Across Programming Language Silos: A Study on Cross-Lingual Retrieval-Augmented Code Generation

Qiming Zhu[1,2], **Jialun Cao**[3*], **Xuanang Chen**[1*], **Weili Zhang**[1], **Yaojie Lu**[1], **Hongyu Lin**[1], **Xianpei Han**[1], **Le Sun**[1], **Shing-Chi Cheung**[3]

[1]Chinese Information Processing Laboratory, Institute of Software, Chinese Academy of Sciences

[2]University of Chinese Academy of Sciences

[3]The Hong Kong University of Science and Technology

{zhuqiming2022,chenxuanang,weili,luyaojie,hongyu,xianpei,sunle}@iscas.ac.cn

{jcaoap,scc}@cse.ust.hk

Abstract

Current research on large language models (LLMs) with retrieval-augmented code generation (RACG) has largely focused on single-language settings, leaving their cross-lingual effectiveness underexplored. Multilingual RACG systems are increasingly important for migrating and reusing code across programming languages (*PL*s), a common yet challenging task in modern software development. To systematically study cross-lingual code knowledge transfer in RACG, we construct a dataset covering 13 *PL*s with nearly 14K instances. Our experiments reveal three key insights: (1) Knowledge transfer in RACG across *PL*s is non-trivial even using direct injection. (2) RACG exhibits unequal cross-lingual knowledge transfer, and its efficacy depends on linguistic affinity of *PL* pair and diversity of LLM pretraining corpus. (3) RACG shows limited reliance on natural language information embedded in code when equipped with a code-specific retriever. These findings provide practical guidance for designing effective multilingual RACG systems. https://github.com/icip-cas/Cross-Lingual-RACG.

1 Introduction

With the emergence and rapid development of large language models (LLMs), significant progress has been made in natural language (NL) to code generation task (Chen et al., 2021; Rozière et al., 2023). Despite these advances, code-oriented LLMs still struggle to generate reliable and correct programs in complex real-world scenarios, largely due to limitations in model capacity and reasoning ability (Zhang et al., 2023; Jimenez et al., 2024). Retrieval-Augmented Generation (RAG) (Lewis et al., 2020; Guu et al., 2020) has emerged as a powerful paradigm to enhance LLMs, and demonstrates efficacy in code task (Tan et al., 2026). Building on this paradigm, Retrieval-Augmented Code Generation (RACG) extends RAG to code generation by retrieving relevant code as external knowledge.

In particular, advancing cross-lingual RACG is increasingly necessary in modern software development, where code reuse and migration across programming languages are common yet remain challenging. A key obstacle lies in the asymmetric distribution of code knowledge across programming languages (*PL*s) (Yang et al., 2024; Cao et al., 2025). Widely adopted *PL*s like Python benefit from extensive documentation, active communities, and abundant repositories, whereas less popular *PL*s such as Scala often suffer from sparse resources and limited maintenance (DJurdjev, 2024). This disparity creates inconvenience for developers working with less popular *PL*s. At the same time, the demand for multilingual RACG is further amplified as enterprises modernize their technology stacks (Yang et al., 2024). Migrating to emerging *PL*s can offer performance optimization and security compliance, creating demand for cross-lingual code transformation tools (Mayer et al., 2017; Cassano et al., 2024).

Although existing works (Wang et al., 2025; Filkov et al., 2024; Parvez et al., 2021) explore the RACG, they are confined to limited *PL*s such as Python and Java, leaving RACG across multiple *PL*s and cross-lingual code knowledge transfer unexplored. In this paper, we study the cross-lingual knowledge transfer in RACG and aim to answer 3 research questions (RQs) about it. Our investigation shows knowledge transfer performance of current code LLMs, while uncovering interesting findings for cross-lingual RACG.

RQ1. How effective is RACG with direct oracle cross-*PL* code injection? We investigates the efficacy upper bound of cross-lingual knowledge transfer for code generation tasks under idealized conditions, implemented via RACG with oracle retrieval, which is one of the direct knowledge in-

*Corresponding Authors.

Findings of the Association for Computational Linguistics: ACL 2026, pages 24283–24296

July 2-7, 2026 ©2026 Association for Computational Linguistics

jection. To strictly control variables, we conducted preliminary injection experiments on a *Small but Parallel Multi-lingual Code Dataset*, which contains only 5 *PL*s and 164 questions per *PL*. We find that while cross-lingual knowledge transfer is feasible by RACG, it is still a non-trivial process with fundamental limitations, motivating further exploration of the knowledge transfer mechanisms.

RQ2. How effective is RACG at cross-*PL* knowledge transfer via code retrieval? To investigate the effectiveness of cross-lingual knowledge transfer in RACG, we construct a *Large Multilingual Code Dataset* covering 13 *PL*s. Each instance includes a NL prompt, a verified reference solution, and executable test cases. Using a complete RACG pipeline, we systematically evaluate enhancement from a source *PL* to a different target *PL*. Our experiments reveal that RACG enables unequal knowledge transfer—its efficacy depends to some extent on the linguistic affinity between the PL pair and the diversity of the LLM's pretraining corpus. Notably, while multi-lingual LLMs show the ability of knowledge transfer, python-specialized LLMs struggle to leverage cross-lingual knowledge in context, underscoring the importance of broad pretraining for effective knowledge transfer.

RQ3. How does NL information in code affect cross-*PL* knowledge transfer in RACG? In this RQ, we simulate a setting where the retrieval corpus consists of pure code snippets without NL descriptions—reflecting the common scenario of isolated code fragments online. Our experiments show that removing NL from the corpus has only a marginal effect on cross-lingual knowledge transfer performance, with overall performance declining very slightly. We also systematically evaluate three retrieval paradigms and domain-specific code retrievers outperform, which suggests that while the existence of NL is not essential for enabling knowledge transfer, specialized code retrieval is critical for effectively bridging NL intent and code semantics in RACG applications.

The contributions are summarized as follows:

- We present the first study to explore the code knowledge transfer in RACG across multiple *PL*s, which sheds light on the possibility of going across *PL* silos.
- Our study deepens the understanding of cross-lingual knowledge transfer in RACG via extensive experiments (3 retrieval experimental settings and 5 code LLMs and 13 *PL*s).
- We construct nearly 14k high-quality code generation instances and document annotations spanning 13 *PL*s. We also plan to release the dataset and experiment code to boost the transparency and facilitate further study.

2 Study Design

2.1 Task Formulation

We formulate the problem as follows: Let L denote the set of *PL*s. For $\text{RACG}_{l_{\text{src}} \to l_{\text{tgt}}}$ where $l_{\text{src}}, l_{\text{tgt}} \in L$, which is a RACG task that transfers knowledge from the code documentation corpus in the source *PL* l_{src} to the target *PL* l_{tgt} for code generation, the process is formalized as:

$$G\Big(p(q, l_{\text{tgt}}),\ \underbrace{R(D_{l_{\text{src}}}, q)}_{\text{top-}K}\Big) \to C_{l_{\text{tgt}}} \qquad (1)$$

where the components are defined as:

- q: User's query in NL
- $D_{l_{\text{src}}}$: Code documentation corpus in source programming language l_{src}
- $R(D_{l_{\text{src}}}, q) \to \mathcal{K}$: Retrieve top-$K$ code documents $\mathcal{K}$ related to q from $D_{l_{\text{src}}}$
- $p(q, l_{\text{tgt}})$: Task prompt combining query q and target programming language l_{tgt} specification
- $G(p, \mathcal{K}) \to C_{l_{\text{tgt}}}$: Generate code in l_{tgt} using prompt and retrieved knowledge from $\mathcal{K}$

2.2 Study Settings

To investigate cross-lingual knowledge transfer in RACG, we conduct the study across three retrieval experimental settings as shown in Figure 1.

