Multi-Agent Collaboration for Multilingual Code Instruction Tuning

Anonymous ACL submission

Abstract

Recent advancement in code understanding and generation demonstrates that code LLMs fine-tuned on a high-quality instruction dataset can gain powerful capabilities to address wide-ranging code-related tasks. However, most previous existing methods mainly view each programming language in isolation and ignore the knowledge transfer among different programming languages. To bridge the gap among different programming languages, we introduce a novel multi-agent collaboration framework to enhance multilingual instruction tuning for code LLMs, where multiple language-specific intelligent agent components with generation memory work together to transfer knowledge from one language to another efficiently and effectively. Specifically, we first generate the language-specific instruction data from the code snippets and then provide the generated data as the seed data for language-specific agents. Multiple language-specific agents discuss and collaborate to formulate a new instruction and its corresponding solution (A new programming language or existing programming language), To further encourage the cross-lingual transfer, each agent stores its generation history as memory and then summarizes its merits and faults. Finally, the high-quality multilingual instruction data is used to encourage knowledge transfer among different programming languages to train Qwen2.5-xCoder. Experimental results on multilingual programming benchmarks demonstrate the superior performance of Qwen2.5-xCoder in sharing common knowledge, highlighting its potential to reduce the cross-lingual gap.

1 Introduction

Recent advancements (OpenAI, 2023; Gunasekar et al., 2023; Li et al., 2023b; Rozière et al., 2023; Lozhkov et al., 2024; Hui et al., 2024) in code understanding and synthesis have seen a transformative shift from small machine learning or deep learning models toward large language models

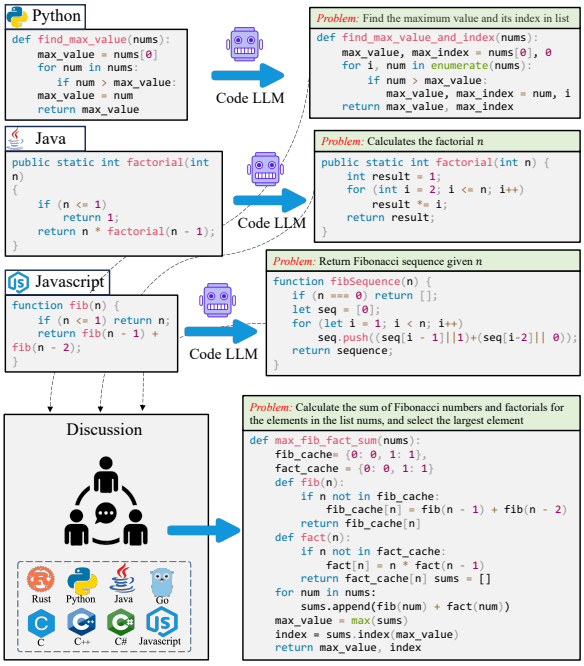


Figure 1: An example of Qwen2.5-xCoder. The Code LLM solves the code generation question by "translating" the pseudocode description (Universal Code) into executable code of the target programming language.

(LLMs) based on the Transformer architecture. The emergence of code LLMs equipped with instruction tuning has advanced a revolutionary step in many code downstream tasks, where LLMs are first trained on massive codebases with autoregressive objectives and then aligned to human preferences and downstream tasks. Code LLMs can understand complex programming problems and produce code closely mirroring user intents.

In the landscape of AI-driven code-related tasks, proprietary models such as ChatGPT and GPT-4 have gained dominance. The open-source community is making strides to narrow this gap, where Self-Instruct (Wang et al., 2023b) enhances the instruction-following capabilities of open-source LLMs. Code Alpaca (Chaudhary, 2023) uses ChatGPT to synthesize instructions with Self-Instruct. Further, Evol-Instruct evolves Code Alpaca by at-

tempting to make code instructions more complex and fine-tune the code LLM with the evolved data. A series of instruction data construction methods are proposed to generate diverse, high-quality instruction data from code snippets. However, these methods mainly focus on each programming language in isolation, ignoring the knowledge transfer among different programming languages.

To minimize the gap among different programming languages, we propose a novel multi-agent collaboration framework to generate the high-quality instruction dataset X-INSTRUCT of multilingual programming languages, which is used to fine-tune our proposed model Qwen2.5-xCoder. Specifically, we employ a multi-agent system where each agent is specialized in a different programming language to facilitate efficient and effective knowledge transfer across languages. Initially, we generate language-specific instruction data from code snippets, with each sample serving as the basis for an individual language-specific agent. These agents then engage in a collaborative discussion to synthesize new instructions, applicable to either a new or an existing programming language, along with their corresponding solutions. To enhance cross-lingual learning, each agent retains a record of its generation history, using this memory to assess its strengths and weaknesses. This iterative process results in high-quality, multilingual instruction data that is instrumental in training our method and fostering knowledge exchange among diverse programming languages.

Qwen2.5-xCoder is evaluated on the Python benchmark, including HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021), and the extended multilingual benchmark MultiPL-E, comprised of Python, Java, CPP, C-sharp, Typescript, PHP, and Bash. The Experimental results demonstrate that Qwen2.5-xCoder consistently outperforms the previous baselines. Empirical studies show that Qwen2.5-xCoder can effectively transfer knowledge of data in different languages to each other and thus help to alleviate the negative language interference among various languages. The contributions are summarized as follows:

- We introduce a multilingual multi-agent framework, where multiple agents participate in a collaborative discussion to synthesize new instructions and the corresponding answers. These cooperative agents work together towards a shared goal, typically exchanging information to enhance a collective solution.
- Based on the code snippets extracted from the open-source code, we leverage the multilingual multi-agent framework to create a multilingual programming instruction dataset X-INSTRUCT to improve the cross-lingual capabilities of the code LLM.
- To validate the effectiveness of our method, we introduce a series of Qwen2.5-xCoder models fine-tuned on our data generation strategy based on Code Llama, and Deepseek-Coder.

