Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 10, No. 1, 2024

Research on Code Generation Technology based on LLM Pre-training

Ling Chen

Department of Big Data and Artificial Intelligence, Wuhu University, Wuhu, Anhui, 241000, *China*

Abstract: In recent years, the continuous improvement and rapid development of large language model (LLM) technology, and the pre-trained code generation technology involved in it has attracted extensive attention in the industry. Through LLM, the conversion from the well-known natural language (NL) to the programming language (PL) written by professional code practitioners can be realized, which greatly reduces the threshold of programming language, and has demonstrated significant performance and advantages in code generation tasks through pre-training. This paper systematically sorts out, researches and summarizes the pre-trained code generation techniques in recent years. Firstly, the development time roadmap of the pre-trained model related to code generation is extracted from the relevant research results. Secondly, the characteristics of different code generation pre-trained models are sorted out and summarized. At the same time, the evaluation mechanism and dataset for different pre-trained code generation models are given, and the research data are compared and analyzed. Finally, combined with the current development situation, the future development direction of code generation technology is prospected.

Keywords: LLM; pre-Training; Code Generation; NL-PL.

1. Introduction

In recent years, the development of LLM technology has attracted extensive attention from industry and academia, and since the early representative language model GPT-3 was proposed in July 2020, code models have been emerging due to their excellent natural language processing task processing capabilities, question answering and code capabilities. GPT-3 is based on the Transformer model, which reflects the advantages of the Transformer model's long text sequence processing ability, processes the input sequence through the self-attention mechanism, and combines parallel computing to speed up the training process. In addition, the autoregressive pre-training adopted by GPT-3 has outstanding performance in generation tasks, which has also prompted more scholars to devote themselves to the research of language models, expanding the types of programming language support for language models on code generation tasks (Java, Python, C/C++, Go, Ruby, JavaScript, PHP, etc.), and the research direction has also been continuously broadened to the fields of program understanding, code filling, code repair, clone detection, code translation and code optimization. Since the introduction of LLM, the pre-trained model of NL has rapidly influenced the process of software development, and gradually influenced the process of hardware development, and achieved success from NL to PL. The pre-trained models in LLMs can be architecturally divided into three categories: encoder-decoder architecture, encoder-only architecture, and decoder-only architecture. The core idea of all three architectures is that knowledge can be learned from large-scale, unlabeled data through self-supervised training. The pre-trained model needs to be pre-trained on large-scale unscaled data, and then fine-tuned on the annotated dataset for specific downstream tasks (such as code generation, code translation, and code repair) according to the representations obtained from the pre-training, so as to achieve better generalization ability and better performance on specific downstream tasks.

In the process of sorting out the research work of LLM pre-trained models on code generation tasks, the literature search keywords used in this paper were code generation, code synthesis, program synthesis, code completion, and text-to-code. The main ways of literature retrieval were: (1) CNKI database, Google Scholar, aiXiv and other databases, (2) enter keywords through the Connected papers webpage to find keyword-related correlation, high citation, and recent literatures, (3) combine Hugging Face and Github to find the publication link, star number and influence of the pre-trained model, (4) use Papers With Code searches keywords to understand the evaluation dataset of the pre-trained model in code generation, high-quality papers, and the latest papers, and (5) obtains the comparative data of key code evaluation through the EvalPlus Leaderboard. The retrieved literature was combed through and 49 high-quality papers were finally selected for this research. This paper mainly studies the pre-trained model of large language model, sorts out and summarizes the code generation technology of programming language and the quality evaluation of generated code, and looks forward to future research directions.