(1) **Golden Solution Document**. We simulate an oracle retriever to isolate the generation, where code LLMs are guaranteed to receive the most relevant code snippets from the specific source *PL* corpora $D_{l_{\text{src}}}$, which can be regarded as a direct knowledge injection means. This injection setup enables us to explore cross-lingual knowledge transfer during generation, independent of retrieval imperfections. In other words, this is also the upper bound of the cross-language knowledge transfer effect in RACG. Specifically, we examine whether code LLMs can leverage knowledge from retrieved code snippets in the source *PL* l_{src}, even when the target generation *PL* l_{tgt} differs from the l_{src}.

(2) **Top-k Retrieved Code Documents**. We implement an end-to-end complete RACG pipeline

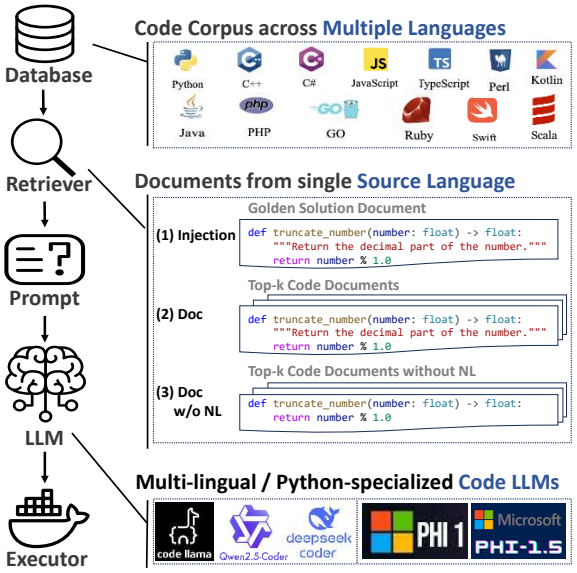

Figure 1: RACG pipeline construction and three retrieval experimental settings for cross-lingual knowledge transfer in RACG

including multi-lingual code retrievers to evaluate transfer performance with practical retrieval. Top-K relevant code snippets selected by retrievers are provided to LLMs in order to generate better solutions for the code generation task. This setting assesses knowledge transfer performance of code LLMs across *PL*s when the retrieval process is not oracle-based (*e.g.*potential mismatches in programming paradigms or API conventions between retrieved and target *PL*s).

(3) **Top-k Retrieved Code Documents (without NL)**. Code snippets found online are often isolated fragments lacking corresponding comments or NL descriptions. The only difference between this setting *Doc w/o NL* and the setting *Doc* is that the $D_{l_{\text{src}}}$ provided is pure code without NL. This real-world scenario presents challenges for both the retriever and the LLM: the retriever must effectively identify relevant pure code snippets based on NL queries q ($R(D_{l_{\text{src}}}, q) \rightarrow \mathcal{K}$), while the LLM needs to leverage these code-only fragments to improve the output quality ($G(p, \mathcal{K}) \rightarrow C_{l_{\text{tgt}}}$). The knowledge transfer in such harsh condition deserves investigation.

Furthermore, we analyze two distinct categories of code LLMs: multi-lingual LLMs (trained on multiple *PL*s) to assess their inherent capacity for cross-lingual knowledge fusion and python-specialized LLMs (trained just on Python) to evaluate their adaptability when augmented with cross-lingual retrieved contexts. Through this dual-lens settings, we aim to uncover the relationship between LLM types (multi-lingual and python-specialized) and cross-lingual transfer efficiency.

2.3 Experimental Details

LLMs for Evaluation. RACG has two phases: retrieval and generation. In the retrieval phase, we have oracle retrieval and practical model retrieval. In the generation phase, we evaluate 5 LLMs, including three representative approximately 7B parameter instruction-tuned multi-lingual LLMs (*CodeLlama-7B-Instruct* (Rozière et al., 2023), *Deepseek-Coder-6.7B-Instruct* (Guo et al., 2024), *Qwen2.5-Coder-7B-Instruct* (Hui et al., 2024)) and two python-specialized LLMs (*Phi-1* (Gunasekar et al., 2023) and *Phi-1.5* (Li et al., 2023)). For multi-lingual/python-specialized LLMs, we evaluate and report their averaged performance. Appendix F presents the experimental results of a larger code LLM.

Parameter Configuration. To ensure reproducibility, we implement greedy decoding with the temperature 0.0 during inference to generate the most probable responses (Chen et al., 2025). The evaluation metric adopts Pass@1 (*i.e.*K=1) under the deterministic setting. Additionally, we report the results of Pass@5 on the data subset in Appendix D. We evaluate the LLMs by using a unified prompt template to instruct LLMs to utilize the code corpus, thereby enhancing code generation performance, as illustrated in Appendix A, following the design in papers (Wang et al., 2025; Athiwaratkun et al., 2023).

Evaluation Metrics. Following the metrics of HumanEval-X and most code generation work, we use Pass@K (Chen et al., 2021) to evaluate the correctness of code generated. We quantify the benefits of different corpus by the increase in Pass@K values, which also reflects the knowledge transfer effect across *PL*s in RACG.

3 RQ1. How effective is RACG with direct oracle cross-*PL* code injection?

3.1 Dataset and Setup

To study generation mechanisms under *controlled direct knowledge injection*, we use HumanEval-X (Zheng et al., 2023) as *Small but Parallel Multi-lingual Code Dataset*, a parallel multi-lingual code dataset containing 164 programming problems with verified reference solutions aligned across 5 *PL*s

Multi-lingual LLMs Pass@k		Target Programming Language C++	Go	Java	JS	Python	Mean
Baseline (without injection)		54.27	42.68	61.79	58.33	59.35	55.28
Source Programming Language of Corpus	C++	\	+4.47	+20.33	+18.90	+15.04	+14.68
	Go	+9.15	\	+14.63	+21.14	+16.26	+15.29
	Java	+8.54	+12.40	\	+23.58	+18.50	+15.75
	JS	+13.62	+4.88	+11.38	\	+13.01	+10.72
	Python	+2.24	+5.49	+9.15	+14.02	\	+7.72
Mean		+8.38	+6.81	+13.87	+19.41	+15.70	+12.84

Python-specialized LLMs Pass@k		Target Programming Language C++	Go	Java	JS	Python	Mean
Baseline (without injection)		15.55	8.54	17.38	19.51	39.63	20.12
Source Programming Language of Corpus	C++	\	+2.74	-2.13	+10.67	-2.74	+2.13
	Go	+5.49	\	-1.22	+3.66	-7.32	+0.15
	Java	+8.23	+7.62	\	+13.41	+2.44	+7.93
	JS	+4.88	+4.27	0	\	-3.66	+1.37
	Python	+3.35	+2.13	+1.22	+8.23	\	+3.73
Mean		+5.49	+4.19	-0.53	+8.99	-2.82	+3.06

Table 1: Baseline (without injection) and Cross-lingual direct injection performance of multi-lingual LLMs and python-specialized LLMs across different source and target programming languages on *Small but Parallel Multilingual Code Dataset*.

