# Systematic Literature Review on Solving Competitive Programming Problem with Artificial Intelligence (AI)

1st Francis Alexander
*Computer Science Department*
*School of Computer Science*
*Bina Nusantara University*
Jakarta, Indonesia 11480
francis.alexander@binus.ac.id

2nd Edwin Ario Abdiwijaya
*Computer Science Department*
*School of Computer Science*
*Bina Nusantara University*
Jakarta, Indonesia 11480
edwin.abdiwijaya@binus.ac.id

3rd Felix Pherry
*Computer Science Department*
*School of Computer Science*
*Bina Nusantara University*
Jakarta, Indonesia 11480
felix.pherry@binus.ac.id

4th Alexander Agung Santoso Gunawan
*Computer Science Department*
*School of Computer Science*
*Bina Nusantara University*
Jakarta, Indonesia 11480
aagung@binus.edu

5th Anderies
*Computer Science Department*
*School of Computer Science*
*Bina Nusantara University*
Jakarta, Indonesia 11480
anderies@binus.edu

2022 1st International Conference on Software Engineering and Information Technology (ICoSEIT) | 978-1-6654-7303-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICOSEIT55604.2022.10029949

***Abstract*— Computer programming has emerged in research, industry, and everyday life as a general-purpose problem-solving tool. From this expansion, there has been a steady rise in demand for tools that can help programmers be more productive or make programming easier using auto-completion and code-generation tools. Hence, the development of Artificial Intelligence (AI) systems capable of successfully understanding and generating code has the potential to change these tools and the way humans interact with them. Systems that produce code are not only beneficial but also serve as stepping-stones toward a better understanding of AI and how it relates to programming. One benchmark that is necessary for completing this development, is solving competitive programming (CP) problems which require a mastery of algorithms and complicated natural language, proves to be difficult for any AI model because it requires thinking like a human brain. In this study, the authors conducted a Systematic Literature Review (SLR) to better understand the evolution of program synthesis and also how different models such as AlphaCode and Codex perform when solving CP problems. In general, the authors find that even with the models that are specifically trained for solving CP problems, they still cannot think like a human when solving those problems. Thus, AI still has a long way to go before competing at the highest level of CP. From this literature review, it can be concluded that the code auto-completion and code-generation tools that are available now still do not meet the necessary benchmark which is solving CP tasks.**

***Keywords—Competitive Programming, Artificial Intelligence, Program Synthesis, Code Auto-Completion, and Code-Generation.***

## I. INTRODUCTION

Humanity has believed that we are the most intelligent living things for centuries. No other specimens or mechanisms could ever approach human intellectual abilities. However, things started to change recently. First, Garry Kasparov, a Russian chess grandmaster and former world number 1 for 255 months, lost to a chess engine, Deep Blue which was developed by IBM's scientist, in 1997. In 2017, poker players were dominated by Artificial Intelligence. In 2018, DOTA 2 video game veterans got steamrolled by an AI team [1]. It can be seen that human intelligence has started getting dominated in every single aspect. However, it turns out that there is one area that artificial intelligence has not completely conquered which is computer programming.

Computer programming is used in almost every aspect of society. Programming is an outstanding universal instrument with applications that are extensive in reach, spanning entertainment, health care, education, and more [23]. With the growth of demand in the computer programming fields, there has been a lot of demand to make tools that can make programmers more productive in doing their work. So, developing an AI which can understand code and write code just like we interact with programming languages would be such a useful tool. It could also be a stepping stone to improve the technology industry drastically [13].

One of the most important scopes to develop such a tool is competitive programming. Competitive programming is a mind sport where participants solve well-defined problems that require algorithms, data structure, and critical thinking. Competitive programming creates an opportunity to help train and improve programmers' skills, critical thinking, and problem-solving. Competitive programming provides complicated, complex problems using many aspects of computer science such as algorithms and data structures [13]. Also, competitive programming is an industry standard now in conducting technical interviews. Using these kinds of problems which are challenging for even humans will give a meaningful benchmark on how well an AI understands and generates code [13].

In this paper, the authors performed a systematic literature review (SLR) of prior studies to understand AI's ability to solve competitive programming problems. The authors wanted to know the limitations of the current AI models based on how well AI can comprehend, learn, and solve competitive programming problems. From this discussion, the authors also predicted whether the AI model that is available right now can compete at the highest level of competitive programming. The following is the next section of the paper's composition: Part 2 discusses our methods for conducting a literature review, and Part 3 presents the findings of a comprehensive systematic literature review (SLR). Part 4 continued our discussion of the research questions the authors had posed. Finally, chapter 5 concludes with conclusions, bringing our research to a close.

