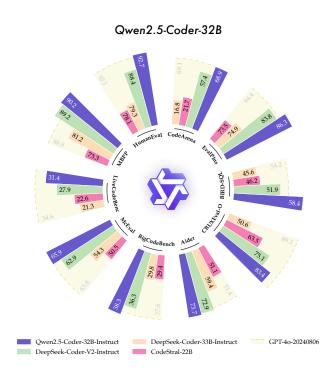# Qwen2.5-Coder Technical Report

**Binyuan Hui\*    Jian Yang\*    Zeyu Cui\*    Jiaxi Yang\***

Dayiheng Liu    Lei Zhang    Tianyu Liu    Jiajun Zhang    Bowen Yu    Keming Lu
Kai Dang    Yang Fan    Yichang Zhang    An Yang    Rui Men    Fei Huang
Bo Zheng    Yibo Miao    Shanghaoran Quan    Yunlong Feng
Xingzhang Ren    Xuancheng Ren    Jingren Zhou    Junyang Lin[†]
**Qwen Team    Alibaba Group**

🤗 https://hf.co/Qwen/Qwen2.5-Coder-32B-Instruct
⭘ https://github.com/QwenLM/Qwen2.5-Coder

## Abstract

In this report, we introduce the Qwen2.5-Coder series, a significant upgrade from its predecessor, CodeQwen1.5. This series includes six models: Qwen2.5-Coder-(0.5B/1.5B/3B/7B/14B/32B). As a code-specific model, Qwen2.5-Coder is built upon the Qwen2.5 architecture and continues pretrained on a vast corpus of over 5.5 trillion tokens. Through meticulous data cleaning, scalable synthetic data generation, and balanced data mixing, Qwen2.5-Coder demonstrates impressive code generation capabilities while retaining general and math skills. These models have been evaluated on a wide range of code-related tasks, achieving state-of-the-art (SOTA) performance across more than 10 benchmarks, including code generation, completion, reasoning, and repair, consistently outperforming larger models of the same model size. We believe that the release of the Qwen2.5-Coder series will advance research in code intelligence and, with its permissive licensing, support wider adoption by developers in real-world applications.

Qwen2.5-Coder-32B

---

\*Equal core contribution, [†]Corresponding author

## Contents

# 1 Introduction

With the rapid development of large language models (LLMs) (Brown, 2020; Achiam et al., 2023; Touvron et al., 2023; Dubey et al., 2024; Jiang et al., 2023; Bai et al., 2023; Yang et al., 2024; Anthropic, 2024; OpenAI, 2024), code-specific language models have garnered significant attention in the community. Built upon pre-trained LLMs, code LLMs such as the StarCoder series (Li et al., 2023; Lozhkov et al., 2024), CodeLlama series (Roziere et al., 2023), DeepSeek-Coder series (Guo et al., 2024a), CodeQwen1.5 (Qwen, 2024), and CodeStral (MistralAI, 2024), have demonstrated superior performance in coding evaluations (Chen et al., 2021; Austin et al., 2021; Cassano et al., 2022; Jain et al., 2024; Liu et al., 2024a; Li et al., 2024b; Guo et al., 2024b; Wu et al., 2024b). However, in comparison with the recently state-of-the-art proprietary LLMs, Claude-3.5-Sonnet (Anthropic, 2024) and GPT-4o (OpenAI, 2024), the code LLMs are still falling behind, either open-source or proprietary models.

Building upon our previous work, CodeQwen1.5, we are excited to introduce **Qwen2.5-Coder**, a new series of language models designed to achieve top-tier performance in coding tasks at various model sizes. Qwen2.5-Coder models are derived from the Qwen2.5 LLMs, inheriting their advanced architecture and tokenizer. These models are trained on extensive datasets and further fine-tuned on carefully curated instruction datasets specifically designed for coding tasks. We are committed to fostering research and innovation in the field of code LLMs, coding agents, and coding assistant applications. Therefore, we release the *Powerful*, *Diverse*, and *Practical* Qwen2.5-Coder series, dedicated to continuously promoting the development of Open CodeLLMs. (1) *Powerful*: Qwen2.5-Coder-32B-Instruct has become the current SOTA open-source code model, matching the coding capabilities of GPT-4o. While demonstrating strong and comprehensive coding abilities, it also possesses good general and mathematical skills. (2) *Diverse*: Qwen2.5-Coder series brings six model sizes, including 0.5B/1.5B/3B/7B/14B/32B. Qwen2.5-Coder has covered six mainstream model sizes to meet the needs of different developers. (3) *Practical*: We explore the practicality of Qwen2.5-Coder in two scenarios, including code assistants and Artifacts, with some examples showcasing the potential applications of Qwen2.5-Coder in real-world scenarios

Significant efforts have been dedicated to constructing a large-scale, coding-specific pretraining dataset comprising over 5.5 trillion tokens. This dataset is sourced from a broad range of public code repositories, such as those on GitHub, as well as large-scale web-crawled data containing code-related texts. We have implemented sophisticated procedures to recall and clean potential code data and filter out low-quality content using weak model based classifiers and scorers. Our approach encompasses both file-level and repository-level pretraining to ensure comprehensive coverage. To optimize performance and balance coding expertise with general language understanding, we have carefully curated a data mixture that includes code, mathematics, and general texts. To transform models into coding assistants for downstream applications, we have developed a well-designed instruction-tuning dataset. This dataset includes a wide range of coding-related problems and solutions, sourced from real-world applications and synthetic data generated by code-focused LLMs, covering a broad spectrum of coding tasks.

To evaluate the effectiveness of Qwen2.5-Coder, we conducted an extensive evaluation on a suite of popular benchmarks. The results highlight Qwen2.5-Coder's superior code generation capabilities, achieving state-of-the-art performance across more than ten code-focused benchmarks while maintaining robust general and mathematical reasoning abilities. This model outperforms larger code models on a variety of tasks. The release of these models aims to advance code intelligence research and promote widespread adoption in real-world applications, facilitated by permissive licensing.

# 2 Model Architecture

**Architecture** The architecture of Qwen2.5-Coder is derived directly from Qwen2.5. Table 1 outlines the architecture of Qwen2.5-Coder across six different model sizes: 0.5B, 1.5B, 3B, 7B, 14B, and 32B parameters. While all sizes share the same architecture in terms of head size, they differ in several other key aspects. With exceptions like the 1.5B model having a larger intermediate size and the 3B model having more layers, most parameters generally

increase as the model size scales up. Comparing the 7B and 32B models for instance: the 7B model features a hidden size of 3,584, whereas the 32B model has a hidden size of 5,120. The 7B model uses 28 query heads and 4 key-value heads, while the 32B model uses 40 query heads and 8 key-value heads, reflecting its enhanced capacity. Similarly, the intermediate size scales with model size, being 18,944 for the 7B model and 27,648 for the 32B model. Additionally, smaller models use embedding tying, while larger models do not. Both models have a vocabulary size of 151,646 tokens and are trained on 5.5 trillion tokens.

**Tokenization**  Qwen2.5-Coder inherits the vocabulary from Qwen2.5 but introduces several special tokens to help the model better understand code. Table 2 presents an overview of the special tokens added during training to better capture different forms of code data. These tokens serve specific purposes in the code-processing pipeline. For instance, <|endoftext|> marks the end of a text or sequence, while the <|fim_prefix|>, <|fim_middle|>, and <|fim_suffix|> tokens are used to implement the Fill-in-the-Middle (FIM) (Bavarian et al., 2022) technique, where a model predicts the missing parts of a code block. Additionally, <|fim_pad|> is used for padding during FIM operations. Other tokens include <|repo_name|>, which identifies repository names, and <|file_sep|>, used as a file separator to better manage repository-level information. These tokens are essential in helping the model learn from diverse code structures and enable it to handle longer and more complex contexts during both file-level and repo-level pretraining.

| Configuration | 0.5B | 1.5B | 3B | 7B | 14B | 32B |
|---|---|---|---|---|---|---|
| Hidden Size | 896 | 1,536 | 2048 | 3,584 | 5120 | 5120 |
| # Layers | 24 | 28 | 36 | 28 | 48 | 64 |
| # Query Heads | 14 | 12 | 16 | 28 | 40 | 40 |
| # KV Heads | 2 | 2 | 2 | 4 | 8 | 8 |
| Head Size | 128 | 128 | 128 | 128 | 128 | 128 |
| Intermediate Size | 4,864 | 8,960 | 4,864 | 18,944 | 13824 | 27648 |
| Embedding Tying | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Vocabulary Size | 151,646 | 151,646 | 151,646 | 151,646 | 151,646 | 151,646 |
| # Trained Tokens | 5.5T | 5.5T | 5.5T | 5.5T | 5.5T | 5.5T |

Table 1: Architecture of Qwen2.5-Coder.

| Token | Token ID | Description |
|---|---|---|
| <|endoftext|> | 151643 | end of text/sequence |
| <|fim_prefix|> | 151659 | FIM prefix |
| <|fim_middle|> | 151660 | FIM middle |
| <|fim_suffix|> | 151661 | FIM suffix |
| <|fim_pad|> | 151662 | FIM pad |
| <|repo_name|> | 151663 | repository name |
| <|file_sep|> | 151664 | file separator |

Table 2: Overview of the special tokens.

## 3 Pre-training

### 3.1 Pretraining Data

Large-scale, high-quality, and diverse data forms the foundation of pre-trained models. To this end, we constructed a dataset named Qwen2.5-Coder-Data. This dataset comprises five key data types: Source Code Data, Text-Code Grounding Data, Synthetic Data, Math Data and Text Data. In this section, we provide a brief overview of the sources and cleaning methods applied to these datasets.

### 3.1.1 Data Composition

**Source Code**  We collected public repositories from GitHub created before February 2024, spanning 92 programming languages. Similar to StarCoder2 (Lozhkov et al., 2024) and DS-Coder (Guo et al., 2024a), we applied a series of rule-based filtering methods. In addition to raw code, we also collected data from Pull Requests, Commits, Jupyter Notebooks, and Kaggle datasets, all of which were subjected to similar rule-based cleaning techniques.
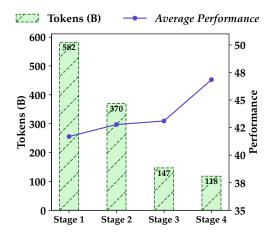


Figure 1: Number of data tokens across different cc-stages, and the validation effectiveness of training Qwen2.5-Coder using corresponding data.

**Text-Code Grounding Data**  We curated a large-scale and high-quality text-code mixed dataset from Common Crawl, which includes code-related documentation, tutorials, blogs, and more. Instead of the conventional URL-based multi-stage recall method, we developed a coarse-to-fine hierarchical filtering approach for raw data. This method offers two key advantages:

1. It enables precise control over each filter's responsibility, ensuring comprehensive handling of each dimension.
2. It naturally assigns quality scores to the dataset, with data retained in the final stage being of higher quality, providing valuable insights for quality-driven data mixing.

We designed a cleaning pipeline for the Text-Code Grounding Data, where each filter level is built using smaller models, such as fastText. Although we experimented with larger models, they did not yield significant benefits. A likely explanation is that smaller models focus more on surface-level features, avoiding unnecessary semantic complexity.

In Qwen2.5-Coder, we applied this process iteratively. As shown in Figure 1, each iteration resulted in improvement for Qwen2.5-Coder-1.5B. Through 4-stage filtering, the average scores on HumanEval and MBPP increased from 41.6% to 46.8% compared to the baseline, demonstrating the value of high-quality Text-Code Grounding Data for code generation.

**Synthetic Data**  Synthetic data offers a promising way to address the anticipated scarcity of training data. We used CodeQwen1.5, the predecessor of Qwen2.5-Coder, to generate large-scale synthetic datasets. To mitigate the risk of hallucinations during this process, we introduced an executor for validation, ensuring that only executable code was retained.

**Math Data**  To enhance the mathematical capabilities of Qwen2.5-Coder, we integrated the pre-training corpus from Qwen2.5-Math into the Qwen2.5-Coder dataset. Importantly, the inclusion of mathematical data did not negatively impact the model's performance on code tasks. For further details on the collection and cleaning process, please refer to the Qwen2.5-Math technical report.

**Text Data** Similar to the Math Data, we included high-quality general natural language data from the pre-training corpus of the Qwen2.5 model to preserve Qwen2.5-Coder's general capabilities. This data had already passed stringent quality checks during the cleaning phase of Qwen2.5's dataset, so no further processing was applied. However, all code segments were removed from the general Text data to avoid overlap with our code data, ensuring the independence of different data sources.

