Inference-Time Masking for Retrieval-Augmented Code Generation

Anonymous ACL submission

Abstract

Retrieval-Augmented Code Generation utilizes relevant code examples as auxiliary context to improve model performance in code generation tasks. To gradually refine the generated code, an iterative strategy can be used, in which the code generated in the last iteration is utilized to retrieve relevant code examples. However, the effectiveness of this strategy diminishes after the second iteration, as the retrieved code examples remain the same and the generated code becomes similar. To address this issue, we propose an **I**nference-time **M**asking strategy for **R**etrieval-**A**ugmented **C**ode **G**eneration (IM-RACG), where the retrieved code examples are masked before being used as auxiliary context. By masking parts of the examples, the diversity of the auxiliary context is increased and the context length is reduced effectively. Given the low information density of code, the remaining context still contains valuable information. As a result, this strategy encourages the model to generate more diverse code, leading performance to scale with the number of iterations. Experimental results on MBPP and HumanEval datasets demonstrate that IM-RACG significantly enhances all tested model's performance across, with an average improvement of approximately **4.5%** in pass rate compared to the original iterative RACG. Additionally, IM-RACG shows the greatest enhancement on MBPP using Llama-8b, with an increasement of the pass rate from 76.2% to **83.4%**.

1 Introduction

Large language models (LLMs) (Radford, 2018; Brown et al., 2020; Touvron et al., 2023) have demonstrated impressive capabilities across various tasks (Fatemi et al., 2024), including code generation (Chen et al., 2021). However, LLMs struggle with factual errors due to the limitations in their training data and a potential lack of real-time or domain-specific knowledge (Mallen et al., 2023; Min et al., 2023). Retrieval-Augmented Generation (RAG) (Lewis et al., 2020; Guu et al., 2020) addresses this issue by retrieving from extensive sources and construct an auxiliary context using the retrieved content. This approach enables LLMs to access external information, hence improve the relevance, coherence and factual accuracy of the generated content (Gao et al., 2023).

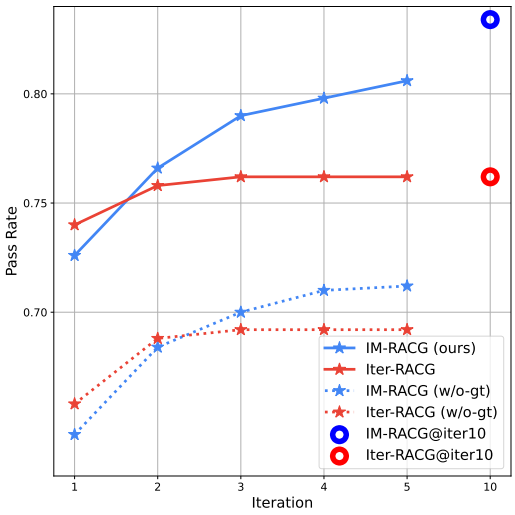

Figure 1: Comparison of pass rates between Iter-RACG and IM-RACG on code generated using Llama-8b on the MBPP dataset. "w/o-gt" indicates that the ground truth answer for the current question cannot be retrieved.

Although RAG has been extensively studied for natural language processing (NLP) tasks (Lewis et al., 2020), its application to code generation remains in the early stages of exploration. Existing retrieval-augmented code generation (RACG) works (Zhang et al., 2023; Wang et al., 2024) directly use the whole retrieved content as the auxiliary context as that in the NLP, neglecting the difference in information density between natural language and programming language. Specifically, compared to natural language, code exhibits a lower information density as it contains more

repetitive patterns. This characteristic arises from the design of programming languages, which prioritize high re-usability to mitigate the cognitive challenges involved in reading and understanding (Casalnuovo et al., 2019). This means that some components are essential for the functionality of code, but carry limited semantic information independently, such as variable names or keywords. For instance, some `return` statement is not meaningful without other statements in the function body. Another example of low information density data is image (He et al., 2022), where large amount of background pixels are often similar and repetitive, such as grass or sky. While background pixels provide contextual information, they can also introduce noise that interferes with object detection (Xiao et al., 2021). Likewise, code with similar repetitive patterns are retrieved and used as a contextual information for RACG. Neglecting the subtle differences in their implementation logic can introduce noise, potentially leading to a decline in generation quality.