## II. METHODOLOGY

### A. Literature Search

This paper uses Systematic Literature Review (SLR) and Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA). The authors searched multiple websites to identify research papers which discuss the use of AI in solving competitive programming problems. The authors searched on Google Scholar and Arxiv. The keyword which was used to conduct this search was: Program Synthesis, AI in Competitive Programming, Code-generation. The authors also have three inclusion and exclusion criteria, including:

TABLE I. INCLUSION EXCLUSION CRITERIA

| Included Criteria | Excluded Criteria |
|---|---|
| Paper must be relevant about AI in solving Competitive Programming problems. | Paper is not relevant to help AI in solving competitive programming problems. |
| Paper must be internationally accepted. | Paper is using another language other than English. |
| Paper is published in 2017 - 2022. | Paper is published before 2017. |

### B. Research Question

In our study, the authors created six research questions to outline the specific goals of our study. These are the research questions:

TABLE II. RESEARCH QUESTIONS

| ID | Research Question | Motivation |
|---|---|---|
| RQ1 | What is the implementation of program synthesis in the real world? | To identify the implementation of program synthesis in the real world |
| RQ2 | How does code autocompletion assist developers and programmers to do their jobs? | To comprehend how code completion assist developers and programmers in their work |
| RQ3 | How does code-generation give an impact in the computer science field? | To comprehend how code-generation give an impact in the computer science field |
| RQ4 | What approaches does AI use when it comes to understanding competitive programming problem descriptions? | To identify the approaches that AI use to understand competitive programming problems descriptions |
| RQ5 | What approaches does AI employ when it comes to solving competitive programming problems? and which AI model gives the best result? | To identify the approaches that AI use to solve competitive programming problems and to determine which model performs the best |
| RQ6 | Can AI surpass humans in competitive programming in the near future? | To determine whether AI will soon be able to outperform humans in competitive programming. |

### C. Results

Based on our selection criteria, the authors examine the papers that are eligible based on the research questions after collecting the publications. The papers are as follows:

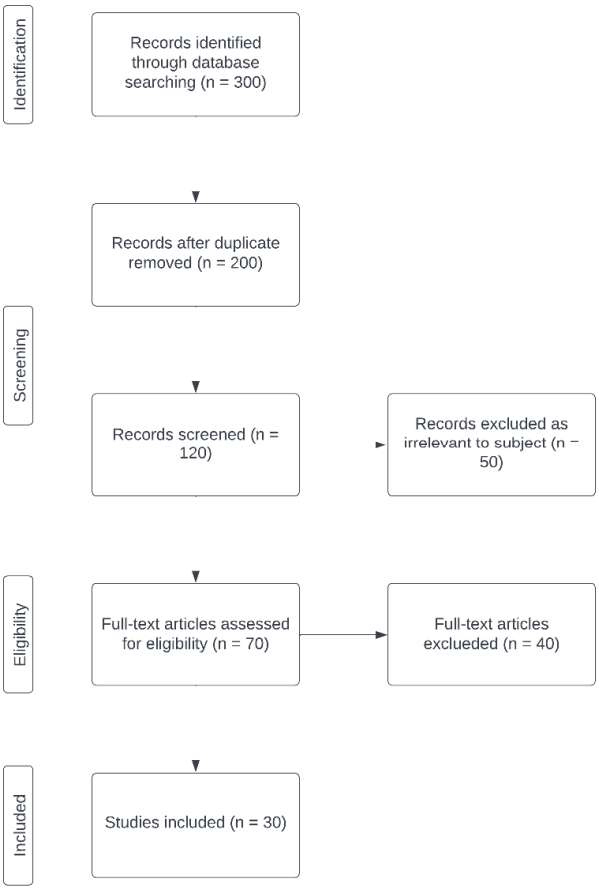


Fig. 1. PRISMA Method Flowchart

From Fig. 1, in the Identification part, the authors search based on required keywords in sites such as Google Scholar and Arxiv. In the Screening part, the authors perform criteria based on Table I. In the Eligibility part, from n=70 to n=30, the authors read papers thoroughly and decide whether the authors can use the papers for reference or not. In the Included part, there is a total of 30 papers that the authors used.