### 3.1.2 Data Mixture

Balancing Code, Math, and Text data is crucial for building a foundational model. Although the research community has explored this balance before, there is limited evidence regarding its scalability to large datasets. To address this, we conducted empirical experiments with different ratios of Code, Math, and Text data, designing multiple experiments to identify an optimal combination rapidly. Specifically, as shown in Table 3, we compared three different Code for Qwen2.5-Coder-7B: Text ratios — 100:0:0, 85:10:5, and 70:20:10.

Interestingly, we found that the 7:2:1 ratio outperformed the others, even surpassing the performance of groups with a higher proportion of code. A possible explanation is that Math and Text data may positively contribute to code performance, but only when their concentration reaches a specific threshold. In future work, we plan to explore more efficient ratio mechanisms and investigate the underlying causes of this phenomenon. Ultimately, we selected a final mixture of 70% Code, 20% Text, and 10% Math. The final training dataset comprises 5.2 trillion tokens.

| Token Ratio | | | Coding | | Math | | General | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|
| Code | Text | Math | Common | BCB | MATH | GSM8K | MMLU | CEval | HellaSwag | |
| 100 | 0 | 0 | **49.8** | **40.3** | 10.3 | 23.8 | 42.8 | 35.9 | 58.3 | 31.3 |
| 85 | 15 | 5 | 43.3 | 36.2 | 26.1 | 52.5 | 56.8 | 57.1 | 70.0 | 48.9 |
| 70 | 20 | 10 | 48.3 | 38.3 | **33.2** | **64.5** | **62.9** | **64.0** | **73.5** | **55.0** |

Table 3: The performance of Qwen2.5-Coder training on different data mixture policy.

## 3.2 Training Policy



Figure 2: The three-stage training pipeline for Qwen2.5-Coder.

As shown in 2, we employed a three-stage training approach to train Qwen2.5-Coder, including file-level pretraining, repo-level pretraining, and instruction tuning.

### 3.2.1 File-Level Pretraining

File-level pretraining focuses on learning from individual code files. In this stage, the maximum training sequence length is set to 8,192 tokens, covering 5.2T of high-quality data. The training objectives include next token prediction and fill-in-the-middle (FIM) (Bavarian et al., 2022). The specific FIM format is shown in Figure 3.
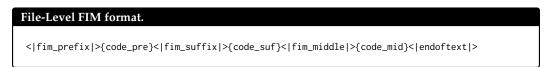
```
File-Level FIM format.

<|fim_prefix|>{code_pre}<|fim_suffix|>{code_suf}<|fim_middle|>{code_mid}<|endoftext|>
```

Figure 3: File-Level FIM format.

### 3.2.2 Repo-Level Pretraining

After file-level pretraining, we turn to repo-level pretraining, aimed at enhancing the model's long-context capabilities. In this stage, the context length is extended from 8,192 tokens to 32,768 tokens, and RoPE's base frequency is adjusted from 10,000 to 1,000,000. To further leverage the model's extrapolation potential, we applied the YARN mechanism (Peng et al., 2023), enabling the model to handle sequences up to 131,072 (128K) tokens.

In this stage, we used a large amount of high-quality, long-context code data ($\approx$ 300B) and extended file-level FIM to the repo-level FIM followed by methods described in Lozhkov et al. (2024), with the specific format shown in Figure 4.

---

**Repo-Level FIM format.**

```
<|repo_name|>{repo_name}
<|file_sep|>{file_path1}
{file_content1}
<|file_sep|>{file_path2}
{file_content2}
<|file_sep|>{file_path3}
<|fim_prefix|>{code_pre}<|fim_suffix|>{code_suf}<|fim_middle|>{code_fim}<|endoftext|>
```

Figure 4: Repo-Level FIM format.

## 4   Post-training

### 4.1   A Recipe for Instruction Data

**Multilingual Programming Code Identification**   We fine-tune a CodeBERT (Feng et al., 2020) to perform the language identification model to categorize documents into nearly 100 programming languages. We keep the instruction data of the mainstream programming languages and randomly discard a portion of the instruction data of the long-tail languages. If a given sample contains very little code data or even no code snippets, the sample will possibly be classified into "No Programming Language" tag. Since too many instruction samples without code snippets hurt the model performance on code generation tasks (e.g. MultiPL-E, McEval, and MdEval), we remove most of the samples without code snippets to keep the code generation capability of our instruction model.

**Instruction Synthesis from GitHub**   For the unsupervised data (code snippets) massively existing in many websites (e.g. GitHub), we try to construct the supervised instruction dataset using LLM. Specifically, we use the LLM to generate the instruction from the code snippets within 1024 tokens and then we use the code LLM to generate the response (Wei et al., 2024; Sun et al., 2024; Yu et al., 2024). Finally, we use the LLM scorer to filter the low-quality ones to obtain the final pair. Given the code snippets of different programming languages, we construct an instruction dataset from the code snippets. To fully unleash the potential of our proposed method, we also include the open-source instruction dataset (e.g. McEval-Instruct for massively multilingual code generation and debugging[1]) in the seed instruction dataset. Finally, we combine the instruction data from the GitHub code snippet and open-source instructions for supervised fine-tuning.

**Multilingual Code Instruction Data**   To bridge the gap among different programming languages, we propose a multilingual multi-agent collaborative framework to synthesize the multilingual instruction corpora. We introduce language-specific agents, where a set of

---

[1]https://huggingface.co/datasets/Multilingual-Multimodal-NLP/McEval-Instruct

specialized agents are created and each dedicated to a particular programming language. These agents are initialized with language-specific instruction data derived from the limited existing multilingual instruction corpora. The multilingual data generation process can be split into: (1) Language-Specific Intelligent Agents: We create a set of specialized agents, each dedicated to a particular programming language. These agents are initialized with language-specific instruction data derived from curated code snippets. (2) Collaborative Discussion Protocol: Multiple language-specific agents engage in a structured dialogue to formulate new instructions and solutions. This process can result in either enhancing existing language capabilities or generating instructions for a novel programming language. (3) Adaptive Memory System: Each agent maintains a dynamic memory bank that stores its generation history to avoid generating the similar samples. (4) Cross-Lingual Discussion: We implement a novel knowledge distillation technique that allows agents to share insights and patterns across language boundaries, fostering a more comprehensive understanding of programming concepts. (5) Synergy Evaluation Metric: We develop a new metric to quantify the degree of knowledge sharing and synergy between different programming languages within the model. (6) Adaptive Instruction Generation: The framework includes a mechanism to dynamically generate new instructions based on identified knowledge gaps across languages.

**Checklist-based Scoring for Instruction Data**    To completely evaluate the quality of the created instruction pair, we introduce several scoring points for each sample: (1) Question&Answer Consistency: Whether Q&A are consistent and correct for fine-tuning. (2) Question&Answer Relevance: Whether Q&A are related to the computer field. (3) Question&Answer Difficulty: Whether Q&A are sufficiently challenging. (4) Code Exist: Whether the code is provided in question or answer. (5) Code Correctness: Evaluate whether the provided code is free from syntax errors and logical flaws. (6) Consider factors like proper variable naming, code indentation, and adherence to best practices. (7) 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. (8) Code Comments: Evaluate the presence of comments and their usefulness in explaining the code's functionality. (9) Easy to Learn: determine its educational value for a student whose goal is to learn basic coding concepts. After gaining all scores $(s_1, \ldots, s_n)$, we can get the final score with $s = w_1 s_1 + \cdots + w_n s_n$, where $(w_1, \ldots, w_n)$ are a series of pre-defined weights.

**A multilingual sandbox for code verification**    To further verify the correctness of the code syntax, we use the code static checking for all extracted code snippets of programming languages (e.g. Python, Java, and C++). We parse the code snippet into the abstract syntax tree and filter out the code snippet, where the parsed nodes in code snippet have parsing errors. We create a multilingual sandbox to support the code static checking for the main programming language. Further, the multilingual sandbox is a comprehensive platform designed to validate code snippets across multiple programming languages. It automates the process of generating relevant unit tests based on language-specific samples and evaluates whether the provided code snippets can successfully pass these tests. Especially, only the self-contained (e.g. algorithm problems) code snippet will be fed into the multilingual sandbox. The multilingual verification sandbox is mainly comprised of five parts:

1. **Language Support Module:**
   - Implements support for multiple languages (e.g., Python, Java, C++, JavaScript)
   - Maintains language-specific parsing and execution environments
   - Handles syntax and semantic analysis for each supported language

2. **Sample Code Repository:**
   - Stores a diverse collection of code samples for each supported language
   - Organizes samples by language, difficulty level, and programming concepts
   - Regularly updated and curated by language experts

3. **Unit Test Generator:**
   - Analyzes sample code to identify key functionalities and edge cases

- Automatically generates unit tests based on the expected behavior
- Produces test cases covering various input scenarios and expected outputs

4. **Code Execution Engine:**

- Provides isolated environments for executing code snippets securely
- Supports parallel execution of multiple test cases
- Handles resource allocation and timeout mechanisms

5. **Result Analyzer:**

- Compares the output of code snippets against expected results from unit tests
- Generates detailed reports on test case successes and failures
- Provides suggestions for improvements based on failed test cases

## 4.2 Training Policy

**Coarse-to-fine Fine-tuning**    We first synthesized tens of millions of low-quality but diverse instruction samples to fine-tune the base model. In the second stage, we adopt millions of high-quality instruction samples to improve the performance of the instruction model with rejection sampling and supervised fine-tuning. For the same query, we use the LLM to generate multiple candidates and then use the LLM to score the best one for supervised fine-tuning.

**Mixed Tuning**    Since most instruction data have a short length, we construct the instruction pair with the FIM format to keep the long context capability of the base model. Inspired by programming language syntax rules and user habits in practical scenarios, we leverage the `tree-sitter-languages`[2] to parse the code snippets and extract the basic logic blocks as the middle code to infill. For example, the abstract syntax tree (AST) represents the structure of Python code in a tree format, where each node in the tree represents a construct occurring in the source code. The tree's hierarchical nature reflects the syntactic nesting of constructs in the code and includes various elements such as expressions, statements, and functions. By traversing and manipulating the AST, we can randomly extract the nodes of multiple levels and use the code context of the same file to uncover the masked node. Finally, we optimize the instruction model with a majority of standard SFT data and a small part of FIM instruction samples.

**Direct Preference Optimization for Code**    After obtaining the SFT model, we further align the Qwen2.5-Coder with the help of offline direct preference optimization (DPO) (Rafailov et al., 2023). Given that human feedback is highly labor-intensive, we use a multilingual code sandbox to provide code execution feedback, while an LLM is utilized for human judgment feedback. For the algorithm-like and self-contained code snippets, we generate the test cases to check the correctness of the code as the code execution feedback, including Python, Java, and other languages. For other complex code snippets, we use LLM-as-a-judge (Zheng et al., 2023) to decide which code snippet is better. Further, we combine the code DPO data and common data for offline DPO training.

## 5  Decontamination

To ensure that Qwen2.5-Coder does not produce inflated results due to test set leakage, we performed decontamination on all data, including both pre-training and post-training datasets. We removed key datasets such as HumanEval, MBPP, GSM8K, and MATH. The filtering was done using a 10-gram overlap method, where any training data with a 10-gram word-level overlap with the test data was removed.

---

[2] https://pypi.org/project/tree-sitter-languages/

# 6 Evaluation on Base Models

For the base model, we conducted a comprehensive and fair evaluation in six key aspects, including code generation, code completion, code reasoning, mathematical reasoning, general natural language understanding and long-context modeling. To ensure the reproducibility of all results, we made all evaluation codes publicly available[3]. For comparing models, we chose the most popular and powerful open source language models, including the StarCoder2 and DeepSeek-Coder series. Below is the list of artifacts used in the evaluation for this section.

| Artifact | Public link |
| --- | --- |
| Qwen2.5-Coder-0.5B | https://hf.co/Qwen/Qwen2.5-Coder-0.5B |
| Qwen2.5-Coder-1.5B | https://hf.co/Qwen/Qwen2.5-Coder-1.5B |
| Qwen2.5-Coder-3B | https://hf.co/Qwen/Qwen2.5-Coder-3B |
| Qwen2.5-Coder-7B | https://hf.co/Qwen/Qwen2.5-Coder-7B |
| Qwen2.5-Coder-14B | https://hf.co/Qwen/Qwen2.5-Coder-14B |
| Qwen2.5-Coder-32B | https://hf.co/Qwen/Qwen2.5-Coder-32B |
| CodeQwen1.5-7B | https://hf.co/Qwen/CodeQwen1.5-7B |
| StarCoder2-3B | https://hf.co/bigcode/starcoder2-3b |
| StarCoder2-7B | https://hf.co/bigcode/starcoder2-7b |
| StarCoder2-15B | https://hf.co/bigcode/starcoder2-15b |
| DS-Coder-1.3B-Base | https://hf.co/deepseek-ai/deepseek-coder-1.3b-base |
| DS-Coder-6.7B-Base | https://hf.co/deepseek-ai/deepseek-coder-6.7b-base |
| DS-Coder-33B-Base | https://hf.co/deepseek-ai/deepseek-coder-33b-base |
| DS-Coder-V2-Lite-Base | https://hf.co/deepseek-ai/DeepSeek-Coder-V2-Lite-Base |
| DS-Coder-V2-Base | https://hf.co/deepseek-ai/DeepSeek-Coder-V2-Base |

Table 4: All artifacts released and used in this section.