2. Code Generates Language Models

Code is a special data format, which is not only a simple code language sequence, but also contains a variety of internal code structures, such as Abstract Syntax Tree (AST), Data Flow Graph (DFG), and Control Flow Graph (CFG). In the past, Natual Language Processing (NLP) technology was better at processing language sequences, and the internal structure of the code has been continuously paid attention to, valued and practiced by researchers in the era of the rise of LLMs. The traditional supervised learning method needs to learn a large number of annotated code data and extract code feature information when dealing with code generation tasks, but the generalization ability is insufficient, and the fundamental reason is that the annotated code data of code generation tasks requires expert knowledge of each downstream task of the code, which requires a large cost and difficulty of dataset annotation. With the development of technology and the improvement of computing power, pre-

trained models that can perform self-supervised learning on large-scale unlabeled data have shown advantages and have been rapidly developed. After the pre-trained model, the dataset can be fine-tuned after a small amount of annotation in the later stage, so that it can be better applied to downstream tasks. Because of this, a variety of pre-trained models for code generation have emerged in recent years. According to the characteristics of the model framework, the model is divided into encoder-decoder architecture, encoder-only architecture and decoder-only architecture. The timeline of the code-generated by LLM since 2020 is shown in Fig.1.

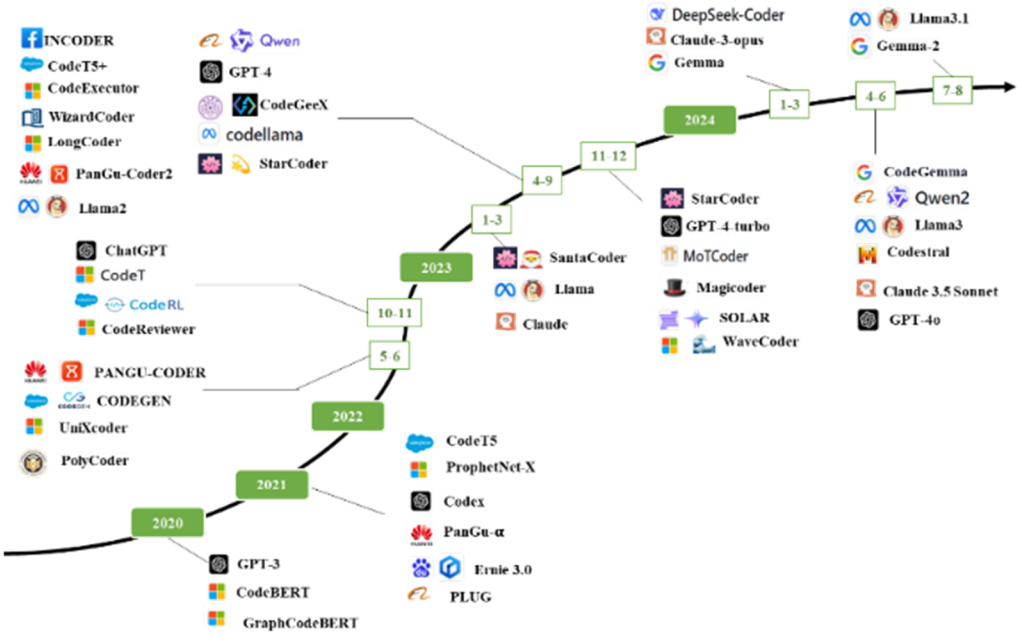

Fig 1. Timeline of code generation by LLM

2.1. Encoder-decoder Code Generation Models

In 2021, Wang et al.[1] proposed a pre-trained model called CodeT5, which supports multi-task learning through an encoder-decoder architecture and introduces an identifier-aware pre-training task, which can more effectively capture the semantic information of identifiers in the code, enhance the alignment between natural language and programming language, and improve the performance of code generation tasks. The following year, Le et al.[2] further developed on the basis of CodeT5 and proposed the CodeRL model. The model combines enhanced learning objectives, larger model sizes, and better pre-trained data, and demonstrates excellent code generation capabilities in APP and MBPP benchmarks by integrating pre-trained language models with deep reinforcement learning techniques. In 2023, Wang et al.[3] again contributed the CodeT5+ model, which mitigates the difference between the pre-training and fine-tuning phases by introducing hybrid pre-training objectives, including span denoising, contrastive learning, text code matching, and causal language model pre-training tasks. The experimental results show that the CodeT5+ 16B version has achieved excellent results in the HumanEval[4] code generation task, indicating that the model has strong generalization ability when dealing with actual programming tasks. Overall, these studies not only demonstrate the potential of encoder-decoder architecture in code generation tasks, but also provide valuable experience and technical reference for subsequent research. The above research demonstrates a trend to improve the performance of code generation models by improving model architecture, introducing task-specific pre-training, and combining different learning methods. From CodeT5 to CodeRL to CodeT5+, each step marks the progress of bridging the gap between NL and PL, and points the way to future code generation research.

