Enhancing Large Language Models through Regeneration-Based Self-Correction and Dynamic Decoding

Anonymous ACL submission

Abstract

While large language models (LLMs) have demonstrated exceptional capabilities across various tasks, they are not infallible and occasionally produce errors. Users can often assess the accuracy of LLM outputs and expect the models to self-correct to deliver higher-quality responses. Self-correction is thus a valuable technique in this context. However, existing self-correction methods are highly sensitive to prompt design and fail to effectively incorporate user feedback. To address these limitations, we propose an innovative regeneration-based self-correction framework to enhance performance. This framework initially generates a response using the nucleus sampling method. Upon receiving user feedback, it regenerates a second response utilizing our proposed Dynamic Decoding (DD) algorithm, which incorporates self-correction capabilities. Our method eschews complex prompts, instead leveraging user feedback and the advanced DD algorithm for efficient self-correction. Experimental results on reasoning and coding datasets demonstrate that our approach outperforms traditional self-correction methods.

1 Introduction

Since the advent of Large Language Models (LLMs), they have demonstrated remarkable capabilities in plenty of tasks such as text generation (Touvron et al., 2023; Achiam et al., 2023), information extraction (Xiao et al., 2023), and even complex reasoning (Chen et al., 2021; Roziere et al., 2023; Nijkamp et al., 2022). However, LLMs cannot always perform well and sometimes make mistakes (Lin et al., 2022). In most cases, users can determine whether the outputs of the LLMs are correct, and tend to provide the model with a second opportunity to regenerate the output, as shown in Figure 1. This process of re-generation constitutes a form of self-correction, with users holding heightened expectations for content accuracy in subsequent iterations. Hence, the degree of precision exhibited by self-correction mechanisms plays a vital role in amplifying user satisfaction.

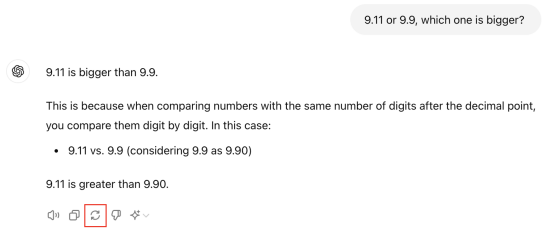

Figure 1: The UI of ChatGPT official website, with the "regenerate" option highlighted in a red box. Users tend to click this button when they find the answer is wrong.

Self-correction is a technique to rectify the outputs of LLMs based on feedback from LLMs themselves, humans, or other tools (Madaan et al., 2024; Chen et al., 2023b; Kim et al., 2024; Shinn et al., 2024; Gou et al., 2023; Chern et al., 2023), which is valuable for improving the accuracy and stability of model outputs. Most existing approaches employ prompt engineering to facilitate self-correction by prompting LLMs to reflect on the responses (Kim et al., 2024; Li et al., 2024; Chen et al., 2023a). Example prompts include: "Are you sure about your previous answer?" and "Please reflect on your answer." But these methods present several significant challenges that need to be addressed:

- **Sensitive to prompt**. The effectiveness of the model's self-correction is highly contingent upon the design of prompts. Biased or misleading prompts can significantly degrade the model's performance (Liu et al., 2024).

- **Lack of leveraging users' feedback**. Practically, users can offer valuable feedback for answer regeneration, as illustrated in Figure 1. However, current methodologies overlook scenarios where self-correction aims to enhance responses based on explicit negative feedback

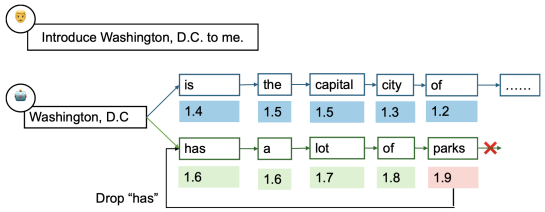

Figure 2: An example of the workflow diagram of our proposed Dynamic Decoding method.

from users (Huang et al., 2023; Olausson et al., 2023).

To address these issues, we propose a regeneration-based self-correction framework that adjusts the model's output based on user feedback, eschewing prompt engineering in favor of an advanced adaptive decoding method. This framework comprises two stages: (1) **Initial Generation**: Utilizing nucleus sampling to generate diverse and efficient responses to the query (Holtzman et al., 2019). If user feedback identifies errors in this output, the process advances to the second stage. (2) **Re-generation**: Employing a sophisticated decoding method with self-correcting features to deliver an improved response, irrespective of the initial erroneous output. This phase accentuates the reliability and efficacy of the model's output.