2 Qwen2.5-xCoder

2.1 Model Overview

In Figure 2, we develop a multi-agent framework to construct a multilingual instruction dataset from code snippets. Each agent in the framework specializes in a different programming language, facilitating effective knowledge transfer between languages. The code snippets are assigned to the respective language-specific agents, who then generate individual instructions. These agents collaborate, leveraging their expertise to create new instructions that can be applied to various programming languages, along with corresponding solutions. To enhance cross-lingual learning, agents keep a record of their generated instructions to identify their strengths and areas for development. This collaborative approach allows us to produce a high-quality multilingual instruction dataset that can be used for instruction tuning.

2.2 Seed Instruction Dataset

Instruction from Code Snippet. For the unsupervised data (code snippets) massively existing in many websites (e.g. GitHub), we try to construct the supervised instruction dataset. Specifically, we use the LLM to generate the instruction q from the code snippets within 1024 tokens and then we use the code LLM to generate the response a. Finally, we use the LLM scorer in Figure 4 to filter the low-quality ones to obtain the final pair (q, a). Given the code snippets of different programming languages $L_k \in \{L_k\}_{k=1}^{K}$, we construct instruction dataset $D_{s_1} = \{D_{s_1}^{L_k}\}_{k=1}^{K}$ from the code snippets. (K is the number of programming languages)

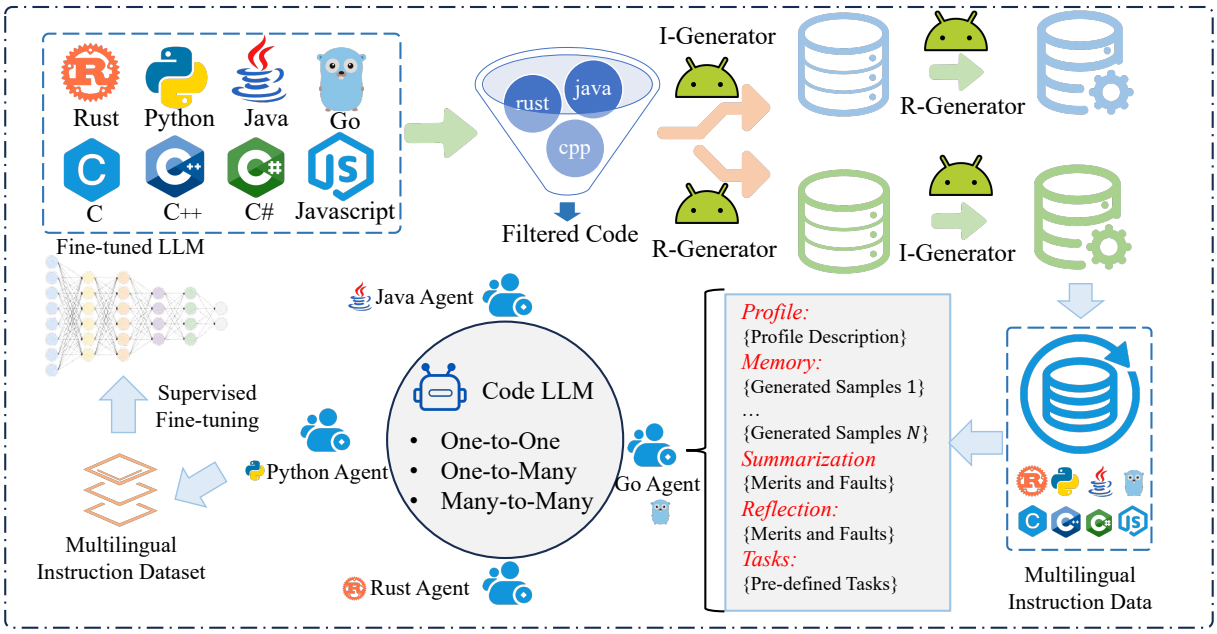


Figure 2: Overview of multilingual multi-agent data generation framework. we first construct the multilingual instruction dataset from the code snippets. We introduce a multi-agent framework, with each agent possessing expertise in a different programming language, allowing for efficient knowledge transfer across various languages. "R-Generator" generates the responses based on the instruction while "I-Generator" generates the instruction based on the responses. Each snippet is assigned to a language-specific agent who uses it to create individual instructions. The agents then collaborate, using their specialized knowledge to create new instructions that can be applied to either a new or existing programming language, along with the appropriate solutions. To improve cross-lingual learning, agents maintain a history of their generated instructions, allowing them to identify their strengths and areas for improvement. Through this collaborative process, we produce high-quality multilingual instruction data for instruction tuning.

Response from Code Snippet. To increase the diversity of the instruction dataset. Conversely, we first generate the responses from the code and then prompt the LLM to generate instructions. Then we use the LLM scorer to filter the low-quality to obtain the final pair (q, a). Similarly, given the code snippets of different programming languages $L_k \in \{L_k\}_{k=1}^{K}$, we can construct instruction dataset $D_{s_2} = \{D_{s_2}^{L_k}\}_{k=1}^{K}$ from the code snippets. To fully unleash the potential of our proposed method, we combine two parts instruction dataset $D_{s_1} \cup D_{s_2}$ as the seed data for multi-agent data generation framework.

2.3 Language Agent

Figure 2 shows the overall framework of the multi-agent framework to generate the new samples.

