KareCoder: A New Knowledge-Enriched Code Generation System

Tao Huang
Shandong Normal University, China
2022317095@stu.sdnu.edu.cn

Zhihong Sun
Shandong Normal University, China
2022021002@stu.sdnu.edu.cn

Zhi Jin*
Peking University, China
zhijin@pku.edu.cn

Ge Li
Peking University, China
lige@pku.edu.cn

Chen Lyu*†
Shandong Normal University, China
lvchen@sdnu.edu.cn

ABSTRACT

Large Language Models (LLMs) demonstrate proficiency in handling fundamental programming problems but struggle with complex programming in new types. The study presents KareCoder, integrating programming knowledge into code generation. Initial tests reveal KareCoder's significant success in the Pass@1 metric for complex competitive programming problems.

1 INTRODUCTION

Code generation has attracted academic and industrial attention. The effectiveness is linked to the size of models, dependency on vast <Text,Code> pairs and significant computational resources. However, the transparency of the specific training data is often lacking. Furthermore, current datasets like APPS [1] and MBPP [2], are confronted with benchmark leakage [3] issues in LLMs.

To tackle the benchmark leakage, we createded CodeF dataset, split into two part based on ChatGPT 3.5's [4] training data cutoff, September 2021. Using ChatGPT to generate code on the Simple, Medium and Hard subsets of the two parts, the Pass@1 on CodeF pre2021-9 reached 50.3%, 11.1% and 4.7%, respectively, while they achieved 26.1%, 7.0% and 6.0% on CodeF post2021-9, the former part exhibits an average relative improvement of 69.1% in the Pass@1 metrics over the latter. This highlights LLMs' ability to learn from exposed problems but still lack in learning problems that are novel. Enhancing LLMs performance remains a challenge, we anticipate that introducing contextual knowledge will prompt LLMs.

We introduce KareCoder, a tool to assist generating complex code for programming problems involving the use of diverse algorithms and data structures, similar to competition-level problems. Initially, a dataset encompassing algorithms and data structures tags and a Knowledge Library are constructed. KareCoder leverages these to enhance LLMs' planning and integrate knowledge. Subsequently, KareCoder amalgamates knowledge to generate step-by-step prompts, guiding the final code generation. Initial evaluations demonstrate KareCoder's effectiveness.

*Zhi Jin and Chen Lyu are the corresponding authors.
†This work was done when Chen Lyu was a visiting scholar at Peking University.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04
https://doi.org/10.1145/3639478.3643076

2 DATASET AND KNOWLEDGE LIBRARY

2.1 CodeF Dataset

To avoid benchmark leakage[3] and incorporate programming knowledge, we created CodeF. This dataset is informed by the methodologies and insights gained from creating the TACO dataset [5]. CodeF is annotated with algorithms and data structures tags and problems release time tags, comprises 1,523 problems across 33 tags, 805 before and 718 after the September 2021 cutoff.

In data collection phase, considering anti-crawling measures and the need of tags, we selected Codeforces. Additionally, We cleaned the data by removing comments and duplicates and confirmed accuracy with unit tests. During data processing stage, we divided CodeF into two subsets: CodeF pre2021-9 and CodeF post2021-9 based on the September 2021 cutoff time for ChatGPT 3.5's training data. Furthermore, we conducted an analysis of both subsets to verify and ensure their consistency, focusing on the distribution of algorithms and data structures tags, average number of problems's tokens and tags. We categorized problems according to the difficulty, delineating three subsets of difficulty: Simple, Medium and Hard.

2.2 Knowledge Library

To integrate knowledge into the model, we structured the Knowledge Library around the 33 tags of CodeF as a useful supplement to LLMs. We employed dictionary method for one-to-one matching of tags and knowledge, e.g."tag": "knowledge". We developed three Knowledge Library versions: Knowledge Description, Knowledge Pseudo-Code, and Knowledge Step of Pseudo-Code. In tests, KareCoder with Knowledge Description performed best.

3 METHODOLOGY

As illustrated in Figure 1, we developed a ChatGPT-based tag generator preceding KareCoder to provide tags for problems lacking them, essential for integrating knowledge into KareCoder. We evaluated the tag generator's accuracy with Cohen's Kappa and manual evaluation methods. KareCoder operates in two phases: prompt engineering stage and coding stage, each with tailored system prompts and one-shot examples for improved efficiency and effectiveness.

3.1 Prompt Engineering Stage

In this phase, we designate ChatGPT as a prompt engineer that matches problems with relevant content in the Knowledge Library based on their tags. ChatGPT generates Knowledge-aware prompt to guide the solution of the problem. Let Q represent a problem, P as a Knowledge-aware prompt, K stand for the Knowledge Library, K' denote a knowledge description, and X signify as example sets for prompt generation, i.e., $X = \{\langle Q_x, K'_x, P_x \rangle\}_{x=1}^{n}$. Consequently,