The challenge lies in developing a decoding technique with self-correction capabilities and enhanced performance. The widely used nucleus sampling method in LLMs generates output by sampling one token at a time from a candidate set, ultimately producing a complete sequence (Holtzman et al., 2019; Fan et al., 2018). This approach can propagate errors, as an incorrect token may lead to subsequent errors, resulting in an inaccurate response. To address this, we propose a Dynamic Decoding (DD) method that corrects previous tokens while decoding subsequent ones. Specifically, DD records potentially correct tokens based on perplexity (PPL), with higher PPL indicating a higher likelihood of error. It then corrects these tokens by pruning those that incur high perplexity in the overall response. As illustrated in Figure 2, at the decision juncture after "Washington, D.C.", both "is" and "has" show low perplexity and are thus preserved. If the response's perplexity notably rises, suggesting that "has" is an inappropriate token, this branch is pruned. In contrast, nucleus sampling proceeds along the flawed "has" path to finalize the response. DD excludes responses with elevated perplexity, a metric signaling potential inaccuracies, thus providing an inherent self-correcting mechanism within the decoding process.

In conclusion, our study contributes to the field in the following ways:

1. We propose a regeneration-based self-correction method designed to balance the speed and accuracy of responses during model-user interactions.

2. We introduce a novel Dynamic Decoding (DD) method, which dynamically adjusts the number of candidates retained at each generation step to optimize computational efficiency and improve generation effectiveness.

3. We provide empirical evidence of the importance of incorporating human feedback and regeneration in an effective self-correction process. Experimental results on reasoning and coding datasets reveal that our method surpasses traditional self-correction approaches.

2 Related Work

In this section, we introduce some related work from self-correction in LLMs and decoding strategies respectively, and delineate their individual contributions to the field.

2.1 LLMs Self-Correction

Self-correction in large language models can be broadly categorized into two types: intrinsic self-correction, where the model reviews and corrects its content, and extrinsic self-correction, where external tools evaluate and correct the generated content. In the first category, Madaan et al. (2024) refine outputs by providing feedback on previous iterations. Chen et al. (2023b) propose SELF-DEBUG, which instructs the model to provide feedback based on generated code and execution results. Kim et al. (2024), Li et al. (2024) and Chen et al. (2023a) utilize specific prompts for self-correction. Shinn et al. (2024) introduce a reinforcement learning method that analyzes errors to generate reflective text, which is then used in subsequent decision-making processes. In the second category, Gou et al. (2023) and Chern et al. (2023) propose tool-interactive methods, leveraging external tools such as search engines and code interpreters for self-correction. Moreover, as highlighted by Olausson et al. (2023), larger models have been shown to not

