Java Multi-Lingual Code Generation

Ronit Singh – Pramit Saha

RonitSingh2158@gmail.com

Abstract. Generating code from natural language descriptions is a complex challenge in the field of Natural Language Processing (NLP). This task requires a deep understanding of both the syntax and semantics of the target programming language. Java Code Generation, in particular, focuses on automatically creating Java code based on text descriptions. By providing quick solutions to simple and routine programming tasks, this NLP application has the potential to significantly boost the productivity of Java developers. Although there have been numerous attempts to tackle this intricate problem, existing text-to-code generation systems mainly rely on English-language datasets. In this study, we present the first multi-lingual code generation systems, capable of handling five different languages: Spanish, French, German, Japanese, and Hindi. We also created multilingual versions of the well-known CONCODE dataset and made them publicly available. By doing so, we aim to encourage and facilitate further research in code generation from multiple languages, enabling the research community to continue making strides in this exciting and promising field.

1 Introduction

In software development, the emergence of code generation from natural language has marked a significant milestone. This advancement has the potential to revolutionize the way developers work, making the development process more efficient and accessible. By allowing developers to express their ideas in natural language and automatically generating corresponding code, code generation tools can significantly reduce the time and effort required for software development, particularly for tasks that are repetitive or require a substantial amount of boilerplate code.

The impact of code generation from natural language extends beyond saving time and effort. It can also help bridge the gap between the intention of the developer and the actual implementation. Many developers, especially those who are new to programming or working on unfamiliar codebases, often struggle with translating

their ideas into syntactically correct and efficient code. Code generation tools can assist in this process by providing a more intuitive and user-friendly interface for expressing programming logic, reducing the cognitive load on developers and allowing them to focus on higher-level concepts and problem solving.

Moreover, the generation of natural language code has the potential to democratize software development by making it more accessible to individuals of diverse backgrounds and skill levels. Traditional programming languages often have steep learning curves and require a significant amount of time and effort to master. Using natural language, which is more familiar and intuitive to most people, code generation tools can lower the barrier to entry for programming and enable a wider range of individuals to participate in software development. This inclusivity can lead to more diverse perspectives and ideas in the software development community, fostering innovation and creativity.

However, despite the promising potential of code generation from natural language, there is a significant challenge that limits its widespread adoption and usability: the language barrier. The vast majority of existing code generation models and tools are primarily designed and trained on English datasets, assuming that the input natural language will be English. This assumption poses a significant barrier for developers who are not native English speakers or who prefer to communicate in their native languages. The lack of multilingual support in code generation tools hinders their accessibility and usefulness for a global audience, preventing developers from leveraging these powerful tools effectively.

Recognizing this critical gap in the field, we present our groundbreaking work on developing the first five multilingual datasets specifically designed for code generation. Our approach involves taking the widely used CONCODE dataset, which consists of English natural language descriptions and their corresponding Java code snippets, and translating it into five different languages: Spanish, French, German, Japanese, and Hindi. By creating these multilingual datasets, we aim to enable the development of code generation models that can handle a variety of natural languages, making the tools more inclusive and accessible to developers around the world.

Our work represents a significant step towards breaking the language barrier in code generation and empowering developers across the globe to leverage these powerful tools effectively. The multilingual datasets we have created will be made publicly available to the research community, fostering further advancements and innovations in this field. We believe that our contributions will pave the way for more inclusive and accessible code generation tools, ultimately leading to a more diverse and efficient software development ecosystem.

In the following sections, we will dive into the details of our approach, including the methodology used for the translation of the data set, the challenges encountered, and the potential implications of our work. We will also discuss the future directions and opportunities that our multilingual datasets open up for the field of code generation from natural language.

2 Background and Related Work

Code generation from natural language has gained significant attention in recent years due to its potential to streamline software development processes. Early work in this area focused on generating code from structured specifications or domain-specific languages [3, 8]. However, these approaches required developers to learn and adhere to specific syntax and rules, limiting their usability and flexibility.

To address these limitations, researchers began exploring the use of natural language processing (NLP) techniques for code generation. Iyer et al. [5] proposed a neural machine translation approach that learns to map natural language descriptions to corresponding code snippets. Their model, trained on a dataset of Python code and associated comments, demonstrated the feasibility of generating code from free-form text.

Subsequent studies have focused on improving the quality and accuracy of generated code. Yin and Neubig [9] introduced a syntax-aware neural model that explicitly captures the hierarchical structure of code. By incorporating abstract syntax trees (ASTs) into the generation process, their approach achieved better syntactic correctness and code quality compared to sequence-to-sequence models.