## III. RESULTS

From all the papers we have read, we have derived the outcomes of our study to answer our research question based on Table II.

### A. RQ1: What is the implementation of program synthesis in the real world?

As time passes, people are increasingly using devices daily. Available software has also aided humans in their daily lives, such as through cell phones or similar devices [4]. One of the most important tools to develop is autocompletion and also text generation. Some examples that we use daily are if we write a word incorrectly on social media, they will bring up suggestions for the correct possibility. Things like this help people to use computers more easily and more effectively. Programs that can produce these things are called language models. So far, many language models have been developed such as GPT-3 [19] which do not have a specific purpose but can do various things, such as autocomplete text, text generation, and even code-generation, although it is limited.

Then there is another language model that beats GPT-3 both in accuracy and memory, called RETRO [18].

Artificial Intelligence (AI) enables machines to learn from data, adapt new inputs, and generate predictions/models based on the given inputs. However, for artificial intelligence to be useful for humans, AI capabilities need to be aligned with what humans want [28]. One of the most popular AI applications in recent times is the synthesis program. Program synthesis is the task of automatically generating a program that is consistent with an instruction [27]. To build a neural model to construct a synthesis program, the model itself needs a deep vast understanding of searching algorithms to explore and learn previous programs into new programs [2]. The Program Synthesis Benchmark Suite (PSBS) has been used by researchers in genetic programming and other program synthesis disciplines to compare several elements of automated program synthesis systems during the last six years [26]. To put it another way, PSBS is a metric for assessing a model's synthesis program's capability.

Deep Learning (DL)-based natural language processing algorithms have advanced at breakneck speed. The recent advancement of DL in language modeling, machine translation, and paragraph understanding has made its promise in Software Engineering, notably in the field of program learning, cannot be ignored [24]. The vast bulk of software development involves visualizing or explaining the program's basic processes and logic. Translating thoughts into code is one of the most difficult components of programming, especially when working with new library APIs [25].

### B. *RQ2: How does code autocompletion assist developers and programmers to do their jobs?*

In today's society, many people are also starting to become developers and use IDEs to write code. Many of these developers also want to get help in solving or simplifying problems in programming. Bidirectional Encoder Representation from Transformation (BERT) is a Machine Learning (ML) model that can be implemented into various programming problems in the realm of Natural Language Processing (NLP). BERT can maintain the performance and results by using a few resources [8]. One of the deeper variations is CodeBERT. CodeBERT is similar to the BERT models or their associations, but CodeBERT provides a wider scope and does not focus solely on tasks or problems that only exist in NLP. CodeBERT can complete NLP tasks as well as Programming Language (PL) tasks in general [7]. The software referred to are GO, Java, JavaScript, PHP, Python, and Ruby languages.

With the growing demand for jobs in the technology industry, companies are demanding higher productivity standards for each of their developers [13]. Therefore, an Autocomplete Tool is also needed which is useful in the world of work so that programmers can focus more on making software than memorizing syntax. One of the Autocomplete Tools that is often used in the IDE is IntelliCode. IntelliCode uses one of the NLP domains, namely N-Gram so that it can provide advice to developers and help developers Autocomplete several lines of code directly at the same time [9]. However, some argue that this autocomplete is useless because it does not symbolize its use in the original world [16]. Therefore, fine-tuning of the autocompletion model in the IDE was carried out using datasets in the real world of work and obtained an increase of 13.8% for accuracy [16].

In addition to the Autocomplete Tool, as mentioned above, the language model can perform text generation if the training data is coded in a programming language and fine-tuned using the dataset, code-generation will also be possible [21]. There are already programs that can assist developers in answering probability and statistical questions, one of which is the Synthesis Program. The Synthesis program created with the help of OpenAI's Codex can answer and generate codes automatically based on probability and statistical questions [10]. The model or program can generate its code automatically, which can be understood using Deep Learning Reinforcement (DLR) [11]. This problem can also be solved with Generative Pre-trained Transformer 3 (GPT-3). GPT-3 is used to solve very simple math problems, such as addition, subtraction, multiplication, and division [5].

