# Artificial Intelligence-Assisted Programming Tasks

Subjects: Computer Science, Artificial Intelligence | Computer Science, Software Engineering

Contributor: Man-Fai Wong , Shangxin Guo , Ching-Nam Hang , Siu-Wai Ho , Chee-Wei Tan

Artificial intelligence (AI)-assisted programming can enable software engineers to work more efficiently and effectively with the existing software tools such as OpenAI ChatGPT, Github Copilot, DeepMind AlphaCode, Amazon Codewhisperer, Replit Ghostwriter, Microsoft IntelliCode and Codedium, especially in situations where complex algorithms are being used that involve large amounts of code (i.e., Big Code regime). It also strikes a balance between productivity and ensuring safety, security, and reliability within the programming development environment. There are two main categories of AI-assisted programming tasks related to software naturalness: generation and understanding. The former includes code generation, code completion, code translation, code refinement, and code summarization. The latter is concerned with understanding code and includes defect detection and clone detection.

software naturalness large language models AI-assisted programming

## 1. Code Generation

Program synthesis, also known as source code generation, is the process of automatically generating source code from a programming language based on user-specified constraints [1][2]. This research focuses on text-to-code generation for code generation, while code-to-code generation is referred to as code translation. The history of code generation dates back to the use of theorem provers to construct a proof of user-provided specifications and extract corresponding logical programs [3][4]. With the increasing popularity of deep learning methods, neural methods, including Long Short–Term Memory (LSTM) [5] and Recursive–Reverse–Recursive Neural Network [6], have been adopted to generate output programs with specific inductive biases given sufficient program samples. More recently, transformer-based LLMs such as GPT-3 [7] and T5 [8] have shown impressive performance in code generation tasks by leveraging contextual representations learned from large amounts of code, as well as public code sources and natural language data, to improve program synthesis. These approaches incorporate systematic pre-training and fine-tuning tasks to develop a deep understanding of code structure and meaning, making them well-suited for software development tasks. To evaluate the models for code generation tasks, different metrics are available such as $pass@k$

[9], which measures the percentage of problems solved using $k$ generated programs per problem, BLEU-4 [10], and exact match accuracy on program synthesis benchmarks such as APPS [11], MBPP [12], and CodeBLEU [8], which consider both syntactic and semantic matches based on code structure in addition to $N$-gram matches.

## 2. Code Completion

Code completion, also known as autocompletion, is a software development feature that suggests possible code completions as a programmer types [13]. Its goal is to save time and reduce errors by providing suggestions for method names, variable names, and even entire code snippets [14]. Previous research on code completion started with statistical language models [15][16]. Later, LSTM-based deep learning approaches were applied to the task, aiming to learn the semantic information of source code without considering its syntactic structure [17]. To address the limitations of LSTM-based language models, transformer architecture was introduced for code completion. Normally, the language models for code completion are trained using a causal language model that predicts the unknown token after a sequence of known tokens. Recent work on code completion using LLMs [9][18] has shown impressive performance on benchmarks, such as CodeXGLUE [19], compared to existing statistical language models and deep learning approaches.

## 3. Code Translation

Code translation is the process of converting code from one programming language to another, with the goal of migrating legacy software. While theoretically possible, building a code translator is challenging due to differences in syntax and platform APIs between programming languages. Most current translation tools are rule-based, requiring handcrafted rewrite rules applied to an abstract syntax tree (AST) derived from the input source code. However, creating such tools demands significant expertise in both the source and target languages. Recent studies have explored using statistical machine translation [20][21] as well as deep learning approaches [22][23] for programming language translation. Quality evaluation for generated functions often uses the BLEU score, while the exact match is used to compare generated output with reference ground truth.

## 4. Code Refinement

Code refinement, which can be referred to as automated program repair (APR), is the process of automatically fixing bugs or vulnerabilities by converting a buggy function into a correct one. Deep learning models have a strong learning capability that enables them to learn various patterns for transforming buggy programs into patched ones from large code corpora. Many studies [24][25] have demonstrated the superior performance of deep learning-based techniques over traditional template-based [26][27], heuristic-based [28][29][30], and constraint-based [31][32] APR techniques. LLM is used to generate plausible patches or modifications to a given incorrect code. The model can be trained on a large corpus of correct code to learn the patterns and structures of correct code. When LLMs are given a faulty code, the model can then generate suggestions for how to correct it as one of the downstream tasks. The LLMs for code refinement can be evaluated by CodeXGLUE [19] or HumanEval [9] as the abstracted codes or the classical APR benchmarks such as Defects4J [33] and QuixBugs [34] as real-world codes, but the understanding and generation of concrete variable and function names is still mandatory and challenging [35].