More recently, pre-trained language models have been adapted for code generation tasks. Feng et al. [2] proposed CodeBERT, a pre-trained model that learns from a large corpus of programming languages and natural language descriptions. CodeBERT has shown promising results in various code-related tasks, including code generation, code summarization, and code search.

Several studies have explored the use of retrieval-based techniques to enhance code generation. Hashimoto et al. [4] proposed a retrieve-and-edit framework that retrieves relevant code snippets from a database and then edits them to match the given natural language description. This approach takes advantage of the vast amount of existing code available online and has shown improvements in the quality and diversity of generated code.

Another line of research focuses on generating code from multiple input modalities, such as natural language and examples. Chen et al. [1] introduced a system that generates Python code from a combination of natural language descriptions and input-output examples. By leveraging both modalities, their approach achieves better accuracy and generalizability compared to using either modality alone.

Despite these advancements, the majority of existing code generation models are trained and evaluated on English datasets. This limits their applicability to developers who primarily communicate in other languages. Few studies have explored multilingual code generation. Karaivanov et al. [6] proposed a framework for generating code from natural language descriptions in multiple programming languages. However, their approach relies on language-specific parsers and does not address the challenge of handling multiple natural languages.

Some recent work has started to address the multilingual aspect of code generation. Nguyen et al. [7] proposed a multilingual code generation model that can handle both English and Vietnamese input descriptions. Their approach involves translating the Vietnamese descriptions into English using machine translation techniques and then generating code using an English-based code generation model. While this study demonstrates the feasibility of multilingual code generation, it relies on an intermediate translation step, which may introduce errors and limit the model's performance.

In this context, our work aims to bridge the gap by creating multilingual datasets for code generation. By translating the widely-used CONCODE dataset into five different languages, we enable the development and evaluation of code generation models that can handle a variety of natural languages. Our datasets will foster research on multilingual code generation and contribute to the creation of more inclusive and accessible tools for developers worldwide.

3 Problem Formulation

The global coding community is characterized by significant linguistic diversity, with a large proportion of the estimated 28 million professional coders worldwide being non-native English speakers. This diversity is further amplified by the growing population of learners in non-English speaking countries who are striving to learn computer science. However, the field of computer science has historically been anglocentric, with most resources and terminologies being in English. This poses a significant challenge for non-English speakers, who often have to grapple with the dual task of learning both computer science concepts and the English language simultaneously.

Despite the increasing importance of natural language processing in computer science and its potential to make coding more accessible and intuitive, there is a lack of research and resources specifically targeting multilingual code generation. Existing code generation models and datasets primarily focus on English, limiting their applicability and usefulness for non-English speaking coders and learners.

To address this gap and contribute to a more inclusive and accessible coding environment, we focus on developing multilingual datasets and benchmarks for code generation in the Java programming language. Our work aims to support and encourage linguistic diversity in the field of computer science by providing resources that enable the development and evaluation of code generation models across different natural languages.

Specifically, our research objectives are:

1. To create multilingual datasets for code generation by translating the widely-used CONCODE dataset from English to five other languages: Spanish, French, German, Japanese, and Hindi.

2. To establish benchmarks for evaluating the performance of code generation models across these languages.
3. To investigate the challenges and opportunities associated with multilingual code generation and provide insights for future research and development in this area.

By addressing these objectives, we aim to contribute to the development of more inclusive and accessible coding tools and resources, ultimately fostering greater participation and innovation in the global coding community.

4 Material and Method

4.1 CONCODE Dataset

The foundation of our research is the CONCODE dataset, a well-established resource in the realm of code generation [5]. This dataset comprises 100,000 training samples, 2,000 test samples, and 2,000 validation samples, providing a vast array of examples that help minimize extraneous errors and enhance the robustness of our models. The CONCODE dataset contains a diverse range of code examples, including snippets from various programming languages, such as Python, Java, and C++, along with their corresponding natural language descriptions. The dataset was originally created by mining open-source repositories on GitHub and manually annotating the code snippets with their respective descriptions.