(Python, Java, JavaScript, C++ and Go). This parallel structure eliminates cross-language corpus inconsistencies, enabling rigorous variable control.

For knowledge injection on the *Small but Parallel Multilingual Code Dataset*, we directly include one canonical solution from the source *PL* into the prompt to help code LLMs try to generate solutions in the target *PL*. In this oracle retrieval setup, code LLMs directly access golden source *PL* solutions from the dataset to isolate the influence of the retriever. The aligned problems and solutions ensure fair evaluation across *PL*s.

3.2 Cross-*PL* Knowledge Injection in RACG Is Non-Trivial

Table 1 shows the experiment results of baseline (without injection) and cross-*PL* direct injection performance of multi-lingual LLMs and python-specialized LLMs across different source and target *PL*s on *Small but Parallel Multilingual Code Dataset*. Knowledge transfer in RACG across *PL*s is not simple and worth exploration.

Finding 1: Direct injection can enable knowledge transfer across *PL*s.

From cross-lingual knowledge direct injection results in the Table 1, compared to the Baseline (without injection), both the average Pass@K improvement of multi-lingual LLMs (+12.84%) and python-specialized LLMs (+3.06%) show positive values, indicating that the cross-lingual knowledge generally enhances code generation capabilities across multiple *PL*s. Through direct injection, code LLMs can transfer the knowledge across *PL*s in general. This is reasonable because providing reference solutions from other *PL*s has a positive effect on model generation overall.

Finding 2: For Python-specialized LLMs, direct injection of cross-language knowledge is not always effective.

According to Table 1, the enhancement effects exhibit significant variations between the two distinct types of LLMs. Python-specialized LLMs (+3.06%) demonstrate much smaller improvements compared to multi-lingual LLMs (+12.84%), and even show performance degradation in several *PL* pairs (*e.g.*C++→Java). This is because python-specialized LLMs have a weaker ability to utilize knowledge from other *PL*s, which can instead introduce interference.

Finding 3: Oracle retrieval does not mean transfer cross-language knowledge without loss.

In this experimental setting of *Direct Knowledge Injection*, we assume that the retrieval process is oracle and the corpus knowledge is complete, representing the upper bound of effectiveness for cross-lingual knowledge transfer. However, even under these ideal conditions, the final generative performance of multi-lingual LLMs (55.28% + 12.84% in Table 1) does not exceed 70%, indicating a significant gap from lossless knowledge transfer (100% performance). This demonstrates that knowledge transfer across *PL*s through RACG is not straightforward and warrants further exploration.

4 RQ2. How effective is RACG at cross-*PL* knowledge transfer via code retrieval?

4.1 Dataset and Setup

To support broader language coverage and code documents retrieval in Figure 1, we extend existing datasets from previous work (Athiwaratkun et al., 2023; Chai et al., 2025), which initially contained partially aligned problems across 13 *PL*s but lacked reference solutions. To ensure reproducibility and solution quality, we employ the powerful open-source LLM, *i.e. Qwen2-72B-Instruct-GPTQ-Int4* to generate missing reference solutions using multi-lingual RACG, followed by verification through unit testing. We conduct five iterations of solution

Multi-lingual LLMs Pass@k		Target Programming Language C++	C#	Go	Java	JS	Kotlin	Perl	PHP	Python	Ruby	Scala	Swift	TS	Mean
Baseline (without RACG)		60.21	61.62	51.42	60.49	62.19	55.15	51.39	59.15	59.05	58.69	57.97	52.61	59.79	57.67
Source Programming Language of Corpus	C++	\	+15.87	+17.46	+20.87	+21.78	+15.41	+14.60	+10.25	+9.85	+13.93	+9.96	+12.38	+18.59	+15.08
	C#	+18.69	\	+15.28	+18.59	+17.69	+21.01	+16.42	+9.42	+7.90	+15.80	+14.10	+14.80	+17.76	+15.62
	Go	+19.30	+18.06	\	+18.99	+15.13	+16.15	+15.40	+10.48	+8.18	+14.32	+13.16	+9.68	+14.84	+14.47
	Java	+18.79	+19.01	+14.66	\	+20.54	+18.77	+20.54	+12.73	+8.86	+14.66	+12.43	+14.06	+17.90	+16.08
	JS	+11.08	+14.16	+9.10	+8.40	\	+12.64	+11.55	+6.42	+7.62	+14.14	+5.63	+6.12	+28.30	+11.26
	Kotlin	+13.87	+17.84	+9.76	+14.37	+17.19	\	+12.75	+8.67	+9.37	+18.89	+6.64	+4.88	+16.36	+12.55
	Perl	+11.05	+16.51	+12.74	+18.06	+9.18	+15.19	\	+9.79	+13.40	+20.91	+9.84	+11.64	+12.49	+13.40
	PHP	+4.21	+4.82	+2.32	+5.59	+3.18	+1.56	+3.69	\	+2.99	+6.92	-1.20	+0.08	+5.96	+3.34
	Python	+7.58	+14.28	+9.50	+13.84	+3.06	+10.08	+10.72	+6.42	\	+6.98	+7.24	+6.48	+3.35	+8.29
	Ruby	-0.16	+14.38	+6.81	+9.10	+1.21	+13.60	+7.27	+4.87	+2.03	\	+7.37	+6.73	+5.30	+6.54
	Scala	+14.42	+15.78	+9.35	+9.98	+9.71	-1.71	+11.36	+6.88	+7.45	+16.74	\	+4.95	+10.46	+9.61
	Swift	+16.09	+16.86	+10.35	+15.92	+15.95	+12.70	+13.80	+9.33	+10.66	+15.80	+11.35	\	+15.32	+13.68
	TS	+14.87	+12.48	+10.02	+12.53	+20.04	+11.67	+9.30	+13.99	+8.38	+12.99	+5.98	+8.51	\	+11.73
Mean		+12.48	+15.00	+10.61	+13.85	+12.89	+12.26	+12.28	+9.10	+8.06	+14.34	+8.54	+8.36	+13.89	+11.67