To improve the effectiveness of RACG, it is reasonable to retrieve more examples or examples with higher relevance (Du et al., 2024). However, including more examples increases the risk of introducing more potential noise and raises the computational cost (Wu et al., 2024). Additionally, retrieving examples with higher relevance requires additional data processing and are typically applicable only to specific scenarios (Liu et al., 2024a). Another approach to enhancing the effectiveness of RACG is to use an iterative strategy to retrieve more relevant examples and gradually refine the generated code (Zhang et al., 2023). However, as the number of iterations increases, the effectiveness of iterative strategy diminishes after the second iteration. As illustrated in Figure 1, the pass rate achieved by Iter-RACG stabilizes after the second iteration, while the pass rate achieved by IM-RACG (our method) surpasses that of Iter-RACG starting from the third iteration. Furthermore, as the number of iterations increases, the performance gap between Iter-RACG and IM-RACG continues to widen.

Based on our observation that code has lower information density, we conjecture that only a portion of the code in the retrieved relevant examples contributes to the improvement of current code generation. However, given the same retrieved document, different LLMs may generate varying code outputs. It is challenging to predict in advance which parts of the retrieved code will be most beneficial for the model in practical use.

To address the challenges in iterative RACG, we leverage the low information density of code to propose the **I**nference-time **M**asking strategy for **R**etrieval-**A**ugmented **C**ode **G**eneration (IM-RACG), which contains two key strategies: (1) *an inference-time masking strategy* to randomly remove portions of the retrieved code, and (2) *an iterative retrieval-augmented strategy* for code generation. These two strategies mutually reinforce each other. By randomly masking portions of the retrieved code, we increase the diversity of the auxiliary context and eliminates the need to identify which specific segment of the code is most beneficial. Furthermore, the iterative strategy can effectively addresses scenarios where the most beneficial segment may be masked in certain iterations. By combining the masking strategy, the performance improves steadily as the number of iterations increases.

Our paper presents three main contributions:

1. To the best of our knowledge, this paper is the first to explore how to use the low information density of code in designing RACG strategies.

2. We propose the IM-RACG, which combines the inference-time masking strategy and iterative retrieval-augmented strategy, significantly reduces context length and improves model performance.

3. Experimental results on five code LLMs and two datasets demonstrate that our method significantly improves the pass rate for generating code from natural language descriptions.

2 Related Work

2.1 Information Density

Information density (Pinsker, 1964) is used to quantify the amount of information about the underlying probability distribution given a specific outcome of a random variable. It differs between different data distributions. For example, the information density of natural language differs from that of images (He et al., 2022). This means that infilling missing words within natural language requires sophisticated linguistic understanding ability, whereas missing patches in an image can be more easily reconstructed based on neighboring patches. This can be reflected by masking rate, He et al. (2022) shows that a masking rate of 75% achieves the optimal result for pre-training an image encoder. In

contrast, pre-training a text encoder requires an optimal masking rate of 15% (Devlin et al., 2019).

Compared to natural language, code contains more repetitive patterns and has lower information density (Casalnuovo et al., 2019). Existing RACG methods neglect this difference (Zhang et al., 2023; Du et al., 2024), resulting in redundant or even detrimental segments in the retrieved code examples, which introduce potential risks for code generation. Hence, we propose an inference-time masking strategy to make the generation model leverage incomplete code as the auxiliary context. We determine the mask ratio by relevance, as more relevant code is likely to contain more useful segments. This approach enables the model to balance its inherent knowledge and the retrieved information.

2.2 Retrieval-Augmented Code Generation

Retrieval-augmented generation (RAG) is an emerging strategy designed to address the limitations of relying solely on parametric knowledge during generation. By integrating a retrieval and a generation module, RAG strategy not only can generate fluent texts but also ground their outputs in real-world and up-to-date data. This approach has been applied to code generation task, named retrieval-augmented code generation (RACG) (Parvez et al., 2021), primarily to address the discrepancy between training and test data, such as the model's inability to update its internal knowledge with real-time code repositories (Zhou et al., 2023), or the exclusion of private code-bases from its training data (Zhang et al., 2023).

However, research suggests that most retrieved content provides little benefit for code generation (Wu et al., 2024). Even when the retrieved context includes the correct code for the question, it does not guarantee accurate code generation (Wang et al., 2024). In addition, to further improve the performance of RACG, Zhang et al. (2023) introduces an iterative strategy. However, their experiments show that the effectiveness of this strategy diminishes as the iteration exceeds two. Based on these observations and the low information density of code, we propose to integrate a masking strategy to further enhance iterative RACG.

3 Methodology

3.1 Problem Formulation

Our method addresses the problem of generating code based on a natural language instruction or description by leveraging the strengths of LLMs and RACG. Given a natural language description $S = \{s_1, s_2, \ldots, s_n\}$, the goal is to generate an executable code $C = \{c_1, c_2, \ldots, c_m\}$ that accurately matches the intent described in the natural language and passes all the corresponding tests. The code is generated auto-regressively by a large language model, written as $c_i = \text{LLM}(S, c_{<i})$. RACG enhances the generation process by leveraging relevant context retrieved from a database.