2.2. Encoder-only Code Generation Models

In September 2020, Feng et al.[5] proposed the CodeBERT model, a bimodal pre-trained model based on the Transformer architecture, which aims to provide a universal representation for the transformation between natural language (NL) and programming language (PL). CodeBERT uses a hybrid objective function to evaluate NL-PL applications by fine-tuning model parameters by leveraging both bimodal and unimodal data of NL-PL pairs during training. Around the same time, Guo et al.[6] also developed the GraphCodeBERT model based on the Transformer architecture. Different from the traditional AST, GraphCodeBERT introduces the DFG in the pre-training stage, which simplifies the coding of the relationship between variables, and integrates the code structure information through the graph-guided masking attention mechanism, which improves the efficiency of the language model.

In 2023, Li et al.[7] proposed the MoTCoder model and its MoT instruction tuning framework. MoTCoder enhances the modularity and correctness of the build solution by breaking down the task into logical subtasks and submodules, thereby improving the quality of code generation. These studies not only demonstrate technological innovations in NL-PL conversion, but also provide new perspectives for future developments in the field of code generation.

The research on Encoder-only code generation models has shown diversified characteristics, from basic model architecture innovations, such as CodeBERT and GraphCodeBERT, to task-oriented optimization frameworks, such as MoTCoder, all of which are constantly promoting technological progress and development in this field. These research results not only provide a new solution for the bridge between NL and PL, but also provide rich theoretical basis and technical support for future scientific research.

2.3. Decoder-only Code Generation Models

The field of code generation has undergone significant developments, with a series of advanced pre-trained models emerging to advance programming-assisted technologies. In 2022, Erik Nijkamp et al.[8] proposed the CODEGEN model, which has 1.6 billion parameters, is able to decompose a single program into multiple sub-problems, and performs well in the HumanEval benchmark. In addition, they constructed the Multi-Round Programming Benchmark (MTPB), which consists of 115 different problem sets and became one of the commonly used evaluation criteria in subsequent studies. In the same year, Fried et al.[9] proposed the InCoder model, which achieves bidirectional context awareness through code filling, improves the performance of tasks such as type inference and annotation generation, and is comparable to other models in terms of program generation capabilities. Huawei's PanGu-Coder[10] implements text-to-code conversion on the PanGu-Alpha architecture, and enhances the performance of the model in a small context window through a combination of causal language modeling (CLM) and masking language modeling (MLM). In terms of automatic test case generation, Chen et al. launched the CodeT[11] model in 2022, which is able to perform a dual execution protocol to ensure the consistency of test cases, and has a pass@1 of 65.8% on HumanEval, which is better than previous research results. The UniXcoder[12] proposed by Guo et al. is a unified cross-modal pre-trained model, which enhances the code representation ability by using information such as AST encoding and code comments, and achieves good performance on multiple code tasks. The year 2023 saw the birth of the CodeGeeX[13] model, which has reached 13 billion parameters, supports not only Python, but also proposes HumanEval-X, an evaluation benchmark that supports multiple programming languages, and builds extensions on platforms such as Visual Studio Code. Loubna Ben Allal et al. [14] proposed the SantaCoder model, which improves model performance by training on a specific subset of programming languages and using a preprocessing method to filter duplicates. The Magicoder model by Wei et al.[15] was trained on synthetic instruction data using OSS-I NSTRUCT and performed well on the HumanEval+ benchmark. During the same period, the StarCoder model proposed by Li et al.[16] performed well in a variety of programming languages, while the WizardCoder proposed by Luo et al.[17] improved the fine-tuning ability of complex instructions through the evolution-directive approach. In addition, Microsoft's wavecoder [18]model solves the problem of repetition in instruction data generation and enhances the generalization ability of the model by classifying instruction data and introducing the Generator-Discriminator framework. In early 2024, Guo et al.[19] open-sourced the DeepSeek Coder model, which is pre-trained on a large-scale code corpus and further optimized for performance with fine-tuning. OpenAI's GPT-4o[20] marks a major breakthrough in multimodal technology, with performance similar to that of humans on a variety of tasks, especially in visual and audio understanding, have better performance than GPT-4[21][22]. In the same year, the Codestral[23] model released by the Mistral AI team focuses on code generation tasks and supports multiple programming languages. The Claude 3.5 Sonnet[24] model launched by AnthropicClaude has reached a high level in terms of reasoning ability, knowledge mastery and coding ability. In 2024, the release of the Qwen 2[25] model further boosted the development of code generation technology. The Qwen 2 model covers a wide range of parameters from hundreds of millions to billions of parameters and has demonstrated comparable or even better performance than proprietary models like Qwen[26] in multiple benchmarks such as coding. Meta's Llama 3.1series[27][28][29][30] models have achieved an important breakthrough in multi-language support, and their powerful instruction tuning capabilities and wide applicability make them ideal for advanced applications. The Gemma 2[31] model, through lightweight design and the application of advanced technology, not only surpasses similar models in multiple tasks, but also pays special attention to the safety and responsibility of the model, pointing out the direction for the development of future models[32][33][34]. In short, from 2022 to the present, significant progress has been made in the field of code generation, and the launch of a variety of models has not only enriched the technical means, but also provided a solid theoretical foundation and technical support for practical applications. These research results not only show technological innovation, but also reveal that code generation models are moving towards more practical and professional directions.