### C. *RQ3: How does code-generation give an impact in the computer science field?*

With the emergence of a model that can write code by itself in a particular language, of course, people start to try to apply it to solve problems such as in technical interviews or competitive programming. This was first done by DeepCoder where they succeeded in taking the first step in solving competitive programming problems, namely making a way for computers to understand the description of the problem along with the input and output given [17]. Later, it was further developed by the team from Google [15] and also Codex [21]. How the model created by the Google team works, it uses a language model left-to-right decoder-only transformer using a dataset that contains math problems and competitive programming questions. However, both models are inferior to the model made by DeepMind, named AlphaCode. AlphaCode itself has proven that it can beat 46% of the contestants on the code forces platform [13]. This is the only code-generation model that can compete with the average human doing competitive programming [13]. For how it works, all the descriptions of the questions are changed from test-based to text-to-text using the transformer model [14]. After that, the data will then be trained in the encoder-decoder transformer model [13].

Based on the above developments, large-scale language models have shown promise in generalizing to a wide range of cognitive tasks, including linguistic inference, common sense reasoning, logical deduction, mathematics, and a general understanding of various domains of human knowledge. However, whether large-scale language models can write code reliably remains an open question [23]. High-capacity language (LM) models have recently shown that machine learning models are capable of producing coherent and realistic text, but it is often difficult to guide them towards a specific goal, especially when explaining intent is complex or more expensive than generating target output manually. [22]. The dominant approach to text generation is to use an autoregressive model that studies maximally probable estimation (MLE) on the monitored data. However, this approach introduces two well-known differences between training and evaluation goals that lead to unwanted generation [29]. Then, some of these model languages have some satisfactory results based on certain benchmarks. However, almost all existing language models still do not care about data security [20]. That way, a lot of data can be retrieved by hackers if they attack in any form, even black-

box query access can get hundreds of data based on one training data sample [20].

### D. *RQ4: What approaches does AI use when it comes to understanding competitive programming problem descriptions?*

By using NLP (Natural Language Processing), AI can understand problem descriptions [21]. AI needs to understand language to process input-output. Human intellect is dependent on language; we use it as part of the system to perceive and communicate about things [30]. Humans' ability to comprehend and communicate about a subject improves over time and is based on the general principles of neural networks in biology: connection-based learning, distributed representation, and context-sensitive learning. Artificial neural networks are built on the same domain principle as humans and are employed in artificial language processing systems. Sometimes coding documentation may be in a language other than English, for example, Japanese. This problem can be solved by using one of the NLP domain methods, namely tokenization and detokenization, and using Neural Machine Translation (NMT) to tokenize sentences and perform translations [12]. Neural Machine Translation (NMT) correctness can be tested with SoftMax Tempering to train NMT more accurately [6].

One of the methods that AI uses to understand text or competitive programming problem descriptions is from the NLP branch of BERT (Bidirectional Encoder Representations from Transformers). BERT tokenizes the input, then by using WordPiece embeddings, models/BERT can understand the text or the competitive programming problem descriptions and give the output [8]. There are 2 steps in the BERT model, namely pre-training and fine-tuning [8]. At the pre-training stage, the model will be trained with data that does not have a label. The data will be processed in 2 pre-training tasks, namely Masked LM (MLM) and Next Sentence Prediction (NSP).

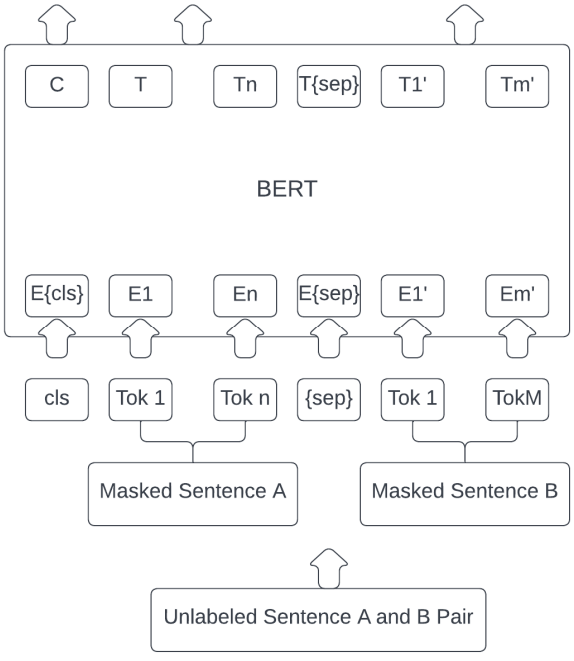


Fig. 2. BERT Pre-training Diagram

In the MLM task, the model masked the token into a random sequence. To train the data in two ways or bidirectional, this can be done by masking some small portion of a random token, then the token predicts itself. In this way, the model can learn more specifically to create a variety of understandings with great output. While in the NSP task, this task holds important things that are Question Answering (QA) and Natural Languages Interface (NLI). Both QA and NLI do not comprehend the relationship between both QA and NLI with the sentences. To fix this, the model labeled the data to guide the model itself to predict the next sentence/sequence token. In the fine-tuning stage, this task still uses the same structures as in the pre-trained step. The model will receive specific input and specific output, then the model fine-tuned all the processes end-to-end.