4.2 Data Preprocessing

Our first step was to preprocess the CONCODE dataset. We identified and removed extraneous special tokens, such as con_elem_sep, using a custom program we developed in Google Colaboratory. This step was crucial in streamlining the dataset and ensuring its compatibility with our models. Here's a code snippet illustrating the removal of special tokens:

```
def remove_special_tokens(text):
    return re.sub(r'con_elem_sep', '', text)
```

However, during the data loading process, we encountered an issue with extraneous symbols, such as backslashes, appearing in the data. Upon analyzing the data for patterns, we discovered that the lines causing errors were appearing at inconsistent intervals. To address this, we developed another piece of code to identify any lines causing issues during the model reading process and manually edited out the symbols:

```
def identify_problematic_lines(file_path):
    with open(file_path, 'r') as file:
        for line_num, line in enumerate(file, 1):
            try:
                json.loads(line)
            except json.JSONDecodeError:
                print(f"Line {line_num}: {line}")
```

This preprocessing procedure was repeated five times, corresponding to the five different natural languages - Spanish, French, German, Japanese, and Hindi - that we aimed to incorporate into our model. We observed that some languages, such as Japanese, caused more errors, while others, such as Hindi, resulted in fewer errors.

4.3 Overall Idea

Code generation, a critical application of machine learning, has been the focus of numerous studies and has found its place in various datasets, such as CodeXGlue/Concode. This process involves the automatic creation of source code based on a given set of specifications, which can significantly enhance the efficiency of software development.

In our research, we have embarked on an ambitious journey to extend the capabilities of code generation systems beyond the confines of the English language. Our primary objective is to establish a robust metric for evaluating the performance of code generation in several natural languages, namely Spanish, French, German, Japanese, and Hindi. This multilingual approach aims to make code generation more accessible and inclusive, thereby catering to a diverse global community of developers.

The importance of multilingual code generation lies in its potential to bridge the language barrier and empower developers worldwide to leverage the benefits of code generation, regardless of their native language. By enabling developers to express their ideas and requirements in their preferred language, multilingual code generation can foster increased collaboration, creativity, and productivity within the global developer community.

Furthermore, we aspire to set a preliminary benchmark for multilingual code generation. This benchmark will serve as a reference point for the code generation community, enabling researchers and developers to gauge the effectiveness of their models and strive for continual improvement. By providing this benchmark, we hope to foster a culture of excellence and innovation in the field of code generation, encouraging the community to push the boundaries of what is possible in this exciting domain.

4.4 Datasets

In this research paper, we leverage the CONCODE dataset, a resource generously provided by [5]. This dataset is a robust collection of data, comprising 100,000 training samples, 2,000 test samples, and 2,000 validation samples. The CONCODE dataset is an exceptional resource for code generation tasks, given its extensive variety of output code. It has been utilized in numerous scholarly papers, demonstrating its value and reliability in the field.

Our research task necessitates the use of multiple languages as inputs for our model. Consequently, we undertook the task of translating the English-based dataset into several languages, namely French, Spanish, Hindi, German, and Japanese. To accomplish this, we employed the translation function provided by TXTAI, a powerful text AI library. TXTAI is a comprehensive library that offers a wide range of text processing capabilities, including text embedding, similarity search, and translation. The translation function in TXTAI utilizes state-of-the-art neural machine translation models, such as the Transformer architecture, to provide accurate and fluent translations between languages.

We translated all natural language (NL) components of each given CodeXGlue dataset. Subsequently, we integrated the original

code and our newly translated NL into the same formatting structure. This process ensured the preservation of the original dataset's structure while accommodating the newly translated components. Here's an example of the translation process using TXTAI:

```
from TXTAI.pipeline import Translation

translator = Translation()
translated_text = translator("Hello, how are you?", "es")
print(translated_text)  # Output: "Hola, ¿cómo estás?"
```

During this process, we made the decision to remove certain special tokens, such as con_elem_sep. Our analysis indicated that these tokens unnecessarily lengthened the natural language section of the datasets. Furthermore, their contribution to improving the training process was minimal. Therefore, their removal streamlined our dataset without compromising the effectiveness of our training process.

4.5 Natural Language Translation

In the current scope of our research, we are focusing on five distinct languages as inputs to our model: Spanish, French, German, Japanese, and Hindi. The choice of these languages is strategic, aiming to cover a broad spectrum of linguistic families and thus ensure the versatility and applicability of our model across diverse linguistic contexts.

Given the inherent complexity of natural languages and the intricate syntactic and semantic rules they follow, creating an encoder and decoder that can understand languages other than English is a significant challenge. Each language has its unique grammatical structures, idiomatic expressions, and cultural nuances, all of which need to be accurately captured and translated into English. For example, Spanish and French have gendered nouns and adjectives, while Japanese employs a complex system of honorifics that reflect social hierarchies and relationships. Hindi, on the other hand, has a relatively free word order compared to English, which can pose challenges for accurate translation.