## 6.1 Code Generation

**HumanEval and MBPP** Code generation serves as a fundamental capability for code models to handle more complex tasks. We selected two popular code generation benchmarks to evaluate Qwen2.5-Coder, namely HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). HumanEval consists of 164 manually written programming tasks, each providing a Python function signature and a docstring as input to the model. MBPP, on the other hand, comprises 974 programming problems created by crowdsource contributors. Each problem includes a problem statement (i.e., a docstring), a function signature, and three test cases.

To further ensure accurate evaluation, EvalPlus (Liu et al., 2023) extends HumanEval into HumanEval+ by adding 80 times more unique test cases and correcting inaccurate ground-truth solutions in HumanEval. Similarly, MBPP+ offers 35 times more test cases than the original MBPP.

Additionally, we should notice that MBPP 3-shot is particularly suitable for monitoring model convergence during training. Early in the convergence process, the model tends to be unstable, causing significant fluctuation in metrics, and simple 3-shot examples effectively mitigate it. Therefore, we also report the results of MBPP 3-shot performance.

As shown in Table 5, Qwen2.5-Coder have shown impressive performance in basic code generation, achieving state-of-the-art results among open-source models of the same size and surpassing even larger models. In particular, Qwen2.5-Coder-7B outperforms the previous best dense model, DS-Coder-33B, across all five metrics.

**BigCodeBench-Complete** BigCodeBench (Zhuo et al., 2024) is a recent and more challenging benchmark for code generation, primarily aimed at evaluating the ability of tool-use and complex instruction following. The base model generates the expected code through a

---

[3]https://github.com/QwenLM/Qwen2.5-Coder

| Model | Size | HumanEval | | MBPP | | | BigCodeBench | |
|---|---|---|---|---|---|---|---|---|
| | | HE | HE+ | MBPP | MBPP+ | 3-shot | Full | Hard |
| **0.5B+ Models** | | | | | | | | |
| **Qwen2.5-Coder-0.5B** | 0.5B | **28.0** | **23.8** | **52.9** | **47.1** | **40.4** | **16.1** | **4.7** |
| **1B+ Models** | | | | | | | | |
| DS-Coder-1.3B | 1.3B | 34.8 | 26.8 | 55.6 | 46.9 | 46.2 | 26.1 | 3.4 |
| **Qwen2.5-Coder-1.5B** | 1.5B | **43.9** | **36.6** | **69.2** | **58.6** | **59.2** | **34.6** | **9.5** |
| **3B+ Models** | | | | | | | | |
| StarCoder2-3B | 3B | 31.7 | 27.4 | 60.2 | 49.1 | 47.4 | 21.4 | 4.7 |
| **Qwen2.5-Coder-3B** | 3B | **52.4** | **42.7** | **72.2** | **61.4** | **65.2** | **41.1** | **11.5** |
| **6B+ Models** | | | | | | | | |
| StarCoder2-7B | 7B | 35.4 | 29.9 | 54.4 | 45.6 | 51.8 | 27.7 | 8.8 |
| DS-Coder-6.7B-Base | 6.7B | 47.6 | 39.6 | 70.2 | 56.6 | 60.6 | 41.1 | 11.5 |
| DS-Coder-V2-Lite-Base | 2.4/16B | 40.9 | 34.1 | 71.9 | 59.4 | 62.6 | 30.6 | 8.1 |
| CodeQwen1.5-7B | 7B | 51.8 | 45.7 | 72.2 | 60.2 | 61.8 | 45.6 | 15.5 |
| **Qwen2.5-Coder-7B** | 7B | **61.6** | **53.0** | **76.9** | **62.9** | **68.8** | **45.8** | **16.2** |
| **14B+ Models** | | | | | | | | |
| StarCoder2-15B | 15B | 46.3 | 37.8 | 66.2 | 53.1 | 57.0 | 38.4 | 12.2 |
| **Qwen2.5-Coder-14B** | 14B | **64.0** | **57.9** | **81.0** | **66.7** | **71.4** | **51.8** | **22.3** |
| **20B+ Models** | | | | | | | | |
| DS-Coder-33B-Base | 33B | 54.9 | 47.6 | 74.2 | 60.7 | 66.0 | 49.1 | 20.3 |
| DS-Coder-V2-Base | 21/236B | 50.0 | 43.3 | 82.5 | 65.7 | 71.2 | 48.7 | 21.6 |
| **Qwen2.5-Coder-32B** | 32B | **65.9** | **60.4** | **83.0** | **68.2** | **76.4** | **53.6** | **26.4** |

Table 5: Performance of various models on HumanEval, MBPP and the "complete" task of BigCodeBench.

completion mode, given a function signature and documentation, which is referred to as BigCodeBench-Complete. It consists of two subsets: the full set and the hard set. Compared to HumanEval and MBPP, BigCodeBench is suited for out-of-distribution (OOD) evaluation.

Table 5 illustrates that Qwen2.5-Coder continues to show strong performance on BigCodeBench-Complete, underscoring the model's generalization potential.

**Multi-Programming Language**   The evaluations mentioned above focus on the Python language. However, we expect a strong code model to be not only proficient in Python but also versatile across multiple programming languages to meet the complex and evolving demands of software development. To more comprehensively evaluate Qwen2.5-Coder's proficiency in handling multiple programming languages, we selected the MultiPL-E (Cassano et al., 2022) and chose to evaluate eight mainstream languages from this benchmark, including Python, C++, Java, PHP, TypeScript, C#, Bash and JavaScript.

As shown in the table 6, Qwen2.5-Coder also achieved state-of-the-art results in the multi-programming language evaluation, with its capabilities well-balanced across various languages. It scored over 60% in five out of the eight languages.

## 6.2   Code Completion

Many developer aid tools rely on the capability to autocomplete code based on preceding and succeeding code snippets. Qwen2.5-Coder utilizes the Fill-In-the-Middle (FIM) training strategy, as introduced in Bavarian et al. (2022), enabling the model to generate code that is contextually coherent. To assess its code completion proficiency, we utilize the HumanEval-FIM benchmark (Allal et al., 2023), CrossCodeEval (Ding et al., 2024), Cross-CodeLongEval (Wu et al., 2024a), RepoEval (Zhang et al., 2023) and SAFIM (Gong et al.,

| Model | Size | Python | C++ | Java | PHP | TS | C# | Bash | JS | Average |
|---|---|---|---|---|---|---|---|---|---|---|
| **0.5B+ Models** | | | | | | | | | | |
| **Qwen2.5-Coder-0.5B** | 0.5B | **28.0** | **25.5** | **22.8** | **23.6** | **30.8** | **31.0** | **7.0** | **29.2** | **24.7** |
| **1B+ Models** | | | | | | | | | | |
| DS-Coder-1.3B-Base | 1.3B | 34.8 | 31.1 | 32.3 | 24.2 | 28.9 | 36.7 | 10.1 | 28.6 | 28.3 |
| **Qwen2.5-Coder-1.5B** | 1.5B | **42.1** | **42.9** | **38.6** | **41.0** | **49.1** | **46.2** | **20.3** | **49.1** | **41.1** |
| **3B+ Models** | | | | | | | | | | |
| StarCoder2-3B | 3B | 31.7 | 30.4 | 29.8 | 32.9 | 39.6 | 34.8 | 13.9 | 35.4 | 31.1 |
| **Qwen2.5-Coder-3B** | 3B | **52.4** | **52.8** | **44.9** | **49.1** | **55.4** | **51.3** | **24.7** | **53.4** | **48.0** |
| **6B+ Models** | | | | | | | | | | |
| StarCoder2-7B | 7B | 35.4 | 40.4 | 38.0 | 30.4 | 34.0 | 46.2 | 13.9 | 36.0 | 34.3 |
| DS-Coder-6.7B-Base | 6.7B | 49.4 | 50.3 | 43.0 | 38.5 | 49.7 | 50.0 | 28.5 | 48.4 | 44.7 |
| DS-Coder-V2-Lite-Base | 2.4/16B | 40.9 | 45.9 | 34.8 | 47.2 | 48.4 | 41.7 | 19.6 | 44.7 | 40.4 |
| CodeQwen1.5-7B | 7B | 51.8 | 52.2 | 42.4 | 46.6 | 52.2 | 55.7 | 36.7 | 49.7 | 48.4 |
| **Qwen2.5-Coder-7B** | 7B | **61.6** | **62.1** | **53.2** | **59.0** | **64.2** | **60.8** | **38.6** | **60.3** | **57.5** |
| **14B+ Models** | | | | | | | | | | |
| StarCoder2-15B | 15B | 46.3 | 47.2 | 46.2 | 39.1 | 42.1 | 53.2 | 15.8 | 43.5 | 41.7 |
| **Qwen2.5-Coder-14B** | 14B | **64.0** | **69.6** | **46.8** | **64.6** | **69.2** | **63.3** | **39.9** | **61.5** | **59.9** |
| **20B+ Models** | | | | | | | | | | |
| DS-Coder-33B-Base | 33B | 56.1 | 58.4 | 51.9 | 44.1 | 52.8 | 51.3 | 32.3 | 55.3 | 50.3 |
| DS-Coder-V2-Base | 21/236B | 50.0 | 59.6 | 50.0 | 55.3 | 58.5 | 45.6 | 36.1 | 59.6 | 51.8 |
| **Qwen2.5-Coder-32B** | 32B | **65.9** | **68.3** | **70.9** | **64.6** | **66.0** | **68.4** | **39.9** | **67.1** | **63.9** |

Table 6: Performance of different models on MultiPL-E.

2024). Figure 5 shows the overall evaluation results of Qwen2.5-Coder-32B on different code completion benchmarks.



Figure 5: The code completion performance of competitive models on five benchmarks, Humaneval-FIM, SAFIM, CrossCodeEval, RepoEval, CrossCodeLongEval.

Humaneval-FIM benchmark challenges the model to accurately predict missing sections of code within tasks derived from Humaneval. We use the single-line infilling settings across Python, Java, and JavaScript, focusing on predicting a single line of code within given contexts. Performance was measured using the Exact Match metric, which determines the proportion of the first generated code line that precisely match the ground truth. The table 7 illustrates that Qwen2.5-Coder surpasses alternative models concerning model size. Specifically, Qwen2.5-Coder-1.5B achieves an average performance improvement of 3.7%, rivaling the majority of models exceeding 6 billion parameters. Moreover, Qwen2.5-Coder-7B stands as the leading model among those over 6 billion parameters, matching the performance of the formidable 33 billion parameter model, DS-Coder-33B-Base. Notably, we excluded DS-Coder-v2-236B from comparison due to its design focus not being on code completion tasks.

| Model | Size | Humaneval-FIM | | | |
|---|---|---|---|---|---|
| | | *Python* | *Java* | *JavaScript* | *Average** |
| **0.5B+ Models** | | | | | |
| **Qwen2.5-Coder-0.5B** | 0.5B | **70.3** | **78.1** | **81.2** | **77.7** |
| **1B+ Models** | | | | | |
| DS-Coder-1.3B-Base | 1.3B | 72.8 | 84.3 | 81.7 | 80.7 |
| **Qwen2.5-Coder-1.5B** | 1.5B | **77.0** | **85.6** | **85.0** | **83.5** |
| **3B+ Models** | | | | | |
| StarCoder2-3B | 3B | 70.9 | 84.4 | 81.8 | 80.4 |
| **Qwen2.5-Coder-3B** | 3B | **78.7** | **88.0** | **87.4** | **85.7** |
| **6B+ Models** | | | | | |
| StarCoder2-7B | 7B | 70.8 | 86.0 | 84.4 | 82.0 |
| DS-Coder-6.7B-Base | 6.7B | 78.1 | 87.4 | 84.1 | 84.0 |
| DS-Coder-V2-Lite-Base | 2.4/16B | 78.7 | 87.8 | 85.9 | 85.0 |
| CodeQwen1.5-7B | 7B | 75.8 | 85.7 | 85.0 | 83.3 |
| **Qwen2.5-Coder-7B** | 7B | **79.7** | **88.5** | **87.6** | **86.2** |
| **14B+ Models** | | | | | |
| StarCoder2-15B | 15B | 74.2 | 85.2 | 84.6 | 82.6 |
| **Qwen2.5-Coder-14B** | 14B | **80.5** | **91.0** | **88.5** | **87.7** |
| **20B+ Models** | | | | | |
| CodeStral-22B | 22B | 76.7 | 82.5 | 86.0 | 82.7 |
| DS-Coder-33B-Base | 33B | 80.1 | 89.0 | 86.8 | 86.2 |
| **Qwen2.5-Coder-32B** | 32B | **81.5** | **91.0** | **89.4** | **88.3** |

Table 7: Performance of different approaches on the Humaneval-FIM Tasks. *$^*$Average* refers to a weighted mean calculated based on the number of samples for each language.