## 5. Code Summarization

Code summarization is a technique used to generate English descriptions of code snippets at the function level, which can then be used to generate documentation. Typically, this involves taking the source code as input and producing a natural language summary as output. In AI-assisted programming tools, code summarization can be used to analyze code and identify optimization opportunities, such as using a binary Euclid algorithm instead of a traditional modular arithmetic-based algorithm, which can significantly improve software performance. In recent years, there has been promising research into the automatic generation of natural language descriptions of programs, with studies such as [36][37][38] making notable progress in this area. The rise of deep learning, coupled with the abundance of data from open-source repositories, has made automatic code summarization an area of interest for researchers. Many of the neural approaches [39][40] use a sequence-to-sequence approach to generate source code summaries, with some models converting the source code into various types of representations, such as token-based [41][42], tree-based [43][44], and graph-based [45][46], before passing it through language models.

## 6. Defect Detection

As software systems increase in complexity, it becomes more challenging to identify errors. Defect detection aims to enhance software reliability by predicting whether a piece of code is susceptible to bugs or not, by detecting previously unknown errors. Rule-based approaches have been defined in existing defect detection frameworks by inferring likely programming rules from various sources such as code, version histories, and comments [23][47][48]. Statistical language models based on *N*-gram language models have also been widely used in this area [49][50][51]. More recently, many deep learning-based solutions [27][52][53][54][55][56][57] have been proposed to bridge the gap by suggesting different feature sets from which the detection framework can learn, attempting to imitate how a practitioner looks for vulnerabilities. However, LLMs, such as CodeBERT [58], have recently emerged as a promising technique in this field due to their ability to understand code structure. These models can be trained on a large corpus of error-free code and used to identify patterns and structures in source code that deviate from those learned from the error-free code as a binary classification task [59][60]. To evaluate the model predictions, accuracy, precision, recall, and F1 scores can be used.

## 7. Clone Detection

Clone detection involves identifying identical or similar code fragments, known as clones, within or across software systems. The goal of clone detection is to measure the similarity between two code snippets and determine if they have the same functionality. Clones can be classified into four types [61][62], with types 1–3 being syntactic clones that differ in minor ways, while type 4 clones, known as semantic clones, are difficult to detect since they have different syntax but the same semantics and, thus, require manual validation. With the increasing amount of source code, large-scale and automatic clone detection has become essential. Several tools have been developed to perform clone detection [63][64][65][66][67][68], using techniques such as comparison of the AST, tokens, or source code text. Notable clone detection datasets include BigCloneBench [69], which contains Java code snippets.

**Table 1.** Summary of language models for AI-assisted programming tasks.

| Framework | Year | Task(s) | Baseline(s) | Supported Language(s) | Open Sourced |
|---|---|---|---|---|---|
| Refactory [70] | 2019 | Defect Detection | BLEU | Java | ✗ |
| CuBERT [71] | 2020 | Code Refinement, Defect Detection | BERT | Python | ✓ |
| CugLM [72] | 2020 | Code Completion | BERT | Java, TypeScript | ✓ |
| Intellicode [73] | 2020 | Code Generation, Code Completion | GPT-2 | Python, C#, JavaScript, and TypeScrip | ✗ |
| Great [74] | 2020 | Defect Detection | Vanilla Transformers | Python | ✓ |
| TreeGEN [75] | 2020 | Code Generation | Vanilla Transformers | Python | ✓ |
| C-BERT [59] | 2020 | Defect Detection | BERT | C | ✗ |
| TransCoder [76] | 2020 | Code Translation | Vanilla Transformers | C++, Java, and Python | ✗ |
| GraphCodeBERT [77] | 2020 | Code Summarization, Code Refinement | BERT | Java | ✗ |
| Codex [9] | 2021 | Code Generation, Code Completion, Code Summarization, Benchmark | GPT-3 | JavaScript, Go, Perl, and 6 more | ✗ |

| Framework | Year | Task(s) | Baseline(s) | Supported Language(s) | Open Sourced |
|---|---|---|---|---|---|
| Copilot [78] | 2021 | Code Generation, Code Completion | Codex | Java, PHP, Python, and 5 more | ✗ |
| CodeT5 [79] | 2021 | Code Summarization, Code Generation, Code Translation, Code Refinement, Defect Detection, Clone Detection | T5 | Python, Java | ✓ |
| Tfix [80] | 2021 | Code Refinement, Defect Detection | T5 | JavaScript | ✓ |
| CodeRL [81] | 2021 | Code Summarization, Code Generation, Code Translation, Code Refinement, Defect Detection, Clone Detection | T5 | Java | ✓ |
| TreeBERT [82] | 2021 | Code Summarization | Vanilla Transformers | Python, Java | ✓ |
| BUGLAB [83] | 2021 | Code Refinement, Defect Detection | GREAT | Python | ✓ |
| TBCC [84] | 2021 | Clone Detection | Vanilla Transformers | C, Java | ✓ |
| APPS [11] | 2021 | Benchmark | N/A | Python | ✓ |