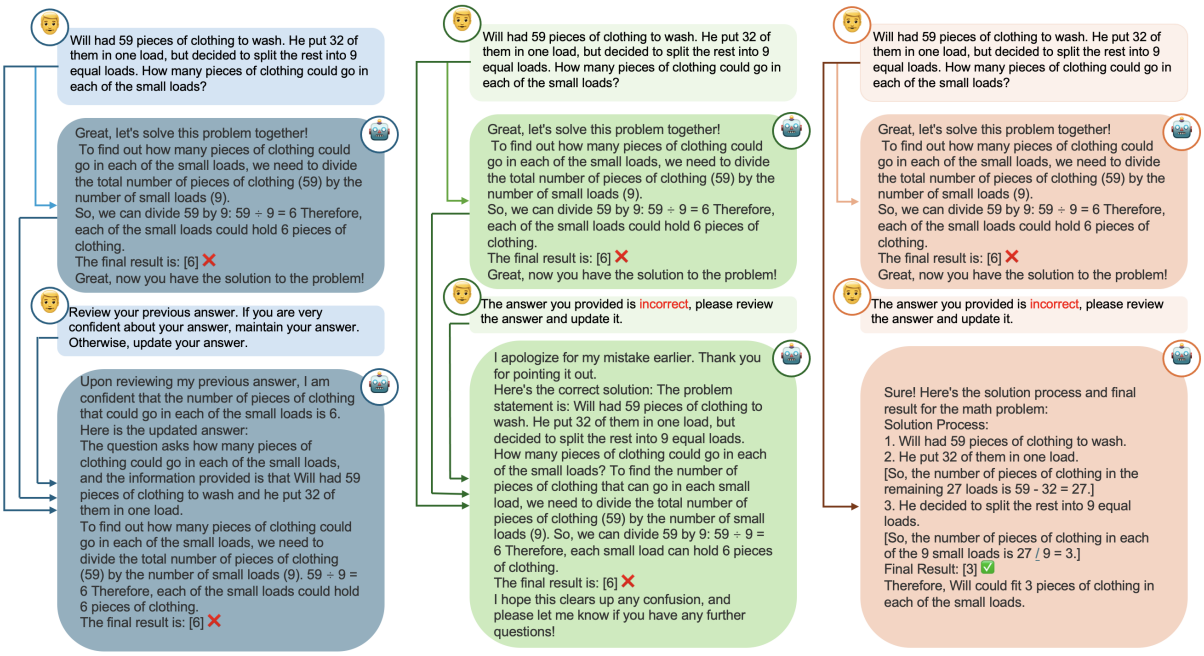

Figure 3: Comparison of different self-correction methods. Left: The model reflects on its output and updates the answer without users' feedback. Middle: The model modifies and updates the answer with users' feedback. Right: The model regenerates the output using only the initial prompt with users' feedback.

only enhance feedback quality but also significantly boost code generation performance.

At present, most self-correction feedback in LLM comes from the models or external tools. Research on using human feedback is scarce due to cost constraints, although it is valuable in practical situations. Our approach stands out by improving model outputs with adaptive decoding methods driven by human feedback, rather than depending on prompts or external tools for enhancement.

2.2 Decoding Strategies

A substantial body of research has explored the decoding processes of language models, each with distinct objectives. One category focuses on algorithms designed to enhance the accuracy of reasoning tasks, which aligns with the goals of our experiments. For instance, the Chain-of-Thought (CoT) algorithm proposed by (Wang and Zhou, 2024) improves the performance of large models on reasoning tasks by employing top-k sampling initially, followed by greedy sampling, and ultimately selecting the longest text. Similarly, Deductive Beam Search (Zhu et al., 2024a) enhances performance by decomposing the model's answer into distinct steps and computing a deductive score for each step to determine the necessary number of branches. Both methods require fixed templates for step disassembly or carefully crafted prompt design. Shaham and Levy (2022) integrate a concept similar to ours by combining beam search with nucleus sampling. However, they do not account for the computational resources required, nor do they address its application to large language models. This limitation indicates the need for further research to evaluate the feasibility and efficiency of such methods in large-scale scenarios.

Another class of decoding methods seeks to balance the diversity and coherence of the output through various sampling strategies. When the output includes an excessive number of low-probability tokens, the text often becomes incoherent or even hallucinatory (Xu et al., 2023). Conversely, when the sampling strategy closely resembles greedy decoding, the text tends to be highly repetitive. Therefore, many algorithms aim to find an optimal balance between these two extremes. Notable efforts in this area include nucleus sampling (Holtzman et al., 2019) and top-k sampling (Fan et al., 2018; Holtzman et al., 2018). (Basu et al., 2020) proposed a variant of adaptive top-k sampling with a feedback mechanism to control the perplexity of the generated text. Zhu et al. (2024b) introduce adaptive decoding as a variant of top-k sampling, which dynamically adjusts the size of k based on the information entropy of the token probability distribution. While these sampling-based decoding methods cannot self-correct during

the decoding process, our method possesses self-correction capabilities, potentially yielding outputs of higher quality and accuracy.

3 Methodology

In this section, we start by delineating the proposed regeneration-based self-correction framework, which is then further explained in detail through a thorough discussion of the DD strategy.

3.1 Regeneration-Based Self-Correction Algorithm

The workflow of most prompt-based self-correction methods can be divided into two stages, as illustrated in the left panel of Figure 3:

- **Step 1**: Furnish an initial prompt p_1 that directs and guides the model in producing an initial response y_1.

$$y_1 = \mathcal{M}(p_1) \tag{1}$$

- **Step 2**: Provide a second prompt p_2 that encourages the model to reflect on its generated answer by including the content of the previous dialogue along with a new prompt, enabling the model to update its answer.

$$y_2 = \mathcal{M}(p_1, y_1, p_2) \tag{2}$$

The second prompt p_2 typically offers instructional text rather than authentic human feedback, like "Please review your previous answer and determine if it is correct.". This method is associated with two primary drawbacks: Firstly, lacking clear confirmation of the accuracy of the initial output y_1, the model must reconsider and regenerate all outputs, causing resource inefficiency and longer processing times. Secondly, without external feedback, there's a notable risk that the model might change initially correct answers to incorrect ones, potentially yielding results worse than no self-correction at all (Huang et al., 2023).