To address these challenges, we have leveraged the translation capabilities of TXTAI, a state-of-the-art tool that employs advanced machine learning algorithms and large-scale language models to provide highly accurate translations. TXTAI has been extensively covered in several research papers (to be added), demonstrating its effectiveness in preserving the essence of the original text in the source language when translated into English.

In addition to using TXTAI for translation, we have also customized our datasets to align with the specific requirements of our research. These datasets have been meticulously curated and preprocessed to ensure they are representative of the coding scenarios that our model is likely to encounter in real-world applications. This includes removing unnecessary tokens, which slow down model training.

4.6 Model Training

Following the preprocessing steps, we embarked on the model training phase. Initially, we trained five distinct models, each tailored to one of the five languages, with no combination of languages. This approach allowed us to develop specialized models that could accurately generate code based on inputs in a specific language.

Subsequently, we trained another model that incorporated all five languages into a single model. This comprehensive model offers the greatest flexibility in real-world applications, as it can generate code based on inputs in any of the five languages.

In the process of training our model, we were presented with several options once the datasets were translated into the target languages. One such option was to train the model directly using the non-English datasets. However, this approach would necessitate translating each line during the training process, which could potentially increase computational load and decrease the speed of model training.

To circumvent these potential issues, we opted for a different approach. We chose to translate each of our datasets back into English. It's important to clarify that the reason for translating back to English, despite originating from English, is to ensure that the model trains only on the translated content, and not on perfect English.

This approach ensures that our model is trained on realistic translations, thereby making our results comparable to the first option discussed.

Following this, we proceeded to train the model line by line, utilizing the load[] method. The model was trained over 750,000 steps, with a batch size of 32 and a learning rate of 0.0001. We employed the Adam optimizer and used the cross-entropy loss function to measure the model's performance. The training process was conducted on an NVIDIA Tesla V100 GPU, which significantly accelerated the computation.

During the training process, we monitored the model's performance using various metrics, such as perplexity and BLEU score. We also employed techniques like early stopping and model checkpointing to prevent overfitting and ensure the best possible results.

Each model training process was conducted in a Google Colaboratory environment, which provided the most cost-effective resources for our large-scale project. This environment enabled us to efficiently train our models while minimizing computational costs.

With this, we have detailed the methodological approach that underpins our research. This approach has enabled us to pioneer the development of multilingual code generation systems, thereby contributing to a more inclusive and efficient coding environment. As we move forward, we will delve deeper into the results and implications of our study.

5 Discussions

Using our datasets, we trained different models for different languages and number of steps. We received varying scores by using the BLEU and EM evaluation scores. After training and scoring, certain observations were apparent. The models using complex languages - languages that have most characters not included on the standard QWERTY keyboard - scored very poorly, almost as if they had not been trained at all. On the other hand, languages that used the standard English letters returned much higher BLEU and EM scores.

For example, Hindi scored a mere 0.14 BLEU, and a 0 EM. Surprisingly, Japanese had the exact same numbered outcome. The re-

sults for German and French were the most surprising, though. German - our highest scorer - was trained for only 75,000 steps, and returned BLEU score of 23.27 and EM score of 12.05.

Below are the scores in table format for ease of comparison:

Language	BLEU	EM Scores
Hindi (750,000 steps)	0.14	0.0
Japanese (250,000 steps)	0.14	0.0
German (75,000 steps)	23.27	12.05
French (28,000 steps)	5.25	0.75
French (30,000 steps)	7.22	1.30
French (35,000 steps)	9.54	2.20
French (45,000 steps)	13.32	11.70

Table 1: Performance Scores by Language and Training Steps

This work provides 5 datasets, which serve as a valuable resource for training models in various languages. This dataset, born out of rigorous research and meticulous curation, holds the potential to be utilized by other models, contributing to the advancement of multilingual code generation.

As this dataset is employed in diverse research contexts and real-world applications, there is an opportunity for its further refinement and expansion. The continuous usage and testing of the dataset across different models will inevitably reveal areas for improvement, allowing us to enhance its accuracy and overall quality.

While we acknowledge that we have set a very low score in terms of BLEU and EM, there is always room for improvement. The dynamic nature of machine learning and the continuous advancements in the field present numerous opportunities for enhancing the performance of our model. With further implementations and refinements of our dataset, we anticipate improvements in these scores, thereby pushing the boundaries of what is currently achievable in code generation.