In order to retrieve relevant examples, we construct a query $Q = \{S; C\}$, by concatenating the natural language description S and the generated code C. This query is used to retrieve k relevant examples $R = \{R^{(1)}, R^{(k)}, \ldots, R^{(k)}\}$ from the database. Each example $R^{(i)} = \{S^{(i)}; C^{(i)}\}$ contains a natural language description $S^{(i)}$ and the corresponding code $C^{(i)}$. Given the retrieved examples, an auxiliary context $A = \{R^{(1)}, R^{(k)}, \ldots, R^{(k)}, S\}$ is obtained by combining the retrieved examples with the original description. This context provides rich information for the LLM, leading the generation process to $c_i = \text{LLM}(A, c_{<i})$. To facilitate this process, we utilize an instruction tuned version of code LLMs, which exhibits strong instruction following capabilities and can leverage additional context effectively.

3.2 IM-RACG

We propose an iterative retrieval-augmented strategy for code generation and refine the auxiliary context after each retrieval using a masking strategy, named **IM-RACG** (**I**nference-time **M**asking strategy for **R**etrieval-**A**ugmented **C**ode **G**eneration). By integrating the iterative retrieval strategy with the masking strategy, the auxiliary context is more diverse and compact, leading to the improvement of effectiveness in code generation.

IM-RACG comprises three main steps as depicted in Figure 2: (1) generating code using a zero-shot approach based on the system prompt and the natural language description; (2) if the zero-shot generation fails, the RACG mode begins. We retrieve relevant examples using a database query, which is a concatenation of the current problem description and the generated code in the first step. These retrieved examples are then randomly masked and used to construct a new auxiliary context. (3) if the RACG fails, the iterative RACG begins. During each iteration, we use the current problem description and the code generated in the last iteration as a database query to retrieve relevant

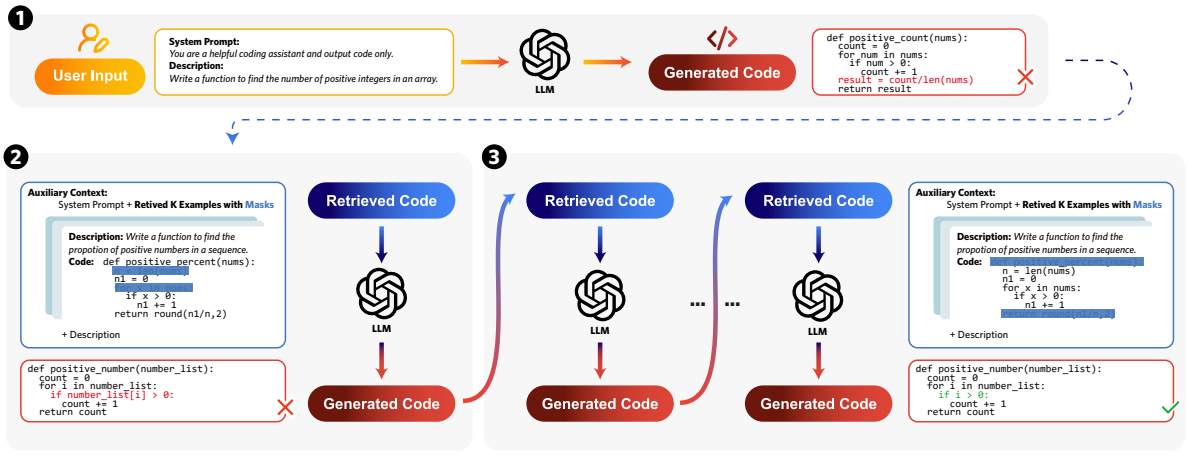

Figure 2: The overall workflow of IM-RACG.

examples. Then the auxiliary context is updated using a masked version of these examples. Notably, the generated code is tested against a set of test cases. Once the code passes all the tests, the iterative process is terminated. This strategy aligns with the principles of Test-Driven Development (TDD) where test code is written prior to the production code (Mäkinen and Münch, 2014). The maximum number of iterations is a hyperparameter, which will be discussed in detail in Section 5.1.

In the following, we introduces three key aspects of our approach: (1) the methodology for retrieving relevant examples from a database; (2) the determination of the mask ratio; (3) the approach for utilizing an LLM to generate code.

