Dear reviewers,

We thank you for the feedback and have revised our manuscript to address your concerns. This document outlines the main points of criticism, our responses, and our revisions. We believe that the new version of the paper addresses the reviewers' concerns to the extent possible within the scope of a single publication.

Please note that the line numbers referenced in the original reviews correspond to the previous draft, whereas the line numbers cited in our responses refer to the current version submitted.

Best regards

Reviewer avUN

Summary of Weakness 1: "it is not very clear to me what the key differences are compared to prior work"

Response: We thank the reviewers for pointing out this potential concern. We will provide two differences compared to previous works below. Filtering Effectiveness. During sampling, filtering gives reranking close alignment with the changes in oracle, giving users a chance to select a more aggressive temperature (figure 3). During reranking, we showcase that filtering itself is a powerful reranking strategy already, and it rocks when combined with MBR-Exec, which is missing from previous works of MBR-Exec (Shi et al., 2022), LEVER (Ni et al., 2023), and Self-Debug (Chen et al., 2024). Moreover we show that filtering should be a better strategy and makes Self-Debug (Chen et al., 2024) being marginal if not ineffective, unless we use our proposed approach of SD-Multi. All these modules closely link with each other, and unravels the fundamentals for getting better code generation performance on execution during inference time. Notably, our approach achieves superior performance compared to prior works, as shown in Table 2. Novel Self-Debugging Setup. Unlike previous studies, we apply self-debugging to all candidates (one round), yielding better results than those reported in the original Self-Debug paper (Chen et al., 2024).

Revision: This concern is addressed with substantial changes in the new paper.

1. Theoretically, we unify all previous reranking methods in Section 2 (Importantly, L181-L221 presents how we unify and prove that all previous methods can be either n-best or MBR reranking).

2. Empirically, examples of key differences are explained in L400-L404.
3. We also perform behavior analysis on unit testing (See Section 4.3, L417-445) studying reranking robustness on different unit tests.

Comments Suggestions And Typos 1: "assumed to be able to access the hidden test input"

Response: Thank you for the valuable question. In terms of comparability with other methods, it can be comparable with MBR-Exec (as one of their setting includes hidden unit tests, Shi et al., 2022), LEVER (Ni et al., 2023), Self-Debug (Chen et al., 2024) as these papers adopt the same setting. Comparing with CoderReviewer (Zhang et al., 2023) is a bit different as they compare with MBR-Exec using only MBR on the trial unit tests. We think there's a need to adopt our setting since the trial unit tests can be directly used for filtering, which utilizes both the inputs and the outputs. We will include these details.

Revision: The new version of the paper includes both cases with/without access to some unit test inputs.

Comments Suggestions and Typos 2: "No trial unit tests in MBPP"

Response: We use the same prompting format as EvalPlus and LEVER, where they use the first unit test for evaluation as the trial unit tests. Since it is provided in the prompt, filtering based on them is sensible.

Revision: See L855-L858

"Note that when prompting the model to generate programs for MBPP, EvalPlus adopts the format that uses the first private unit test as the trial unit test."

Comments Suggestions and Typos 3: "Is the MBR-Exec the same as majority vote based on execution results in the experiments?"

Response: Yes but there are nuances. We select the majority of code candidates, even though some of their execution outputs are not in the majority.

Revision: See L188-L202 and Table 2 (Page 3).

Limitations And Societal Impact: "not trying models larger than 16B"

Response: We will include experiments with model up to the size of ~33B in the paper.

Revision: See Table 5 (Page 7) for experiments on models at the size of ~33B. We do not include more experiments due to the page limit.

References

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida I. Wang, and Xi Victoria Lin. 2023. LEVER: learning to verify language-tocode generation with execution. In International Conference on Machine Learning, ICML 2023, 23- 29 July 2023, Honolulu, Hawaii, USA, volume 202 of Proceedings of Machine Learning Research, pages 26106–26128. PMLR.

Tianyi Zhang, Tao Yu, Tatsunori Hashimoto, Mike Lewis, Wen-Tau Yih, Daniel Fried, and Sida Wang. 2023. Coder reviewer reranking for code generation. In International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA, volume 202 of Proceedings of Machine Learning Research, pages 41832–41846. PMLR.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. 2022. Natural language to code translation with execution. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022, pages 3533–3546. Association for Computational Linguistics.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023.