Our work does discuss and execute the translation of multiple natural languages into Java code. However, it is important to note that our current scope is limited to five languages: Spanish, French, German, Japanese, and Hindi. While this already represents a signif-

icant stride towards multilingual code generation, we recognize that there are numerous other languages that are yet to be included in our model.

As we continue to refine our model and expand its capabilities, we anticipate a significant expansion in the compatibility of these datasets. This will not only enhance the versatility of code generation models but also make them accessible to a broader global audience. The potential for growth is immense, and we are excited about the future prospects of our work in the field of multilingual code generation. Our ongoing efforts aim to break down language barriers in coding and create a more inclusive coding environment for developers worldwide. We believe that our work will inspire further research and innovation in this domain, contributing to the continual evolution of code generation.

6 Future Work

Our research on multilingual code generation systems has opened up exciting avenues for future exploration and advancement. In this section, we outline several key areas where further research and development efforts can be directed to enhance the capabilities, usability, and impact of multilingual code generation technologies.

6.1 Expanding Language Coverage

One of the primary directions for future work is to expand the language coverage of multilingual code generation systems. While our study focused on five languages (Spanish, French, German, Japanese, and Hindi), there is a vast linguistic diversity that remains untapped. Future research should aim to include a wider range of languages from different language families, scripts, and regions. To achieve this, collaborative efforts can be made to create and curate multilingual datasets that represent the breadth of global linguistic diversity. This may involve partnering with language experts, open-source communities, and industry stakeholders to collect and annotate code snippets and their corresponding natural language descriptions in various languages. By leveraging the linguistic similarities and shared features across languages, it may be possible to develop more robust and inclusive code generation models.

6.2 Improving Translation Quality and Robustness

Another key area for future research is enhancing the quality and robustness of the translation component in multilingual code generation systems. While machine learning translations have made significant strides, there is still room for improvement in terms of accuracy, fluency, and handling of domain-specific terminology and idiomatic expressions. Future work can explore advanced translation techniques, such as domain adaptation, where translation models are fine-tuned on programming-specific corpora to better capture the nuances and terminology of coding contexts. Additionally, incorporating techniques like back-translation and iterative refinement can help improve the quality and consistency of translations. Investigating the use of language-specific pre-processing and post-processing steps can also contribute to more accurate and natural translations.

6.3 Exploring Advanced Model Architectures and Training Strategies

Future research can delve into the exploration of advanced model architectures and training strategies to improve the performance and generalizability of multilingual code generation systems. While our study employed a specific model architecture, there is a wide range of possibilities to be investigated. Transformer-based models, such as BERT and GPT, have shown remarkable success in natural language processing tasks. Adapting these architectures for multilingual code generation and exploring their potential for capturing the relationships between natural language and code can lead to more powerful and expressive models. Moreover, techniques like unsupervised pre-training, multi-task learning, and meta-learning can be explored to leverage the vast amounts of unlabeled code and natural language data available across languages. By learning from diverse data sources and tasks, models can potentially acquire a more comprehensive understanding of the mapping between natural language and code, leading to improved generalization and adaptability.

6.4 Developing Comprehensive Evaluation Frameworks

Evaluating the quality, correctness, and usefulness of generated code in a multilingual setting is a complex challenge. Future work should

focus on developing comprehensive evaluation frameworks that go beyond traditional metrics like perplexity and BLEU score. This may involve designing evaluation tasks that assess the compilability, runtime behavior, and functional correctness of the generated code. Collaborating with programming language experts and software engineering researchers can help define rigorous evaluation criteria and benchmarks. Additionally, incorporating human evaluation, such as user studies, expert reviews and community uses, can provide valuable insights into the practical utility and usability of multilingual code generation systems. Engaging with developers from diverse linguistic backgrounds and collecting their feedback can guide the refinement and optimization of these systems to better meet real-world needs.

6.5 Exploring Real-World Applications and Integration

To maximize the impact of multilingual code generation research, future work should explore real-world applications and integration scenarios. This involves collaborating with industry partners, open-source communities, and educational institutions to identify practical use cases and develop tools and platforms that leverage multilingual code generation capabilities. Potential applications include code completion systems, code search engines, and educational tools for programming learning and assistance. By integrating multilingual code generation into popular integrated development environments (IDEs) and code editors, developers can benefit from language-agnostic coding support and automation. Moreover, exploring the potential of multilingual code generation in domains beyond traditional software development, such as data analysis, scientific computing, and digital humanities, can broaden the scope and impact of this technology. Collaborating with domain experts and understanding their specific requirements can drive the development of tailored multilingual code generation solutions.