Instance-level Agent. Given the seed instruction dataset D_s, each instance $(q, a) \in D_s$ is used to initialize the agent $\mathcal{A} = \{p, m, r, O\}$, where $\mathcal{A}$ contains the pre-defined agent profile p (task definition), the agent operation $o \in O = \{o_1 \ldots, o_a\}$, the memory of the generated history $m = \{m_1, \ldots, m_T\}$, the reflection $r = \{r_1, \ldots, r_c\}$. O contains different evolution operations, such as increasing difficulty and adding more reasoning steps. m is comprised of T history generated samples, where $m_i = (q_i, a_i) \in m$ (q_i and a_i are the question and answer generated by the agent $\mathcal{A}$).

Memory Initialization&Update. The memory module m is essential for the abilities of an agent to gather, retain, and apply knowledge gained from interactions. Initially, the agent $\mathcal{A}$ does not generate any samples, and thus the memory is initialized by the seed data ($m = \{q_s, a_s\}$). Memory updating is the process of recording new information, thereby updating the knowledge base of the agent with fresh insights or observations. When the agent generates the new sample (q, a), the pair is added into the memory m, where the similarity between (q, a) and the samples already in m is less than a certain threshold to prevent duplicates. m is a priority queue with capacity size T. When the capacity is full, the sample with the lowest evaluated score is first removed from the queue.

You are a programming expert, very good at designing high-quality engineering problems and algorithmic problems.

[Memory]:
Here are the algorithmic problems and solutions you've generated in the past:
{Sample 1}
…
{Sample T}

[Reflection]:
Here are the merits and faults of each sample in [memory] you've generated in the past:
{Sample 1}
…
{Sample T}

[Communication]:
{Agent 1}
…
{Agent M}

[Goal]:
Please rewrite the problem to specify a programming language {target_language}.
The created problem ({target_language}) needs to fulfill the following requirements compared to the given problem.

[Guidelines]
* Please create a new prompt based on the given prompts in [Communication]
* Please increase the difficulty by proposing higher time or space complexity requirements.
* Please try to avoid generating new questions that are similar to the original questions.
* Please avoid creating the same problem description as in [memory] and generating more novel problems based on previous generation history.
[Created Prompt]:

Figure 3: Prompt of the multilingual multi-agent framework.

As a code expert, your task is to evaluate the provided data based on specific criteria. Please generate a JSON list that includes the scores for the data, following the given guidelines.

Return Format:
[{"expert": "", "score": "", "reason": ""}]
Scoring Range:
1.0: very bad, 2.0: bad, 3.0: Neutral, 4.0: good, 5.0: very good, N/A: Not applicable

Experts of different aspects:
* Question&Answer Consistency: Whether Q&A are consistent and correct for fine-tuning.
* Question&Answer Relevance: Whether Q&A are related to the computer field.
* Question&Answer Difficulty: Whether Q&A are sufficiently challenging
* Code Exist: Whether the code is provied in question or answer.
* Code Correctness: Evaluate whether the provided code is free from syntax errors and logical flaws. Consider factors like proper variable naming, code indentation, and adherence to best practices.
* Code Clarity: Assess how clear and understandable the code is. Evaluate if it uses meaningful variable names, proper comments, and follows a consistent coding style.
* Code Comments: Evaluate the presence of comments and their usefulness in explaining the code's functionality.
* Easy to Learn: determine its educational value for a student whose goal is to learn basic coding concepts
* Total Score: Give a final rating based on the quality of the data.
Please score the following Q&A (score the data as strictly as possible and identify errors):

[Question]:
{question}
[Answer]:
{response}

Figure 4: Prompt of evaluation.

Memory Relfection. The goal is to equip agents with the ability to autonomously generate summaries of their experiences and draw inferences that reach beyond simple data processing. After the agent $\mathcal{A}$ synthesizes one new sample (q, a), we use the LLM scorer to evaluate the sample in many aspects and obtain the reflection r in Figure 4.

2.4 Cross-lingual Discussion

The communication between agents in our proposed multi-agent framework is the critical infrastructure supporting collective intelligence. We introduce two types of communication structures, including centralized communication and distributed communication. When multiple agents try to synthesize new samples, each agent will randomly select a sample from its memory for further data synthesis. (for the first data synthesis, the agents uses the provided seed data $D_{s_1} \cup Ds_2$ to synthesize new samples.)

Centralized Discussion. Centralized communication involves a central agent coordinating the communication, with other agents primarily interacting through this central node. Given d agents $\{\mathcal{A}_1, \ldots, \mathcal{A}_d\}$($\mathcal{A}_1$ is the main agent and others are auxiliary agents), the prompt $\mathcal{F}$ is used to generate the new sample (q, a) mainly based on $\mathcal{A}_1$, where the process can be described as $\mathcal{F}(\mathcal{A}_1; \mathcal{A}_2, \ldots, \mathcal{A}_d)$.

Parallel Discussion. Parallel discussion equally regards all agents $\mathcal{A}_1, \ldots, \mathcal{A}_d$, the prompt is used to consider all agents and generate a new sample, where the process is described as $\mathcal{F}(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_d)$.

Data Generation. Based on the seed instruction dataset D_s, we adopt the multilingual agent framework to create the multilingual instruction dataset $D = \{D^{L_k}\}_{k=1}^{K}$ (K is the number of languages). Finally, the generated multilingual data from different agents comprise the corpus D_{s_3}.

2.5 Multilingual Code Instruction Tuning

Given the multilingual corpora $D = \{D^{L_k}\}_{k=1}^{K}$, the training objective of the $\mathcal{L}_{\text{SFT}}$ can be described:

$$\mathcal{L}_{\text{SFT}} = -\sum_{k=1}^{K} \mathbb{E}_{q,a\sim D_{L_k}} \left[\log P(a|q;\mathcal{M})\right] \quad (1)$$

where q and a are the question and answer pair. $\mathcal{L}_{\text{SFT}}$ is the instruction fine-tuning objective.