In real-world scenarios, code completion often depends on accessing cross-file context and dependencies. CrossCodeEval is a benchmark that requires a deep understanding of this cross-file context to accurately complete the code. In our evaluation, we set a maximum sequence length of 8192 tokens, designate a maximum output length of 50 tokens, and impose a limit of 2048 tokens for the cross-file context. For the cross-file context, we use the official BM25 search results provided by Ding et al. (2024). We evaluate performance using Exact Match (EM) and Edit Similarity (ES) metrics. Table 8 shows that the Qwen2.5-Coder-32B achieves state-of-the-art performance with a 3.7% improvement. Qwen2.5-Coder outperforms all the models with a comparable model size. Meanwhile, Qwen2.5-Coder-7B has a comparable performance with other models exceeding 20 billion parameters.

CrossCodeLongEval is a long context benchmark on cross file code completion tasks. In our evaluation, we set a maximum sequence length of 8192 tokens and set the maximum output as 256 tokens for function completion and 50 tokens for other tasks. The cross-file context is truncated to 2048 tokens. For the cross-file context, we use the official BM25 search results provided by Wu et al. (2024a). We evaluate performance using Exact Match (EM) and Edit Similarity (ES) metrics. Qwen2.5-Coder-32B achieves state-of-the-art performance, as detailed in Table 9. The Qwen2.5-Coder series surpasses all other models of a similar size. All models demonstrate low Exact Match (EM) results on function completion tasks, likely due to the complexity of generating multi-line code snippets that are challenging to match precisely.

RepoEval is a benchmark designed to evaluate repository-level code completion capabilities across three granularities: line, API invocation, and function body completion. In our evaluation, we set a maximum sequence length of 8192 tokens, set the maximum output as 256 tokens for function completion and 50 tokens for other tasks, and impose a limit of 2048

| Model | Python | | Java | | TypeScript | | C# | | Average | |
|---|---|---|---|---|---|---|---|---|---|---|
| | EM | ES | EM | ES | EM | ES | EM | ES | EM | ES |
| **0.5B+ Models** | | | | | | | | | | |
| **Qwen2.5-Coder-0.5B** | **22.7** | **66.2** | **21.7** | **66.8** | **21.9** | **67.2** | **32.1** | **75.4** | **24.6** | **68.9** |
| **1B+ Models** | | | | | | | | | | |
| DS-Coder-1.3B-Base | 33.4 | 72.6 | 34.9 | 74.5 | 36.7 | 76.4 | 46.6 | 83.5 | 37.9 | 76.8 |
| **Qwen2.5-Coder-1.5B** | **35.5** | **74.3** | **37.9** | **76.5** | **37.6** | **77.4** | **49.8** | **84.5** | **40.2** | **78.2** |
| **3B+ Models** | | | | | | | | | | |
| StarCoder2-3B | 11.0 | 62.7 | 11.6 | 69.7 | 8.8 | 75.8 | 8.2 | 71.2 | 9.9 | 69.8 |
| **Qwen2.5-Coder-3B** | **38.4** | **76.1** | **42.8** | **79.8** | **41.6** | **80.5** | **56.7** | **87.1** | **44.9** | **80.9** |
| **6B+ Models** | | | | | | | | | | |
| StarCoder2-7B | 10.9 | 63.1 | 8.3 | 71.0 | 6.7 | 76.8 | 7.3 | 72.1 | 8.3 | 70.8 |
| DS-Coder-6.7B-Base | 41.1 | 79.2 | 39.9 | 80.1 | 46.3 | 82.4 | 55.0 | 86.9 | 45.6 | 82.1 |
| DS-Coder-V2-Lite-Base | 41.8 | 78.3 | 46.1 | 81.2 | 44.6 | 81.4 | 58.7 | 87.9 | 47.8 | 82.2 |
| CodeQwen1.5-7B | 40.7 | 77.8 | 47.0 | 81.6 | 45.8 | 82.2 | 59.7 | 87.6 | 48.3 | 82.3 |
| **Qwen2.5-Coder-7B** | **42.4** | **78.6** | **48.1** | **82.6** | **46.8** | **83.4** | **59.7** | **87.9** | **49.3** | **83.1** |
| **14B+ Models** | | | | | | | | | | |
| StarCoder2-15B | 28.2 | 70.5 | 26.7 | 71.0 | 24.7 | 76.3 | 25.2 | 74.2 | 26.2 | 73.0 |
| **Qwen2.5-Coder-14B** | **47.7** | **81.7** | **54.7** | **85.7** | **52.9** | **86.0** | **66.4** | **91.1** | **55.4** | **86.1** |
| **20B+ Models** | | | | | | | | | | |
| CodeStral-22B | **49.3** | **82.7** | 44.1 | 71.1 | 51.0 | 85.0 | 53.7 | 83.6 | 49.5 | 80.6 |
| DS-Coder-33B-Base | 44.2 | 80.4 | 46.5 | 82.7 | 49.2 | 84.0 | 55.2 | 87.8 | 48.8 | 83.7 |
| **Qwen2.5-Coder-32B** | 49.2 | 82.1 | **56.4** | **86.6** | **54.9** | **87.0** | **68.0** | **91.6** | **57.1** | **86.8** |

Table 8: Performance of different approaches on the CrossCodeEval Tasks.

tokens for the cross-file context. Besides, we utilize the official sparse retriever (Lu et al., 2022) to extract the cross-file context. We evaluate performance using Exact Match (EM) and Edit Similarity (ES) metrics. As shown in Table 10, Qwen2.5-Coder-32B achieves state-of-the-art performance with an average improvement of 7.9% EM and 4.2% ES compared to DS-Coder-33B-Base. Furthermore, Qwen2.5-Coder-14B and Qwen2.5-Coder-7B achieve comparable performance to models with more than 20B parameters, while maintaining state-of-the-art results among models of similar size.

SAFIM is a syntax-aware fill-in-the-middle benchmark that emphasizes AST-based code completion, specifically targeting algorithmic blocks, control-flow expressions, and API function calls. The benchmark consists of 17,720 examples from 8,590 code files created after April 2022, deliberately avoiding overlap with mainstream pretraining corpora. For evaluation, we use pass@1 rate as the metric for algorithmic and control-flow tasks, and Exact Match (EM) for API completion tasks.

## 6.3 Code Reasoning

Code is a highly abstract form of logical language, and reasoning based on code helps us determine whether a model truly understands the reasoning flow behind the code. We selected CRUXEval (Gu et al., 2024) as the benchmark, which includes 800 Python functions along with corresponding input-output examples. It consists of two distinct tasks: CRUXEval-I, where the large language model (LLM) must predict the output based on a given input; and CRUXEval-O, where the model must predict the input based on a known output. For both CRUXEval-I and CRUXEval-O, we used a chain-of-thought (CoT) approach, requiring the LLM to output steps sequentially during simulated execution.

| Model | Chunk Completion | | Function completion | | Average | |
|---|---|---|---|---|---|---|
| | *EM* | *ES* | *EM* | *ES* | *EM* | *ES* |
| **0.5B+ Models** | | | | | | |
| **Qwen2.5-Coder-0.5B** | **29.8** | **64.2** | **9.5** | **38.0** | **19.7** | **51.1** |
| **1B+ Models** | | | | | | |
| DS-Coder-1.3B-Base | 40.6 | 71.9 | 9.6 | 39.4 | 25.1 | 55.7 |
| **Qwen2.5-Coder-1.5B** | **44.2** | **73.9** | **12.4** | **44.4** | **28.3** | **59.2** |
| **3B+ Models** | | | | | | |
| StarCoder2-3B | 18.5 | 62.0 | 10.2 | 39.2 | 14.3 | 50.6 |
| **Qwen2.5-Coder-3B** | **46.6** | **76.1** | **13.5** | **46.4** | **30.0** | **61.3** |
| **6B+ Models** | | | | | | |
| StarCoder2-7B | 19.4 | 63.6 | 10.2 | 40.0 | 14.8 | 51.8 |
| DS-Coder-6.7B-Base | 48.4 | 78.2 | 10.7 | 42.4 | 29.6 | 60.3 |
| DS-Coder-V2-Lite-Base | 49.5 | 77.1 | 11.4 | 43.1 | 30.4 | 60.1 |
| CodeQwen1.5-7B | 48.2 | 77.5 | 6.4 | 30.6 | 27.3 | 54.1 |
| **Qwen2.5-Coder-7B** | **52.4** | **79.3** | **14.4** | **48.4** | **33.4** | **63.8** |
| **14B+ Models** | | | | | | |
| StarCoder2-15B | 21.3 | 53.7 | 7.8 | 30.5 | 14.6 | 42.1 |
| **Qwen2.5-Coder-14B** | **56.9** | **81.8** | **15.4** | **49.8** | **36.1** | **65.8** |
| **20B+ Models** | | | | | | |
| CodeStral-22B | 56.7 | 81.8 | 10.5 | 37.8 | 33.6 | 59.8 |
| DS-Coder-33B-Base | 52.0 | 79.9 | 11.9 | 44.3 | 32.0 | 62.1 |
| **Qwen2.5-Coder-32B** | **57.3** | **82.1** | **16.4** | **50.8** | **36.9** | **66.4** |

Table 9: Performance of different approaches on the CrossCodeLongEval Tasks.

As shown in Table 11, Qwen2.5-Coder delivered highly promising results, achieving a score of 56.5 on CRUXEval-I and 56.0 on CRUXEval-O, thanks to our focus on executable quality during the code cleaning process.

## 6.4 Math Reasoning

Mathematics and coding have always been closely intertwined. Mathematics forms the foundational discipline for coding, while coding serves as a vital tool in mathematical fields. As such, we expect an open and powerful code model to exhibit strong mathematical capabilities as well. To assess Qwen2.5-Coder's mathematical performance, we selected five popular benchmarks, including MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), MMLU-STEM (Hendrycks et al., 2020) and TheoremQA (Chen et al., 2023). Table 12 highlights Qwen2.5-Coder's strengths in mathematics, which likely stem from two key factors: first, the model's strong foundation built on Qwen2.5, and second, the careful mixing of code and mathematical data during training, which has ensured a well-balanced performance across these domains.

## 6.5 General Natural Language

In addition to mathematical ability, we aim to retain as much of the base model's general-purpose capabilities as possible, such as general knowledge. To evaluate general natural language understanding, we selected MMLU (Hendrycks et al., 2021) and its variant MMLU-Redux (Gema et al., 2024), along with four other benchmarks: ARC-Challenge (Clark et al., 2018), TruthfulQA (Lin et al., 2021), WinoGrande (Sakaguchi et al., 2019), and HellaSwag (Zellers et al., 2019). Similar to the results in mathematics, Table 14 highlights Qwen2.5-

| Model | Line | | Function | | API | | Average | |
|---|---|---|---|---|---|---|---|---|
| | *EM* | *ES* | *EM* | *ES* | *EM* | *ES* | *EM* | *ES* |
| **0.5B+ Models** | | | | | | | | |
| **Qwen2.5-Coder-0.5B** | **44.2** | **72.6** | **4.6** | **48.0** | **35.6** | **68.5** | **28.1** | **63.0** |
| **1B+ Models** | | | | | | | | |
| DS-Coder-1.3B-Base | 58.7 | 80.4 | 6.2 | 48.8 | 45.8 | 75.0 | 36.9 | 68.1 |
| **Qwen2.5-Coder-1.5B** | **59.8** | **82.6** | **10.6** | **52.4** | **51.0** | **80.1** | **40.5** | **71.7** |
| **3B+ Models** | | | | | | | | |
| StarCoder2-3B | 22.3 | 67.4 | 3.1 | 51.6 | 20.6 | 70.1 | 15.3 | 63.0 |
| **Qwen2.5-Coder-3B** | **64.9** | **85.0** | **12.3** | **55.8** | **54.7** | **81.3** | **44.0** | **74.0** |
| **6B+ Models** | | | | | | | | |
| StarCoder2-7B | 19.5 | 67.6 | 4.0 | 53.5 | 19.1 | 72.8 | 14.2 | 64.7 |
| DS-Coder-6.7B-Base | 63.1 | 85.5 | 9.9 | 53.3 | 52.3 | 81.7 | 41.7 | 73.5 |
| DS-Coder-V2-Lite-Base | 66.5 | 85.4 | 10.8 | 53.9 | 53.1 | 81.3 | 43.4 | 73.5 |
| CodeQwen1.5-7B | 59.7 | 81.5 | 4.8 | 44.3 | 46.1 | 77.5 | 36.9 | 67.8 |
| **Qwen2.5-Coder-7B** | **67.3** | **86.1** | **13.2** | **55.2** | **58.4** | **83.9** | **46.3** | **75.1** |
| **14B+ Models** | | | | | | | | |
| StarCoder2-15B | 30.9 | 62.5 | 5.5 | 43.7 | 21.7 | 60.3 | 19.4 | 55.5 |
| **Qwen2.5-Coder-14B** | **74.3** | **90.1** | **14.1** | **59.5** | **63.4** | **87.3** | **50.6** | **79.0** |
| **20B+ Models** | | | | | | | | |
| Codestral-22B-v0.1 | 40.9 | 51.7 | 9.9 | 49.2 | 24.8 | 40.8 | 30.0 | 46.6 |
| DS-Coder-33B-Base | 66.5 | 86.6 | 10.3 | 52.9 | 54.2 | 83.5 | 43.7 | 74.3 |
| **Qwen2.5-Coder-32B** | **76.1** | **90.5** | **13.6** | **57.5** | **65.1** | **87.6** | **51.6** | **78.5** |

Table 10: Performance of different approaches on the RepoEval Tasks.