6.6 Addition of Previously Removed Code

Earlier, we discussed the removing of special tokens for ease of cleaning our datasets. One further avenue of improvements may be re-

addition of the special tokens. While we did not compare the differences of the special tokens in this research, it may be of potential interest in terms of higher BLEU and EM scores.

6.7 Addressing Ethical Considerations and Biases

As multilingual code generation systems become more advanced and widely deployed, it is crucial to proactively address ethical considerations and potential biases. Future research should prioritize the development of frameworks and guidelines for the responsible development and deployment of these technologies. This may involve investigating techniques for detecting and mitigating biases in training data, ensuring the fairness and inclusivity of generated code, and establishing ethical standards for the use of multilingual code generation in various contexts. Engaging with the community can help shape and promote the development of socially responsible and beneficial technologies. Additionally, research efforts should be directed towards understanding the potential risks and misuses of multilingual code generation, such as the generation of malicious code. Developing safeguards, monitoring mechanisms, and accountability measures can help ensure the safe deployment of these systems.

6.8 Fostering Collaboration and Knowledge Sharing

To accelerate progress in multilingual code generation research, it is essential to foster collaboration and knowledge among researchers, developers, and stakeholders from diverse backgrounds. Future work should prioritize the establishment of collaborative platforms, open-source initiatives, and community-driven efforts. Encouraging the sharing of datasets, code, and research findings can enable researchers to build upon each other's work and collectively advance state of the art technologies. Organizing workshops, conferences, and hackathons focused on multilingual code generation can provide opportunities for networking and collaboration. In addition, promoting the participation of underrepresented communities and languages in multilingual code generation research can bring diverse perspectives and insights to the field. Providing resources, mentorship, and support for researchers and developers from these communities can help bridge

the language gap and ensure the inclusive development of these technologies.

7 Conclusions

In this paper, we have ventured into the relatively unexplored domain of multilingual code generation, with the aim of improving access to programming assistance for speakers of various languages. By developing code generation systems for five different languages — Spanish, French, German, Japanese, and Hindi — we have taken a significant step towards breaking down language barriers in the world of coding. Our work is not just about facilitating code generation for non-English speakers. It is about fostering inclusivity and diversity in the tech industry. By making coding assistance accessible in multiple languages, we hope to empower individuals from different linguistic backgrounds to participate in and contribute to the global coding community. Furthermore, we have made our multilingual versions of the well-known CONCODE dataset publicly available. This is a significant contribution to the research community, as it provides a valuable resource for further studies in this area. We hope that our datasets will ignite more research into multilingual code generation and lead to the development of more sophisticated models. We also hope that our work will inspire others to set benchmarks for performance metrics such as the BLEU, EM, and CodeBLEU scores. Establishing these benchmarks is crucial for assessing the effectiveness of code generation models and driving advancements in this field. In conclusion, this paper represents a effort in the field of multilingual code generation. We believe that our findings will serve as a foundation for future research and look forward to seeing how our work will be built upon to further expand the horizons of code generation. We envision a future where code generation systems can cater to speakers of any language, thereby making coding a truly global and inclusive endeavor.

References

1. Chen, X., Liu, C., Song, D.: Execution-guided neural program decoding. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32 (2018)

2. 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: Findings of the Association for Computational Linguistics: EMNLP 2020. pp. 1536–1547 (2020)
3. Gulwani, S., Marron, M.: Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. pp. 803–814 (2014)
4. Hashimoto, T.B., Guu, K., Oren, Y., Liang, P.: A retrieve-and-edit framework for predicting structured outputs. In: Advances in Neural Information Processing Systems. pp. 10052–10062 (2018)
5. Iyer, S., Konstas, I., Cheung, A., Zettlemoyer, L.: Mapping language to code in programmatic context. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1643–1652 (2018)
6. Karaivanov, S., Raychev, V., Vechev, M.: Phrase-based statistical translation of programming languages. In: Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software. pp. 173–184 (2014)
7. Nguyen, A.T., Nguyen, T.D., Nguyen, T.N.: Multilingual code generation from natural language descriptions. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 947–958 (2021)
8. Raza, M., Gulwani, S., Milic-Frayling, N.: Compositional program synthesis from natural language and examples. In: Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15). pp. 792–800 (2015)
9. Yin, P., Neubig, G.: A syntactic neural model for general-purpose code generation. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL). pp. 440–450 (2017)