Python-specialized LLMs Pass@k		Target Programming Language C++	C#	Go	Java	JS	Kotlin	Perl	PHP	Python	Ruby	Scala	Swift	TS	Mean
Baseline (without RACG)		23.65	25.29	7.73	21.07	26.48	14.85	10.54	24.31	37.85	23.25	15.75	14.78	25.56	20.85
Source Programming Language of Corpus	C++	\	+13.04	+4.54	+8.52	+11.60	+3.64	+3.60	+16.06	-1.14	+6.26	+1.09	+2.35	+11.30	+6.74
	C#	+6.02	\	+3.32	+1.05	+1.59	+3.87	+2.03	+7.30	-4.36	+4.63	-1.47	+1.50	-1.16	+2.03
	Go	-0.48	+4.00	\	-0.75	+0.84	-4.35	+1.89	+6.61	-0.72	-5.57	-5.55	-6.40	-4.21	-1.22
	Java	+5.40	+4.00	+2.66	\	-1.94	+0.98	+2.72	+13.17	-9.53	+1.05	+0.57	+0.32	+1.72	+1.76
	JS	+10.26	+8.38	+2.88	+8.74	\	+3.12	+3.65	+11.61	-4.57	+5.44	+3.18	+1.39	+45.40	+8.29
	Kotlin	+0.58	+6.95	+3.21	+6.76	+8.29	\	+1.11	+10.06	-5.46	-0.18	+5.93	+3.63	+5.24	+3.84
	Perl	-8.33	+2.14	-0.88	+2.94	-8.39	-6.35	\	-2.63	-2.12	+0.10	-6.45	-7.15	-8.25	-3.78
	PHP	+5.73	+4.47	+0.45	+2.50	+1.94	+3.92	+3.42	\	-7.37	-0.27	-0.29	+0.37	+2.02	+1.41
	Python	-2.84	-0.34	-0.49	-1.49	-9.00	-6.91	+3.92	-2.89	\	-3.35	-4.46	-8.27	-9.88	-3.83
	Ruby	-7.66	+10.61	-1.16	-2.54	-5.34	-6.59	+3.23	-5.53	-2.37	\	-3.23	-5.28	-5.67	-2.63
	Scala	-2.70	+2.71	-1.60	-0.75	-4.06	-13.59	-0.24	+2.72	-8.21	+0.19	\	-4.91	-8.16	-3.22
	Swift	-0.24	-9.53	+1.61	+3.47	+0.26	-6.26	+2.49	+8.72	-5.89	+0.14	-6.88	\	+2.70	-0.78
	TS	+9.30	+4.09	+8.35	+5.22	+28.90	+0.98	-0.19	+3.84	-3.05	-0.09	+1.85	-0.85	\	+4.86
Mean		+1.25	+4.21	+1.91	+2.81	+2.06	-2.30	+2.30	+5.75	-4.57	+0.70	-1.31	-1.94	+2.59	+1.04

Table 2: RACG performance compared to baseline (without RACG) when the retrieval corpus and generation task involve different programming languages on *Large Multilingual Code Dataset* in *Doc* setting.

generation with verification in total. Finally, we construct a *Large Multilingual Code Dataset* and unify both the data format and evaluation methodology according to the HumanEval-X (Zheng et al., 2023) benchmark. Our datasets provide not only test cases but also verified reference solutions, establishing a foundation for constructing a multi-*PL* code generation benchmark and retrieval repository. Following the canonical document setup described in (Wang et al., 2025), we create code documents for the retrieval corpus (i.e., reference solutions with NL comments).

Through rigorous unit testing and manual review, we ensure the correctness of our data. This data synthesis method does not introduce potential dataset bias; details can be found in the Appendix C. This pipeline produces 13910 high-quality datapoints, approximately 1,000 datapoints per language across 13 *PL*s (Python: 1181; Kotlin: 1071; Java: 1139; Ruby: 1103; JavaScript (JS): 1133; PHP: 1158; TypeScript (TS): 1059; C++: 1038; C#: 1050; Go: 905; Perl: 1082; Scala: 1054; Swift: 937), providing balanced coverage while maintaining high degree of alignment.

For the complete RACG pipeline on *Large Multilingual Code Dataset*, to control for variables, we fix the retrieval window size to 3 due to the context length limitation of Python-specialized LLMs, that is, using the Top-3 relevant code documents for each query. We also report the experiment results of different retrieval window sizes in Appendix E. In the retrieval phase, we employ a state-of-the-art embedding model *CodeRankEmbed* (Suresh et al., 2024) for the code retrieval task, which supports multi-lingual code retrieval. In the LLM generation stage, we use a unified prompt template to evaluate as shown in Appendix A.

4.2 Cross-*PL* Knowledge Transfer in RACG Is Unequal.

From Table 2, we report the RACG performance improvements on *Large Multilingual Code Dataset*

in *Doc* setting. RACG enables unequal cross-lingual knowledge transfer, and its efficacy depends on linguistic affinity of *PL* pair and diversity of LLM pretraining corpus.

Finding 4: Balanced code generation abilities do not mean equal knowledge transfer abilities.

While multi-lingual LLMs show relatively balanced code generation performance across 13 *PL*s (aligning with findings from the work (Cassano et al., 2023)), their capacity to leverage retrieved cross-lingual knowledge shows significant disparities. Overall, the results in Table 2 show that the enhancement effects vary considerably across different *PL* pairs, with a maximum improvement of +28.30%, while the use of PHP and Scala corpora even degrades the generation performance for the specific target *PL*.

The mainstream *PL* used for training is not always a suitable knowledge transfer vehicle. While both Python and Java dominate multi-lingual LLMs pretraining, Java demonstrates superior efficacy as a retrieval corpus (Java→others average improvement: +16.08% in comparison with Python→others: +8.29%). This reveals an unexpected difference between training and retrieval.

Finding 5: Knowledge transfer is always very effective when two *PL*s are very similar.

We also observe linguistic affinity patterns: JavaScript and TypeScript exhibit outstanding bidirectional enhancement effects. When using TypeScript as the corpus, JavaScript shows the most significant improvement (+20.04% and +28.90% in Table 2). Meanwhile, TypeScript demonstrates gains the strongest enhancement (+28.30% and +45.40% in Table 2) when corpus is JavaScript. This is because TypeScript is a superset of JavaScript, and the two are very similar.

Finding 6: RACG is not good at enabling python-specialized LLMs to transfer knowledge; Diverse training corpus is key to transfer knowledge across *PL*s for LLMs in RACG.

Horizontally, python-specialized LLMs performs poorly when Python is source *PL* because they lack the capability to generate solutions in other *PL*s. Vertically, python-specialized LLMs improve in several non-specialized *PL*s through cross-lingual RACG but suffer 4.57% degradation (Table 2) in their native *PL* tasks (*i.e.*, Python task performance for a Python-specialized LLM), indicating that knowledge in other *PL*s acts as interference because they cannot comprehend and utilize it.. Moreover, the overall improvement is minimal, only +1.04% in Table 2, indicating that knowledge transfer across *PL*s is rather ineffective.

This contrasts with multi-lingual LLMs, which demonstrate all positive gains across 13 *PL*s. Furthermore, under the same *PL*s settings, multi-lingual LLMs universally achieve greater improvements compared to python-specialized LLMs in cross-lingual RACG, suggesting stronger cross-lingual knowledge transfer capabilities derived from their diverse pretraining.