Coder's advantage in general natural language capabilities compared to other coders, further validating the effectiveness of Qwen2.5-Coder data mixing strategy.

### 6.6 Long-Context Evaluation

Long context capability is crucial for code LLMs, serving as the core skill for understanding repository-level code and becoming a code agent. However, most of the current code models still have very limited support for length, which hinders their potential for practical application. Qwen2.5-Coder aims to further advance the progress of open-source code models in long context modeling. To achieve this, we have collected and constructed long sequence code data at the repository level for pre-training. Through careful data proportioning and organization, we have enabled it to support input lengths of up to 128K tokens.

**Needle in the Code**   We created a simple but basic synthetic task called *Needle in the Code*, inspired by popular long-context evaluations in the text domain. In this task, we inserted a very simple custom function at various positions within a code repo (we chose Megatron [4] to honor its contributions to open-source LLMs!) and tested whether the model could replicate this function at the end of the codebase. The figure below shows that Qwen2.5-Coder is capable of successfully completing this task within a 128k length range.

## 7   Evaluation on Instruct Models

For the evaluation of the instruct models, we rigorously assessed six core areas: *code generation*, *code reasoning*, *code editing*, *text-to-sql*, *mathematical reasoning* and *general natural language*

---

[4]https://github.com/NVIDIA/Megatron-LM

| Model | Size | CRUXEval | |
| --- | --- | --- | --- |
| | | *Input-CoT* | *Output-CoT* |
| **0.5B+ Models** | | | |
| **Qwen2.5-Coder-0.5B** | 0.5B | **35.2** | **23.0** |
| **1B+ Models** | | | |
| DS-Coder-1.3B-Base | 1.3B | 32.1 | 28.2 |
| **Qwen2.5-Coder-1.5B** | 1.5B | **43.8** | **34.6** |
| **3B+ Models** | | | |
| StarCoder2-3B | 3B | 42.1 | 29.2 |
| **Qwen2.5-Coder-3B** | 3B | **46.5** | **43.8** |
| **6B+ Models** | | | |
| StarCoder2-7B | 7B | 39.5 | 35.1 |
| DS-Coder-6.7B-Base | 6.7B | 39.0 | 41.0 |
| DS-Coder-V2-Lite-Base | 2.4/16B | 53.4 | 46.1 |
| CodeQwen1.5-7B | 7B | 44.8 | 40.1 |
| **Qwen2.5-Coder-7B** | 7B | **56.5** | **56.0** |
| **14B+ Models** | | | |
| StarCoder2-15B | 15B | 46.1 | 47.6 |
| **Qwen2.5-Coder-14B** | 14B | **60.6** | **66.4** |
| **20B+ Models** | | | |
| DS-Coder-33B-Base | 33B | 50.6 | 48.8 |
| DS-Coder-V2-Base | 21/236B | 62.7 | 67.4 |
| **Qwen2.5-Coder-32B** | 32B | **62.5** | **69.4** |

Table 11: Performance of different models on CRUXEval with *Input-CoT* and *Output-CoT* settings.
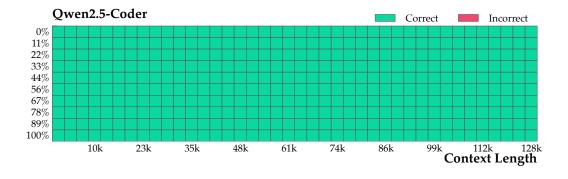


Figure 6: The long context ability of Qwen2.5-Coder, evaluated by Needle in the Code.

*understanding*. The evaluation was structured to ensure a fair and thorough comparison across models. All evaluation code is publicly accessible for reproducibility[5]. To ensure a broad comparison, we included some of the most popular and widely-used open-source instruction-tuned models, notably versions from the DeepSeek-Coder series and Codestral models. Below is a list of all artifacts referenced in this section.

---

[5] https://github.com/QwenLM/Qwen2.5-Coder

| Model | Size | MATH<br>*4-shot* | GSM8K<br>*4-shot* | MMLU STEM<br>*5-shot* | TheoremQA<br>*5-shot* |
|---|---|---|---|---|---|
| **0.5B+ Models** | | | | | |
| **Qwen2.5-Coder-0.5B** | 0.5B | **15.4** | **34.5** | **34.4** | **14.3** |
| **1B+ Models** | | | | | |
| DS-Coder-1.3B-Base | 1.3B | 4.6 | 4.4 | 24.5 | 8.9 |
| **Qwen2.5-Coder-1.5B** | 1.5B | **30.9** | **65.8** | **49.0** | **21.4** |
| **3B+ Models** | | | | | |
| StarCoder2-3B | 3B | 10.8 | 21.6 | 34.9 | 12.1 |
| **Qwen2.5-Coder-3B** | 3B | **40.0** | **75.7** | **56.0** | **29.5** |
| **6B+ Models** | | | | | |
| StarCoder2-7B | 7B | 14.6 | 32.7 | 39.8 | 16.0 |
| DS-Coder-6.7B-Base | 6.7B | 10.3 | 21.3 | 34.2 | 13.6 |
| DS-Coder-V2-Lite-Base | 2.4/16B | 39.0 | 67.1 | 58.5 | 29.3 |
| CodeQwen1.5-7B | 7B | 10.6 | 37.7 | 39.6 | 15.8 |
| **Qwen2.5-Coder-7B** | 7B | **46.6** | **83.9** | **67.6** | **34.0** |
| **14B+ Models** | | | | | |
| StarCoder2-15B | 15B | 23.7 | 57.7 | 49.2 | 20.5 |
| **Qwen2.5-Coder-14B** | 14B | **52.8** | **88.7** | **73.9** | **39.6** |
| **20B+ Models** | | | | | |
| DS-Coder-33B-Base | 33B | 14.4 | 35.4 | 39.5 | 17.5 |
| DS-Coder-V2-Base | 21/236B | 50.6 | 85.8 | **76.0** | 39.4 |
| **Qwen2.5-Coder-32B** | 32B | **57.2** | **91.1** | 75.1 | **43.1** |

Table 12: Performance of various models on four math benchmarks, named MATH, GSM8K, MMLU STEM and TheoremQA respectively.

## 7.1 Code Generation

Building on the performance improvements of the Qwen2.5-Coder series base models, our Qwen2.5-Coder series instruct models similarly demonstrated outstanding performance in code generation tasks.

**HumanEval and MBPP**  We also assessed the code generation capabilities of the Qwen2.5-Coder series instruction models using the EvalPlus (Liu et al., 2023) dataset. As shown by the results in Table 16, our Qwen2.5-Coder-7B-Instruct model demonstrated exceptional accuracy, significantly outperforming other models with a comparable parameter count. Remarkably, it even surpassed larger models with over 20 billion parameters, such as CodeStral-22B and DS-Coder-33B-Instruct. Furthermore, our Qwen2.5-Coder-32B-Instruct model achieved the highest performance on EvalPlus, even outperforming DS-Coder-V2-Instruct, making it the most powerful open-source code model to date.

**BigCodeBench-Instruct**  The *instruct* split provided by BigCodeBench (Zhuo et al., 2024) is designed to evaluate the code generation capabilities of instruction-based models. We evaluated the Qwen2.5-Coder series instruct models on the BigCodeBench-Instruct dataset. As indicated in Table 16, the Qwen2.5-Coder-7B-Instruct model outperformed other instruct models with comparable parameter sizes, achieving notably high accuracy scores on both the full and hard subsets, reaching 41.0% on the full subset and 18.2% on the hard subset. This highlights the robust code generation capabilities of the Qwen2.5-Coder instruct models. Furthermore, the Qwen2.5-Coder-32B-Instruct achieved accuracy rates of 49.6% on the complete split and 27.0% on the hard split, establishing it as the best-performing open-source code generation model and surpassing several closed-source APIs.

| Model | Size | MMLU | | |
|-------|------|------|-----|-------|
| | | **Base** | **Pro** | **Redux** |
| **0.5B+ Models** | | | | |
| **Qwen2.5-Coder-0.5B** | 0.5B | **42.0** | **13.3** | **40.6** |
| **1B+ Models** | | | | |
| DS-Coder-1.3B-Base | 1.3B | 25.8 | 11.4 | 24.5 |
| **Qwen2.5-Coder-1.5B** | 1.5B | **53.6** | **23.1** | **50.9** |
| **3B+ Models** | | | | |
| StarCoder2-3B | 3B | 36.6 | 15.5 | 37.0 |
| **Qwen2.5-Coder-3B** | 3B | **61.2** | **32.0** | **59.5** |
| **6B+ Models** | | | | |
| StarCoder2-7B | 7B | 38.8 | 17.2 | 38.6 |
| DS-Coder-6.7B-Base | 6.7B | 36.4 | 16.7 | 36.5 |
| DS-Coder-V2-Lite-Base | 2.4/16B | 60.5 | 33.4 | 58.3 |
| CodeQwen1.5-7B | 7B | 40.5 | 17.2 | 41.2 |
| **Qwen2.5-Coder-7B** | 7B | **68.0** | **40.1** | **66.6** |
| **14B+ Models** | | | | |
| StarCoder2-15B | 15B | 64.1 | 24.3 | 48.8 |
| **Qwen2.5-Coder-14B** | 14B | **75.2** | **49.3** | **72.4** |
| **20B+ Models** | | | | |
| DS-Coder-33B-Base | 33B | 39.4 | 18.4 | 38.7 |
| **Qwen2.5-Coder-32B** | 32B | **79.1** | **50.4** | **77.5** |

Table 13: MMLU results of different models, a general benchmark for common knowledge.

**LiveCodeBench**    LiveCodeBench (Jain et al., 2024) is a comprehensive and contamination-free benchmark designed to evaluate the coding capabilities of LLMs. It continuously gathers new problems from leading competitive programming platforms like LeetCode[6], AtCoder[7], and CodeForces[8], ensuring an up-to-date and diverse set of challenges. Currently, it hosts over 600 high-quality coding problems published between May 2023 and September 2024.

To further demonstrate our model's effectiveness on real-world competitive programming tasks, we evaluated the Qwen-2.5-Coder series instruct models on the LiveCodeBench (2407-2409) dataset. As shown in Table 16, the Qwen-2.5-Coder-7B-Instruct model achieved an impressive Pass@1 accuracy of 37.6%, significantly outperforming other models with similar parameter counts. Notably, it also outperformed larger models, such as CodeStral-22B-v0.1 and DS-Coder-33B-Instruct. Additionally, our Qwen-2.5-Coder-32B-Instruct model achieved an accuracy of 31.4%, surpassing all open-source code generation models and reaching a level comparable to many closed-source APIs.

**Multi-Programming Language**    The Qwen2.5-Coder series instruct models have inherited the high performance of the base model on the Multi-Programming Language. To further evaluate their capabilities, we tested the instruct models on two specific benchmarks: MultiPL-E (Cassano et al., 2022) and McEval (Chai et al., 2024).