| Framework | Year | Task(s) | Baseline(s) | Supported Language(s) | Open Sourced |
|---|---|---|---|---|---|
| CodeXGLUE [19] | 2021 | Benchmark | N/A | Python | ✓ |
| CoTexT [85] | 2021 | Code Summarization, Code Generation, Code Refinement, Defect detection | T5 | Python, Java, Javascript, PHP, Ruby, Go | ✓ |
| SynCoBERT [86] | 2021 | Code Translation, Defect Detection, Clone Detection | BERT | Ruby, Javascript, Go, Python, Java, PHP | ✗ |
| TravTrans [87] | 2021 | Code Completion | Vanilla Transformers | Python | ✗ |
| CCAG [88] | 2021 | Code Completion | Vanilla Transformers | JavaScript, Python | ✗ |
| DeepDebug [89] | 2021 | Defect Detection | Reformer | Java | ✓ |
| Recoder [25] | 2021 | Defect Detection | TreeGen | Java | ✓ |
| PLBART [90] | 2021 | Code Summarization, Code Generation, Code Translation, Code Refinement, Clone Detection, Detect Detection | BART | Java, Python | ✗ |
| CODEGEN [91] | 2022 | Code Generation | GPT-NEO & GPT-J | Python | ✓ |

| Framework | Year | Task(s) | Baseline(s) | Supported Language(s) | Open Sourced |
|---|---|---|---|---|---|
| GPT-2 for APR [92] | 2022 | Code Refinement | GPT-2 | JavaScript | ✓ |
| CERT [93] | 2022 | Code Generation | CODEGEN | Python | ✓ |
| PyCoder [18] | 2022 | Code Generation | GPT-2 | Python | ✓ |
| AlphaCode [94] | 2022 | Code Generation | GPT | Java | ✗ |
| InCoder [95] | 2022 | Code Generation, Code Completion, Code Summarization | GPT-3 | Java, JavaScript, Python | ✓ |
| RewardRepair [96] | 2022 | Code Refinement, Defect Detection | T5 | Java | ✓ |
| CodeParrot [97] | 2022 | Code Generation | GPT-2 | Python | ✓ |
| AlphaRepair [98] | 2022 | Code Refinement, Defect Detection | CodeBERT | Java | ✓ |
| CodeReviewer [60] | 2022 | Code Summarization, Code Refinement, Defect Detection | CodeT5 | Java | ✓ |
| TransRepair [99] | 2022 | Code Refinement, Defect Detection | BLEU | Java | ✗ |
| NatGen [100] | 2022 | Code Generation, Code Translation, Code Refinement | CodeT5 | Java, Python, Go, JavaScript, Ruby, PHP | ✓ |

| Framework | Year | Task(s) | Baseline(s) | Supported Language(s) | Open Sourced |
|---|---|---|---|---|---|
| DualSC [101] | 2022 | Code Generation, Code Summarization | T5 | Shellcode | ✓ |
| VulRepair [102] | 2022 | Code Refinement, Defect Detection | T5 | C, C++ | ✓ |
| CoditT5 [103] | 2022 | Code Summarization, Defect Detection | CodeT5 | Java, Python, Ruby, PHP, Go, JavaScript | ✓ |
| C4 [104] | 2022 | Clone Detection | CodeBERT | C++, C#, Java, Python | ✓ |
| SPT-Code [105] | 2022 | Code Summarization, Code Completion, Code Refinement, Code Translation | CodeBERT & GraphCodeBERT | Python, Java, JavaScript, PHP, Go | ✓ |
| ExploitGen [106] | 2023 | Code Generation | CodeBERT | Python, Assembly | ✓ |
| Santacoder [107] | 2023 | Code Summarization, Code Generation | GPT-2 | Python, Java, and Javascript | ✓ |
| xCodeEval [108] | 2023 | Benchmark | N/A | Python, Java, C++, PHP, and 8 more | ✓ |
| StarCoder [109] | 2023 | Code Generation, Code Completion, Code | BERT & SantaCoder | HTML, Python, Java, and 83 | ✓ |

Res. 2020, 21, 5485–5551.

9. Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H.P.d.O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. Evaluating Large language Models Trained on Code. arXiv 2021,

| Framework | Year | Task(s) | Baseline(s) | Supported Language(s) | Open Sourced |
|---|---|---|---|---|---|
| | | Summarization | | more | |


11. Hendrycks, D.; Basart, S.; Kadavath, S.; Mazeika, M.; Arora, A.; Guo, E.; Burns, C.; Puranik, S.; He, H.; Song, D.; et al. Measuring Coding Challenge Competence With APPS. arXiv 2021, arXiv:2105.09938.
12. Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. Program Synthesis with Large Language Models. arXiv 2021, arXiv:2108.07732.
13. Dong, Y.; Gu, T.; Tian, Y.; Sun, C. SnR: Constraint-based Type Inference for Incomplete Java Code Snippets. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 25–27 May 2022; pp. 1982–1993.
14. Amazon, C. AI Code Generator—Amazon CodeWhisperer. Available online: https://aws.amazon.com/codewhisperer (accessed on 18 May 2023).
15. Robbes, R.; Lanza, M. How Program History Can Improve Code Completion. In Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, L'aquila, Italy, 15–16 September 2008; pp. 317–326.
16. Bruch, M.; Monperrus, M.; Mezini, M. Learning from Examples to Improve Code Completion Systems. In Proceedings of the 7th Joint Meeting of The European Software Engineering Conference and The ACM SIGSOFT Symposium on The Foundations of Software Engineering, Amsterdam, The Netherlands, 24–28 August 2009; pp. 213–222.
17. Svyatkovskiy, A.; Zhao, Y.; Fu, S.; Sundaresan, N. Pythia: Ai-assisted code completion system. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 2727–2735.
18. Takerngsaksiri, W.; Tantithamthavorn, C.; Li, Y.F. Syntax-Aware On-the-Fly Code Completion. arXiv 2022, arXiv:2211.04673.
19. Lu, S.; Guo, D.; Ren, S.; Huang, J.; Svyatkovskiy, A.; Blanco, A.; Clement, C.B.; Drain, D.; Jiang, D.; Tang, D.; et al. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. arXiv 2021, arXiv:2102.04664.
20. Koehn, P.; Federico, M.; Shen, W.; Bertoldi, N.; Bojar, O.; Callison-Burch, C.; Cowan, B.; Dyer, C.; Hoang, H.; Zens, R.; et al. Open Source Toolkit for Statistical Machine Translation: Factored Translation Models and Confusion Network Decoding. In Proceedings of the CLSP Summer Workshop Final Report WS-2006, Baltimore, MD, USA, 1 June–1 August 2007.

21. Artetxe, M.; Labaka, G.; Agirre, E. Unsupervised Statistical Machine Translation. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 31 October–4 November 2018.
22. Allamanis, M.; Barr, E.T.; Bird, C.; Sutton, C. Learning Natural Coding Conventions. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, China, 16–21 November 2014; pp. 281–293.
23. Acharya, M.; Xie, T.; Pei, J.; Xu, J. Mining API Patterns as Partial Orders from Source Code: From Usage Scenarios to Specifications. In Proceedings of the 6th Joint Meeting of The European Software Engineering Conference and The ACM SIGSOFT Symposium on The Foundations of Software Engineering, Dubrovnikm, Croatia, 3–7 September 2007; pp. 25–34.
24. Jiang, N.; Lutellier, T.; Tan, L. Cure: Code-aware Neural Machine Translation for Automatic Program Repair. In Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering, Madrid, Spain, 22–30 May 2021; pp. 1161–1173.
25. Zhu, Q.; Sun, Z.; Xiao, Y.a.; Zhang, W.; Yuan, K.; Xiong, Y.; Zhang, L. A Syntax-guided Edit Decoder for Neural Program Repair. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, 23–28 August 2021; pp. 341–353.
26. Jiang, J.; Xiong, Y.; Zhang, H.; Gao, Q.; Chen, X. Shaping Program Repair Space with Existing Patches and Similar Code. In Proceedings of the 27th ACM SIGSOFT International Symposium On Software Testing And Analysis, Amsterdam, The Netherlands, 16–21 July 2018; pp. 298–309.
27. Liu, K.; Koyuncu, A.; Kim, D.; Bissyandé, T.F. TBar: Revisiting Template-based Automated Program Repair. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, Beijing China, 15–19 July 2019; pp. 31–42.
28. Yuan, Y.; Banzhaf, W. Arja: Automated Repair of Java Programs via Multi-objective Genetic Programming. IEEE Trans. Softw. Eng. 2018, 46, 1040–1067.
29. Wen, M.; Chen, J.; Wu, R.; Hao, D.; Cheung, S.C. Context-aware patch generation for better automated program repair. In Proceedings of the 40th International Conference on Software Engineering, Gothenburg, Sweden, 27 May–3 June 2018; pp. 1–11.
30. Saha, R.K.; Lyu, Y.; Yoshida, H.; Prasad, M.R. Elixir: Effective Object-oriented Program Repair. In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, Urbana-Champaign, IL, USA, 30 October–3 November 2017; pp. 648–659.
31. Xiong, Y.; Wang, J.; Yan, R.; Zhang, J.; Han, S.; Huang, G.; Zhang, L. Precise Condition Synthesis for Program Repair. In Proceedings of the IEEE/ACM 39th International Conference on Software Engineering, Buenos Aires, Argentina, 20–28 May 2017; pp. 416–426.