5 RQ3. How does NL information in code affect cross-*PL* knowledge transfer in RACG?

5.1 Dataset and Setup

Code corpora on the Internet most exist as isolated fragments without corresponding comments or aligned NL descriptions (Song et al., 2019). For simulation of this setting (*i.e.Doc w/o NL* in Figure 1), we provide a *pure code corpus* version of the *Large Multilingual Code Dataset* dataset stripped of NL comments. We also annotate 1 golden document per *PL* for each query. This design ensures that each query corresponds to 13 golden documents across *PL*s, enabling calculation of retrieval precision and recall rates. The large dataset scale and intentional information deprivation create challenging yet realistic conditions for examining cross-lingual knowledge transfer mechanisms.

To explore the different retrieval performance in this *Doc w/o NL* setting, we investigate three distinct retrieval paradigms in RACG: *First*, lexical sparse retrieval (*e.g.*, BM25), which is computationally efficient and relies on keyword matching and term frequency statistics. *Second*, generic text embedding models, where code is treated as normal text using a pre-trained language model (*e.g.*, BERT-style architectures) to encode. *Third*, domain-specific code retrievers, which are trained on NL-to-code alignment tasks and can bridge NL intent and code semantics.

Specifically, we evaluate retrieval strategies on the *Large Multilingual Code Dataset* in *Doc w/o NL* setting, where, for each query, one most relevant code snippet is annotated per *PL*. The evaluated models span the three paradigms:

Multi-lingual LLMs Pass@k		Target Programming Language C++	C#	Go	Java	JS	Kotlin	Perl	PHP	Python	Ruby	Scala	Swift	TS	Mean
Baseline (without RACG)		60.21	61.62	51.42	60.49	62.19	55.15	51.39	59.15	59.05	58.69	57.97	52.61	59.79	57.67
Source Programming Language of Corpus	C++	\	+14.06	+11.97	+14.11	+15.92	+11.67	+9.40	+16.70	+7.71	+11.45	+8.86	+9.36	+12.34	+11.96
	C#	+13.49	\	+12.30	+14.57	+15.59	+16.87	+9.03	+15.83	+7.87	+11.51	+11.42	+10.25	+13.06	+12.65
	Go	+12.52	+13.08	\	+14.49	+13.92	+12.82	+9.83	+14.31	+9.12	+9.70	+9.80	+7.15	+12.86	+11.63
	Java	+14.32	+16.41	+10.42	\	+16.21	+15.38	+14.11	+18.91	+8.10	+11.82	+10.59	+11.70	+12.26	+13.35
	JS	+7.87	+11.21	+5.56	+7.46	\	+10.15	+7.30	+9.38	+5.67	+9.40	+7.43	+2.95	+18.18	+8.55
	Kotlin	+10.89	+14.92	+6.52	+11.30	+13.77	\	+6.13	+12.12	+7.56	+14.02	-8.29	-4.41	+10.82	+7.95
	Perl	+12.30	+11.24	+10.24	+14.63	+17.03	+11.89	\	+12.98	+11.21	+15.53	+9.39	+7.44	+14.89	+12.40
	PHP	+1.57	+3.37	+1.80	+5.15	+4.15	+2.89	+0.18	\	+2.20	+3.60	+2.94	+0.68	+4.12	+2.72
	Python	+10.08	+10.10	+4.53	+10.36	+9.86	+7.35	+4.68	+7.71	\	-5.71	+6.17	+2.77	+7.73	+6.30
	Ruby	+7.06	+9.87	+4.20	+8.05	+9.44	+8.75	+0.86	+6.36	+0.48	\	+7.97	+3.66	+10.19	+6.41
	Scala	+11.85	+11.68	+7.51	+11.15	+13.42	+0.37	+4.87	+11.89	+6.63	+12.21	\	+2.63	+9.36	+8.63
	Swift	+11.62	+12.98	+7.15	+13.46	+14.24	+7.28	+8.26	+12.92	+8.35	+10.73	+9.23	\	+11.43	+10.64
	TS	+11.88	+6.86	+7.26	+9.42	+14.80	+7.56	+4.28	+10.22	+6.44	+8.25	+2.62	+3.63	\	+7.77
Mean		+10.46	+11.31	+7.46	+11.18	+13.20	+9.41	+6.58	+12.44	+6.78	+9.38	+6.51	+4.82	+11.44	+9.30

Python-specialized LLMs Pass@k		Target Programming Language C++	C#	Go	Java	JS	Kotlin	Perl	PHP	Python	Ruby	Scala	Swift	TS	Mean
Baseline (without RACG)		23.65	25.29	7.73	21.07	26.48	14.85	10.54	24.31	37.85	23.25	15.75	14.78	25.56	20.85
Source Programming Language of Corpus	C++	\	+7.19	+3.87	+0.04	+6.40	-0.84	-0.51	+1.38	-0.59	+2.54	+1.66	+0.21	+2.49	+1.99
	C#	+8.72	\	+3.04	+3.91	-1.81	+2.66	-1.20	-0.78	+0.04	+2.67	+1.76	-0.91	+1.25	+1.61
	Go	+1.69	-0.38	\	-1.45	-7.11	-4.39	-2.54	-5.09	-3.39	-6.66	-3.65	-6.99	-4.12	-3.67
	Java	+7.08	+3.71	+4.97	\	+0.09	-1.96	-0.88	+0.69	-4.36	+1.22	+1.52	+0.21	+3.65	+1.33
	JS	+7.37	+4.00	+2.71	-0.79	\	+1.45	-0.18	+0.35	-1.52	+0.14	+0.14	-1.87	+23.24	+2.92
	Kotlin	+1.78	+3.00	+3.09	+0.44	+6.00	\	-0.55	-0.69	-1.48	+1.04	+1.61	-1.23	+3.99	+1.42
	Perl	+3.71	+0.29	+0.33	+1.45	-0.79	+1.63	\	-0.60	+1.91	-1.31	+0.76	-3.95	+1.07	+0.37
	PHP	+3.95	-0.24	+0.88	-0.79	-0.79	+1.59	+0.88	\	-2.07	-1.99	+0.09	-2.40	+0.47	-0.04
	Python	+2.26	-0.14	+3.15	-1.54	-2.52	-0.75	+0.05	-2.85	\	-5.58	-0.95	-2.08	+0.69	-0.85
	Ruby	+3.76	+3.62	+0.77	-0.35	+8.61	+1.49	+1.39	-2.50	-0.80	\	-1.23	-0.96	+6.74	+1.71
	Scala	+1.11	+0.14	+1.55	+1.54	-2.16	-6.07	-1.34	-2.29	-1.10	0	\	-1.17	-1.93	-0.98
	Swift	+0.39	+2.10	+2.21	-2.81	+0.75	-3.27	-0.79	-0.60	-0.97	-1.59	-4.13	\	+2.10	-0.55
	TS	+4.96	-1.43	+4.86	+0.83	+15.53	-1.63	-2.54	-4.23	-2.03	-1.09	-1.47	-4.48	\	+0.61
Mean		+3.90	+1.82	+2.62	+0.04	+1.85	-0.84	-0.69	-1.44	-1.37	-0.88	-0.32	-2.13	+3.30	+0.45

Table 3: RACG performance compared to baseline (without RACG) when the retrieval corpus and generation task involve different programming languages on *Large Multilingual Code Dataset* in *Doc w/o NL* setting.