Retrieve Relevant Examples IM-RACG uses a dense retriever, which retrieves relevant code based on vector similarity. Specifically, given a retrieval dataset $D = \{d_1, d_2, \ldots, d_j\}$ containing J entries, we encode each entry into a vector h_{d_i} using a pre-trained sequence encoder. Given a query Q, we use the same encoder to encode it, yielding the representation h_Q. The similarity between the query and the data in the retrieval database is calculated as following:

$$sim(Q, d_j) = \frac{h_Q \cdot d_j}{||h_Q|| \cdot ||h_{d_j}||} \quad (1)$$

Mask Ratio Determination After retrieving the relevant examples, masking is applied to the code in these examples. The mask ratio is determined based on the similarity between the retrieved example and the query. For example, the masking ratio for an example is calculated by:

$$M_{R^{(i)}} = M_{base} * (1 - sim(R^{(i)}, Q)) \quad (2)$$

where $R^{(i)}$ is the i-th retrieved example, M_{base} is the base mask ratio which is a hyperparameter and $M_{R^{(i)}} \in [0, 1]$. According to Equation 2, a higher similarity between a retrieved example and the query corresponds to a lower masking ratio. This adjustment facilitates the retention of greater amounts of information from examples that are more similar to the query.

Code Generation After masking is applied, the retrieved examples contains a masked version of code, $\tilde{R}^{(i)} = \{S^{(i)}; \tilde{C}^{(i)}\}$. These examples are used to construct an auxiliary context $\tilde{A} = \{\tilde{R}^{(1)}, \tilde{R}^{(k)}, \ldots, \tilde{R}^{(k)}, S\}$ to facilitate code generation. The probability of generating a code sequence is given by:

$$P(C) = \prod_{t=1}^{T} P_\theta(c_t | \tilde{A}, c_{<t}) \quad (3)$$

where θ is the parameters of a code generation model and T is the length of the code sequence.

4 Experiments

Section 4.1 details the experimental setups, including datasets, evaluation metrics, models, and tools. Sections 4.2 and 4.3 demonstrate the effectiveness of IM-RACG in scenarios where the ground truth can or cannot be retrieved, respectively.

4.1 Experimental Setup

We use the widely used MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021) datasets to validate the effectiveness of IM-RACG. MBPP includes 974 samples, of which 500 are test samples. HumanEval only has a test set which contains 164

samples. Each sample contains a natural language description, a corresponding code, and several test cases. Pass rate is used as the evaluation metric. Since we use an iterative approach to generate code, if the generated code passes all the test cases in any of the iteration, the sample is considered as a "pass". All the samples in MBPP and HumanEval are used to construct the retrieval database and the evaluation is conducted on the test set.

For each sample in the dataset, we concatenate the natural language description with the code and use Bge-base[1] to encode it into a vector. As a result, we get a total of 1,138 records in our vector database. We use Faiss[2] as our database construction and querying tool. Instruction tuning version of LLMs is used for code generation, as they can utilize retrieved context more effectively.

To study the effectiveness of IM-RACG, we conduct experiments on LLMs with parameter scales ranging from 3B to 12B, including *Phi-3.5-mini-instruct* (Phi-3B) (Abdin et al., 2024), *Llama-3.2-3B-Instruct* (Llama-3B) (Liu et al., 2024b), *deepseek-coder-7b-instruct-v1.5* (DeepSeek-7B) (Guo et al., 2024), *Llama-3.1-8B-Instruct* (Llama-8B) (Liu et al., 2024b), and *Mistral-Nemo-Instruct-2407* (Mistral-12B) [3].

All experiments were conducted on an NVIDIA RTX A6000. All the models used in the experiments are publicly available and the specific URLs for each model can be found in Appendix A. The default number of retrieval candidate is three and greedy decoding strategy is used for code generation for reproducibility.

4.2 Effectiveness of IM-RACG

The main results are shown in Table 1. We conclude that, IM-RACG consistently improves the code generation pass rate across all baseline models. On MBPP dataset, Llama-8B shows the strongest performance across all configurations, achieving a pass rate of 80.6% with IM-RACG. On HumanEval dataset, Phi-3B achieves the highest pass rate of 76.2% with IM-RACG. It can be notice that all RACG-based methods outperform the Zero-Shot approach significantly. This improvement is partially due to the inclusion of test set samples in the database, ensuring that the ground truth answer is often present in the auxiliary context. However, it is important to note that, even when the ground truth is included, RACG-based methods still generate incorrect results in some cases. Compared to Iter-RACG which does not apply masking strategy, IM-RACG shows consistent improvements across all models with a maximum iteration of five[4], highlighting the effectiveness of our strategy.