3. Evaluation Datasets for Code Generation

Before 2020, most of the code generated by the model could not be directly executed, and was only evaluated by the similarity between the generated code and the reference code, and common metrics include BLEU[35], CodeBLEU [36], and accuracy. Since 2020, with the rise of pre-trained big oracle model technology, test datasets to evaluate the quality of code generation have been emerging. Commonly used datasets include: HumanEval, MBPP [37], CoNaLa (The Code/Natural Language Challenge) [38], APPS [39], CodeXGLUE [40], DS-1000 (Data Science) [41], CodeContests [42], and CONCODE[43]. Table 1 shows the optimal model situation of the dataset commonly used for pre-trained LLMs [44].

Table 1. The best model of the dataset commonly used by pretrained LLMs

Dataset	Best Model
HumanEval	LDB (GPT-4o, based on seed programs from Reflexion)
MBPP	GPT-4 + AgentCoder
CoNaLa	PanGu-Coder-FT-I
APPS	MapCoder (GPT-4)
RES-Q	QurrentOS-coder + Claude 3.5 Sonnet
PECC	Claude 3 Haiku
CoNaLa-Ext	BART Base
Turbulence	GPT-4

The CoNaLa dataset, which was proposed by Yin P et al. in 2018, was scraped from Stack Overflow, filtered and organized, and divided into 2379 training examples and 500 test examples. CodeXGLUE, presented in February 2021, the benchmark dataset pair accelerates re-searching in programming language tasks. CodeXGLUE, a benchmark dataset searcher used to facilitate machine learning comprehension and generation. CodeXGLUE includes a collection of 10 tasks across 14 datasets and a platform for model evaluation and comparison. CodeXGLUE also features three baseline systems, including BERT-style, GPT-style, and encoder-decoder models, making it easy for researchers to use the platform. The availability of such data and baselines can aid in the development and validation of issues that can be used for a variety of program understanding and generation issues. The HumanEval dataset was released by OpenAI in July 2021 along with the code model Codex, and contains 164 programming questions, all of which include function signature properties, document strings, function bodies, and several unit tests. They are all handwritten to ensure that they are not included in the training set of the code generation model. Programming questions are written in Python and include natural text in English in comments and docstrings. For example, the task_id is HumalEval/0, which consists of seven unit test cases [45]. Table 2 shows the test rankings of different code models on the HumanEval dataset [44] (as of August 6, 2024). In 2021, the majority of the MBPP (Mostly Basic Python Programming) dataset was basic Python programming questions, and the benchmark consisted of 974 crowdsourced Python programming questions, each of which consisted of a task description, a code solution, and 3 automated test cases. Specifically, task ID 11 ~ 510 is used for testing, task ID 1~10 is used for a small number of short prompts, task ID 511~ 600 is used for verification during fine-tuning, and task ID 601~ 974 is used for training.