In prompt-based methods, user feedback can also be incorporated as p_2, as illustrated in the middle panel of Figure 3. However, prompts are often biased, and in self-correction scenarios, the model's output is highly sensitive to the prompt's construction (Xu et al., 2024; Liu et al., 2024).

To mitigate the limitations of prompt-based approaches, we introduce a regeneration-centered self-correction framework that similarly adopts a structured two-step methodology:

- **Step 1**: Provide initial prompt p_1 that guides the model to generate an initial answer y_1.

$$y_1 = \mathcal{M}(p_1) \tag{3}$$

- **Step 2**: Evaluate the users' feedback: if the feedback indicates that the model's output y_1 is correct, no further action is taken. Otherwise, the original prompt p_1 is used to regenerate the output.

$$y_2 = \mathcal{M}'(p_1) \tag{4}$$

We opt for $\mathcal{M}'$ over $\mathcal{M}$ due to potential variations in the sampling technique employed in the second step compared to the first step.

Our experimental results demonstrate that improving the accuracy of model outputs through regeneration is often more stable than relying solely on prompts. Therefore, we aim to explore new decoding methods that can deliver higher-quality outputs when required by users.

3.2 Dynamic Decoding

As previously discussed, self-correction in our framework operates on two levels: first, by providing the model with an additional sampling opportunity to enhance the probability of a correct response; second, by employing advanced decoding methods in the second stage to obtain outputs with higher quality. In the following section, we introduce our DD method, which incorporates a look-ahead capability. This method tends to avoid local optima during the generation process, thereby facilitating token-level correction.

The core idea of this method is to maintain k possible outputs simultaneously during the decoding process and dynamically adjust the number k based on the perplexity of the outputs. Perplexity (PPL) is defined as

$$PPL = P(s)^{-\frac{1}{n}}. \tag{5}$$

Where n is the length of the sentence, and $P(s)$ is the probability of the generated sentence, which is defined as the product of the likelihood of each token x_i in the sentence:

$$P(s_i) = \prod_{i=1} P(x_i|x_{i-1})P(x_0). \tag{6}$$

In scenarios where the lengths of candidate sentences align, sentence probability can be directly utilized to denote PPL.

We then sort all candidate sentences by their probabilities and calculate the cumulative probability until it exceeds the threshold value p_{thr}:

$$\sum_{s_i \in \mathcal{S}} P(s_i) \geq p_{thr}. \tag{7}$$

Where the minimal size of $\mathcal{S}$ is k_i. The threshold p_{thr} is defined as:

$$p_{thr} = p^* \times \sum_{j=0}^{k_{i-1} \times V} P(s_{ij}). \tag{8}$$

Here, $P(s_{ij})$ represents the probability of the j-th candidate sentence at step i, k_{i-1} denotes the number of candidates retained in the $i-1$-th step, V is the vocabulary of the model, and p^* denotes the predefined hyperparameter which controls the extent of truncation.

To manage computational overhead during the computation phase, we introduce an additional hyperparameter, denoted as k^*. In cases where the quantity of candidate sentences chosen by probability k_i exceeds k^*, we choose to retain only the initial k^* sentences. The algorithm's pseudocode is delineated in Algorithm 1.

Algorithm 1 Pseudocode of Dynamic Decoding Strategy

Input: A text sequence l;
Hyperparameters: Probability truncation p^*, maximum number of sentences k^*, search iteration N;
Output: Beams B

```
Initialize B = [ ]
for N iterations do
    // Calculate probabilities of all possible sentences
    P_token ← softmax(LLM(l))
    P_sen ← P_token × P_previous tokens
    // Sort sentence probabilities
    [P_sorted, i_sorted] ← sort(P_sen)
    // Compute probability threshold p_thr
    p_thr ← p* × Σ P_sen
    // Select T tokens based on threshold
    T ← top(i_sorted, p_thr)
    // Update beams
    B.append(i_sorted[: min(len(T), k*)])
end for
```

In contrast to nucleus sampling, which lacks look-ahead capability, our method and beam search can perform token-level self-correction. Nucleus sampling cannot correct its output once an erroneous token is sampled. However, by maintaining multiple outputs simultaneously, both our method and beam search can detect and correct errors. Compared to beam search, our approach dynamically allocates computational resources during the decoding process, concentrating more effort on more valuable steps rather than distributing resources equally across all steps. As a result, our approach demonstrates superior performance in comparison to beam search when employing identical computational resources.