Model	Method	MBPP	HumanEval
Phi-3B	Zero-Shot	45.4%	68.3%
Phi-3B	RACG	62.8%	70.1%
Phi-3B	Iter-RACG	67.4%	70.1%
Phi-3B	IM-RACG	72.6%	**76.2%**
Llama-3B	Zero-Shot	36.6%	50.0%
Llama-3B	RACG	69.8%	57.3%
Llama-3B	Iter-RACG	71.4%	59.2%
Llama-3B	IM-RACG	74.8%	62.2%
DeepSeek-7B	Zero-Shot	56.4%	42.1%
DeepSeek-7B	RACG	73.6%	51.8%
DeepSeek-7B	Iter-RACG	76.0%	53.7%
DeepSeek-7B	IM-RACG	78.8%	62.2%
Llama-8B	Zero-Shot	53.2%	56.1%
Llama-8B	RACG	74.2%	62.2%
Llama-8B	Iter-RACG	76.2%	62.8%
Llama-8B	IM-RACG	**80.6%**	69.5%
Mistral-12B	Zero-Shot	38.2%	48.8%
Mistral-12B	RACG	64.4%	58.5%
Mistral-12B	Iter-RACG	65.4%	59.8%
Mistral-12B	IM-RACG	67.0%	63.4%

Table 1: Comparison of different methods for code generation. Zero-Shot means code generation without RAG context. Both RACG and Iter-RACG use the entire retrieved code as the context. In Iter-RACG and IM-RACG, the maximum number of iterations is set to 5. The base mask ratio for IM-RACG is 0.5. The highlighted rows represent the optimal methods for each model and **bold** represent the best result on each dataset.

It can be observed that the model performance varies significantly under the Zero-Shot setting and the pass rate does not increase monotonically with parameter scale. Llama-3B performs poorly in the Zero-Shot setting, but shows strong capability in leveraging the auxiliary context. On the other hand, although Mistral-12B contains a greater number of parameters compared to other models, its performance in the Zero-Shot setting is only comparable with Llama-3B. A possible explanation for this could be that Mistral-12B does not prioritize code generation tasks as highly as other tasks.

[1] https://huggingface.co/BAAI/bge-base-en-v1.5
[2] https://github.com/facebookresearch/faiss
[3] https://mistral.ai/news/mistral-nemo/

[4] As the number of iterations increases, the performance advantage of IM-RACG over Iter-RACG becomes progressively more pronounced.

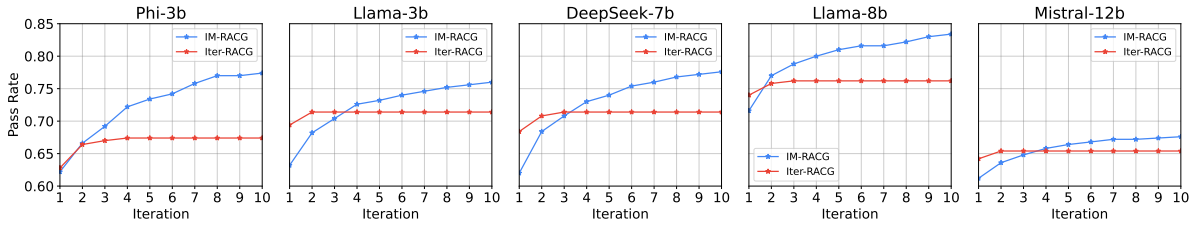

Figure 3: A comparison between IM-RACG and Iter-RACG as the maximum number of iteration increases on MBPP dataset, with a base mask ratio of 0.5 for IM-RACG. In both settings, the number of retrieval examples is 3.

4.3 IM-RACG without Ground Truth

As the test examples are included in the vector database, the ground truth question-answer pair is usually the first retrieval candidate. This is an ideal case for RACG (Wang et al., 2024), however, it is more common to retrieve relevant but not identical examples in practice. Therefore, we further verify the effectiveness of IM-RACG when the ground truth example cannot be retrieved.

As shown in Table 2, we evaluate the performance of various RACG strategies across all baseline models by excluding the ground truth example from the retrieval candidates. Overall, the results demonstrate that IM-RACG consistently outperforms RACG and Iter-RACG on MBPP and HumanEval datasets. This indicates that IM-RACG is effective scenarios where ground truth examples are unavailable, positioning it as a reliable alternative compared to other RACG strategies.

Model	Method	MBPP	HumanEval
Phi-3B	Zero-Shot	45.4%	68.3%
Phi-3B	RACG	60.0%	70.1%
Phi-3B	Iter-RACG	63.0%	70.1%
Phi-3B	IM-RACG	67.2%	**74.4%**
Llama-3B	Zero-Shot	36.6%	50.0%
Llama-3B	RACG	56.4%	50.0%
Llama-3B	Iter-RACG	60.0%	54.9%
Llama-3B	IM-RACG	63.6%	57.3%
DeepSeek-7B	Zero-Shot	56.4%	42.1%
DeepSeek-7B	RACG	63.4%	46.3%
DeepSeek-7B	Iter-RACG	67.0%	51.8%
DeepSeek-7B	IM-RACG	71.0%	61.0%
Llama-8B	Zero-Shot	53.2%	56.1%
Llama-8B	RACG	65.8%	61.6%
Llama-8B	Iter-RACG	69.2%	62.8%
Llama-8B	IM-RACG	**71.2%**	69.5%
Mistral-12B	Zero-Shot	38.2%	48.8%
Mistral-12B	RACG	56.4%	52.4%
Mistral-12B	Iter-RACG	58.0%	53.6%
Mistral-12B	IM-RACG	60.4%	59.2%

Table 2: Comparison of different methods when ground truth candidate cannot be retrieved.