Reviewer HufT

Summary of Weakness 1: “It seems the authors only investigate the importance of choosing better methods for each components. It will be complicated, indeed, to find out if there is a best combination for these components for better code

generation. I assume current conclusion is kind of sub-optimal for this framework?"

Response: We thank the reviewer for giving us the opportunity to explain the contributions. First of all, one of our major contributions is the effectiveness of filtering on trial unit tests during sampling, reranking, and self-debugging. Additionally, with the current best configuration of all components, we further perform self-debug on all candidates (only 1 round) which is missing from the original self-debug paper, with better performance (See Table 2).We will include experiments with model up to the size of ~33B in the paper.

Revision: Same as revision to weakness 1 proposed by avUN , this concern is addressed with substantial changes in the new paper.

1. Theoretically, we unify all previous reranking methods in Section 2 (L151-L221) Importantly, L181-L221 presents how we unify and prove that all previous methods can be either n-best or MBR reranking. No works before have tried this unification.
2. Empirically, examples of key differences are explained in L400-L404.
3. We also perform behavior analysis on unit testing (See Section 4.3, L417-445) studying reranking robustness on different unit tests.

Summary of weakness 2: "It is not sure if the specifications of the methods, such unit test, could effect the results. It is not revealed the complications of the settings and combinations of all the methods."

Response: We thank the reviewer for this comment. We will include more explanations, along with the difference between our work and previous works, in the next draft.We thank the reviewer for giving us the opportunity to explain the contributions. First of all, one of our major contributions is the effectiveness of filtering on trial unit tests during sampling, reranking, and self-debugging. Additionally, with the current best configuration of all components, we further perform self-debug on all candidates (only 1 round) which is missing from the original self-debug paper, with better performance (See Table 2).We will include experiments with model up to the size of ~33B in the paper.

Revision: Section 4.2 (L381-L416) explains the effect of different specifications. Importantly, we adopt all possible contexts.

Reviewer VpeV

Summary of Weakness 1: "...cannot find any novelty nor critical contribution…, we followed…,we followed."

Response: We thank the reviewer for giving us the opportunity to restate the intention of the paper even though it has been stated in the introduction section. The paper is highly motivated by a lack of apples-to-apples comparison of the existing methods. First of all, there is a high necessity to perform this comparison (L55-78), i.e., to point out when and how one method works (or not). We think that, compared to proposing new methods for better reranking results (which is already shown by MBR-Exec+Filtering as close-to-upper-bound, L400-406), performing systematic analysis presents significantly higher importance for solid research progress in inference-time compute. In the next paragraph, we will present an example of filtering, including where it is missing from previous works, and why it shouldn't be missed and should be reported.

In terms of crucial contributions, one of them is that we present the effectiveness of filtering, which is crucial with increased temperature (see Figure 3) compared to MBR-Exec (without filtering) which only gives moderate improvement. Even though filtering has been proposed, it was proposed in a different context (L55-78), and missing in the original papers of MBR-Exec (Shi et al., 2022), LEVER (Ni et al., 2023) and CoderReviewer (Zhang et al., 2023). It is included in the Self-Debug paper (Chen et al., 2024), yet its effect is not reported. One major difference between our work and these works is that we systematically study filtering.

We will clarify the difference between our work and previous works.

Revision: Same as revision to weakness 1 proposed by avUN , this concern is addressed with substantial changes in the new paper.

1. Theoretically, we unify all previous reranking methods in Section 2 (L151-L221) Importantly, L181-L221 presents how we unify and prove that all previous methods can be either n-best or MBR reranking. No works before have tried this unification.
2. Empirically, examples of key differences are explained in L400-L404.
3. We also perform behavior analysis on unit testing (See Section 4.3, L417-445) studying reranking robustness on different unit tests.

Summary of weakness 2: "access to 50+ test cases for filtering?...used to evaluate the correctness of the generated code?..."

Response: Both no. Trial unit tests are provided in the input to the LLM as standard practice. Only MBPP+ includes 1 trial unit test for evaluation as we use that one for

prompting the LLM as we adopt the same experimental setting in EvalPlus (Liu et al., 2023) and LEVER (Ni et al., 2023), while the rest do not. Average trial unit test counts are as follows:

HumanEval+: 2.80; MBPP+: 1; LiveCodeBench: 2.47.