2.6 Multilingual Code DPO

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}} \left[\log \sigma(\beta(\frac{\pi_\theta(y^+|x)}{\pi_{\text{r}}(y^+|x)} - \frac{\pi_\theta(y^-|x)}{\pi_{\text{r}}(y^-|x)})\right] \quad (2)$$

where (y^+, y^-) is the positive and negative pair. σ denotes the sigmoid function. After generating the SFT data, we further leverage the multi-agent collaboration data generation framework to synthesize the DPO data. For the sample query, we prompt the SFT model to sample 128 responses and use the code execution to verify the correctness of the generated code snippet. We feed generated test cases from LLM into the code snippet to verify the correctness of the function under the multilingual environment. The DPO data is denoted as the D_{s_4}. Finally, we use $D_{s_1} \cup D_{s_2} \cup D_{s_3}$ for supervised fine-tuning and D_{s_4} for preference learning.

3 Experiments

3.1 Implementation Details

For the code snippets collected from GitHub, we choose nearly 100K code snippets from different languages (Python, Java, CPP, C-sharp, Typescript, PHP, and Bash) to construct the synthetic instruction dataset. Finally, we obtain the instruction dataset X-INSTRUCT containing nearly 95K training samples. We utilize Qwen2.5-Coder as the foundational code LLMs for supervised fine-tuning. We fine-tune these foundation LLMs on X-INSTRUCT. Qwen2.5-xCoder (32B) is fine-tuned on the 128 NVIDIA H800-80GB GPUs[1]. The learning rate first increases into 5e-5 with 100 warmup steps and then adopts a cosine decay scheduler. We adopt the Adam optimizer with a global batch size of 1024 samples, truncating sentences to 2048 tokens.

[1] https://github.com/QwenLM/Qwen2.5-Coder/tree/main/finetuning/sft

3.2 Evaluation Metrics

Pass@k. We adopt the Pass@k metric (Zheng et al., 2023) for evaluation by using test cases to verify the correctness of the generated code. For the fair comparison, we report the greedy Pass@1 scores for all LLMs in our work.

3.3 Instruction Dataset

We use the created instruction dataset we combine three parts instruction dataset $D_{s_1} \cup D_{s_2} \cup D_{s_3}$ for supervised fine-tuning, comprising nearly 97.3K instruction samples. The created samples $D_{s_1} \cup D_{s_2}$ from code snippets contain 30K samples. The existing instruction dataset D_{s_3} contains nearly 67K samples. The data used for DPO is comprised of 133K samples.

3.4 Baselines

Proprietary Models. GPT-3.5 and GPT-4 (OpenAI, 2023) are both LLMs developed by OpenAI based on a neural architecture known as generative pre-trained Transformers (GPT) (Radford et al., 2018). They are both trained on massive datasets of text and code, allowing them to generate human-quality text, translate languages, and write different kinds of creative content. GPT-3.5 Turbo (i.e., ChatGPT) and GPT4 achieve excellent performance in various code understanding and generation tasks.

Open-Source Models. To narrow the gap between open-source and closed-source models, a series of open-source models and instruction datasets are proposed to improve the code foundation LLMs and bootstrap the instruction-following ability of code LLMs. We compare the following code LLMs: Qwen-Coder (Hui et al., 2024), Code Llama (Rozière et al., 2023), and DeepSeek-Coder (Guo et al., 2024b) with different model sizes are introduced into the based model.

3.5 Evaluation Benchmark

EvalPlus. The HumanEval test set (Chen et al., 2021) is a crafted collection of 164 Python programming problems to test the abilities of code generation models. For each problem, there are roughly 9.6 test cases to check whether the generated code works as intended. Humaneval has become the most popular benchmark to measure how well these code-writing AI models perform, making it a key tool in the field of AI and machine learning for coding. The MBPP dataset (Austin et al.,

2021), comprising approximately 1,000 Python programming challenges sourced from a crowd of contributors, is tailored for beginners in programming, focusing on core principles and the usage of the standard library. The MBPP test set comprised of 500 problems is selected to evaluate the few-shot inference of the code LLMs.

MultiPL-E. The MultiPL-E test set (Cassano et al., 2023) translates the original HumanEval test set to other languages and we report the scores of the languages Python, Java, CPP, Typescript, Javascript, PHP, and Bash.

3.6 Baselines

Proprietary Models. GPT-3.5 and GPT-4 (OpenAI, 2023) are both LLMs developed by OpenAI based on a neural architecture known as generative pre-trained Transformers (GPT). They are both trained on massive datasets of text and code, allowing them to generate human-quality text, translate languages, and write different kinds of creative content. GPT-3.5 Turbo (i.e., ChatGPT) and GPT4 achieve excellent performance in various code understanding and generation tasks.

Open-Source Models. To narrow the gap between open-source and closed-source models, a series of open-source models and instruction datasets are proposed to improve the code foundation LLMs and bootstrap the instruction-following ability of code LLMs. Starcoder (Li et al., 2023a), Code Llama (Rozière et al., 2023), and DeepSeek-Coder (Guo et al., 2024b) with different model sizes are introduced into the based model. OctoCoder, WiazrdCoder, MagiCoder, WaveCoder

4 Base Models

Qwen2.5-Coder. The Qwen2.5-Coder (Guo et al., 2024b) series contains a range of open-source code models with sizes from 0.5B to 32B, pretrained on 5.5 trillion tokens from the Qwen2.5 base model. These models are fine-tuned on a high-quality code corpus, using a 32K token window for a fill-in-the-middle task to improve code generation and completion.