**MultiPL-E**    As shown by the evaluation results in Table 17, Qwen2.5-Coder-7B-Instruct consistently outperforms other models with similar parameter counts, such as DS-Coder-

---

[6]https://leetcode.com
[7]https://atcoder.jp
[8]https://codeforces.com

| Model | Size | ARC-Challenge | TruthfulQA | WinoGrande | HellaSwag |
|---|---|---|---|---|---|
| **0.5B+ Models** | | | | | |
| **Qwen2.5-Coder-0.5B** | 0.5B | **34.4** | **42.7** | **54.8** | **48.4** |
| **1B+ Models** | | | | | |
| DS-Coder-1.3B-Base | 1.3B | 25.4 | 42.7 | 53.3 | 39.5 |
| **Qwen2.5-Coder-1.5B** | 1.5B | **45.2** | **44.0** | **60.7** | **61.8** |
| **3B+ Models** | | | | | |
| StarCoder2-3B | 3B | 34.2 | 40.5 | 57.1 | 48.1 |
| **Qwen2.5-Coder-3B** | 3B | **52.9** | **49.2** | **67.4** | **70.9** |
| **6B+ Models** | | | | | |
| StarCoder2-7B | 7B | 38.7 | 42.0 | 57.1 | 52.4 |
| DS-Coder-6.7B-Base | 6.7B | 36.4 | 40.2 | 57.6 | 53.8 |
| DS-Coder-V2-Lite-Base | 2.4/16B | 57.3 | 38.8 | **72.9** | 76.1 |
| CodeQwen1.5-7B | 7B | 35.7 | 42.2 | 59.8 | 56.0 |
| **Qwen2.5-Coder-7B** | 7B | **60.9** | **50.6** | **72.9** | **76.8** |
| **14B+ Models** | | | | | |
| StarCoder2-15B | 15B | 47.2 | 37.9 | 64.3 | 64.1 |
| **Qwen2.5-Coder-14B** | 14B | **66.0** | **55.2** | **76.8** | **80.2** |
| **20B+ Models** | | | | | |
| DS-Coder-33B-Base | 33B | 42.2 | 40.0 | 62.0 | 60.2 |
| DS-Coder-V2-Base | 21/236B | 64.3 | 41.4 | **83.7** | **86.0** |
| **Qwen2.5-Coder-32B** | 32B | **70.5** | **54.2** | 80.8 | 83.0 |

Table 14: General performance of different models on four popular general benchmarks, ARC-Challenge, TruthfulQA, WinoGrande and HellaSwag.

| Artifact | Public link |
|---|---|
| Qwen2.5-Coder-0.5B-Instruct | https://hf.co/Qwen/Qwen2.5-Coder-0.5B-Instruct |
| Qwen2.5-Coder-1.5B-Instruct | https://hf.co/Qwen/Qwen2.5-Coder-1.5B-Instruct |
| Qwen2.5-Coder-3B-Instruct | https://hf.co/Qwen/Qwen2.5-Coder-3B-Instruct |
| Qwen2.5-Coder-7B-Instruct | https://hf.co/Qwen/Qwen2.5-Coder-7B-Instruct |
| Qwen2.5-Coder-14B-Instruct | https://hf.co/Qwen/Qwen2.5-Coder-14B-Instruct |
| Qwen2.5-Coder-32B-Instruct | https://hf.co/Qwen/Qwen2.5-Coder-32B-Instruct |
| CodeQwen1.5-7B-Chat | https://hf.co/Qwen/CodeQwen1.5-7B-Chat |
| CodeLlama-7B-Instruct | https://hf.co/meta-llama/CodeLlama-7b-Instruct-hf |
| CodeLlama-13B-Instruct | https://hf.co/meta-llama/CodeLlama-13b-Instruct-hf |
| CodeLlama-34B-Instruct | https://hf.co/meta-llama/CodeLlama-34b-Instruct-hf |
| CodeLlama-70B-Instruct | https://hf.co/meta-llama/CodeLlama-70b-Instruct-hf |
| DS-Coder-1.3B-instruct | https://hf.co/deepseek-ai/deepseek-coder-1.3b-instruct |
| DS-Coder-6.7B-instruct | https://hf.co/deepseek-ai/deepseek-coder-6.7b-instruct |
| DS-Coder-33B-instruct | https://hf.co/deepseek-ai/deepseek-coder-33b-instruct |
| DS-Coder-V2-Lite-Instruct | https://hf.co/deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct |
| DS-Coder-V2-Instruct | https://hf.co/deepseek-ai/DeepSeek-Coder-V2-Instruct |
| Starcoder2-15B-Instruct-v0.1 | https://hf.co/bigcode/starcoder2-15b-instruct-v0.1 |
| CodeStral-22B-v0.1 | https://hf.co/mistralai/Codestral-22B-v0.1 |
| Yi-Coder-1.5B-Chat | https://hf.co/01-ai/Yi-Coder-1.5B-Chat |
| Yi-Coder-9B-Chat | https://hf.co/01-ai/Yi-Coder-9B-Chat |

Table 15: All artifacts released and used in this section.

V2-Lite-Instruct, in code generation tasks across eight programming languages. Both Qwen2.5-Coder-7B-Instruct and Qwen2.5-Coder-14B-Instruct even surpass larger models, like CodeStral-22B and DS-Coder-33B-Instruct (which have over 20 billion parameters), underscoring their strong code generation capabilities across multiple languages. Our Qwen2.5-Coder-32B-Instruct model achieves comparable performance to the DS-Coder-V2-

| Model | Size | HumanEval | | MBPP | | BigCodeBench | | LiveCodeBench |
|---|---|---|---|---|---|---|---|---|
| | | HE | HE+ | MBPP | MBPP+ | Full | Hard | Pass@1 |
| **0.5B+ Models** | | | | | | | | |
| **Qwen2.5-Coder-0.5B-Instruct** | 0.5B | **61.6** | **57.3** | **52.4** | **43.7** | **11.1** | **1.4** | **2.0** |
| **1B+ Models** | | | | | | | | |
| DS-Coder-1.3B-Instruct | 1.3B | 65.9 | 60.4 | 65.3 | 54.8 | 22.8 | 3.4 | 5.1 |
| Yi-Coder-1.5B-Chat | 1.5B | 69.5 | 64.0 | 65.9 | 57.7 | 23.8 | **11.5** | 4.8 |
| **Qwen2.5-Coder-1.5B-Instruct** | 1.5B | **70.7** | **66.5** | **69.2** | **59.4** | **32.5** | 6.8 | **6.1** |
| **3B+ Models** | | | | | | | | |
| **Qwen2.5-Coder-3B-Instruct** | 3B | **84.1** | **80.5** | **73.6** | **62.4** | **35.8** | **14.2** | **10.8** |
| **6B+ Models** | | | | | | | | |
| CodeLlama-7B-Instruct | 7B | 40.9 | 33.5 | 54.0 | 44.4 | 21.9 | 3.4 | 7.1 |
| DS-Coder-6.7B-Instruct | 6.7B | 74.4 | 71.3 | 74.9 | 65.6 | 35.5 | 10.1 | 15.5 |
| CodeQwen1.5-7B-Chat | 7B | 83.5 | 78.7 | 77.7 | 67.2 | 39.6 | 18.9 | 7.9 |
| Yi-Coder-9B-Chat | 9B | 82.3 | 74.4 | 82.0 | 69.0 | 38.1 | 11.5 | 17.2 |
| DS-Coder-V2-Lite-Instruct | 2.4/16B | 81.1 | 75.6 | 82.8 | 70.4 | 36.8 | 16.2 | 16.3 |
| **Qwen2.5-Coder-7B-Instruct** | 7B | **88.4** | **84.1** | **83.5** | **71.7** | **41.0** | **18.2** | **18.2** |
| **13B+ Models** | | | | | | | | |
| CodeLlama-13B-Instruct | 13B | 40.2 | 32.3 | 60.3 | 51.1 | 28.5 | 9.5 | 6.1 |
| Starcoder2-15B-Instruct-v0.1 | 15B | 67.7 | 60.4 | 78.0 | 65.1 | 37.2 | 11.5 | 12.1 |
| **Qwen2.5-Coder-14B-Instruct** | 14B | **89.6** | **87.2** | **86.2** | **72.8** | **48.4** | **22.2** | **23.4** |
| **20B+ Models** | | | | | | | | |
| CodeLlama-34B-Instruct | 34B | 48.2 | 40.2 | 61.1 | 50.5 | 29.0 | 8.8 | 8.4 |
| CodeStral-22B-v0.1 | 22B | 81.1 | 73.2 | 78.2 | 62.2 | 41.8 | 16.9 | 22.6 |
| DS-Coder-33B-Instruct | 33B | 81.1 | 75.0 | 80.4 | 70.1 | 42.0 | 17.6 | 21.3 |
| CodeLlama-70B-Instruct | 70B | 72.0 | 65.9 | 77.8 | 64.6 | 40.7 | 11.5 | 3.3 |
| DS-Coder-V2-Instruct | 21/236B | 85.4 | 82.3 | 89.4 | **75.1** | 48.2 | 24.3 | 27.9 |
| **Qwen2.5-Coder-32B-Instruct** | 32B | **92.7** | **87.2** | **90.2** | **75.1** | **49.6** | **27.0** | **31.4** |
| **Closed-APIs** | | | | | | | | |
| Claude-3.5-Sonnet-20240620 | - | 89.0 | 81.1 | 87.6 | 72.0 | 45.3 | 25.7 | 32.1 |
| Claude-3.5-Sonnet-20241022 | - | 92.1 | 86.0 | 91.0 | 74.6 | 45.3 | 23.6 | 31.6 |
| GPT-4o-mini-2024-07-18 | - | 87.8 | 84.8 | 86.0 | 72.2 | 46.9 | 23.6 | 28.3 |
| GPT-4o-2024-08-06 | - | 92.1 | 86.0 | 86.8 | 72.5 | 50.1 | 25.0 | 34.6 |
| o1-mini | - | **97.6** | **90.2** | **93.9** | **78.3** | 46.3 | 23.0 | **60.0** |
| o1-preview | - | 95.1 | 88.4 | 93.4 | 77.8 | 49.3 | **27.7** | 43.1 |

Table 16: The performance of different instruct models on code generation by HumanEval, MBPP, bigcodebench and livecodebench. For bigcodebench here, we report "instruct" tasks score.

Instruct model with only 32 billion parameters, bringing it very close to the performance of several closed-source APIs.

**McEval**    To comprehensively assess the code generation capabilities of the Qwen2.5-Coder series models across a broader range of programming languages, we evaluated them on the McEval benchmark (Chai et al., 2024), which spans 40 programming languages and includes 16,000 test cases. As shown in Figure 7, the Qwen2.5-Coder-32B-Instruct model excels when compared to other open-source models on the McEval benchmark, particularly across a wide range of programming languages.

**MdEval**    Qwen2.5-Coder is further evaluated on the comprehensive multilingual code debugging benchmark MdEval (Liu et al., 2024b) across 18 languages. Compared to the multilingual code generation benchmark McEval (Chai et al., 2024), MdEval provides the buggy code with example test cases (1.2K samples) to LLM for generating the correct code. Figure 8 demonstrates that the Qwen2.5-Coder-32B-Instruct achieves a comparable or better performance even compared to LLMs with larger model sizes.