BM25 method for sparse retrieval; *Sup-SimCSE-BERT-large-uncased* (Gao et al., 2021), *BGE-large-en-v1.5* (Xiao et al., 2023) and the larger *Qwen3-Embedding-8B* (Zhang et al., 2025) models for generic text embedding; and *CodeRankEmbed* (Suresh et al., 2024) model for domain-specific code retrieval. We use Precision@K and Recall@K as metrics to assess the code retrieval effectiveness.

5.2 Natural Language Plays a Minor Role under Strong Code Retrieval.

Table 3 shows RACG performance compared to baseline (without RACG) when the retrieval corpus and generation task involve different *PL*s on *Large Multilingual Code Dataset* in *Doc w/o NL* setting, while effectiveness of different retrieval strategies in RACG for the *Doc w/o NL* setting is reported in Table 4. In RACG, removing NL from the retrieval corpus has little impact on cross-lingual knowledge transfer ablility. In this case, domain-specific retriever exhibits best retrieval performance.

Finding 7: NL in the code corpus (*e.g.* code comments) has little impact on knowledge transfer performance cross *PL*s in RACG.

Our investigation reveals a counter-intuitive finding regarding the role of NL in RACG. While it is often assumed that NL documentation (e.g., comments, docstrings) serves as a critical semantic bridge between differing *PL*s, our empirical data suggests its impact is secondary to the code itself.

Table 3 and Table 2 share the same experimental setup, except that Table 3 uses the *Doc w/o NL* setting. By comparing the two tables, it can be seen that after removing NL, the knowledge contained in the corpus decreases. The overall knowledge transfer effect of RACG declines, but is very slight (+11.67%→+9.30% for multi-lingual LLMs overall performance and +1.04%→+0.45% for Python-specialized LLMs).

For multi-lingual LLMs, performance gap is merely 2.37%. While the presence of comments

Strategies	Retrieval Model	Precison@5	Precison@10	Recall@10	Recall@20
Lexical Sparse Retrieval	BM25	6.57%	4.87%	3.74%	4.79%
Generic Text Embedding	Sup-SimCSE-BERT-large	18.65%	13.75%	10.58%	14.44%
	BGE-large-en-v1.5	56.18%	44.64%	34.34%	46.05%
	Qwen3-Embedding-8B	77.29%	66.99%	51.53%	68.38%
Domain-specific NL-to-Code Alignment	CodeRankEmbed	91.60%	87.72%	67.48%	88.04%

Table 4: Effectiveness of different retrieval strategies in RACG for the *Doc w/o NL* setting.

does provide contextual cues that aid retrieval and generation, the vast majority of the improvement (over 9%) is driven solely by the semantic information encoded within the programming logic itself. The "heat" of the improvement (represented by the blue cells in the Tables 3) remains intense even when NL is stripped away.

Finding 8: Different retrieval strategies yield varied effectiveness on pure code corpus. Domain-specific retrievers perform the best.

As shown in Table 4, significant performance variations emerge across strategies. The sparse retriever *BM25*, based on bag-of-words matching, performs worst with only 6.57%Precision@5 and 4.79%Recall@20, confirming the inadequacy of pure lexical matching for code. The generic embedding models show a clear performance hierarchy correlating with their general text embedding strength. *Sup-SimCSE-BERT-large-uncased*, a contrastively fine-tuned BERT model, achieves moderate gains (18.65%P@5, 14.44%R@20). The more advanced general-purpose embedders perform better: *BGE-large-en-v1.5* reaches 56.18%P@5 and 46.05%R@20, while the larger model *Qwen3-Embedding-8B* further improves to 77.29%P@5 and 68.38%R@20, indicating that powerful general text models can capture some code semantics. In contrast, the domain-specific *CodeRankEmbed* delivers the best results, attaining approximately 90%in both precision and recall metrics (91.60%P@5, 88.04%R@20), significantly outperforming all general-purpose models.

6 Related Work

6.1 Retrieval-Augmented Code Generation

Recent years have witnessed significant advances in RACG, where external contexts and documentation are leveraged to enhance code tasks. Classic works such as REDCODER (Parvez et al., 2021), ReACC (Lu et al., 2022), and DocPrompt (Zhou et al., 2023) demonstrate its efficacy for code generation, summarization and completion. Subsequent research expand RACG methodologies and benchmarks, yielding notable contributions (Li et al., 2025a,b; Dutta et al., 2024; Tan et al., 2026; Su et al., 2024). However, existing works such as (Wang et al., 2025; Du et al., 2024; Filkov et al., 2024) remain limited, only focusing on 1-2 mainstream *PL*s. For instance, CodeRAG-Bench (Wang et al., 2025) establishes a comprehensive evaluation framework for Python code RACG; CodeGRAG (Du et al., 2024) explore Python and C++ RACG which use graphical view of code blocks; RRG (Filkov et al., 2024) introduces a code refactorer module in Python and Java RACG. Our work goes beyond existing works by investigating cross-lingual RACG across 13 *PL*s with various settings to explore knowledge transfer.

6.2 Multi-*PL* Code Generation Benchmark

The emergence of multi-lingual evaluation benchmarks has driven progress in cross-lingual code generation research. HumanEval-X (Zheng et al., 2023), MultiPL-E (Cassano et al., 2023) and CruxEval-X (Xu et al., 2025) enable parallel evaluation across multiple *PL*s by translating existing problems, and McEval (Chai et al., 2025) further enhances the diversity of evaluation data. However, existing studies predominantly focus on benchmarking performance, leaving cross-lingual knowledge transfer mechanisms underexplored. The work (Athiwaratkun et al., 2023) explains the mutual augmentation capability between different *PL* corpora through the zero-shot code translation ability of LLMs, which is entirely distinct from the perspective of our study. In comparison, we focuse on cross-lingual knowledge transfer challenges in complete RACG.

7 Conclusion

In this paper, we investigated knowledge transfer across *PL*s through the large-scale empirical study.

By constructing a high-quality dataset spanning 13 *PL*s of nearly 14k code generation instances, we explore three RQs about knowledge transfer in RACG across *PL*s. This study reveals the complexity and imbalance of cross-lingual knowledge transfer in RACG, providing a roadmap for effective multilingual code generation assistant.

Limitations

There are two primary limitations in our study. First, the LLMs evaluated in the RACG are all of relatively small scale (with at most 14B parameters), which may not be fully representative of all code-related models. Second, the scale of the current dataset is constrained, comprising approximately 1,000 samples per programming language. This limitation may affect the comprehensiveness and statistical robustness of the evaluation across diverse linguistic features. Future work can improve the study by using larger code LLMs and benchmark expended through large-scale synthetic data generation techniques.

Acknowledgements

We sincerely thank the reviewers for their insightful comments and valuable suggestions. This work was supported by the National Key R&D Program of China (2024YFC3308000), the Natural Science Foundation of China (No. 62506354, 62306303, 62476265).