32. Xuan, J.; Martinez, M.; Demarco, F.; Clement, M.; Marcote, S.L.; Durieux, T.; Le Berre, D.; Monperrus, M. Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs. IEEE Trans. Softw. Eng. 2016, 43, 34–55.
33. Just, R.; Jalali, D.; Ernst, M.D. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In Proceedings of the International Symposium on Software Testing and Analysis, San Jose, CA, USA, 21–25 July 2014; pp. 437–440.
34. Lin, D.; Koppel, J.; Chen, A.; Solar-Lezama, A. QuixBugs: A Multi-lingual Program Repair Benchmark Set Based on The Quixey Challenge. In Proceedings of the ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity, Vancouver, BC, Canada, 22–27 October 2017; pp. 55–56.
35. Jiang, N.; Liu, K.; Lutellier, T.; Tan, L. Impact of Code Language Models on Automated Program Repair. In Proceedings of the IEEE/ACM 45th International Conference on Software Engineering, Melbourne, Australia, 14–20 May 2023.
36. Sridhara, G.; Hill, E.; Muppaneni, D.; Pollock, L.; Vijay-Shanker, K. Towards Automatically Generating Summary Comments for Java Methods. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, 20–24 September 2010; pp. 43–52.
37. Moreno, L.; Aponte, J.; Sridhara, G.; Marcus, A.; Pollock, L.; Vijay-Shanker, K. Automatic Generation of Natural Language Summaries for Java Classes. In Proceedings of the 21st International Conference on Program Comprehension, San Francisco, CA, USA, 20–21 May 2013.
38. Sridhara, G.; Pollock, L.; Vijay-Shanker, K. Generating Parameter Comments and Integrating with Method Summaries. In Proceedings of the IEEE 19th International Conference on Program Comprehension, Kingston, ON, Canada, 22–24 June 2011; pp. 71–80.
39. Ahmad, W.; Chakraborty, S.; Ray, B.; Chang, K.W. A Transformer-based Approach for Source Code Summarization. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Virtual, 5–10 July 2020; pp. 4998–5007.
40. Iyer, S.; Konstas, I.; Cheung, A.; Zettlemoyer, L. Summarizing Source Code Using a Neural Attention Model. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, Berlin, Germany, 7–12 August 2016; pp. 2073–2083.
41. Allamanis, M.; Peng, H.; Sutton, C. A Convolutional Attention Network for Extreme Summarization of Source Code. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; pp. 2091–2100.
42. Chen, Q.; Zhou, M. A Neural Framework for Retrieval and Summarization of Source Code. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software

Engineering, Montpellier, France, 3–7 September 2018; pp. 826–831.

43. Mou, L.; Li, G.; Zhang, L.; Wang, T.; Jin, Z. Convolutional Neural Networks Over Tree Structures for Programming Language Processing. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
44. Liang, Y.; Zhu, K. Automatic Generation of Text Descriptive Comments for Code Blocks. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
45. Tufano, M.; Watson, C.; Bavota, G.; Di Penta, M.; White, M.; Poshyvanyk, D. Deep Learning Similarities From Different Representations of Source Code. In Proceedings of the 15th International Conference on Mining Software Repositories, Gothenburg, Sweden, 27 May–3 June 2018.
46. Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; Zhu, W. Asymmetric Transitivity Preserving Graph Embedding. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 1105–1114.
47. Livshits, B.; Zimmermann, T. Dynamine: Finding Common Error Patterns by Mining Software Revision Histories. ACM SIGSOFT Softw. Eng. Notes 2005, 30, 296–305.
48. Wasylkowski, A.; Zeller, A.; Lindig, C. Detecting Object Usage Anomalies. In Proceedings of the 6th Joint Meeting of The European Software Engineering Conference and The ACM SIGSOFT Symposium on The Foundations of Software Engineering, Dubrovnik, Croatia, 3–7 September 2007; pp. 35–44.
49. Charniak, E. Statistical Language Learning; MIT Press: Cambridge, MA, USA, 1996.
50. Nessa, S.; Abedin, M.; Wong, W.E.; Khan, L.; Qi, Y. Software Fault Localization Using N-gram Analysis. In Proceedings of the Wireless Algorithms, Systems, and Applications: 3rd International Conference, Dallas, TX, USA, 26–28 October 2008; pp. 548–559.
51. Wang, S.; Chollak, D.; Movshovitz-Attias, D.; Tan, L. Bugram: Bug Detection with N-gram Language Models. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, Singapore, 3–7 September 2016; pp. 708–719.
52. Lin, G.; Zhang, J.; Luo, W.; Pan, L.; Xiang, Y.; De Vel, O.; Montague, P. Cross-project Transfer Representation Learning for Vulnerable Function Discovery. IEEE Trans. Ind. Inform. 2018, 14, 3289–3297.
53. Li, Z.; Zou, D.; Xu, S.; Ou, X.; Jin, H.; Wang, S.; Deng, Z.; Zhong, Y. Vuldeepecker: A Deep Learning-based System for Vulnerability Detection. In Proceedings of the Network and Distributed Systems Security (NDSS) Symposium, San Diego, CA, USA, 18–21 February 2018.