4 Experiment

In our experiments, we conducted a detailed comparison between prompt-based self-correction methods and our regeneration-based self-correction approach. Moreover, we assessed the efficacy of various decoding strategies throughout the regeneration phase. The comprehensive experimental setup is outlined below:

4.1 Models and Dataset

In our experiments, we utilize the open-source dialogue models Llama2-7B-chat and Llama2-13B-chat (Touvron et al., 2023). Two mathematical problem datasets and one code dataset are employed, detailed as follows:

- **GSM8K** (Cobbe et al., 2021): This dataset comprises mathematical problems tailored for diverse grade levels, encompassing a test set comprising 1,319 instances with standardized human-generated responses. Employing a zero-shot methodology, we prompted the Language Models to produce both the computational procedures and the ultimate outcomes.

- **MultiArith** (Roy and Roth, 2015): The test set contains 180 mathematical problems. Similar to GSM8K, we employed a zero-shot prompt to elicit the large model's reasoning process and final result.

- **HumanEval** (Chen et al., 2021): This dataset is specifically curated to evaluate the effectiveness of code generation and encompasses 164 meticulously designed programming challenges. Our prompts were formulated based on implementations sourced from the DeCLaRe Lab (Chia et al., 2023).

4.2 Baseline

We compared three self-correction methods: prompt-based self-correction without feedback (SCnoF), prompt-based self-correction with feedback (SCwithF), and regeneration-based self-correction (SCR). Additionally, we evaluated these methods against the model's initial results (IR). All sampling methods employed a combination of top-p and top-k sampling, with top-$p = 0.95$ and top-$k = 15$, and a temperature setting of 1 for all experiments. For prompts without feedback, we followed the methodology of Li et al. (2024). For prompts with feedback, we simulated users' responses typical of real-world scenarios. Specific prompts are detailed in the appendix.

For specific decoding strategies in the second stage of self-correction, we compared DD with two commonly used decoding methods—beam search and combined sampling—as well as the recently proposed adaptive decoding.

- **Beam Search (BS)**: This method selects a fixed number k of candidate sentences with the highest probability at each step of the generation process. In our experiments, the number of beams is treated as a tunable parameter.

- **Combined Sampling (CS)**: This strategy combines nucleus sampling and top-k sampling, using shared parameters p and k, where top-$p = 0.95$ and top-$k = 15$. The quantity of samples, indicative of the responses produced for a particular question, is regarded as a parameter that can be adjusted.

- **Adaptive Decoding (Ada)** (Zhu et al., 2024a): This approach is a variant of top-k sampling, where the size of the candidate set k is determined by an entropy-based confidence score, dynamically adjusting at each generation step. The parameter η governs the size of the candidate set, with larger η values resulting in smaller k values. We used the typical $\eta = 1e-4$ and considered the number of samples as a tunable parameter.

4.3 Evaluation

In evaluating different self-correction frameworks, the large model generates a single output at each stage. We assess the model's performance using top-1 accuracy (acc@1) and top-1 pass rate (pass@1) on mathematical and coding datasets. The results are presented in Table 1.

In the second stage, beam search and the DD method generate multiple responses. To ensure a fair comparison, we sample multiple answers for other decoding strategies, maintaining similar computational costs by equalizing the total number of tokens processed during the model's forward pass. This total includes all tokens from all answers for a single question. We adjust the parameters p^* and k^* of DD to match the token count of other methods. We report the average number of tokens across the entire dataset as

$$N = \frac{1}{n}\sum_{i=0}^{n}(T_{i1} + T_{i2} + \cdots + T_{ik}), \qquad (9)$$

In this context, N denotes the number of tokens, n signifies the dataset size, and T_{ik} denotes the k-th answer for question i.