### E. *RQ5: What approaches does AI employ when it comes to solving competitive programming problems? and which AI model gives the best result?*

For now, there are two main AIs that have the ability to solve some competition-style programming problems, which are AlphaCode by Deepmind and Codex by OpenAI.

AlphaCode's AI uses an encoder-decoder transformer model, then it uses next-token prediction loss for the encoder and masked language modeling loss for the decoder [13]. This model will then be pre-trained using the GitHub dataset, which consists of several files written in popular programming languages such as C++, Python, and Ruby [13]. Next, the model will go to fine-tuning, where it will use a similar procedure as the one they used in the pre-training phase but they will add three optimization techniques, which are: tempering, value conditioning & prediction, and GOLD [13]. After the model has been through the learning phase, it will face another dataset that was scraped from the codeforces problem set. Then the model will produce a lot of possible solutions.

All of these possible solutions will then be checked against the sample input and output given in the problem description. The ones that are not correct will be deleted, and the other ones will go through the next step, clustering. In the clustering method, AI will generate a bunch of inputs corresponding to the problem description [13]. This input will be tested against all of the possible solutions, and the AI will do clustering based on the input the model gives. After clustering all the models, AI will choose a total of ten solutions from all of the clusters, and then AI will submit all ten to the judge. If one of the solutions is accepted, then the AI will consider that task successfully done.

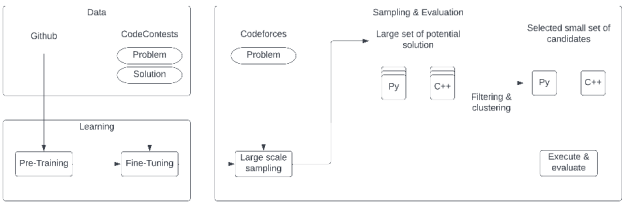


Fig. 3. AlphaCode Process Diagram

Codex works in a way that breaks down a problem into a set of simpler problems [21]. After the problem is divided into subparts, the interface uses existing libraries, APIs, and structures to map those problems. Mapping the problems into existing codes is a menial job, and that's where Codex comes

in handy. Codex is specially designed to be applied to any programming task, including competitive programming.

Both AlphaCode's AI and Codex have the same purpose: to solve programming problems, including competitive programming problems. Codex shows that codex is superior in solving easy programming problems with a success rate of around 25% than AlphaCode's AI with a success rate of around 17% [13]. AlphaCode's AI shows that AlphaCode's AI is superior in solving interview programming problems with a success rate of around 5%–9% than Codex, with a success rate of around 3% [13]. When it comes to competitive programming problems, AlphaCode's AI is better than Codex's. AlphaCode's AI in solving competitive programming problems shows a success rate of 4%-7%, while Codex shows a success rate of 0%-3%.

### F. *RQ6: Can AI surpass humans in competitive programming in the near future?*

In general, AI will never outperform humans; specifically, AI intelligence will behave similarly to human intelligence [3]. However, there is a chance that AI can surpass humans in specific fields. This has happened in many fields, such as chess, poker, and even in complex MOBA games such as DOTA 2. Competitive programming is also not immune to this threat, just like many other fields/jobs in the world.

When compared to other competitive programmers, Alphacode's AI performance at solving competitive programming problems is quite impressive. They used a site called Codeforces to test their AI in ten different competitive programming contests. As a result, they have consistently outperformed 46% of all contestants. After participating in ten of Codeforces' contests, they received a total rating of 1238, which is greater than 72 percent of the people who are actively participating in these contests [13].

Looking at the performance of AlphaCode's AI, it is quite impressive. But, when it comes to solving problems, this AI is still far from decent. AlphaCode's AI does not focus on improving itself to think and solve competitive programming problems; it just generates billions of lines of code, most of which do not even run, and they will do some clustering and filtering to find 10 of the best lines to submit to the contest platform. There are some submissions on the Codeforces platform where the AI submits C++ code to a Python compiler, which means that the AI does not even understand what language it is writing in. In conclusion, AlphaCode's AI has surpassed some competitive programmers, but it still has a long way to go before it can challenge the greatest competitive programmers in the world.