4.1 Main Results

Python Code Generation. Table 1 illustrates that Qwen2.5-xCoder significantly outperforms the base Code Llama and previous strong open-source baselines, minimizing the gap with GPT-3.5 and GPT-4. Particularly, Qwen2.5-xCoder outperforms the WizardCoder with 15B foundation LLM and Evol-Instruct techniques. Magicoder (Wei et al., 2023) and Wavecoder (Yu et al., 2023) both prove the effectiveness of data construction from code snippets.

Multilingual Code Generation. For the multilingual code generation task, our proposed model Qwen2.5-xCoder is evaluated on the MultiPL-E, including Python, Java, CPP, C-sharp, and other languages. The experimental results in Figure 1 show that DS-Coder and Qwen2.5-Coder significantly outperform across all languages. The enhanced version, Qwen2.5-xCoder, further improves the multilingual performance of LLM, rivaling the CodeLlama-70B-Instruct model with only 14B parameters. Remarkably, we introduce the language-specific multi-agent framework to compose new samples for enhancing cross-lingual transfer among different programming languages.

Multilingual Code Understanding. Given the multilingual correct code snippet, the code LLM is tasked to generate an explanation of the code and then regenerate the code only based on its own explanation. For the different backbones Code Llama and Deepseek-Coder, our method beats most previous methods, especially in other languages, which demonstrates that X-INSTRUCT can bring multilingual agreement for different programming languages.

5 Analysis

Ablation Study. Given the synthetic instruction dataset, we can obtain the fine-tuned model Qwen2.5-xCoder based on the Qwen2.5-Coder-7B denoted as ①. Our multilingual instruction dataset contains four parts $\{D_{s_1}, D_{s_2}, D_{s_3}, D_{s_4}\}$. D_{s_1} is created from the code snippets by first generating instructions and then outputting the response while D_{s_2} is developed from the code snippets by first generating responses and then synthesizing the instructions. D_{s_3} is derived from our multilingual multi-agent framework. D_{s_4} is derived from the temperature-based sampling for DPO training. Specially, we can observe that ③ drops a lot compared to ②. It indicates the significance of the dataset D_{s_3} from the multi-agent data generation framework. Table 2 summarizes the results of the ablation study of these datasets, which shows that our multilingual multi-agent framework can leverage code snippets