**Human Preference Alignment**    To evaluate the alignment performance of Qwen2.5-Coder-32B-Instruct with the human preferences, we adopted an internal annotated evaluation benchmark called CodeArena, including nearly 400 human-curated samples. Similar to

| Model | Size | Python | Java | C++ | C# | TS | JS | PHP | Bash | Average |
|-------|------|--------|------|-----|-----|-----|-----|-----|------|---------|
| **0.5B+ Models** | | | | | | | | | | |
| **Qwen2.5-Coder-0.5B-Instruct** | 0.5B | **62.8** | **46.2** | **43.5** | **62.7** | **50.3** | **50.3** | **52.8** | **27.8** | **49.6** |
| **1B+ Models** | | | | | | | | | | |
| DS-Coder-1.3B-Instruct | 1.3B | 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 | 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 | **71.2** | **55.7** | **50.9** | **64.6** | **61.0** | **62.1** | **59.0** | **29.1** | **56.7** |
| **3B+ Models** | | | | | | | | | | |
| **Qwen2.5-Coder-3B-Instruct** | 3B | **83.5** | **74.7** | **68.3** | **78.5** | **79.9** | **75.2** | **73.3** | **43.0** | **72.1** |
| **6B+ Models** | | | | | | | | | | |
| CodeLlama-7B-Instruct | 7B | 34.8 | 30.4 | 31.1 | 21.6 | 32.7 | - | 28.6 | 10.1 | - |
| DS-Coder-6.7B-Instruct | 6.7B | 78.6 | 68.4 | 63.4 | 72.8 | 67.2 | 72.7 | 68.9 | 36.7 | 66.1 |
| CodeQwen1.5-7B-Chat | 7B | 84.1 | 73.4 | 74.5 | 77.8 | 71.7 | 75.2 | 70.8 | 39.2 | 70.8 |
| Yi-Coder-9B-Chat | 9B | 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 | 76.6 | 75.8 | 76.6 | 80.5 | 77.6 | 74.5 | 43.0 | 73.2 |
| **Qwen2.5-Coder-7B-Instruct** | 7B | **87.8** | **76.5** | **75.6** | **80.3** | **81.8** | **83.2** | **78.3** | **48.7** | **76.5** |
| **13B+ Models** | | | | | | | | | | |
| CodeLlama-13B-Instruct | 13B | 42.7 | 40.5 | 42.2 | 24.0 | 39.0 | - | 32.3 | 13.9 | - |
| Starcoder2-15B-Instruct-v0.1 | 15B | 68.9 | 53.8 | 50.9 | 62.7 | 57.9 | 59.6 | 53.4 | 24.7 | 54.0 |
| **Qwen2.5-Coder-14B-Instruct** | 14B | **89.0** | **79.7** | **85.1** | **84.2** | **86.8** | **84.5** | **80.1** | **47.5** | **79.6** |
| **20B+ Models** | | | | | | | | | | |
| CodeLlama-34B-Instruct | 34B | 41.5 | 43.7 | 45.3 | 31.0 | 40.3 | - | 36.6 | 19.6 | - |
| CodeStral-22B-v0.1 | 22B | 81.1 | 63.3 | 65.2 | 43.7 | 68.6 | - | 68.9 | 42.4 | - |
| DS-Coder-33B-Instruct | 33B | 79.3 | 73.4 | 68.9 | 74.1 | 67.9 | 73.9 | 72.7 | 43.0 | 69.2 |
| CodeLlama-70B-Instruct | 70B | 67.8 | 58.2 | 53.4 | 36.7 | 39.0 | - | 58.4 | 29.7 | - |
| DS-Coder-V2-Instruct | 21/236B | 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** | 80.4 | 79.5 | **82.9** | **86.8** | **85.7** | 78.9 | 48.1 | 79.4 |
| **Closed-APIs** | | | | | | | | | | |
| Claude-3.5-Sonnet-20240620 | - | 89.6 | 86.1 | 82.6 | 85.4 | 84.3 | 84.5 | 80.7 | 48.1 | 80.2 |
| Claude-3.5-Sonnet-20241022 | - | 93.9 | 86.7 | 88.2 | **87.3** | 88.1 | 91.3 | 82.6 | 52.5 | 83.8 |
| GPT-4o-mini-2024-07-18 | - | 87.2 | 75.9 | 77.6 | 79.7 | 79.2 | 81.4 | 75.2 | 43.7 | 75.0 |
| GPT-4o-2024-08-06 | - | 90.9 | 83.5 | 76.4 | 81.0 | 83.6 | 90.1 | 78.9 | 48.1 | 79.1 |
| o1-mini | - | 95.7 | **90.5** | **93.8** | 77.2 | **91.2** | 92.5 | 84.5 | **55.1** | 85.1 |
| o1-preview | - | **96.3** | 88.0 | 91.9 | 84.2 | 90.6 | **93.8** | **90.1** | 47.5 | **85.3** |

Table 17: The performance of different models on instruct format MultiPL-E.

Chatbot Arena (Chiang et al., 2024), we use CodeArena to emulate user code-related prompts in realistic environments. We use GPT-4o as the evaluation model for preference alignment, employing an "A vs. B win" evaluation method, which measures the percentage of instances in the test set where the score of A exceeds the score of B. The results in Figure 9 demonstrate the advantage of Qwen2.5-Coder-32B-Instruct in preference alignment.

## 7.2 Code Reasoning

To evaluate the code reasoning capabilities of the Qwen2.5-Coder series instruct models, we conducted an assessment on the CRUXEval (Gu et al., 2024) dataset. As shown in Table 18, the Qwen2.5-Coder-7B-Instruct model achieved Input-CoT and Output-CoT accuracies of 65.8% and 65.9%, respectively—demonstrating a substantial improvement over the DS-Coder-V2-Lite-Instruct model, with gains of 12.8% in Input-CoT accuracy and 13.0% in Output-CoT accuracy. Additionally, the Qwen2.5-Coder-7B-Instruct model outperformed larger models, including CodeStral-22B and DS-Coder-33B-Instruct, highlighting its advanced code reasoning capabilities despite its smaller size. Notably, our Qwen2.5-Coder-32B-Instruct model achieved accuracies of 75.2% and 83.4% on Input-CoT and Output-CoT, respectively, significantly outperforming other open-source code models (including DS-Coder-V2-Instruct) and underscoring its robust performance in code reasoning.

Figure 10 illustrates the relationship between model sizes and code reasoning capabilities. The Qwen2.5-Coder instruct models stand out for delivering superior code reasoning
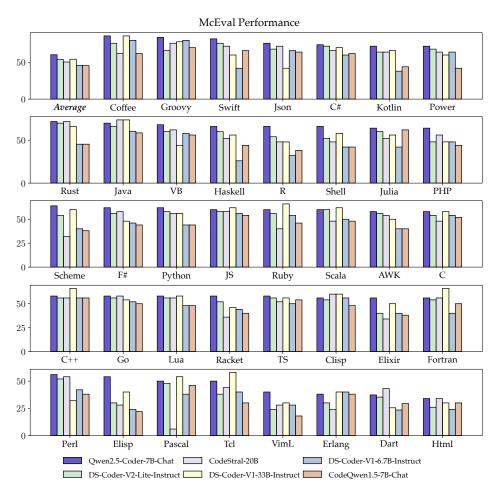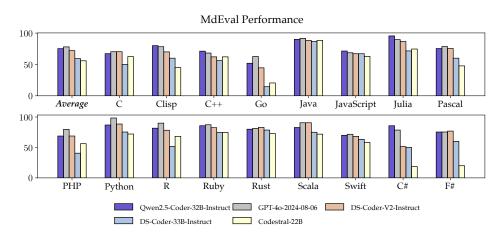
Figure 7: The McEval Performance of Qwen2.5-Coder-32B-Instruct compared with popular open-source large code models with similar size.

performance with the fewest parameters, surpassing the results of other open-source large language models by a significant margin.

## 7.3 Code Editing

**Aider** Aider[9] has created a code editing benchmark designed to quantitatively measure its collaboration with large language models (LLMs). Drawing from a set of 133 Python exercises sourced from Exercism[10], the benchmark tests the ability of Aider and LLMs to interpret natural language programming requests and translate them into executable code that successfully passes unit tests. This assessment goes beyond evaluating raw coding proficiency; it also examines how effectively LLMs can edit existing code and format those modifications for seamless integration with Aider's system, ensuring that local source files can be updated without issues. The comprehensive nature of this benchmark reflects both the technical aptitude of the LLMs and their consistency in task completion. Table 19 highlights the performance of several language models in the Code Editing task. Among these models, Qwen2.5-Coder-7B-Instruct exhibits exceptional code repair capabilities. Despite its relatively modest scale of 7 billion parameters, it achieves an impressive PASS@1 accuracy of 51.9%, significantly outperforming comparable models. Remarkably, it also surpasses larger models such as CodeStral-22B and DS-Coder-33B-Instruct , highlighting

---

[9] https://github.com/paul-gauthier/aider
[10] https://github.com/exercism/python

Figure 8: The MdEval Performance of Qwen2.5-Coder-32B-Instruct compared with popular open-source large code models with similar size.
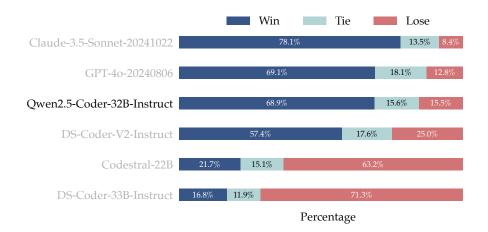


Figure 9: The CodeArena Performance of Qwen2.5-Coder-32B-Instruct compared with popular open-source large code models with similar size.

its remarkable efficiency and effectiveness in code editing tasks. Our Qwen2.5-Coder-32B-Instruct model achieves even higher accuracy, with Pass@1 and Pass@2 rates reaching 60.9% and 73.7%, respectively.

**CodeEditorBench**   An effective code assistant must excel in generating code based on given specifications, as well as in modifying or debugging existing code to meet evolving requirements or resolve issues. In evaluating Qwen2.5-Coders proficiency in code modification tasks, we focused on the CodeEditorBench (Guo et al., 2024b) suite, which assesses performance across four key dimensions: Debugging, Translation, Switching, and Polishing. We employed the same evaluation approach used in the original paper, relying on win rate as the metric for overall performance across diverse problem types. The win rate was computed for each problem category and then averaged across all categories to obtain the overall score. The results in Figure 11 show that Qwen2.5-Coder-32B-Instruct achieves a win rate comparable to DS-Coder-V2-Instruct (86.2% win rate), which features a significantly larger 236 billion parameter scale.

| Model | Size | CRUXEval | |
| --- | --- | --- | --- |
| | | *Input-CoT* | *Output-CoT* |
| **0.5B+ Models** | | | |
| **Qwen2.5-Coder-0.5B-Instruct** | 0.5B | **33.9** | **27.8** |
| **1B+ Models** | | | |
| DS-Coder-1.3B-Instruct | 1.3B | 12.9 | 28.1 |
| Yi-Coder-1.5B-Chat | 1.5B | 19.9 | 24.9 |
| **Qwen2.5-Coder-1.5B-Instruct** | 1.5B | **45.4** | **37.5** |
| **3B+ Models** | | | |
| **Qwen2.5-Coder-3B-Instruct** | 3B | **53.2** | **56.0** |
| **6B+ Models** | | | |
| CodeLlama-7B-Instruct | 7B | 36.1 | 36.2 |
| DS-Coder-6.7B-Instruct | 6.7B | 42.6 | 45.1 |
| CodeQwen1.5-7B-Chat | 7B | 44.0 | 38.8 |
| Yi-Coder-9B-Chat | 9B | 47.5 | 55.6 |
| DS-Coder-V2-Lite-Instruct | 2.4/16B | 53.0 | 52.9 |
| **Qwen2.5-Coder-7B-Instruct** | 7B | **65.8** | **65.9** |
| **13B+ Models** | | | |
| CodeLlama-13B-Instruct | 13B | 47.5 | 41.1 |
| Starcoder2-15B-Instruct-v0.1 | 15B | 45.5 | 50.9 |
| **Qwen2.5-Coder-14B-Instruct** | 14B | **69.5** | **79.5** |
| **20B+ Models** | | | |
| CodeLlama-34B-Instruct | 34B | 48.5 | 47.1 |
| CodeStral-22B-v0.1 | 22B | 61.3 | 63.5 |
| DS-Coder-33B-Instruct | 33B | 47.3 | 50.6 |
| CodeLlama-70B-Instruct | 70B | 56.5 | 57.8 |
| DS-Coder-V2-Instruct | 21/236B | 70.0 | 75.1 |
| **Qwen2.5-Coder-32B-Instruct** | 32B | **75.2** | **83.4** |
| *Closed-APIs* | | | |
| Claude-3.5-Sonnet-20240620 | - | 75.5 | 81.8 |
| Claude-3.5-Sonnet-20241022 | - | 84.4 | 87.2 |
| GPT-4o-mini-2024-07-18 | - | 67.5 | 78.4 |
| GPT-4o-2024-08-06 | - | 78.6 | 89.2 |
| o1-mini | - | 91.6 | 96.2 |
| o1-preview | - | 86.5 | 81.4 |

Table 18: The CRUXEval performance of different instruct models, with *Input-CoT* and *Output-CoT* settings.

### 7.4 Text-to-SQL

SQL is one of the essential tools in daily software development and production, but its steep learning curve often hinders free interaction between non-programming experts and databases. To address this issue, the Text-to-SQL task was introduced, aiming for models to automatically map natural language questions to structured SQL queries. Previous improvements in Text-to-SQL focused primarily on structure-aware learning, domain-specific pre-training, and sophisticated prompt designs.