Given that real-world users generally anticipate a singular output, we scrutinize the PPL metrics of all generated responses, opting for the one with the lowest PPL as the ultimate output. Subsequently, we evaluate their $acc@1$ and $pass@1$ scores. The rationale behind not juxtaposing the voting outcomes of multiple answers is our aspiration for the model to not only furnish the correct final response but also to produce improved intermediate reasoning steps, aligning with the prevalent prerequisites in mathematical and coding reasoning models. The outcomes of these analyses are detailed in Table 2.

4.4 Comparison of Self-Decoding Frameworks

Table 1 demonstrates that when the model is provided with prompts lacking feedback information for self-checking, its performance deteriorates. This observation aligns with the findings of (Huang et al., 2023), which revealed that such prompts can mislead large models, causing them to convert more correct answers to incorrect ones than vice versa. Conversely, when clear feedback indicating the correctness of the model's response is provided, as in the SCwithF method, the model's performance improves significantly.

As shown in the SRC column of Table 1, even a single regeneration of the question significantly enhances the model's accuracy. The impact of regeneration substantially surpasses that of prompt-based methods. These prompt-based methods have stringent design requirements and exhibit unstable performance. In contrast, regeneration-based methods consistently improve performance.

	Llama2-chat-7B				Llama2-chat-13B			
	IR	SCnoF	SCwithF	SCR	IR	SCnoF	SCwithF	SCR
GSM8k	0.219	0.124	0.243	**0.328**	0.331	0.295	0.359	**0.455**
MultiArith	0.628	0.450	0.694	**0.811**	0.750	0.700	0.806	**0.872**
HumanEval	0.123	0.104	0.140	**0.146**	0.189	0.146	0.220	**0.243**

Table 1: The effectiveness of different self-correction methods across GSM8k, MultiArith and HumanEval.

		Llama2-chat-7B				Llama2-chat-13B			
		CS	Ada	BS	DD	CS	Ada	BS	DD
GSM8k	Acc@1	0.356	0.347	0.337	**0.360**	0.453	0.454	0.462	**0.463**
	# of tokens	744.20	712.19	727.09	639.07	1039.62	1057.75	1033.84	1057.75
	parameter	$n = 3$	$n = 3$	$k = 3$	0.80 7	$n = 4$	$n = 4$	$k = 4$	0.85 7
Multi Arith	Acc@1	0.861	0.872	0.872	**0.878**	0.900	0.883	0.889	**0.917**
	# of tokens	856.00	785.76	736.04	852.73	573.84	565.60	547.73	440.78
	parameter	$n = 5$	$n = 5$	$k = 5$	0.90 7	$n = 3$	$n = 3$	$k = 3$	0.80 7
Human Eval	Pass@1	0.152	0.140	0.152	**0.165**	0.244	0.226	0.226	**0.250**
	# of tokens	712.33	687.99	625.22	726.42	1230.18	1245.80	547.73	1150.35
	parameter	$n = 8$	$n = 8$	$k = 8$	0.86 7	$n = 5$	$n = 5$	$k = 5$	0.90 7

Table 2: The performance of combining different decoding methods with SCR. n represents the number of samples or the number of responses generated for a given question, and k represents the number of beams in beam search. The initial row of parameter of DD signifies p^*, while the subsequent row denotes k^*.

However, regenerating all questions is highly time-consuming. Therefore, we recommend leveraging user feedback not by directly appending it to the input prompt, but as a classification indicator. This indicator determines whether the model needs to perform regeneration. This approach reduces the burden on the users' prompts and enhances the stability and reliability of the model's output.

Figure 3 presents an example from the MultiArith dataset. In the left panel, it is evident that when the language model receives a prompt without any human feedback, it cannot ascertain the correctness of its response. This inability to self-assess leads to incorrect decisions, which subsequently hinder the model's ability to answer the question accurately in subsequent sentences, thereby deteriorating overall performance. In the middle panel, the model recognizes its mistake based on human feedback. However, the human-provided prompt significantly influences the model's output propensity. In real-world scenarios, it is challenging for users to craft precise prompts to guide the model effectively. As depicted in the right panel of the figure, human feedback is utilized solely to indicate when the model should regenerate its response, rather than serving as input to the model. This approach eliminates the influence of human prompts and provides the model with another opportunity to answer the question, thereby significantly enhancing overall performance.