Model	Size	HE	HE+	MBPP	MBPP+	Python	Java	C++	C#	TS	JS	PHP	Bash	Avg.
Closed-APIs														
Claude-3.5-Sonnet-20240620	🔒	89.0	81.1	87.6	72.0	89.6	86.1	82.6	85.4	84.3	84.5	80.7	48.1	80.2
Claude-3.5-Sonnet-20241022	🔒	92.1	86.0	91.0	74.6	93.9	86.7	88.2	87.3	88.1	91.3	82.6	52.5	83.8
GPT-4o-mini-2024-07-18	🔒	87.8	84.8	86.0	72.2	87.2	75.9	77.6	79.7	79.2	81.4	75.2	43.7	75.0
GPT-4o-2024-08-06	🔒	92.1	86.0	86.8	72.5	90.9	83.5	76.4	81.0	83.6	90.1	78.9	48.1	79.1
o1-mini	🔒	97.6	90.2	93.9	78.3	95.7	90.5	93.8	77.2	91.2	92.5	84.5	55.1	85.1
o1-preview	🔒	95.1	88.4	93.4	77.8	96.3	88.0	91.9	84.2	90.6	93.8	90.1	47.5	85.3
0.5B+ Models														
Qwen2.5-Coder-0.5B-Instruct	0.5B	61.6	57.3	52.4	43.7	61.6	57.3	52.4	43.7	50.3	50.3	52.8	27.8	49.6
Qwen2.5-xCoder (SFT)	0.5B	69.5	66.5	52.6	45.5	68.9	57.6	53.4	67.7	63.5	66.5	57.8	33.5	57.1
Qwen2.5-xCoder (DPO)	0.5B	72.6	67.1	51.9	45.2	72.0	58.2	54.0	68.4	66.7	64.0	57.8	36.7	58.0
1B+ Models														
DS-Coder-1.3B-Instruct	1.3B	65.9	60.4	65.3	54.8	65.2	51.9	45.3	55.1	59.7	52.2	45.3	12.7	48.4
Yi-Coder-1.5B-Chat	1.5B	69.5	64.0	65.9	57.7	67.7	51.9	49.1	57.6	57.9	59.6	52.2	19.0	51.9
Qwen2.5-Coder-1.5B-Instruct	1.5B	70.7	66.5	69.2	59.4	71.2	55.7	50.9	64.6	61.0	62.1	59.0	29.1	56.7
Qwen2.5-xCoder (SFT)	1.5B	85.4	79.9	69.3	59.5	70.1	68.4	66.5	68.4	71.1	72.7	70.2	44.3	65.9
Qwen2.5-xCoder (DPO)	1.5B	65.2	60.4	70.9	61.6	74.4	69.6	67.7	70.9	75.5	74.5	69.6	44.9	67.5
3B+ Models														
Qwen2.5-Coder-3B-Instruct	3B	84.1	80.5	73.6	62.4	83.5	74.7	68.3	78.5	79.9	75.2	73.3	43.0	72.1
Qwen2.5-xCoder (SFT)	3B	84.8	79.9	69.3	61.1	67.7	57.6	39.8	28.5	43.4	55.3	38.5	31.6	42.1
Qwen2.5-xCoder (DPO)	3B	85.4	80.5	75.7	66.7	80.5	11.4	70.2	80.4	82.4	82.6	74.5	43.7	63.6
6B+ Models														
CodeLlama-7B-Instruct	7B	40.9	33.5	54.0	44.4	34.8	30.4	31.1	21.6	32.7	-	28.6	10.1	-
DS-Coder-6.7B-Instruct	6.7B	74.4	71.3	74.9	65.6	78.6	68.4	63.4	72.8	67.2	72.7	68.9	36.7	66.1
CodeQwen1.5-7B-Chat	7B	83.5	78.7	77.7	67.2	84.1	73.4	74.5	77.8	71.7	75.2	70.8	39.2	70.8
Yi-Coder-9B-Chat	9B	82.3	74.4	82.0	69.0	85.4	76.0	67.7	76.6	72.3	78.9	72.1	45.6	71.8
DS-Coder-V2-Lite-Instruct	2.4/16B	81.1	75.6	82.8	70.4	81.1	76.6	75.8	76.6	80.5	77.6	74.5	43.0	73.2
Qwen2.5-Coder-7B-Instruct	7B	88.4	84.1	83.5	71.7	87.8	76.5	75.6	80.3	81.8	83.2	78.3	48.7	76.5
OpenCoder-8B-Instruct	8B	83.5	78.7	79.1	69.0	83.5	72.2	61.5	75.9	78.0	79.5	73.3	44.3	71.0
Qwen2.5-xCoder (SFT)	7B	86.0	79.9	81.2	67.2	84.1	39.2	78.9	79.7	81.8	78.9	68.3	46.2	67.6
Qwen2.5-xCoder (DPO)	7B	86.0	79.9	82.3	69.3	82.3	23.4	77.0	81.6	81.1	82.0	65.8	47.5	65.5
13B+ Models														
CodeLlama-13B-Instruct	13B	40.2	32.3	60.3	51.1	42.7	40.5	42.2	24.0	39.0	-	32.3	13.9	-
Starcoder2-15B-Instruct-v0.1	15B	67.7	60.4	78.0	65.1	68.9	53.8	50.9	62.7	57.9	59.6	53.4	24.7	54.0
Qwen2.5-Coder-14B-Instruct	14B	89.6	87.2	86.2	72.8	89.0	79.7	85.1	84.2	86.8	84.5	80.1	47.5	79.6
Qwen2.5-xCoder (SFT)	14B	88.4	83.5	84.1	73.3	89.0	81.0	75.2	81.6	84.3	85.1	75.2	46.8	75.6
Qwen2.5-xCoder (DPO)	14B	89.0	84.8	83.1	72.0	91.5	81.0	78.9	82.9	85.5	85.1	77.0	48.1	76.9
20B+ Models														
CodeLlama-34B-Instruct	34B	48.2	40.2	61.1	50.5	41.5	43.7	45.3	31.0	40.3	-	36.6	19.6	-
CodeStral-22B-v0.1	22B	81.1	73.2	78.2	62.2	81.1	63.3	65.2	43.7	68.6	-	68.9	42.4	-
DS-Coder-33B-Instruct	33B	81.1	75.0	80.4	70.1	79.3	73.4	68.9	74.1	67.9	73.9	72.7	43.0	69.2
CodeLlama-70B-Instruct	70B	72.0	65.9	77.8	64.6	67.8	58.2	53.4	36.7	39.0	-	58.4	29.7	-
DS-Coder-V2-Instruct	21/236B	85.4	82.3	89.4	75.1	90.2	82.3	84.8	82.3	83.0	84.5	79.5	52.5	79.9
Qwen2.5-Coder-32B-Instruct	32B	92.7	87.2	90.2	75.1	92.7	80.4	79.5	82.9	86.8	85.7	78.9	48.1	79.4
Qwen2.5-32B-Instruct	32B	87.8	82.9	86.8	70.9	88.4	80.4	81.0	74.5	83.5	82.4	78.3	46.8	76.9
Qwen2.5-72B-Instruct	32B	85.4	79.3	90.5	77.0	82.9	81.0	80.7	81.6	81.1	82.0	77.0	48.7	75.1
Qwen2.5-SynCoder	32B	92.7	87.8	86.2	74.7	92.1	80.4	80.7	81.6	83.0	85.7	77.6	49.4	78.8
Qwen2.5-xCoder (SFT)	32B	87.8	84.1	84.9	74.9	89.6	74.7	73.3	79.1	82.4	81.4	78.3	46.2	73.6
Qwen2.5-xCoder (DPO)	32B	89.0	86.0	85.4	74.9	90.9	76.6	72.7	79.1	83.6	81.4	78.9	49.4	74.5

Table 1: The performance of different instruction LLMs on EvalPlus and MultiPL-E. "HE" denotes the HumanEval, "HE+" denotes the plus version with more test cases, and "MBPP+" denotes the plus version with more test cases.

ID	Methods	Python	Java	C++	C#	Avg.
①	Qwen2.5-xCoder	**90.9**	**76.6**	**72.7**	**79.1**	**79.8**
②	① - D_{s_4}	89.6	74.7	73.3	79.1	79.2
③	② - D_{s_3}	82.9	69.6	68.3	70.3	72.8
③	② - D_{s_2} (D_{s_1})	79.9	67.1	65.8	68.4	70.3

Table 2: Ablation study of our proposed method. Qwen2.5-xCoder is fine-tuned on the combination of all generated instruction datasets.

and existing instruction datasets to transfer knowledge from Python to other languages.

Effect of Instruction Data Size. To discuss the effect of the size of our created instruction dataset X-INSTRUCT (nearly 97.3K sentences), we plot evaluation scores with different training data sizes in Figure 5. We randomly sample $\{1K, \ldots, ALL\}$ sentences from the whole corpora to fine-tune the base Qwen2.5-Coder. With the training data size increasing, the fine-tuned model gets better performance. Surprisingly, only $50K$ pseudo annotated sentences bring large improvement to the multilingual code generation, which benefits from the knowledge transfer of the multilingual agents.