References

Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, Sujan Kumar Gonugondla, Hantian Ding, Varun Kumar, Nathan Fulton, Arash Farahani, Siddhartha Jain, Robert Giaquinto, Haifeng Qian, Murali Krishna Ramanathan, and Ramesh Nallapati. 2023. Multilingual evaluation of code generation models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jialun Cao, Yuk-Kit Chan, Zixuan Ling, Wenxuan Wang, Shuqing Li, Mingwei Liu, Ruixi Qiao, Yuting Han, Chaozheng Wang, Boxi Yu, Pinjia He, Shuai Wang, Zibin Zheng, Michael R. Lyu, and Shing-Chi Cheung. 2025. How should I build A benchmark? revisiting code-related benchmarks for llms. *CoRR*, abs/2501.10711.

Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Anders Freeman, Carolyn Jane Anderson, Molly Q. Feldman, Michael Greenberg, Abhinav Jangda, and Arjun Guha. 2024. Knowledge transfer from high-resource to low-resource programming languages for code llms. *Proc. ACM Program. Lang.*, 8(OOPSLA2):677–708.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. 2023. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Trans. Software Eng.*, 49(7):3675–3691.

Linzheng Chai, Shukai Liu, Jian Yang, Yuwei Yin, Ke Jin, Jiaheng Liu, Tao Sun, Ge Zhang, Changyu Ren, Hongcheng Guo, Noah Wang, Boyang Wang, Xianjie Wu, Bing Wang, Tongliang Li, Liqun Yang, Sufeng Duan, Zhaoxiang Zhang, and Zhoujun Li. 2025. Mceval: Massively multilingual code evaluation.

Jingyi Chen, Songqiang Chen, Jialun Cao, Jiasi Shen, and Shing-Chi Cheung. 2025. When llms meet API documentation: Can retrieval augmentation aid code generation just as it helps developers? *CoRR*, abs/2503.15231.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Darko DJurdjev. 2024. Popularity of programming languages. *AIDASCO Reviews*, 2(2):24–29.

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. *CoRR*, abs/2405.02355.

Avik Dutta, Mukul Singh, Gust Verbruggen, Sumit Gulwani, and Vu Le. 2024. RAR: retrieval-augmented retrieval for code generation in low resource languages. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages

21506–21515. Association for Computational Linguistics.

Vladimir Filkov, Baishakhi Ray, and Minghui Zhou, editors. 2024. *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*. ACM.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 6894–6910. Association for Computational Linguistics.

Alex Gu, Baptiste Rozière, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida Wang. 2024. Cruxeval: A benchmark for code reasoning, understanding and execution. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 16568–16621. PMLR / OpenReview.net.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks are all you need. *CoRR*, abs/2306.11644.

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. *CoRR*, abs/2401.14196.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Proceedings of Machine Learning Research, pages 3929–3938. PMLR.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-coder technical report. *CoRR*, abs/2409.12186.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Jia Li, Chongyang Tao, Jia Li Male, Ge Li, Zhi Jin, Huangzhao Zhang, Zheng Fang, and Fang Liu. 2025a. Large language model-aware in-context learning for code generation. *ACM Trans. Softw. Eng. Methodol.*, 34(7):190:1–190:33.

Xinze Li, Hanbin Wang, Zhenghao Liu, Shi Yu, Shuo Wang, Yukun Yan, Yukai Fu, Yu Gu, and Ge Yu. 2025b. Building a coding assistant via the retrieval-augmented language model. *ACM Trans. Inf. Syst.*, 43(2):39:1–39:25.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023. Textbooks are all you need II: phi-1.5 technical report. *CoRR*, abs/2309.05463.

Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seungwon Hwang, and Alexey Svyatkovskiy. 2022. Reacc: A retrieval-augmented code completion framework. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6227–6240. Association for Computational Linguistics.

Philip Mayer, Michael Kirsch, and Minh Anh Le. 2017. On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers. *J. Softw. Eng. Res. Dev.*, 5:1.

Md. Rizwan Parvez, Wasi Uddin 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, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, Findings of ACL, pages 2719–2734. Association for Computational Linguistics.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Xiaotao Song, Hailong Sun, Xu Wang, and Jiafei Yan. 2019. A survey of automatic generation of source code comments: Algorithms and techniques. *IEEE Access*, 7:111411–111428.

Hongjin Su, Shuyang Jiang, Yuhang Lai, Haoyuan Wu, Boao Shi, Che Liu, Qian Liu, and Tao Yu. 2024. Evor: Evolving retrieval for code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, Findings of ACL, pages 2538–2554. Association for Computational Linguistics.

Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. 2024. Cornstack: High-quality contrastive data for better code ranking. *CoRR*, abs/2412.01007.

Hanzhuo Tan, Qi Luo, Ling Jiang, Zizheng Zhan, Jing Li, Haotian Zhang, and Yuqun Zhang. 2026. Prompt-based code completion via multi-retrieval augmented generation. *ACM Trans. Softw. Eng. Methodol.*, 35(1):9:1–9:28.

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2025. Coderag-bench: Can retrieval augment code generation? In *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, Findings of ACL, pages 3199–3214. Association for Computational Linguistics.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *CoRR*, abs/2309.07597.

Ruiyang Xu, Jialun Cao, Yaojie Lu, Ming Wen, Hongyu Lin, Xianpei Han, Ben He, Shing-Chi Cheung, and Le Sun. 2025. CRUXEVAL-X: A benchmark for multilingual code reasoning, understanding and execution. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 23762–23779. Association for Computational Linguistics.

Haoran Yang, Yu Nong, Shaowei Wang, and Haipeng Cai. 2024. Multi-language software development: Issues, challenges, and solutions. *IEEE Trans. Software Eng.*, 50(3):512–533.

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, EMNLP 2023, Singapore, December 6-10, 2023*, pages 2471–2484. Association for Computational Linguistics.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *CoRR*, abs/2506.05176.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, pages 5673–5684. ACM.

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, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, and et al. 2025. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

A Prompt Template for Benchmark Evaluation

We evaluate the LLMs by using a unified prompt template (as illustrated in Figure 2) that instructs them to utilize the code corpus, thereby enhancing code generation.

```
# Multi-lingual RACG Instruction Template
You will be given a question (problem
specification) and will generate a correct
program that matches the specification and passes
all tests. You will NOT return anything except
for the program. Please do not output any
additional comments, explanations, or text.

## Corpus if exists
Here are some retrieved code snippets or
documents that may be helpful.

${Relevant Code Documents in Source Programming
Language}

## Generation Task
Please complete the function using ${Target
Programming Language} in the format of the
example following.

${Few-shot in Target Programming Language for
Format Following}

${User's question}
```

Figure 2: Unified prompt template for LLMs to generate code in multi-lingual RACG.