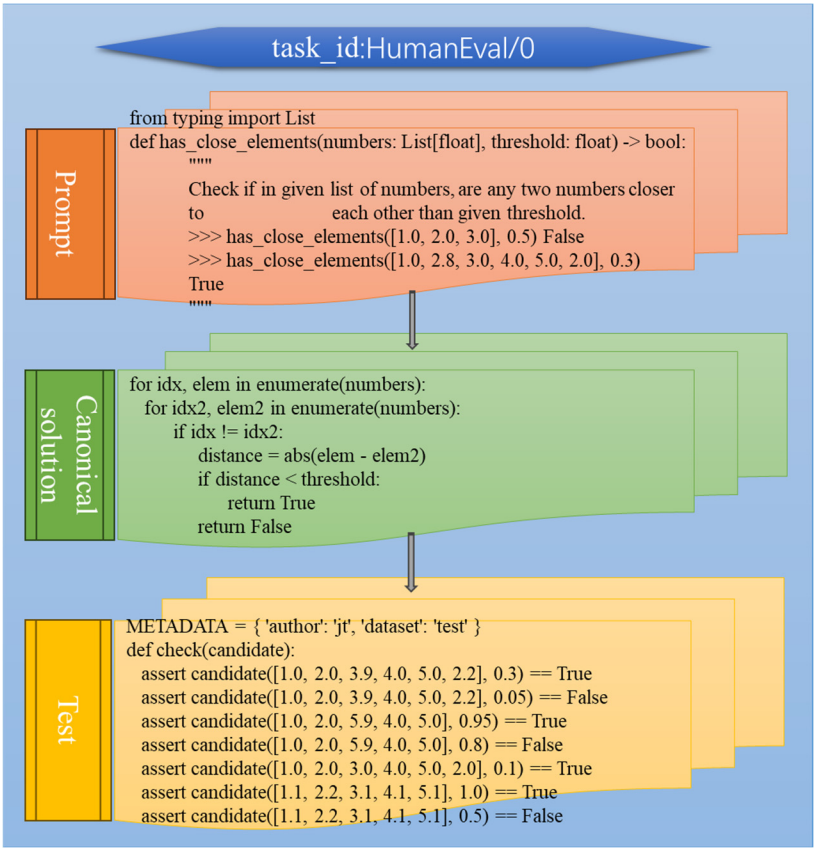

Fig 2. The task_id of the HumanEval dataset consists of HumalEval/0 questions

The APPS (Automated Programming Progress Standard) dataset was proposed by Hendrycks D et al. in 2021 and includes questions collected from different open access coding websites. The APPS benchmark attempts to mirror the way human programmers evaluate by asking coding questions in unrestricted natural language and assessing the correctness of the solutions. The difficulty of the questions ranges from entry-level to collegiate competition level and measures coding ability and problem-solving skills. There are 10,000 coding questions, with 131,836 test cases to check the solution and 232,444 human-written true solutions. Questions can be complicated because the average length of a question is 293.2 words. The data is divided equally into training and test sets, each with 5,000 questions. In the test set, there are multiple test cases for each question, and the average number of test cases is 21.2. Each test case is specifically designed for the respective problem in order to be able to critically evaluate the functionality of the program. In 2021, Orlanski G et al. proposed CoNaLa-ext (CoNaLa Extended with Question Text)[46] to introduce the CoNaLa extension with question text. A key addition is that each example now has a complete issue body from its respective StackOverflow issue. The Turbulence[47]dataset, proposed in 2023 by Honarvar et al., is a new benchmark for systematically evaluating the correctness and robustness of instruction tuning LLMs for code generation, consisting of a large number of natural language question templates, each of which is a programming problem that is parameterized to allow questions to be posed in many different forms. Each issue template has an expert in the associated tests to determine if the code solution returned