Token Count Distribution. Figure 6 illustrates the distribution of lengths for both the generated problems and their corresponding solutions. On the

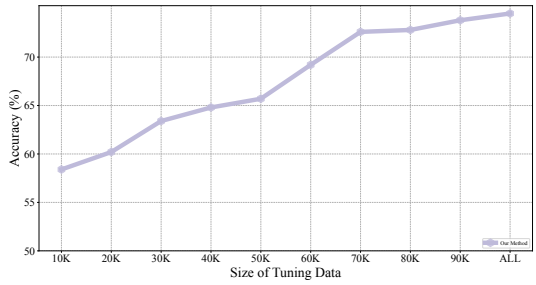
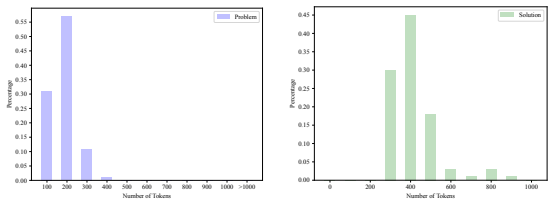
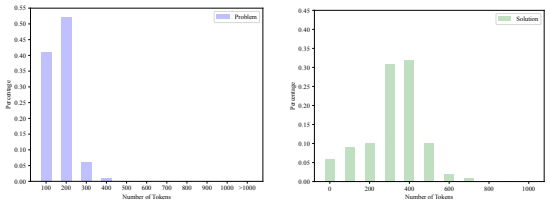


Figure 5: Evaluation results (average scores of 8 programming languages) of Pass@1 on the MultiPL-E with different training sizes by randomly down-sampling.



(a) Problem (SFT) (b) Solution (SFT)



(c) Problem (DPO) (d) Solution (DPO)

Figure 6: Length distribution of problem and solution for the SFT data and DPO data.

horizontal axis, we quantify length in terms of the number of tokens comprising each problem or solution. The solution has a similar length distribution to the distribution of the problem.

6 Related Work

Code-related Tasks. Pre-training has significantly enhanced the model capabilities of code understanding and synthesis in downstream various tasks, such as CodeBERT (Feng et al., 2020) and CodeT5 (Wang et al., 2021). The model architecture and pre-training objectives originating from natural language processing (NLP) (Lu et al., 2021; Yan et al., 2023; Liu et al., 2023; Xie et al., 2023) have been increasingly adopted to synthesize programs from human language and perform code infilling, effectively addressing a multitude of software engineering challenges, such as code summarization, code refinement, and code translation.

Code-specific Large Language Model. Code-specific large language models (LLMs) (Li et al., 2023a; Guo et al., 2024a; Yang et al., 2024a,b) trained on large-scale code corpora show remarkable performance across a diverse set of software engineering tasks. Code LLMs culminate in a foundational competence in general code generation and understanding, such as CodeGen (Nijkamp et al., 2023) and Code Llama (Rozière et al., 2023), which enables them to tackle code-related tasks with better performance.

Multilingual Code Instruction Tuning. Instruction tuning is a powerful paradigm enhancing the performance of LLMs by fine-tuning them with the instruction dataset (Ouyang et al., 2022; Zhang et al., 2023; Wang et al., 2023b). Instruction tuning enables LLMs to generalize better and follow instructions more directly. The previous works (Wang et al., 2023b; Chaudhary, 2023) use a foundation LLM to generate the instruction data and then refine the model through instruction tuning with the synthetic data. To further enhance Self-Instruct, WizardCoder (Luo et al., 2023) introduces code Evol-Instruct to produce more high-quality data by using heuristic prompts to increase the complexity and diversity of synthetic data. Recently, OSS-Instruct (Wei et al., 2023) and CodeOcean (Yu et al., 2023) leveraged real-world code snippets to inspire LLMs to generate more controllable and realistic instruction corpora. A series of multilingual benchmarks (Cassano et al., 2023; Chai et al., 2024; Liu et al., 2024a,b; Zhuo et al., 2024) (e.g. MultiPl-E, McEval, and MdEval) are proposed to evaluate the multilingual capabilities of code LLMs.

7 Conclusion

In this work, we propose a novel multilingual multi-agent collaboration framework to bridge the language divide in programming by generating a comprehensive and high-quality multilingual instruction dataset X-INSTRUCT for fine-tuning Qwen2.5-xCoder. Our proposed model leverages the expertise of individual agents, each fluent in a different programming language, to achieve effective knowledge transfer. The collaborative efforts among multiple agents, informed by their individual generation histories, enable the synthesis of versatile programming instructions and solutions. This enriches the instruction dataset and bolsters the capability of our model to generalize across languages, promising significant advancements in the field of multilingual programming development.

Limitations

We acknowledge the following limitations of this study: (1) This work focuses on exploring instruction tuning for multilingual code-related works. The investigation of this paradigm on other multilingual tasks has not been studied yet. (2) While our approach aims to facilitate knowledge transfer across multiple programming languages, it may not be equally effective for all languages, potentially leading to a bias towards more commonly used or better-represented languages in the dataset.

Ethics Statement

Qwen2.5-xCoder, as a novel multi-agent collaboration framework, enhances multilingual instruction tuning for code LLMs, where multiple language-specific intelligent agent components with generation memory work together to transfer knowledge from one language to another efficiently and effectively. Qwen2.5-xCoder effectively enhances the multilingual code generation capabilities of the LLMs but the constructed SFT and DPO data may contain unsafe queries and thus lead to unsafe code generation and execution. Therefore, to ensure the security and reliability of the code execution, we filter the unsafe queries with an LLM filter and suggest the users should run the code under a sandbox to verify the safety of the generated code.