Pass@k		Multi-lingual LLMs			Mean
		CodeLlama-7B	Deepseek-Coder-6.7B	Qwen2.5-Coder-7B	
Source Programming Language of Corpus	C++	+15.62	+11.56	+8.70	+11.96
	C#	+16.38	+12.59	+8.98	+12.65
	Go	+13.87	+11.78	+9.24	+11.63
	Java	+17.18	+12.96	+9.92	+13.35
	JS	+10.66	+8.25	+6.73	+8.55
	Kotlin	+11.34	+8.79	+3.71	+7.95
	Perl	+14.57	+13.05	+9.57	+12.40
	PHP	+4.45	+3.36	+0.36	+2.72
	Python	+5.06	+8.18	+5.67	+6.30
	Ruby	+6.05	+8.05	+5.13	+6.41
	Scala	+9.38	+10.67	+5.84	+8.63
	Swift	+12.48	+12.34	+7.09	+10.64
	TS	+10.38	+7.34	+5.58	+7.77
Mean		+11.34	+9.92	+6.66	+9.30

Table 5: Multi-lingual LLMs RACG performance compared to baseline (without RAG) when the retrieval corpus and generation task involve different programming languages on *Large Multilingual Code Dataset* in *Doc w/o NL* setting.

B Statistic of *Large Multilingual Code Dataset*

To conduct large-scale empirical experiments and explore the knowledge transfer effects between more *PL* pairs, we construct the *Large Multilingual Code Dataset* based on existing datasets, which will be used as both an evaluation benchmark and a retrieval library. The dataset has nearly 14k high-quality code generation instances across 13 *PL*s. As shown in Table 6, each *PL* has approximately 1,000 datapoints.

Programming Language	Correct Instance
Python	1181
Kotlin	1071
Java	1139
Ruby	1103
JavaScript (JS)	1133
PHP	1158
TypeScript (TS)	1059
C++	1038
C#	1050
Go	905
Perl	1082
Scala	1054
Swift	937
Total	13910

Table 6: Instance distribution across 13 programming languages in *Large Multilingual Code Dataset* for RACG (Total 13,910 instances)

C Dataset Bias for Data-Generating LLM

One might be concerned that the reference solutions generated by *Qwen-72B* could favor models from the *Qwen* series. For example, Table 3 reports the average results for multilingual LLMs in the *Doc w/o NL* setting, where the retrieval corpus and generation task involve different programming languages on the *Large Multilingual Code Dataset*. To address this concern more thoroughly, we present the per-model Pass@K improvements for three models in Table 5. The results show that *Qwen2.5-Coder-7B* obtains a smaller gain (6.66%↑) compared to *CodeLlama-7B* (11.34%↑) and *DeepSeek-Coder-6.7B* (9.92%↑). Since the Qwen-Coder model does not receive a larger advantage, this indicates that the concern about bias is unwarranted. This conclusion is also supported by prior work (Gu et al., 2024; Xu et al., 2025), which suggests that such concerns are unnecessary.

D Evaluation Metrics

We employ greedy decoding (Temperature=0.0) to ensure the reproducibility of our experimental results. In deterministic (greedy) settings, Pass@1 provides a clear and reliable measurement of a model's ability to follow retrieved instructions and generate correct code in a single attempt, which

is crucial for practical RACG applications. While security and efficiency are important, this study primarily focuses on the feasibility and effectiveness of cross-lingual knowledge transfer. Correctness (Pass@K) serves as the primary indicator of successful knowledge transfer across programming language silos. We appreciate the reviewer's suggestion, and we will continue exploring code security and style in future work.

Refering the settings from previous work (Xu et al., 2025; Zhuo et al., 2025), we evaluated Pass@1 (Temperature = 0.0, SampleN = 1) and Pass@5 (Temperature = 0.8, SampleN = 5). We conduct experiments using a subset of the dataset on Multi-lingual LLMs (~7B). The results, briefly summarized in the Table 7, show that increasing the number of samples improves the LLM's performance. In this case, RACG remains a highly effective method for performance enhancement and continues to exhibit consistent patterns in the Pass@K (K=1,5) results. These findings further support the conclusions of our study.

Multi-lingual LLMs (~7B) Window Size K=3		**Pass@1**	**Pass@5**
Baseline (without RAG)		55.75	70.42
Source Programming Language of Corpus	Python	+13.53	+12.38
	Java	+16.86	+15.90
	JS	+13.52	+14.48
	TS	+13.05	+11.90
	Ruby	+12.76	+12.48
	C++	+19.71	+18.57
	Go	+17.71	+13.33
	Swift	+16.48	+12.47
Mean		+15.45	+13.94

Table 7: RACG performance compared to baseline (without RACG) when the retrieval corpus and generation task involve different programming languages on the subset of *Large Multilingual Code Dataset* in *Doc* setting.

E Parameter Configuration

The retrieval window size is fixed at K=3 to strictly control variables across all evaluated models. This specific value is chosen as a pragmatic trade-off to accommodate the context length limitations of certain models, while still providing sufficient external knowledge for effective RACG.

We also conduct experiments on a subset of the dataset to explore the impact of the context retrieval window size K (K=1, 3, 5). The results, briefly summarized in the Table 8, show that when the context length allows, introducing more external knowledge can help the model generate better code. These findings further support the conclusions of our study.

Multi-lingual LLMs (~7B) Pass@1		**K=1**	**K=3**	**K=5**
Baseline (without RAG)		55.75		
Source Programming Language of Corpus	Python	+13.53	+13.71	+15.52
	Java	+16.86	+17.14	+16.19
	JS	+13.52	+15.33	+16.19
	TS	+13.05	+13.91	+14.57
	Ruby	+12.76	+14.95	+15.24
	C++	+19.71	+20.76	+21.33
	Go	+17.71	+18.67	+18.95
	Swift	+16.48	+16.76	+16.38
Mean		+15.45	+16.41	+16.80

Table 8: RACG performance compared to baseline (without RACG) when the retrieval corpus and generation task involve different programming languages on the subset of *Large Multilingual Code Dataset* in *Doc* setting.

F Model Size

Qwen2.5-Coder-14B-Instruct Pass@1		**K=1**	**K=3**	**K=5**
Baseline (without RAG)		72.25		
Source Programming Language of Corpus	Python	+8.86	+10.57	+10.29
	Java	+8.00	+9.43	+7.43
	JS	+8.00	+12.00	+10.29
	TS	+8.86	+9.14	+9.14
	Ruby	+6.57	+9.71	+9.43
	C++	+11.14	+12.57	+12.86
	Go	+10.00	+12.00	+12.29
	Swift	+11.43	+10.29	+10.29
Mean		+9.11	+10.71	+10.25

Table 9: RACG performance compared to baseline (without RACG) when the retrieval corpus and generation task involve different programming languages on the subset of *Large Multilingual Code Dataset* in *Doc* setting.

With similar model sizes controlled for variable control, we evaluate a representative set of code LLMs around 7B size. Given the reviewers' interest in seeing experimental results with larger code LLMs, we add experiments with a larger code LLM (Qwen2.5-Coder-14B-Instruct) using a subset of the datasets. We also explore the impact of different retrieval window sizes (K=1, 3, 5). The results, briefly summarized in the Table 9, show that larger

models exhibit stronger code generation capabilities and still achieve significant performance gains through RACG. These findings continue to support the conclusions of our study.