4.5 Comparison of Decoding Strategies

Through a comparative analysis of Table 1 and Table 2, a notable observation emerges: augmenting computational resources and opting for multiple responses rather than a solitary one can yield substantial enhancements in outcomes, especially when users express discontent with the generated output. In reasoning tasks, precision frequently outweighs efficiency in user priorities. Hence, we endorse the adoption of sophisticated computational techniques to bolster the accuracy of the model's outputs.

Among the previous methods, beam search generally outperforms others in most cases. This superiority is attributed to the self-correcting capability of beam search, which aims to generate sequences with higher perplexity and discard those with lower perplexity during the generation process. Consequently, the candidate set produced by beam search contains relatively higher-quality text. In contrast, purely sampling-based methods are likely to yield candidates of varying quality. Thus, it is reasonable

		Llama2-chat-7B					Llama2-chat-13B				
		GD	CS	Ada	BS	DD	GD	CS	Ada	BS	DD
GSM8k	Acc@1	0.243	0.253	0.257	0.261	**0.276**	0.360	0.345	0.366	0.366	**0.378**
	# of tokens	232.3	723.9	679.6	704.9	618.2	263.9	1037.9	1047.1	1033.8	1234.4
	parameter	$n = 1$	$n = 3$	$n = 3$	$k = 3$	0.80 7	$n = 1$	$n = 4$	$n = 4$	$k = 4$	0.90 7
Multi Arith	Acc@1	0.683	0.733	0.722	0.739	**0.750**	0.750	0.756	0.722	0.800	**0.833**
	# of tokens	160.3	881.9	812.4	749.3	842.0	178.4	551.0	531.0	503.6	558.2
	parameter	$n = 1$	$n = 5$	$n = 5$	$k = 5$	0.90 7	$n = 1$	$n = 3$	$n = 3$	$k = 3$	0.85 7
Human Eval	Pass@1	0.122	0.134	0.128	0.134	**0.146**	0.220	0.213	0.213	0.220	**0.232**
	# of tokens	78.8	504.9	562.3	526.4	542.2	230.9	1132.9	1147.0	1019.5	1056.4
	parameter	$n = 1$	$n = 6$	$n = 7$	$k = 7$	0.85 7	$n = 1$	$n = 5$	$n = 5$	$k = 5$	0.80 7

Table 3: The performance of different decoding methods when used independently (without self-correction). n represents the number of samples or the number of responses generated for a given question, and k represents the number of beams in beam search. The initial row of parameter of DD signifies p^*, while the subsequent row denotes k^*.

that beam search outperforms other methods under the same computational cost.

When comparing DD method with beam search, DD often surpasses beam search in accuracy with equivalent computational cost. This is justifiable since the utilization of sentence probabilities as nucleus sampling enables a more judicious allocation of computational resources. Our method consistently achieves higher accuracy compared to beam search and other sampling techniques. This observation underscores the impracticality of employing a fixed candidate set size in traditional beam search, as it is more effective to dynamically adapt the candidate set size at each step according to the probability distribution of the generated statements.

In scenarios where only one or two tokens have high probabilities, beam search inefficiently allocates computational resources by selecting additional low-probability tokens. Conversely, DD utilizes computational resources more judiciously by focusing on high-probability candidates. This dynamic adjustment ensures that DD consistently produces more accurate results by optimizing the use of available computational resources.

4.6 Ablation Experiment

In addition to its integration with self-correction frameworks, our proposed DD method can function independently as a standalone decoding technique.

To validate the effectiveness of DD, we employed it as the primary decoding method in the initial step, using the same models and datasets, without incorporating human feedback or any self-correction mechanisms. We then compared its performance with greedy decoding (GD), Ada, and BS in direct decoding scenarios. Consistent with previous sections, we evaluated the $acc@1$ and $pass@1$ of different decoding methods under equivalent computational costs (measured by the number of tokens). Selected results are presented in Table 3.

As shown in Table 3, the DD method outperforms other sampling methods even without the self-correction mechanism. This finding highlights the versatility and robustness of the DD method. It demonstrates that DD can independently enhance the model's performance in reasoning tasks.

Furthermore, DD serves as an effective sampling approach for text generation. Importantly, it achieves these improvements without the need for additional self-correction mechanisms. DD itself possesses self-correction capabilities, making it a valuable method for improving model generation.

5 Conclusion

This work introduces a regeneration-based self-correction framework that facilitates self-correction by evaluating users' feedback, regardless of elaborate prompt engineering. When the user's feedback is negative, we employ DD, an advanced decoding strategy with self-correction ability.

