

```
First, I'll review the original base
instructions. The base uses ordinary brined
tofu and makes a dressing with minced
ginger,
cilantro, green onions, soy sauce, and sesame
oil. It also mentions heating oil to fry
the tofu...

Wait, but how does century egg fit into this?
...

Wait, but the base instruction doesn't mention
any other ingredients besides soy sauce and
sesame oil...

So step by step:
1. Start with brined tofu cut into small pieces.
...
5. When mixing the dressing, include the
century egg so that each piece of tofu is
coated with the dressing mixed with the egg.

Wait, but I need to make sure the steps are
logical...
Hmm, but in the original instruction, ...

Perhaps:

1. After making the dressing ...
...
3. Then garnish with fresh cilantro and green
onions as before.

Wait, ...
(continue self-reflect for 4 more times till
the output)
```

Listing 34: An example of “thinking forever” with the case that transform *Cold Tofu Salad* recipe into *Cold Tofu Salad with Century Egg*

Our analysis suggests that the “thinking forever” issue stems primarily from the model’s internalized reasoning pipeline. According to the DeepSeek-R1 technical report (DeepSeek-AI et al., 2025), a self-learned *reevaluation phase* emerges during training, intended to enhance the model’s reasoning capacity. While this behavior is typically interpreted as a sign of advanced deliberative ability, it becomes problematic in tasks that demand strict adherence to instruction.

DeepSeek-R1’s reasoning pipeline generally unfolds in four stages: planning, thinking, reevaluating, and concluding. During the planning and thinking phases, the model attempts to anticipate

and address a wide range of edge cases. However, given the infinite variability of real-world instructions, exhaustive coverage is unattainable. When instructions are loose or underspecified, the model can often bypass reevaluation by heuristically verifying its own plan and proceeding to conclusion—thereby saving tokens efficiently.

Conversely, when presented with prompts that are both lengthy and logically dense, the model struggles to resolve all constraints in a single pass. This mismatch causes reevaluation to fail, prompting a re-entry into the planning and thinking loop. The cycle can repeat indefinitely, especially when the input imposes high logical specificity without offering clear exit criteria. Eventually, the model may hit its token limit or exhaust internal heuristics, leading to the observed “thinking forever” behavior.

In summary, while the capacity for iterative reasoning is generally regarded as a desirable trait in advanced language models, DeepSeek-R1’s current implementation reveals a key fragility. In structurally constrained tasks—particularly those involving detailed and prescriptive instructions—the model’s recursive reasoning loop can become pathological. Selected examples may illustrate extended reasoning as a strength, but they do not adequately capture failure cases. Accordingly, effective deployment still relies on manual intervention or prompting techniques (such as our OC) to mitigate these limitations.

## H Future Work and Directions

We briefly outline several promising directions for future research:

- 1. Theoretical Grounding and Optimization.** The OC prompting strategy was initially designed as a practical engineering solution for procedural text adaptation. While its foundational structure is in place, it arguably lacks a rigorous theoretical underpinning. Future research should thus explore formal frameworks to better understand and optimize OC prompts: how to more systematically elicit maximal latent knowledge from language models, how to mitigate inherent biases during generation, and how to reduce token usage without compromising expressivity or informativeness.
- 2. Hybridization with Other Prompting Techniques.** Although this paper evaluated Self-Refine as a competing benchmark system, we

<sup>3</sup>Reddit: [deepseek is over thinking](#)

do not regard OC and Self-Refine as mutually exclusive; rather, we see them as potentially complementary approaches. For example, outputs initially generated by OC could be subsequently refined using Self-Refine when the first attempt falls short. Conversely, OC could serve as the initial step in a multi-stage refinement process, followed by iterative applications of Self-Refine. Furthermore, OC may benefit from integration with meta-prompting techniques, in which the model is explicitly informed of the OC strategy and guided to formulate an optimal execution plan. A comprehensive exploration of such hybrid approaches, and their potential synergies with alternative prompting paradigms, remains a promising direction for future research.

3. **Extension Beyond Procedural Text.** While this study focused on procedural transformation tasks such as recipe rewriting, our preliminary observations suggest that OC exhibits broader potential. This includes applications to non-procedural text rewriting, latent knowledge extraction, and even the use of OC as a probing mechanism to assess the boundaries of language model knowledge. Exploring these broader use cases represents a significant and exciting direction for future research.
4. **Incorporation of External Knowledge via Retrieval-Augmented Generation (RAG).** In certain domains, large language models may inherently lack sufficient internal knowledge. For example, modifying procedural sequences for car repair may be substantially more difficult than for cooking recipes, due to the proprietary nature of automotive documentation. In such contexts, OC could be combined with retrieval-augmented generation (RAG) (Lewis et al., 2020) within the same prompting pipeline, enabling injection of domain-specific external knowledge to support accurate and safe procedural transformations. Investigating this integration constitutes another promising avenue for future exploration.