References

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. 2023. Multipl-e: a scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*.

Linzheng Chai, Shukai Liu, Jian Yang, Yuwei Yin, Ke Jin, Jiaheng Liu, Tao Sun, Ge Zhang, Changyu Ren, Hongcheng Guo, et al. 2024. Mceval: Massively multilingual code evaluation. *arXiv preprint arXiv:2406.07436*.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison 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. abs/2107.03374.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.

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. 2024a. Deepseek-coder: When the large language model meets programming – the rise of code intelligence.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. 2024b. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024c. Large language model based multi-agents: A survey of progress and challenges. *CoRR*, abs/2402.01680.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Moustafa-Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023a. Starcoder: may the source be with you! *CoRR*, abs/2305.06161.

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

Shukai Liu, Linzheng Chai, Jian Yang, Jiajun Shi, He Zhu, Liran Wang, Ke Jin, Wei Zhang, Hualei Zhu, Shuyue Guo, et al. 2024a. Mdeval: Massively multilingual code debugging. *arXiv preprint arXiv:2411.02310*.

Siyao Liu, He Zhu, Jerry Liu, Shulin Xin, Aoyan Li, Rui Long, Li Chen, Jack Yang, Jinxiang Xia, ZY Peng, et al. 2024b. Fullstack bench: Evaluating llms as full stack coder. *arXiv preprint arXiv:2412.00535*.

Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach Dinh Le, and David Lo. 2023. Refining chatgpt-generated code: Characterizing and mitigating code quality issues. *CoRR*, abs/2307.12596.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

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*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *CoRR*, abs/2306.08568.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

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.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2023a. A survey on large language model based autonomous agents. *CoRR*, abs/2308.11432.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. 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 8696–8708. Association for Computational Linguistics.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source code is all you need. *CoRR*, abs/2312.02120.

Zhuokui Xie, Yinghao Chen, Chen Zhi, Shuiguang Deng, and Jianwei Yin. 2023. Chatunitest: a chatgpt-based automated unit test generation tool. *CoRR*, abs/2305.04764.

Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. 2023. Codetransocean: A comprehensive multilingual benchmark for code translation. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 5067–5089. Association for Computational Linguistics.

Jian Yang, Jiaxi Yang, Ke Jin, Yibo Miao, Lei Zhang, Liqun Yang, Zeyu Cui, Yichang Zhang, Binyuan Hui, and Junyang Lin. 2024a. Evaluating and aligning codellms on human preference. *arXiv preprint arXiv:2412.05210*.

Jian Yang, Jiajun Zhang, Jiaxi Yang, Ke Jin, Lei Zhang, Qiyao Peng, Ken Deng, Yibo Miao, Tianyu Liu, Zeyu Cui, et al. 2024b. Execrepobench: Multi-level executable code completion evaluation. *arXiv preprint arXiv:2412.11990*.

Zhaojian Yu, Xin Zhang, Ning Shang, Yangyu Huang, Can Xu, Yishujie Zhao, Wenxiang Hu, and Qiufeng Yin. 2023. Wavecoder: Widespread and versatile enhanced instruction tuning with refined data generation. *CoRR*, abs/2312.14187.

Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. 2023. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *CoRR*, abs/2303.16199.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*, abs/2303.17568.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

A Related Work

Code-related Tasks. Pre-training has significantly enhanced the model capabilities of code understanding and synthesis in downstream various tasks, such as CodeBERT (Feng et al., 2020) and CodeT5 (Wang et al., 2021). The model architecture and pre-training objectives originating from natural language processing (NLP) (Lu et al., 2021; Yan et al., 2023; Liu et al., 2023; Xie et al., 2023) have been increasingly adopted to synthesize programs from human language and perform code infilling, effectively addressing a multitude of software engineering challenges, such as code summarization, code refinement, and code translation.

Code-specific Large Language Model. Code-specific large language models (LLMs) (Li et al., 2023a; Rozière et al., 2023; Guo et al., 2024a; Yang et al., 2024a,b) trained on large-scale code corpora show remarkable performance across a diverse set of software engineering tasks. Code LLMs culminate in a foundational competence in general code generation and understanding, such as CodeGen (Nijkamp et al., 2023) and Code Llama (Rozière et al., 2023), which enables them to tackle code-related tasks with better performance. Inspired by the success of multi-agent collaboration in other fields (Guo et al., 2024c; Wang et al., 2023a), we introduce the language-specific agent to formulate a multilingual instruction dataset.

Multilingual Code Instruction Tuning. Instruction tuning is a powerful paradigm enhancing the performance of LLMs by fine-tuning them with the instruction dataset (Ouyang et al., 2022; Zhang et al., 2023; Wang et al., 2023b). Instruction tuning enables LLMs to generalize better and follow instructions more directly. The previous works (Wang et al., 2023b; Chaudhary, 2023) use a foundation LLM to generate the instruction data and then refine the model through instruction tuning with the synthetic data. To further enhance Self-Instruct, WizardCoder (Luo et al., 2023) introduces code Evol-Instruct to produce more high-quality data by using heuristic prompts to increase the complexity and diversity of synthetic data. Recently, OSS-Instruct (Wei et al., 2023) and CodeOcean (Yu et al., 2023) leveraged real-world code snippets to inspire LLMs to generate more controllable and realistic instruction corpora. A series of multilingual benchmarks (Cassano et al., 2023; Chai et al., 2024; Liu et al., 2024a,b; Zhuo et al., 2024) (e.g. MultiPl-E, McEval, and MdEval) are proposed to evaluate the multilingual capabilities of code LLMs.