Thanks to the use of finely crafted synthetic data during both pre-training and fine-tuning, we significantly enhanced Qwen2.5-Coder's capability in Text-to-SQL tasks. We selected two well-known benchmarks, Spider (Yu et al., 2018) and BIRD (Li et al., 2024a), for comprehensive evaluation. To ensure a fair comparison between Qwen2.5-Coder and other open-source language models on this task, we used a unified prompt template as input, following the work of Chang & Fosler-Lussier (2023). The evaluation prompt consists of table representations aligned with database instructions, examples of table content, optional additional knowledge, and natural language questions. This standardized prompt template minimizes biases that may arise from prompt variations. As shown in Figure 12, Qwen2.5-Coder outperforms other code models of the same size on the Text-to-SQL task.
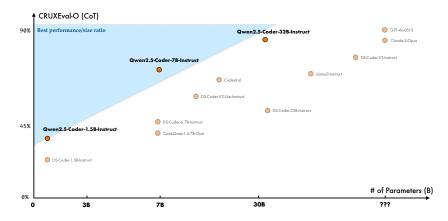
Figure 10: The relationship between model sizes and code reasoning capabilities. The x-axis represents the parameter sizes of different models, and the y-axis indicates the CRUXEval-O (CoT) scores respectively.
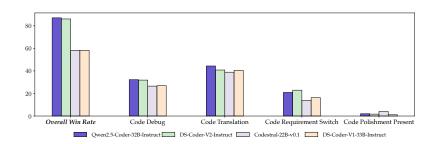


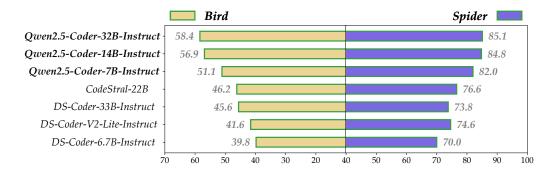Figure 11: The evaluation results on CodeEditBench.



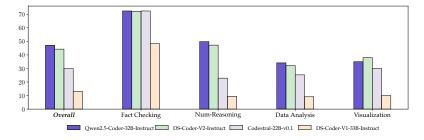Figure 12: The text-to-SQL evaluation on various instruct code models.



Figure 13: The table understanding evaluation on TableBench.

| Model | Size | Aider | |
| --- | --- | --- | --- |
| | | *Pass@1* | *Pass@2* |
| **0.5B+ Models** | | | |
| **Qwen2.5-Coder-0.5B-Instruct** | 0.5B | **14.3** | **14.3** |
| **1B+ Models** | | | |
| DS-Coder-1.3B-Instruct | 1.3B | 18.0 | 18.8 |
| Yi-Coder-1.5B-Chat | 1.5B | 17.3 | 17.3 |
| **Qwen2.5-Coder-1.5B-Instruct** | 1.5B | **28.6** | **31.6** |
| **3B+ Models** | | | |
| **Qwen2.5-Coder-3B-Instruct** | 3B | **33.8** | **39.1** |
| **6B+ Models** | | | |
| CodeLlama-7B-Instruct | 7B | 1.5 | 1.5 |
| DS-Coder-6.7B-Instruct | 6.7B | 37.6 | 44.4 |
| CodeQwen1.5-7B-Chat | 7B | 24.8 | 38.3 |
| Yi-Coder-9B-Chat | 9B | 45.9 | 54.1 |
| DS-Coder-V2-Lite-Instruct | 2.4/16B | 44.4 | 52.6 |
| **Qwen2.5-Coder-7B-Instruct** | 7B | **55.6** | **68.4** |
| **13B+ Models** | | | |
| CodeLlama-13B-Instruct | 13B | 1.5 | 1.5 |
| **Qwen2.5-Coder-14B-Instruct** | 14B | **58.6** | **69.2** |
| **20B+ Models** | | | |
| CodeLlama-34B-Instruct | 34B | 1.5 | 1.5 |
| CodeStral-22B-v0.1 | 22B | 36.8 | 51.1 |
| DS-Coder-33B-Instruct | 33B | 50.4 | 54.5 |
| CodeLlama-70B-Instruct | 70B | 12.8 | 15.0 |
| DS-Coder-V2-Instruct | 21/236B | 51.9 | **73.7** |
| **Qwen2.5-Coder-32B-Instruct** | 32B | **60.9** | **73.7** |
| **Closed-APIs** | | | |
| Claude-3.5-Sonnet-20240620 | - | 59.4 | 66.2 |
| Claude-3.5-Sonnet-20241022 | - | **71.4** | 86.5 |
| GPT-4o-mini-2024-07-18 | - | 43.6 | 55.6 |
| GPT-4o-2024-08-06 | - | 56.8 | 74.4 |
| o1-mini | - | 49.6 | 70.7 |
| o1-preview | - | 69.9 | **88.0** |

Table 19: The code editing ability of different instruct models evaluated by Aider benchmark. The *whole* edit-format was consistently applied across all our experiments.

## 7.5 Math Reasoning and General Natural Language

In this section, we provide a comparative analysis of the performance between our Qwen2.5-Coder series models and the DS-Coder-V2 series models, with a focus on both mathematical computation and general natural language processing tasks. The results in Table 20 highlight the versatility of the Qwen2.5-Coder series, which excels not only in complex coding tasks but also in advanced general-purpose tasks, setting it apart from its competitors.

## 7.6 Table Understanding

To evaluate the understanding capabilities of structured data, we further evaluate the Qwen2.5-Coder on a comprehensive and complex benchmark TableBench (Wu et al., 2024b), which includes 18 fields within four major categories of table question answering (TableQA) capabilities. We compare Qwen2.5-Coder with other LLMs under the textual chain-of-

| Model | Size | MATH | GSM8K | GaoKao2023en | OlympiadBench | CollegeMath | AIME24 |
|---|---|---|---|---|---|---|---|
| DS-Coder-V2-Lite-Instruct | 2.4/16B | 61.0 | 87.6 | 56.1 | 26.4 | 39.8 | 6.7 |
| DS-Coder-V2-Instruct | 21/236B | 74.2 | **94.5** | 65.7 | 37.8 | 45.9 | 6.7 |
| **Qwen2.5-Coder-3B-Instruct** | 3B | 58.1 | 80.7 | 48.8 | 23.6 | 39.7 | 6.7 |
| **Qwen2.5-Coder-7B-Instruct** | 7B | 66.8 | 86.7 | 60.5 | 29.8 | 43.5 | 10.0 |
| **Qwen2.5-Coder-14B-Instruct** | 14B | 66.8 | 94.2 | 66.0 | 40.1 | 47.3 | 10.0 |
| **Qwen2.5-Coder-32B-Instruct** | 32B | **76.4** | 93.0 | **68.3** | **42.5** | **47.7** | **20.0** |

| Model | Size | AMC23 | MMLU | MMLU-Pro | IFEval | CEval | GPQA |
|---|---|---|---|---|---|---|---|
| DS-Coder-V2-Lite-Instruct | 2.4/16B | 40.4 | 42.5 | 60.6 | 38.6 | 60.1 | 27.6 |
| DS-Coder-V2-Instruct | 21/236B | 52.5 | 76.7 | **65.6** | 40.9 | **73.4** | **44.3** |
| **Qwen2.5-Coder-3B-Instruct** | 3B | 25.0 | 56.5 | 35.2 | 44.2 | 53.9 | 28.3 |
| **Qwen2.5-Coder-7B-Instruct** | 7B | 42.5 | 68.7 | 45.6 | 58.6 | 61.4 | 35.6 |
| **Qwen2.5-Coder-14B-Instruct** | 14B | 50.0 | 71.7 | 55.6 | 66.5 | 66.2 | 36.8 |
| **Qwen2.5-Coder-32B-Instruct** | 32B | **55.0** | **77.6** | 62.3 | **79.9** | 68.9 | 41.8 |

Table 20: The performance of math and general.

thought (TCoT) setting. Figure 13 demonstrates that Qwen2.5-Coder-32B-Instruct gets the best performance 45.1 on TableBench.

## 8  Discussion: Scaling is All You Need

In Figure 14, We present a comparison of different sizes of Qwen2.5-Coder with other open-source LLMs on MBPP-3shot and LiveCodeBench. For the base LLM, we choose MBPP-3shot as the evaluation metric. Our extensive experiments show that MBPP-3shot is more suitable for evaluating base models and correlates well with the actual performance of the models. For the instruction model, we select the latest 4 months of LiveCodeBench (2024.07~2024.11) questions as the evaluation to strictly avoid test data contamination, truly reflecting the OOD capabilities of the LLM. There is a positive correlation between model size and model performance, and Qwen2.5-Coder has achieved state-of-the-art performance across all sizes, encouraging us to continue exploring larger sizes of code LLM.
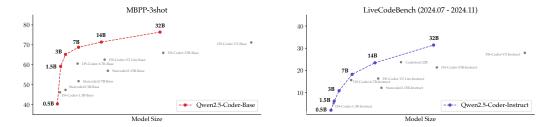


Figure 14: The evaluation results of Qwen2.5-Coder models with different sizes on MBPP-3shot and LiveCodeBench.

## 9  Conclusion

This work introduces Qwen2.5-Coder, the latest addition to the Qwen series. Built upon Qwen2.5, a top-tier open-source LLM, Qwen2.5-Coder has been developed through extensive pre-training and post-training of Qwen2.5-0.5B/1.5B/3B/7B/14B/32B on large-scale datasets. To ensure the quality of the pre-training data, we have curated a dataset by collecting public code data and extracting high-quality code-related content from web texts, while filtering out low-quality data using advanced classifiers. Additionally, we have constructed a meticulously designed instruction-tuning dataset to transform the base code LLM into a strong coding assistant.

Looking ahead, our research will focus on exploring the impact of scaling up code LLMs in terms of both data size and model size. We will also continue to enhance the reasoning capabilities of these models, aiming to push the boundaries of what code LLMs can achieve.

# References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, et al. Santacoder: don't reach for the stars! *arXiv preprint arXiv:2301.03988*, 2023.

Anthropic. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, 2024. 2024.06.21.

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.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

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. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.

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

Shuaichen Chang and Eric Fosler-Lussier. How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. *arXiv preprint arXiv:2305.11853*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. Theoremqa: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7889–7901, 2023.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yangruibo Ding, Zijian Wang, Wasi Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, et al. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. *Advances in Neural Information Processing Systems*, 36, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pp. 1536–1547. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.FINDINGS-EMNLP.139. URL https://doi.org/10.18653/v1/2020.findings-emnlp.139.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, et al. Are we done with mmlu? *arXiv preprint arXiv:2406.04127*, 2024.

Linyuan Gong, Sida Wang, Mostafa Elhoushi, and Alvin Cheung. Evaluation of llms on syntax-aware code fill-in-the-middle tasks. *arXiv preprint arXiv:2403.04814*, 2024.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.

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

Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi Li, Ruibo Liu, Yue Wang, et al. Codeeditorbench: Evaluating code editing capability of large language models. *arXiv preprint arXiv:2404.03543*, 2024b.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

AQ Jiang, A Sablayrolles, A Mensch, C Bamford, DS Chaplot, D de las Casas, F Bressand, G Lengyel, G Lample, L Saulnier, et al. Mistral 7b (2023). *arXiv preprint arXiv:2310.06825*, 2023.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36, 2024a.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tianyu Zheng, Xinyao Niu, Xiang Yue, Yue Wang, Jian Yang, Jiaheng Liu, et al. Autokaggle: A multi-agent framework for autonomous data science competitions. *arXiv preprint arXiv:2410.20424*, 2024b.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.

J Liu, CS Xia, Y Wang, and L Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. arxiv preprint arxiv: 230501210. 2023, 2023.

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

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

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

Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seung-won Hwang, and Alexey Svyatkovskiy. Reacc: A retrieval-augmented code completion framework. *arXiv preprint arXiv:2203.07722*, 2022.

MistralAI. Codestral. https://mistral.ai/news/codestral, 2024. 2024.05.29.

OpenAI. Gpt-4o. https://openai.com/index/hello-gpt-4o, 2024. 2024.05.13.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

Qwen. Code with codeqwen1.5, April 2024. URL https://qwenlm.github.io/blog/codeqwen1.5/.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

Tao Sun, Linzheng Chai, Jian Yang, Yuwei Yin, Hongcheng Guo, Jiaheng Liu, Bing Wang, Liqun Yang, and Zhoujun Li. Unicoder: Scaling code large language model via universal code. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 1812–1824. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.100.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with oss-instruct. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=XUeoOBid3x.

Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. Repoformer: Selective retrieval for repository-level code completion. *arXiv preprint arXiv:2403.10059*, 2024a.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xinrun Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, et al. Tablebench: A comprehensive and complex benchmark for table question answering. *arXiv preprint arXiv:2408.09174*, 2024b.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

Zhaojian Yu, Xin Zhang, Ning Shang, Yangyu Huang, Can Xu, Yishujie Zhao, Wenxiang Hu, and Qiufeng Yin. Wavecoder: Widespread and versatile enhancement for code large language models by instruction tuning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 5140–5153. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024. ACL-LONG.280. URL https://doi.org/10.18653/v1/2024.acl-long.280.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

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. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.