5 Discussion

In this section, we discuss several factors that may affect the effectiveness of IM-RACG.

5.1 Scaling the Iteration

To evaluate whether increasing the maximum number of iterations further improves performance, we conducted experiments by increasing the iteration limit to 10 on MBPP. As shown in Figure 3, IM-RACG exhibits a consistent upward trend as the the number of iteration increases. At the 10^{th} iteration, the pass rate achieved by IM-RACG outperforms Iter-RACG across all the models we tested. Notably, the highest relative gain is obtained by Phi-3B, which is **14.8%** (from 67.4% to 77.4%). The highest pass rate is achieved by Llama-8B, which is **83.4%** at the 10^{th} iteration with IM-RACG.

Compared to IM-RACG, Iter-RACG exhibits a higher pass rate at the first iteration, however, there is only a marginal improvement in the second and third iterations, after which the pass rate plateaus. These results suggest that the iterative strategy alone cannot yield diverse auxiliary contexts, thereby limiting the model's ability to produce varied outputs. Although IM-RACG performs slightly worse in early iterations, it outperforms Iter-RACG across all models by the fourth iteration. Notably, on the best-performing baseline model, Llama-8B, IM-RACG surpasses Iter-RACG as early as the second iteration and the pass rate increases steadily afterwards. This demonstrates that the masking strategy in IM-RACG enhances its compatibility with the iterative retrieval-augmented strategy by introducing diversity.

5.2 Effect of Description for Retrieved Code

We investigate whether including a natural language description in the retrieved examples enhances code generation quality or if the retrieved

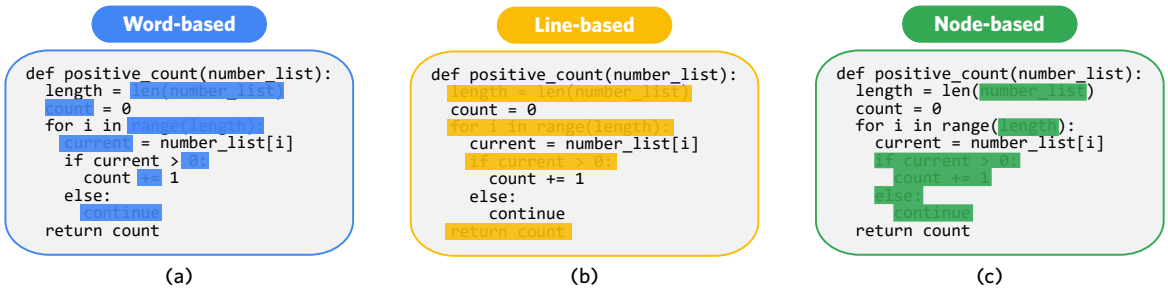

Figure 4: Three mask sampling strategies.

Model	NL	PR
Phi-3B	w/o	52.0%
Phi-3B	w/	67.4%
Llama-3B	w/o	62.2%
Llama-3B	w/	72.0%
DeepSeek-7B	w/o	72.6%
DeepSeek-7B	w/	76.0%
Llama-8B	w/o	71.2%
Llama-8B	w/	**76.2%**
Mistral-12B	w/o	60.6%
Mistral-12B	w/	65.4%

Table 3: Evaluation of the effect of including natural language (NL) description in the context across different models on MBPP. The number of retrieved examples is set to 3 and all the results are obtained using Iter-RACG. PR represents Pass Rate. "w/o" and "w/" means without and with natural language description, respectively.

Model	MSS	BMR	PR
Phi-3B	Word-based	0.7	65.8%
Phi-3B	Line-based	0.7	**70.8%**
Phi-3B	Node-based	0.7	59.0%

Table 4: Comparison of word-based, line-based and node-based mask sampling strategies with a Base Mask Ratio (BMR) of 0.7 on MBPP using Phi-3B. MSS represents mask sampling strategy.

code alone provides sufficient information. To evaluate this, we use MBPP to assess the impact of descriptions on retrieved code. Specifically, we construct a new vector database using only code. When relevant code is retrieved, instead of appending a description of its functionality, we add the sentence, "*This code snippet may be useful.*"

As presented in Table 3, including natural language descriptions significantly improves the pass rate, suggesting that code alone provides insufficient context. This finding implies that the utility of code is enhanced when accompanied by a description of its functionality. Notably, the greatest performance improvement is observed in DeepSeek-7B, suggesting that the model struggles to utilize code without adequate contextual information. This result is intuitive, as identifying relevant components for a given problem becomes more difficult without understanding of the broader context in which the code is applied. This observation highlights the impact of low information density in code, particularly in the context of RACG.