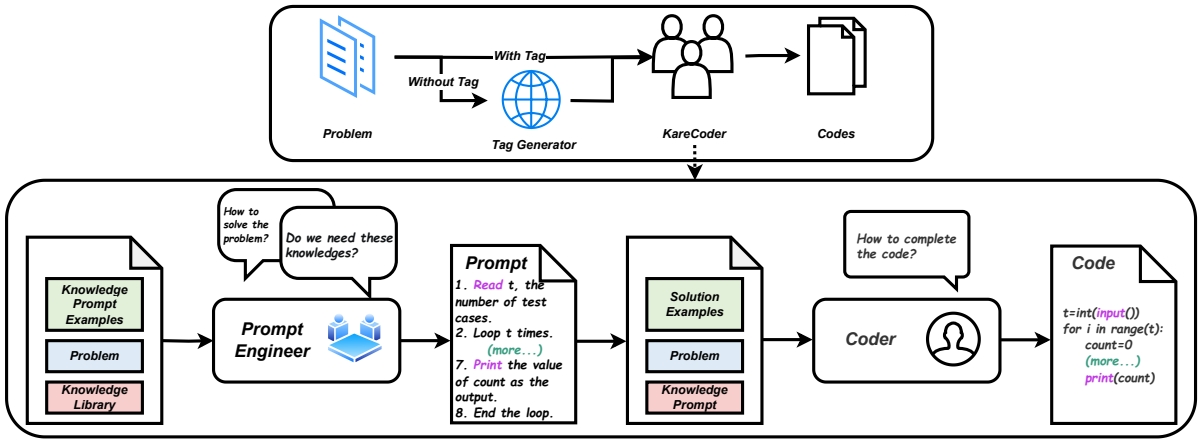

Figure 1: Overview of KareCoder.

the generation of knowledge-aware prompt is defined as:

$$f\left(P \mid Q, K, X\right) \triangleq f\left(P \mid Q, K', X\right) \cdot m\left(K' \mid Q, K\right) \tag{1}$$

Here, f represents the generative task executed by ChatGPT, and m symbolizes the task of knowledge matching.

3.2 Prompt Engineering Stage

In this stage, ChatGPT acts as the coder, reading and understanding the problem and Knowledge-aware prompts from the Prompt Engineer. Subsequently, ChatGPT proceeds to write the code. This phase aims to transform natural language Step-by-Step prompts into executable programs. Let Q symbolize the problem, P represent the Knowledge-aware prompt, C denote the generated code, and Y as a set of code generation examples, i.e., $Y = \left\{\left\langle Q_y, P_y, C_y\right\rangle\right\}_{y=1}^{n}$. The prompt-based code generation is defined as:

$$f\left(C \mid Q, P, Y\right) \tag{2}$$

The Knowledge-aware Code Generation can be encapsulated as:

$$f(C \mid Q, K) \triangleq \underbrace{f(C \mid Q, P, Y)}_{\text{Generate-Code}} \cdot \underbrace{f(P \mid Q, K', X)}_{\text{Generate-Prompt}} \cdot \underbrace{m(K' \mid Q, K)}_{\text{Match-Knowledge}} \tag{3}$$

4 EVALUATION

We assessed KareCoder on CodeF post2021-9 by using "gpt-3.5-turbo-0613" ChatGPT. We conducted comparisons with ChatGPT [4], Self-plan [6] and SCoT [7] to validate KareCoder's superiority by Pass@k metric. After extensive tests, we choosed 1-shot, temperature=1, and top_p=1 configuration, generating five solutions for each problem to compare the Pass@1, Pass@3 and Pass@5 metrics. Notably, Pass@1 is especially important as it aligns more with practical application needs.

Table 1 presents results from experiments on CodeF post2021-9 dataset and its Simple, Medium, and Hard subsets. These experiments validate the efficacy of incorporating programming knowledge. KareCoder outperformed ChatGPT, Self-plan, and SCoT in Pass@1 due to integrating algorithms and data structures knowledge, enhancing the abilities of LLMs. However, in Pass@3 and Pass@5, Self-plan, SCoT, and KareCoder didn't surpass ChatGPT, as these methods' prompt-based approach limits solutions' diversity.

Table 1: Performance Comparision

Method	*Simple*			*Medium*		
	Pass@1	*Pass@3*	*Pass@5*	*Pass@1*	*Pass@3*	*Pass@5*
ChatGPT	26.1	39.8	46.2	7.0	15.0	19.5
ChatGPT+Self-Plan	28.5	31.8	33.6	5.1	7.6	8.4
ChatGPT+SCoT	19.1	24.3	25.2	6.4	11.3	13.1
ChatGPT+KareCoder	30.5	38.8	41.8	10.5	14.4	16.4
Method	*Hard*			*ALL*		
	Pass@1	*Pass@3*	*Pass@5*	*Pass@1*	*Pass@3*	*Pass@5*
ChatGPT	6.0	10.4	12.4	12.9	22.4	26.9
ChatGPT+Self-Plan	7.3	8.6	9.0	14.2	17.3	18.3
ChatGPT+SCoT	7.7	11.6	12.4	11.2	14.2	15.0
ChatGPT+KareCoder	7.4	11.1	12.5	15.9	19.8	21.3

5 CONCLUSION AND FUTURE WORK

The paper presents KareCoder, a system for integrating programming knowledge in code generation. We developed CodeF dataset for the evaluation of LLMs and a Knowledge Library encompassing algorithms and data structures knowledge. Our research explored how to embed knowledge into code. Looking ahead, our future endeavors will focus on new knowledge integration methods and enhance the Knowledge Library.

REFERENCES

[1] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.
[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
[3] Kun Zhou, Yutao Zhu, Zhipeng Chen, Wentong Chen, Wayne Xin Zhao, Xu Chen, Yankai Lin, Ji-Rong Wen, and Jiawei Han. Don't make your llm an evaluation benchmark cheater. *arXiv preprint arXiv:2311.01964*, 2023.
[4] OpenAI. Chatgpt. https://OpenAI.com/chatgpt, 2023
[5] Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.
[6] Xue Jiang, Yihong Dong, Lecheng Wang, Qiwei Shang, and Ge Li. Self-planning code generation with large language model. *arXiv preprint arXiv:2303.06689*, 2023.
[7] Jia Li, Ge Li, Yongmin Li, and Zhi Jin. Structured chain-of-thought prompting for code generation. *arXiv preprint arXiv:2305.06599*, 2023.