by the LLM is correct. Therefore, from a single problem template, it is possible to ask the LLM about the neighborhoods of very similar programming problems, and evaluate the correctness of the returned results of each problem, effectively solving the high robustness problem. Lai, Yuhan, et al. proposed DS-1000 in 2023, a code generation benchmark with 1,000 data science questions for Python. The DS-1000 contains 1,000 questions that originate from 451 unique StackOverflow issues. To prevent potential memories, more than half of the DS-1000 issues were modified from the original StackOverflow issue, these include 152 surface perturbations, 235 semantic perturbations, and 162 difficult rewrites.

In 2024, LaBash B et al. proposed RES-Q [48], a large language model system for code editing at the repository scale, a natural language instruction-based benchmark for evaluating repository editing systems, which consists of 100 hand-crafted repository editing tasks that come to GitHub. Given an edit instruction and code repository, RES-Q evaluates the LLM system's ability to interpret the edit instructions, gather information, and make appropriate edits to the repository. Proposed by Haller et al. in 2024, the PECC (Problem Extraction and Coding Challenges) is an existing benchmark that evaluates tasks in isolation and includes 2,396 questions[49]. Unlike traditional benchmarks, PECC requires LLMs to interpret embedded problems, extract requirements, and generate executable code. A key feature of the dataset is the added complexity of natural language prompts in conversation-based assessments, reflecting real-world instruction ambiguity.

Table 2. Scores of different pre-trained large language models on the HumanEval dataset on the Pass@1

Rank	Model	Pass@1	Year
1	LDB(GPT-4o, based on seed programs from Reflexion)	98.2	2024
2	LDB(GPT-4, based on seed programs from Reflexion)	96.9	2024
3	AgentCoder(GPT-4)	96.3	2023
4	LDB(GPT-3.5, based on seed programs from Reflexion)	95.1	2024
5	MapCoder(GPT-4)	93.9	2024
6	Language Agent Tree Search(GPT-4)	92.7	2023
7	Claude 3.5 Sonnet(0-shot)	92.0	2024
8	FractalResearch : Pioneer-SWO(GPT-4T)	91.6	2023
9	L2MAC(GPT-4)	90.2	2023
10	GPT-4o(0-shot)	90.2	2024

4. Conclusion

This paper summarizes the code generation task models of large language models since 2020, sorts out the main code pre-trained language models at home and abroad according to the timeline, and gives the development chart of the timeline. In addition, the typical datasets of code-generated language models are reviewed and summarized, and the research overview of the methods used to evaluate the code-generated results of language models is summarized and explained.

Although large-scale pre-trained language models have brought a lot of convenience to program design in code generation tasks, there are still many directions that need further attention and continuous improvement in related code generation technologies. The pre-trained language model technology is developed from NLP, and the model will also be affected by the base architecture such as Transformer, which will also continue the syntax correctness problem in NLP, and how to solve the syntax problem in the code generation process is a research direction. In addition, with the rapid development of code technology, it is worth paying attention to how to make the code generation of LLM adapt to various code open datasets and open-source code schemes in a timely manner. Finally, code generation technology will involve data leakage and security issues, and how to better implement code generation technology under the premise of ensuring data security is an important research direction for scientific researchers.

Acknowledgments

This work is supported by Research Project of Fundamentals of Programming for School-level Excellent Course Projects of Wuhu University, Project Number: WHJPKC-202403.

References

[1] Wang, Y., Wang, W., Joty, S. R., & Hoi, S. (2021). CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. ArXiv, abs/2109. 00859.

[2] Le, H., Wang, Y., Gotmare, A. D., Savarese, S., & Hoi, S. (2022). CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. ArXiv, abs/2207.01780.

[3] Wang, Y., Le, H., Gotmare, A. D., Bui, N. D. Q., Li, J., & Hoi, S. C. H. (2023). CodeT5+: Open Code Large Language Models for Code Understanding and Generation.

[4] Chen M, Tworek J, Jun H, et al. Evaluating large language models trained on code[J]. arXiv preprint arXiv:2107.03374, 2021.