54. Russell, R.; Kim, L.; Hamilton, L.; Lazovich, T.; Harer, J.; Ozdemir, O.; Ellingwood, P.; McConley, M. Automated Vulnerability Detection in Source Code Using Deep Representation Learning. In Proceedings of the 17th IEEE International Conference on Machine Learning and Applications, Orlando, FL, USA, 17–20 December 2018; pp. 757–762.
55. Le, T.; Nguyen, T.; Le, T.; Phung, D.; Montague, P.; De Vel, O.; Qu, L. Maximal Divergence Sequential Autoencoder for Binary Software Vulnerability Detection. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
56. Chen, Z.; Kommrusch, S.; Tufano, M.; Pouchet, L.N.; Poshyvanyk, D.; Monperrus, M. Sequencer: Sequence-to-sequence Learning for End-to-end Program Repair. IEEE Trans. Softw. Eng. 2019, 47, 1943–1959.
57. Gupta, R.; Pal, S.; Kanade, A.; Shevade, S. Deepfix: Fixing Common C Language Errors by Deep Learning. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
58. Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In Proceedings of the Findings of the Association for Computational Linguistics (EMNLP 2020), Virtual, 16–20 November 2020; pp. 1536–1547.
59. Buratti, L.; Pujar, S.; Bornea, M.; McCarley, S.; Zheng, Y.; Rossiello, G.; Morari, A.; Laredo, J.; Thost, V.; Zhuang, Y.; et al. Exploring Software Naturalness through Neural Language Models. arXiv 2020, arXiv:2006.12641.
60. Li, Z.; Lu, S.; Guo, D.; Duan, N.; Jannu, S.; Jenks, G.; Majumder, D.; Green, J.; Svyatkovskiy, A.; Fu, S.; et al. Automating Code Review Activities by Large-scale Pre-training. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 14–18 November 2022; pp. 1035–1047.
61. Bellon, S.; Koschke, R.; Antoniol, G.; Krinke, J.; Merlo, E. Comparison and Evaluation of Clone Detection Tools. IEEE Trans. Softw. Eng. 2007, 33, 577–591.
62. Roy, C.K.; Cordy, J.R. A Survey on Software Clone Detection Research. Queen’s Sch. Comput. TR 2007, 541, 64–68.
63. Kontogiannis, K.A.; DeMori, R.; Merlo, E.; Galler, M.; Bernstein, M. Pattern Matching for Clone and Concept Detection. Autom. Softw. Eng. 1996, 3, 77–108.
64. Ducasse, S.; Rieger, M.; Demeyer, S. A Language Independent Approach for Detecting Duplicated Code. In Proceedings of the IEEE International Conference on Software Maintenance, Oxford, UK, 30 August–3 September 1999; pp. 109–118.
65. Baxter, I.D.; Yahin, A.; Moura, L.; Sant’Anna, M.; Bier, L. Clone Detection using Abstract Syntax Trees. In Proceedings of the International Conference on Software Maintenance, Bethesda, MD,

USA, 16–19 November 1998; pp. 368–377.