5.3 Exploration of Mask Sampling Strategies

In order to find an optimal mask sampling strategy, we investigate different strategies with a fixed base mask ratio of 0.7, as shown in Figure 4. These strategies include word-based, line-based, and node-based masking strategy. For word-based masking, we use whitespace characters to split the retrieved code. For line-based masking, newline characters are used to split the retrieved code. Lastly, for node-based masking, we utilize Tree-sitter[5] to parse the retrieved code so that we get a list of nodes from the parse tree. For instance, Figure 4 shows that the condition expression in the `if` statement can be masked by removing the corresponding node.

Experimental results shown in Table 4, reveal that the line-based masking strategy achieves the highest pass rate. This can be intuitively explained by the granularity of the different strategies. Word-based and node-based masking strategies often introduce overly fine-grained perturbations, disrupting reusable code snippets in the retrieved examples. In contrast, a line of code typically represents a meaningful and self-contained statement, making it a more suitable unit for perturbation. Given its simplicity and effectiveness, the line-based masking strategy is selected as the default setting.

[5] https://tree-sitter.github.io/tree-sitter/

Method	MR	PR	AvgLen	AvgIter
Zero-Shot	-	45.4%	0	1
Iter-RACG	0	67.4%	616.57	1.91
IM-RACG	14.9%	**72.6%**	524.66	1.89
IM-RACG	35.7%	68.0%	396.69	2.08
IM-RACG	56.6%	66.8%	267.41	2.16
IM-RACG	78.2%	62.0%	134.30	2.26

Table 5: Comparison of code length in the auxiliary context at different Mask Ratios (MR) on MBPP. AvgLen represents the average number of characters of code in the auxiliary context. AvgIter represents the average number of iterations for answering each question.

Model	Nucleus	Mask
Phi-3B	73.8%	78.4%
Llama-3B	72.2%	74.6%
DeepSeek-7B	81.2%	81.0%
Llama-8B	78.6%	78.8%
Mistral-12B	70.4%	71.0%

Table 6: Compare the pass rate achieved by applying nucleus sampling to that achieved by mask sampling on MBPP dataset with a base mask ratio of 0.7.

5.4 The Role of Base Mask Ratio

As the base mask ratio increases, a larger portion of the code is masked, leading to a reduction in the amount of information available for models to utilize. However, a higher base mask ratio also increases the diversity of the auxiliary context, enabling the model to generate a broader range of outputs. Therefore, the base mask ratio plays a crucial role in balancing information availability and context diversity. It is important to note that a base mask ratio of 1.0 does not imply that the entire code is masked, as the similarity score of the retrieved example is also considered. Iter-RACG can be interpreted as a special case of IM-RACG with a base mask ratio of 0, and Zero-Shot can be viewed as a case where the base mask ratio is ∞.

Another advantage of IM-RACG is the reduction in the length of the code used as auxiliary context, achieved by masking tokens from the original code. As shown in Table 5, at a base mask ratio of 0.5, the average number of characters in the code decreases by 14.93% and the average number of iterations is decreased slightly, while the pass rate improves by 7.72% relatively. Additionally, the pass rate exhibits a non-monotonic trend as the base mask ratio increases. This suggests that a moderate level of masking can improve the performance of code generation. As the base mask ratio continues to rise, performance remains robust. For instance, at a ratio of 2.0, where only an average of 134.3 characters remains, the performance still significantly surpasses that of the Zero-Shot setting.

5.5 Evaluating Mask Sampling Efficiency

Distribution annealing techniques, such as nucleus sampling (Holtzman et al., 2019), can significantly improve the performance of auto-regressive models. As the retrieved code examples are masked randomly, our approach can be viewed as a form of sampling strategy. The goal of the experiments in this section is to compare the effectiveness of nucleus sampling and mask sampling for retrieval-augmented code generation. To verify the effectiveness, we generate five code candidates under two settings with a fixed mask rate. In the first setting, we apply nucleus sampling to a masked version of the retrieval code examples to generate five code candidates. In the second setting, we perform mask sampling five times to obtain five distinct masked versions of the retrieval code examples and then one code candidate is generated using nucleus sampling for each masked version. As shown in Table 6, except for DeepSeek-7B, the pass rate achieved through nucleus sampling is slightly lower than that achieved by mask sampling, which demonstrates the effectiveness of the mask sampling strategy. Finally, as distribution annealing techniques are performed on the output side and mask sampling strategy operates on the input side, these two sampling techniques can coexist, potentially leading to further performance improvements.