## IV. Discussion

Technology is advancing at a breakneck pace, causing dramatic changes in the field of computer science. As evidence, more people are utilizing smartphones regularly as time goes on. Humans have also benefited from the available software, such as through smartphones or such devices. For instance, if we misspell a word on social media, they will suggest alternatives. This feature is also important in developing software, especially when the developer is not familiar with a particular language / framework. As a result, in the workplace, a code autocomplete and code-generation tool is needed so that programmers can just focus on building software rather than learning syntax.

Knowing that two of the most critical parts in improving software developer productivity are code-generation and code autocomplete, it is intuitive to find a good benchmark before publishing these tools to the public. The best way to do this is to tackle challenges like technical interviews or competitive programming with a model that can generate code in a certain language on its own. DeepCoder was the first to do so, and they were able to take the first step in tackling competitive programming difficulties by establishing a way for computers to understand problem descriptions as well as input and output.

To understand competitive programming problem descriptions, AI uses natural language processing (NLP) to assist them in comprehending the problems. When the problem description isn't in English, AI won't have any issues. Tokenize the sentences, execute translations, and finally detokenize are all done by AI using Neural Machine Translation (NMT). BERT, an approach from the NLP branch, is used by AI to aid in the understanding of competitive programming problem descriptions. After analyzing the problems, Deepmind's AlphaCode and OpenAI's Codex can generate the solutions. To solve the problem, AlphaCode employs an encoder-decoder transformer model, whereas Codex divides the problem into smaller parts and solves them as separate problem sets.

AI has showed great achievements by defeating pro-chess players, dominating in poker games, and defeating pro-team DOTA 2 players. When it comes to solving competitive programming problems, AlphaCode does not try to improve its skill to solve those problems but rather just generating more and more code which most of them cannot be executed and optimizing the clustering and filtering technique to find the top 10 best solutions generated.

## V. Conclusion

Natural language processing allows AI to comprehend human speech (NLP). It's being used to interpret competitive programming issue descriptions in this case. AI may automatically develop programs/codes based on problem descriptions by comprehending problem descriptions. AlphaCode and Codex are two examples of AI models that can be used to solve competitive programming problems. Both AI models can solve a range of introduction or easy programming problems to competitive or hard programming problems. To see if the code is correct, the code will be tested with multiple test cases from the problems, and if all of the output from the generated code is the same as the output from the judge then it will be a valid solution.

From those two examples of code-generation AI, the authors can conclude that AlphaCode is way superior compared to Codex in competitive programming questions. AlphaCode even beat 72% of active competitive programmers on the Codeforces platform, which is highly impressive. However, none of this AI can solve problems such as human brain work. Therefore, the authors can conclude that AI still has a long way to go until AI can compete with the highest level of competitive programmers in the world. Also, most existing language models are still unconcerned with data security. In the future, if they continue to use this method, hackers can attack in any way. They can gain a lot of data. Even black-box query access can get hundreds of data based on a single training data sample.

Future research should focus on finding a means to safeguard the data and also improve the language models' accuracy.

## REFERENCES


[1] Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., ... & Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680 Available from https://arxiv.org/abs/1912.06680

[2] Shi, K., Dai, H., Ellis, K., & Sutton, C. (2022). CrossBeam: Learning to search in bottom-up program synthesis. arXiv preprint arXiv:2203.10452. Available from https://arxiv.org/abs/2203.10452

[3] Bartholomew, B. (2020). Why AI Will Never Surpass Human Intelligence. Journal of Consciousness Exploration & Research, 11.