66. Chen, K.; Liu, P.; Zhang, Y. Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets. In Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 31 May–7 June 2014; pp. 175–186.
67. Sajnani, H.; Saini, V.; Svajlenko, J.; Roy, C.K.; Lopes, C.V. Sourcerercc: Scaling code clone detection to big-code. In Proceedings of the 38th International Conference on Software Engineering, Austin, TX, USA, 14–22 May 2016; pp. 1157–1168.
68. Yu, H.; Lam, W.; Chen, L.; Li, G.; Xie, T.; Wang, Q. Neural Detection of Semantic Code Clones via Tree-based Convolution. In Proceedings of the IEEE/ACM 27th International Conference on Program Comprehension, Montreal, QC, Canada, 25–26 May 2019; pp. 70–80.
69. Svajlenko, J.; Roy, C.K. Description2Code Dataset. 2021. Available online: https://github.com/clonebench/BigCloneBench (accessed on 18 May 2023).
70. Hu, Y.; Ahmed, U.Z.; Mechtaev, S.; Leong, B.; Roychoudhury, A. Re-factoring based Program Repair applied to Programming Assignments. In Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, San Diego, CA, USA, 11–15 November 2019; pp. 388–398.
71. Kanade, A.; Maniatis, P.; Balakrishnan, G.; Shi, K. Learning and Evaluating Contextual Embedding of Source Code. In Proceedings of the International Conference on Machine Learning, Virtual, 13–18 July 2020; pp. 5110–5121.
72. Liu, F.; Li, G.; Zhao, Y.; Jin, Z. Multi-task Learning Based Pre-trained Language Model for Code Completion. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, Virtual, 21–25 September 2020; pp. 473–485.
73. Svyatkovskiy, A.; Deng, S.K.; Fu, S.; Sundaresan, N. Intellicode Compose: Code Generation Using Transformer. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual, 8–13 November 2020; pp. 1433–1443.
74. Hellendoorn, V.J.; Sutton, C.; Singh, R.; Maniatis, P.; Bieber, D. Global Relational Models of Source Code. In Proceedings of the International Conference on Learning Representations, Virtual, 26–30 April 2020.
75. Sun, Z.; Zhu, Q.; Xiong, Y.; Sun, Y.; Mou, L.; Zhang, L. Treegen: A Tree-based Transformer Architecture for Code Generation. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; pp. 8984–8991.
76. Roziere, B.; Lachaux, M.A.; Chanussot, L.; Lample, G. Unsupervised Translation of Programming Languages. Adv. Neural Inf. Process. Syst. 2020, 33, 20601–20611.

77. Guo, D.; Ren, S.; Lu, S.; Feng, Z.; Tang, D.; Liu, S.; Zhou, L.; Duan, N.; Svyatkovskiy, A.; Fu, S.; et al. GraphCodeBERT: Pre-training Code Representations with Data Flow. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 3–7 May 2021.
78. Friedman, N. Introducing GitHub Copilot: Your AI Pair Programmer. 2021. Available online: https://github.com/features/copilot (accessed on 18 May 2023).
79. Wang, Y.; Wang, W.; Joty, S.; Hoi, S.C. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Punta Cana, Dominican Republic, 7–11 November 2021; pp. 8696–8708.
80. Berabi, B.; He, J.; Raychev, V.; Vechev, M. Tfix: Learning to Fix Coding Errors with a Text-to-text Transformer. In Proceedings of the International Conference on Machine Learning. PMLR, Virtual, 18–24 July 2021; pp. 780–791.
81. Le, H.; Wang, Y.; Gotmare, A.D.; Savarese, S.; Hoi, S. CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. In Proceedings of the Advances in Neural Information Processing Systems 35 (NeurIPS 2022), New Orleans, LA, USA, 28 November 2022.
82. Jiang, X.; Zheng, Z.; Lyu, C.; Li, L.; Lyu, L. TreeBERT: A Tree-based Pre-trained Model for Programming Language. In Proceedings of the Uncertainty in Artificial Intelligence, Virtual, 27–30 July 2021; pp. 54–63.
83. Allamanis, M.; Jackson-Flux, H.; Brockschmidt, M. Self-supervised Bug Detection and Repair. In Proceedings of the Advances in Neural Information Processing Systems 34 (NeurIPS 2021), Virtual, 6–14 December 2021.
84. Hua, W.; Liu, G. Transformer-based Networks Over Tree Structures for Code Classification. Appl. Intell. 2022, 52, 8895–8909.
85. Phan, L.; Tran, H.; Le, D.; Nguyen, H.; Annibal, J.; Peltekian, A.; Ye, Y. CoTexT: Multi-task Learning with Code-Text Transformer. In Proceedings of the 1st Workshop on Natural Language Processing for Programming, Virtual, 6 August 2021; pp. 40–47.
86. Wang, X.; Wang, Y.; Mi, F.; Zhou, P.; Wan, Y.; Liu, X.; Li, L.; Wu, H.; Liu, J.; Jiang, X. SynCoBERT: Syntax-Guided Multi-Modal Contrastive Pre-Training for Code Representation. arXiv 2021, arXiv:2108.04556.
87. Kim, S.; Zhao, J.; Tian, Y.; Chandra, S. Code Prediction by Feeding Trees to Transformers. In Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering, Madrid, Spain, 22–30 May 2021; pp. 150–162.
88. Wang, Y.; Li, H. Code Completion by Modeling Flattened Abstract Syntax Trees as Graphs. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual, 2–9 February 2021; pp.

14015–14023.