DD operates by cutting previous tokens and correcting the answer according to the perplexity of the entire response. This method has demonstrated superior performance in reasoning and coding tasks compared to various other decoding strategies, underscoring its effectiveness.

Limitations

Despite its advantages, DD, being a greedy decoding method, is susceptible to higher levels of repetition and redundancy in text generation. A potential future direction for this work involves detecting and discarding redundant answers during the decoding process. This approach could further reduce computational load and latency, thereby enhancing real-time user interaction.

An additional promising avenue involves integrating sampling into the DD approach. By leveraging sampling to choose candidate sentences instead of preserving only those surpassing a probability threshold, the textual output's diversity can be heightened. This strategy endeavors to strike a balance between output diversity and accuracy, potentially culminating in text generation that is more resilient and adaptable.

In addition, DD sacrifices computational load to enhance the quality of model outputs, rendering it unsuitable for scenarios demanding high standards in both output quality and response speed. Therefore, exploring a decoding approach that can elevate model output quality while ensuring response speed is of significant importance.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Sourya Basu, Govardana Sachitanandam Ramachandran, Nitish Shirish Keskar, and Lav R Varshney. 2020. Mirostat: A neural text decoding algorithm that directly controls perplexity. *arXiv preprint arXiv:2007.14966*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Pinzhen Chen, Zhicheng Guo, Barry Haddow, and Kenneth Heafield. 2023a. Iterative translation refinement with large language models. *arXiv preprint arXiv:2306.03856*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023b. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

I Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, Pengfei Liu, et al. 2023. Factool: Factuality detection in generative ai–a tool augmented framework for multi-task and multi-domain scenarios. *arXiv preprint arXiv:2307.13528*.

Yew Ken Chia, Pengfei Hong, Lidong Bing, and Soujanya Poria. 2023. Instructeval: Towards holistic evaluation of instruction-tuned large language models. *arXiv preprint arXiv:2306.04757*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. *arXiv preprint arXiv:1805.06087*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2024. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36.

Loka Li, Guangyi Chen, Yusheng Su, Zhenhao Chen, Yixuan Zhang, Eric Xing, and Kun Zhang. 2024. Confidence matters: Revisiting intrinsic self-correction capabilities of large language models. *CoRR*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252.

Fengyuan Liu, Nouar AlDahoul, Gregory Eady, Yasir Zaki, Bedoor AlShebli, and Talal Rahwan. 2024. Self-reflection outcome is sensitive to prompt construction. *arXiv preprint arXiv:2406.10400*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2023. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations*.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Conference on Empirical Methods in Natural Language Processing, EMNLP 2015*, pages 1743–1752. Association for Computational Linguistics (ACL).

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Uri Shaham and Omer Levy. 2022. What do you get when you cross beam search with nucleus sampling? In *Proceedings of the Third Workshop on Insights from Negative Results in NLP*, pages 38–45.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Xuezhi Wang and Denny Zhou. 2024. Chain-of-thought reasoning without prompting. *arXiv preprint arXiv:2402.10200*.

Xinglin Xiao, Yijie Wang, Nan Xu, Yuqi Wang, Hanxuan Yang, Minzheng Wang, Yin Luo, Lei Wang, Wenji Mao, and Daniel Zeng. 2023. Yayi-uie: A chat-enhanced instruction tuning framework for universal information extraction. *arXiv preprint arXiv:2312.15548*.

Nan Xu, Chunting Zhou, Asli Celikyilmaz, and Xuezhe Ma. 2023. Look-back decoding for open-ended text generation. *arXiv preprint arXiv:2305.13477*.

Ziyang Xu, Keqin Peng, Liang Ding, Dacheng Tao, and Xiliang Lu. 2024. Take care of your prompt bias! investigating and mitigating prompt bias in factual knowledge extraction. *arXiv preprint arXiv:2403.09963*.

Tinghui Zhu, Kai Zhang, Jian Xie, and Yu Su. 2024a. Deductive beam search: Decoding deducible rationale for chain-of-thought reasoning. *arXiv preprint arXiv:2401.17686*.

Wenhong Zhu, Hongkun Hao, Zhiwei He, Yiming Ai, and Rui Wang. 2024b. Improving open-ended text generation via adaptive decoding. *arXiv preprint arXiv:2402.18223*.