[5] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages.

[6] Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S. K., Clement, C., Drain, D., Sundaresan, N., Yin, J., Jiang, D., & Zhou, M. (2020). GraphCodeBERT: Pre-training Code Representations with Data Flow.

[7] Li J, Chen P, Jia J. Motcoder: Elevating large language models with modular of thought for challenging programming tasks[J]. arXiv preprint arXiv:2312.15960, 2023.

[8] Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., & Xiong, C. (2022). CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis.

[9] Fried, D., Aghajanyan, A., Lin, J., Wang, S. I., Wallace, E., Shi, F., Zhong, R., Yih, W., Zettlemoyer, L., & Lewis, M. (2022).

InCoder: A Generative Model for Code Infilling and Synthesis. ArXiv, abs/2204.05999.

[10] Christopoulou, F., Lampouras, G., Gritta, M., Zhang, G., Guo, Y., Li, Z., Zhang, Q., Xiao, M., Shen, B., Li, L., Yu, H., Yan, L., Zhou, P., Wang, X., Ma, Y., Iacobacci, I., Wang, Y., Liang, G., Wei, J., Liu, Q. (2022). PanGu-Coder: Program Synthesis with Function-Level Language Modeling. ArXiv, abs/ 2207. 11280.

[11] Chen, B., Zhang, F., Nguyen, A., Zan, D., Lin, Z., Lou, J.-G., & Chen, W. (2022). CodeT: Code Generation with Generated Tests. ArXiv, abs/2207.10397.

[12] Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., & Yin, J. (2022). UniXcoder: Unified Cross-Modal Pre-training for Code Representation.

[13] Zheng, Q., Xia, X., Zou, X., Dong, Y., Wang, S., Xue, Y., Wang, Z., Shen, L., Wang, A., Li, Y., Su, T., Yang, Z., & Tang, J. (2023). CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Benchmarking on HumanEval-X. Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, abs/2303.17568.

[14] Allal, L. ben, Li, R., Kocetkov, D., Mou, C., Akiki, C., Ferrandis, C. M., Muennighoff, N., Mishra, M., Gu, A., Dey, M., Umapathi, L. K., Anderson, C. J., Zi, Y., Poirier, J., Schoelkopf, H., Troshin, S., Abulkhanov, D., Romero, M., Lappert, M., von Werra, L. (2023). SantaCoder: don't reach for the stars! ArXiv, abs/2301.03988, null.

[15] Wei, Y., Wang, Z., Liu, J., Ding, Y., & Zhang, L. (2023). Magicoder: Source Code Is All You Need. ArXiv, abs/2312. 02120.

[16] Li, R., Allal, L. ben, Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., … Vries, H. D. (2023). StarCoder: may the source be with you! ArXiv, abs/2305.06161.

[17] Luo, Z., Xu, C., Zhao, P., Sun, Q., Geng, X., Hu, W., Tao, C., Ma, J., Lin, Q., & Jiang, D. (2023). WizardCoder: Empowering Code Large Language Models with Evol-Instruct. ArXiv, abs/2306.08568.

[18] Yu Z, Zhang X, Shang N, et al. Wavecoder: Widespread and versatile enhanced instruction tuning with refined data generation[J]. arXiv preprint arXiv:2312.14187, 2023.

[19] Guo D, Zhu Q, Yang D, et al. DeepSeek-Coder: When the Large Language Model Meets Programming--The Rise of Code Intelligence[J]. arXiv preprint arXiv:2401.14196, 2024.

[20] Open AI, Hello GPT-4o(2024-05-18) [EB/OL], https:// openai. com/ index/ hello- gpt- 4o/.

[21] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S. and Avila, R., 2023. Gpt-4 technical report. arXiv preprint arXiv:2303.08774.

[22] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33: 1877-1901.

[23] Mistral, model fluent in 80+ programming languages(2024) [EB/OL], https://mistral.ai/news/codestral/.

[24] Anthropic, Claude 3.5 Sonnet[EB/OL], https://www. anthropic. com/news/claude-3-5-sonnet.