89. Drain, D.; Clement, C.B.; Serrato, G.; Sundaresan, N. Deepdebug: Fixing Python Bugs Using Stack Traces, Backtranslation, and Code Skeletons. arXiv 2021, arXiv:2105.09352.
90. Ahmad, W.; Chakraborty, S.; Ray, B.; Chang, K.W. Unified Pre-training for Program Understanding and Generation. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Virtual, 6–11 June 2021; pp. 2655–2668.
91. Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; Xiong, C. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv 2022, arXiv:2203.13474.
92. Lajkó, M.; Csuvik, V.; Vidács, L. Towards Javascript Program Repair with Generative Pre-trained Transformer (GPT-2). In Proceedings of the 3rd International Workshop on Automated Program Repair, Pittsburgh, PA, USA, 19 May 2022; pp. 61–68.
93. Zan, D.; Chen, B.; Yang, D.; Lin, Z.; Kim, M.; Guan, B.; Wang, Y.; Chen, W.; Lou, J.G. CERT: Continual Pre-training on Sketches for Library-oriented Code Generation. In Proceedings of the 31st International Joint Conference on Artificia Intelligence (IJCAI-22), Vienna, Austria, 23–29 July 2022.
94. Li, Y.; Choi, D.; Chung, J.; Kushman, N.; Schrittwieser, J.; Leblond, R.; Eccles, T.; Keeling, J.; Gimeno, F.; Dal Lago, A.; et al. Competition-level Code Generation with Alphacode. Science 2022, 378, 1092–1097.
95. Fried, D.; Aghajanyan, A.; Lin, J.; Wang, S.; Wallace, E.; Shi, F.; Zhong, R.; Yih, W.t.; Zettlemoyer, L.; Lewis, M. Incoder: A Generative Model for Code Infilling and Synthesis. arXiv 2022, arXiv:2204.05999.
96. Ye, H.; Martinez, M.; Monperrus, M. Neural Program Repair with Execution-based Backpropagation. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 25–27 May 2022; pp. 1506–1518.
97. Tunstall, L.; Von Werra, L.; Wolf, T. Natural Language Processing with Transformers; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2022.
98. Xia, C.S.; Zhang, L. Less Training, More Repairing Please: Revisiting Automated Program Repair via Zero-shot Learning. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 14–18 November 2022; pp. 959–971.
99. Li, X.; Liu, S.; Feng, R.; Meng, G.; Xie, X.; Chen, K.; Liu, Y. TransRepair: Context-aware Program Repair for Compilation Errors. In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, Rochester, MI, USA, 10–14 October 2022; pp. 1–13.

100. Chakraborty, S.; Ahmed, T.; Ding, Y.; Devanbu, P.T.; Ray, B. NatGen: Generative Pre-training by “Naturalizing” Source Code. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 14–18 November 2022; pp. 18–30.
101. Yang, G.; Chen, X.; Zhou, Y.; Yu, C. Dualsc: Automatic Generation and Summarization of Shellcode via Transformer and Dual Learning. In Proceedings of the International Conference on Software Analysis, Evolution and Reengineering, Honolulu, HI, USA, 15–18 March 2022.
102. Fu, M.; Tantithamthavorn, C.; Le, T.; Nguyen, V.; Phung, D. VulRepair: A T5-based Automated Software Vulnerability Repair. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 14–18 November 2022; pp. 935–947.
103. Zhang, J.; Panthaplackel, S.; Nie, P.; Li, J.J.; Gligoric, M. CoditT5: Pretraining for Source Code and Natural Language Editing. In Proceedings of the International Conference on Automated Software Engineering, Rochester, MI, USA, 10–14 October 2022.
104. Tao, C.; Zhan, Q.; Hu, X.; Xia, X. C4: Contrastive Cross-language Code Clone Detection. In Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, Virtual, 16–17 May 2022; pp. 413–424.
105. Niu, C.; Li, C.; Ng, V.; Ge, J.; Huang, L.; Luo, B. SPT-code: Sequence-to-sequence Pre-training for Learning Source Code Representations. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 25–27 May 2022; pp. 2006–2018.
106. Yang, G.; Zhou, Y.; Chen, X.; Zhang, X.; Han, T.; Chen, T. ExploitGen: Template-augmented Exploit Code Generation based on CodeBERT. J. Syst. Softw. 2023, 197, 111577.
107. Allal, L.B.; Li, R.; Kocetkov, D.; Mou, C.; Akiki, C.; Ferrandis, C.M.; Muennighoff, N.; Mishra, M.; Gu, A.; Dey, M.; et al. SantaCoder: Don’t Reach for the Stars! arXiv 2023, arXiv:2301.03988.
108. Khan, M.A.M.; Bari, M.S.; Do, X.L.; Wang, W.; Parvez, M.R.; Joty, S. xCodeEval: A Large Scale Multilingual Multitask Benchmark for Code Understanding, Generation, Translation and Retrieval. arXiv 2023, arXiv:2303.03004.
109. Li, R.; Allal, L.B.; Zi, Y.; Muennighoff, N.; Kocetkov, D.; Mou, C.; Marone, M.; Akiki, C.; Li, J.; Chim, J.; et al. StarCoder: May the source be with you! arXiv 2023, arXiv:2305.06161.