Revision: See L858-L859:

The average numbers of trial unit tests in HumanEval/MBPP/LiveCodeBench are 2.8/1/2.47.

For the access to private unit tests, we explain in L288-L291:

“For a fair comparison across contexts, we use either 100 generated tests or 20 private test inputs for most experiments. For experiments on HumanEval, we use 200 generated tests or 50 private test inputs.”

Comments Suggestions And Typos: “suddenly providing Eq 7 and proceeding without further elaboration or justification…It could have been (the only) interesting part.”

Response: Equation 7 integrates features from Equations 1–6 for n-best reranking and MBR. Section 2.2.2 explains how these features are defined.

Moreover, Eq 7 only presents the case without self-debugging. We also define in L209-222 self-debugging with {1/multiple} candidates, which is also important to the framework.

Revision: We replace the entire math session with a new Section 2. We only use verbal explanations for self-debugging (L139-L147) to avoid confusion. We explain our differences to Chen et al. (2024) in Table 3 (Page 4).

References

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida I. Wang, and Xi Victoria Lin. 2023. LEVER: learning to verify language-tocode generation with execution. In International Conference on Machine Learning, ICML 2023, 23- 29 July 2023, Honolulu, Hawaii, USA, volume 202 of Proceedings of Machine Learning Research, pages 26106–26128. PMLR.

Tianyi Zhang, Tao Yu, Tatsunori Hashimoto, Mike Lewis, Wen-Tau Yih, Daniel Fried, and Sida Wang. 2023. Coder reviewer reranking for code generation. In International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA, volume 202 of Proceedings of Machine Learning Research, pages 41832–41846. PMLR.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. 2022. Natural language to code translation with execution. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022, pages 3533–3546. Association for Computational Linguistics.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023.

Meta Reviewer oZ73

Summary Of Suggested Revisions 1: “Highlight and justify the differences between the proposed framework and prior work. Provide concrete examples of novel insights derived from the unified framework.”

Revision: Same as revision to weakness 1 proposed by avUN , this concern is addressed with substantial changes in the new paper.

1. Theoretically, we unify all previous reranking methods in Section 2 (L151-L221) Importantly, L181-L221 presents how we unify and prove that all previous methods can be either n-best or MBR reranking. No works before have tried this unification.
2. Empirically, examples of key differences are explained in L400-L404.
3. We also perform behavior analysis on unit testing (See Section 4.3, L417-445) studying reranking robustness on different unit tests.

Summary Of Suggested Revisions 2: “Address the uniqueness of filtering and self-debugging approaches in the context of existing works.

Revision: See Table 3 (Page 4) for a clear comparison. Note that our work focuses on analysis (L231-L237).

Summary Of Suggested Revisions 3: “Elaborate on how canonical reranking and MBR decoding are combined. Provide theoretical justifications and empirical results to support the approach.

Revision: Theoretical justifications are provided in L151-L221. Importantly, L181-L221 presents how we unify and prove that all previous methods can be either n-best or MBR reranking. Feel free to also check out L802-L842 for some proofs that integrate some reranking metrics into the MBR decoding domain.

Summary Of Suggested Revisions 4: “Include details about the use of test cases for filtering and the implications on real-world scenarios.

Revision:

1. We explain details about using trial unit test cases in L128-L133 and L139-L147.
2. We explain cases regarding access to private unit tests in 162-L180.
3. We explain filtering on trial unit tests in Table 1 (page 3).

Evaluate the framework using larger language models (beyond 16B) to strengthen the generalizability of findings.

Revision: See Table 6 (page 7) for using models beyond 16B parameters.

Explore alternative reranking configurations and their impact on performance.

Revision: We have already included the use of generated test cases with different configurations. See L151-L221 for the theoretical framework, and see Table 3 (Page 4) for a clear comparison. Note that our work focuses on analysis (L231-L237).

Results are provided in Table 4 (Page 5) with discussions in L381-L416.

Also see Table 8 (Page 14) for reranking with non-execution-based signals as preliminary experiments.

Release code and datasets to facilitate reproducibility.

Revision: We have stated in L23-L25, and will release code and datasets upon acceptance. Note that we have also contributed a unit test classifier dataset (L304-L327).