[25] Yang A, Yang B, Hui B, et al. Qwen2 technical report[J]. arXiv preprint arXiv:2407.10671, 2024.

[26] Bai J, Bai S, Chu Y, et al. Qwen technical report[J]. arXiv preprint arXiv:2309.16609, 2023.

[27] Meta AI, Meta Llama 3.1 (2024) [EB/OL], Retrieved from https://ai.meta.com/blog/meta-llama-3-1/.

[28] Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M. P., Ferrer, C. C., Grattafiori, A., Xiong, W., D'efossez, A., Copet, J., Synnaeve, G. (2023). Code Llama: Open Foundation Models for Code. ArXiv, abs/ 2308.12950.

[29] Touvron H, Martin L, Stone K, et al. Llama 2: Open foundation and fine-tuned chat models[J]. arXiv preprint arXiv:2307. 09288, 2023.

[30] Touvron H, Lavril T, Izacard G, et al. Llama: Open and efficient foundation language models[J]. arXiv preprint arXiv: 2302.13971, 2023.

[31] Team G, Riviere M, Pathak S, et al. Gemma 2: Improving open language models at a practical size[J]. arXiv preprint arXiv: 2408. 00118, 2024.

[32] Team C G. Codegemma: Open code models based on gemma[J]. arXiv preprint arXiv:2406.11409, 2024.

[33] Team G, Mesnard T, Hardin C, et al. Gemma: Open models based on gemini research and technology[J]. arXiv preprint arXiv: 2403.08295, 2024.

[34] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv preprint arXiv: 2403.05530.

[35] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, BLEU: a method for automatic evaluation of machine translation[C]// Proceedings of the 40th annual meeting of the Association for Computational Linguistics, 2002:311–318.

[36] Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Zhou, M., Blanco, A., & Ma, S. (2020). CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. ArXiv, abs/2009. 10297.

[37] Austin J, Odena A, Nye M, et al. Program synthesis with large language models[J]. arXiv preprint arXiv:2108.07732, 2021.

[38] Yin P, Deng B, Chen E, et al. Learning to mine aligned code and natural language pairs from stack overflow [C]// Proceedings of the 15th international conference on mining software repositories. 2018: 476-486.

[39] Hendrycks D, Basart S, Kadavath S, et al. Measuring coding challenge competence with apps[J]. arXiv preprint arXiv:2105. 09938, 2021.

[40] Lu S, Guo D, Ren S, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation[J]. arXiv preprint arXiv:2102.04664, 2021.

[41] Lai, Yuhang, et al. DS-1000: A natural and reliable benchmark for data science code generation[C] //International Conference on Machine Learning. PMLR, 2023: 18319-18345.

[42] Li Y, Choi D, Chung J, et al. Competition-level code generation with alphacode[J]. Science, 2022, 378(6624): 1092-1097.

[43] Iyer S, Konstas I, Cheung A, et al. Mapping language to code in programmatic context[J]. arXiv preprint arXiv:1808.09588, 2018.

[44] Papers With Code, Code Generation [EB/OL] (2024-08-06), https: //paperswithcode.com/task/code-generation.

[45] OpenAI, openai/openai_humaneval Datasets at Hugging Face, [EB/OL],https://huggingface.co/datasets/openai/openai_humaneval/viewer/openai_humaneval/test.

[46] Orlanski G, Gittens A. Reading stackoverflow encourages cheating: adding question text improves extractive code generation[J]. arXiv preprint arXiv:2106.04447, 2021.

[47] Honarvar, S., van der Wilk, M. and Donaldson, A., 2023. Turbulence: Systematically and automatically testing instruction-tuned large language models for code[J]. arXiv preprint arXiv:2312.14856.

[48] LaBash B, Rosedale A, Reents A, et al. RES-Q: Evaluating Code-Editing Large Language Model Systems at the Repository Scale[J]. arXiv preprint arXiv:2406.16801, 2024.

[49] Haller P, Golde J, Akbik A. Pecc: Problem extraction and coding challenges[J]. arXiv preprint arXiv:2404.18766, 2024.