[4] Clement, C. B., Drain, D., Timcheck, J., Svyatkovskiy, A., & Sundaresan, N. (2020). PyMT5: multi-mode translation of natural language and Python code with transformers. arXiv preprint arXiv:2010.03150. Available from https://arxiv.org/abs/2010.03150

[5] Cobbe, K., Kosaraju, V., Bavarian, M., Hilton, J., Nakano, R., Hesse, C., & Schulman, J. (2021). Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168. Available from https://arxiv.org/abs/2110.14168

[6] Dabre, R., & Fujita, A. (2020). Softmax Tempering for Training Neural Machine Translation Models. arXiv preprint arXiv:2009.09372. Available from https://arxiv.org/abs/2009.09372

[7] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., ... & Zhou, M. (2020). Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155 Available from https://arxiv.org/abs/2002.08155

[8] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 Available from https://arxiv.org/abs/1810.04805

[9] Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan. 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, pages 1433–1443, 2020 Available from https://doi.org/10.48550/arXiv.2005.08025

[10] Tang, E. Ke, N. Singh, N. Verma, and I. Drori. Solving probability and statistics problems by program synthesis. arXiv preprint arXiv:2111.08267, 2021 Available from https://doi.org/10.48550/arXiv.2111.08267

[11] Trivedi, J. Zhang, S.-H. Sun, and J. J. Lim. Learning to synthesize programs as interpretable and generalizable policies. Advances in neural information processing systems, 2021 Available from https://doi.org/10.48550/arXiv.2108.13643

[12] Kudo, T., & Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. arXiv preprint arXiv:1808.06226 Available from https://arxiv.org/abs/1808.06226

[13] Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., ... & Vinyals, O. (2022). Competition-level code-generation with alphacode. arXiv preprint arXiv:2203.07814. Available from https://arxiv.org/abs/2203.07814

[14] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv preprint arXiv:1910.10683. Available from https://arxiv.org/abs/1910.10683

[15] Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., ... & Sutton, C. (2021). Program synthesis with large language models. arXiv preprint arXiv:2108.07732. Available from https://arxiv.org/abs/2108.07732

[16] Aye, G. A., Kim, S., & Li, H. (2021, May). Learning autocompletion from real-world datasets. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 131-139). IEEE. Available from https://arxiv.org/abs/2011.04542

[17] Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2017). Deepcoder: Learning to write programs. arXiv preprint arXiv:1611.01989. Available from https://arxiv.org/abs/1611.01989

[18] Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., ... & Sifre, L. (2021). Improving language models by retrieving from trillions of tokens. arXiv preprint arXiv:2112.04426. Available from https://arxiv.org/abs/2112.04426

[19] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901. Available from https://arxiv.org/abs/2005.14165

[20] Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., ... & Raffel, C. (2021). Extracting training data from large language models. In 30th USENIX Security Symposium (USENIX Security 21) (pp. 2633-2650). Available from https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting

[21] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. Available from https://arxiv.org/abs/2107.03374

[22] Guo, D., Svyatkovskiy, A., Yin, J., Duan, N., Brockschmidt, M., & Allamanis, M. (2021). Learning to Generate Code Sketches. arXiv preprint arXiv:2106.10158. Available from https://arxiv.org/abs/2106.10158

[23] Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., ... & Steinhardt, J. (2021). Measuring coding challenge competence with apps. arXiv preprint arXiv:2105.09938. Available from https://arxiv.org/abs/2105.09938

[24] Le, T. H., Chen, H., & Babar, M. A. (2020). Deep learning for source code modeling and generation: Models, applications, and challenges. ACM Computing Surveys (CSUR), 53(3), 1-38. Available from https://arxiv.org/abs/2002.05442

[25] Xu, F. F., Vasilescu, B., & Neubig, G. (2021). In-ide code-generation from natural language: Promise and challenges. arXiv preprint arXiv:2101.11149. Available from https://arxiv.org/abs/2101.11149

[26] Helmuth, T., & Kelly, P. (2021, June). PSB2: the second program synthesis benchmark suite. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 785-794). Available from https://arxiv.org/abs/2106.06086

[27] Bunel, R., Hausknecht, M., Devlin, J., Singh, R., & Kohli, P. (2018). Leveraging grammar and reinforcement learning for neural program synthesis. arXiv preprint arXiv:1805.04276. Available from https://arxiv.org/abs/1805.04276

[28] Kenton, Z., Everitt, T., Weidinger, L., Gabriel, I., Mikulik, V., & Irving, G. (2021). Alignment of language agents. arXiv preprint arXiv:2103.14659. Available from https://arxiv.org/abs/2103.14659

[29] Pang, R. Y., & He, H. (2020). Text generation by learning from demonstrations. arXiv preprint arXiv:2009.07839. Available from https://arxiv.org/abs/2009.07839

[30] McClelland, J. L., Hill, F., Rudolph, M., Baldridge, J., & Schütze, H. (2019). Extending machine language models toward human-level language understanding. arXiv preprint arXiv:1912.05877. Available from https://arxiv.org/abs/1912.05877