6 Conclusion

In this work, we introduce IM-RACG, a novel RACG strategy that combines iterative retrieval-augmented generation with an inference-time masking strategy based on the relevance of the retrieved examples to leverage the low information density of code. Experimental results on various LLMs demonstrate that IM-RACG significantly outperforms top-notch baseline methods in the code generation task, not only improving pass rates but also reducing the length of the auxiliary context when compared to traditional iterative RACG approaches. Furthermore, the performance of IM-RACG continues to improve as the number of iterations increases, highlighting its stability and potentiality for handling more complex tasks.

7 Limitations

This study explores a scenario where a large language model is required to generate code given a natural language description. However, other scenarios such as repository-level code completion (Shrivastava et al., 2023) and code repair (Lu et al., 2021) may also be explored. In the current experiments, the whole MBPP and HumanEval datasets are used as the retrieval source and a relatively small number of retrieval examples are utilized. Future research could examine the application of IM-RACG with a larger, more diverse database and incorporate more retrieval examples.

Further investigation is needed to explore the limits of IM-RACG's effectiveness, particularly in tasks with higher complexity(Hendrycks et al., 2021). Additionally, predicting which portions of code are the most beneficial for a model, given the current question, presents an intriguing area of study. Successful identification of the most relevant code segments could significantly enhance performance and potentially eliminate the need for iterative retrieval, thereby improving both efficiency and accuracy in code generation tasks.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *CoRR*, abs/2404.14219.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Casey Casalnuovo, Kenji Sagae, and Prem Devanbu. 2019. Studying the difference between natural and programming language corpora. *Empirical Software Engineering*, 24:1823–1868.

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*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Kounianhua Du, Renting Rui, Huacan Chai, Lingyue Fu, Wei Xia, Yasheng Wang, Ruiming Tang, Yong Yu, and Weinan Zhang. 2024. Codegrag: Extracting composed syntax graphs for retrieval augmented cross-lingual code generation. *arXiv preprint arXiv:2405.02355*.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2024. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *CoRR*, abs/1904.09751.

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.

Jiaheng Liu, Ken Deng, Congnan Liu, Jian Yang, Shukai Liu, He Zhu, Peng Zhao, Linzheng Chai, Yanan Wu, Ke Jin, et al. 2024a. M2rc-eval: Massively multilingual repository-level code completion evaluation. *arXiv preprint arXiv:2410.21157*.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024b. Spinquant: Llm quantization with learned rotations. *Preprint*, arXiv:2405.16406.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Simo Mäkinen and Jürgen Münch. 2014. Effects of test-driven development: A comparative analysis of empirical studies. In *Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering*, pages 155–169, Cham. Springer International Publishing.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822, Toronto, Canada. Association for Computational Linguistics.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12076–12100, Singapore. Association for Computational Linguistics.

Md Rizwan Parvez, Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval augmented code generation and summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2719–2734.

Mark S Pinsker. 1964. Information and information stability of random variables and processes. *Holden-Day*.

Alec Radford. 2018. Improving language understanding by generative pre-training.

Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. 2023. Repository-level prompt generation for large language models of code. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 31693–31715. PMLR.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2024. Coderag-bench: Can retrieval augment code generation? *arXiv preprint arXiv:2406.14497*.

Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. 2024. Repoformer: Selective retrieval for repository-level code completion. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Kai Yuanqing Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. 2021. Noise or signal: The role of image backgrounds in object recognition. In *International Conference on Learning Representations*.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. Repocoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2471–2484.

Shuyan Zhou, Uri Alon, Frank F. Xu, Zhengbao Jiang, and Graham Neubig. 2023. Docprompting: Generating code by retrieving the docs. In *The Eleventh International Conference on Learning Representations*.

A Appendix

A.1 Implementation Details

We use *Phi-3.5-mini-instruct (Phi-3B)*[6], *Llama-3.2-3B-Instruct (Llama-3B)*[7], *deepseek-coder-7b-instruct-v1.5 (DeepSeek-7B)*[8] *Llama-3.1-8B-Instruct (Llama-8B)* [9] and *Mistral-Nemo-Instruct-2407 (Mistral-12B)*[10].

[6] https://huggingface.co/microsoft/Phi-3.5-mini-instruct

[7] https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

[8] https://huggingface.co/deepseek-ai/deepseek-coder-7b-instruct-v1.5

[9] https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

[10] https://huggingface.co/mistralai/Mistral-Nemo-Instruct-2407