Kütahya Dumlupınar University
Institute of Graduate Studies

Journal of Scientific
Reports-A
E-ISSN: 2687-6167
Number 49, June 2022

RESEARCH ARTICLE

CODE GENERATION USING TRANSFORMER BASED LANGUAGE MODEL

Umut Can ALAÇAM[1], Çağla GÖKGÖZ[2], Cahit PERKGÖZ[3,*]

[1]Eskisehir Technical University, Faculty of Engineering, Department of Computer Engineering,,
umutcanalacam@eskisehir.edu.tr, ORCID: 0000-0002-6376-0352
[2]Eskisehir Technical University, Faculty of Engineering, Department of Computer Engineering,, caglagokgoz@eskisehir.edu.tr,
ORCID: 0000-0003-1214-3546
[3,*]Eskisehir Technical University, Faculty of Engineering, Department of Computer Engineering,,
cahitperkgoz@eskisehir.edu.tr, ORCID: 0000-0003-0424-7046

Receive Date:09.03.2022 *Accepted Date: 12.05.2022*

ABSTRACT

Machine Learning has attracted researchers in the last decades and has been applied to different problems in many fields. Deep Learning methods which is a subfield of Machine Learning have started to be utilized to solve complex and hard problems with the improvement of computer technologies. Natural language processing is one of the challenging tasks that still needs to be improved for different applications such as code generation. Recently, general-purpose transformer based autoregressive language models achieved promising results on natural language generation tasks. Code generation from natural utterance using deep learning methods could be a promising development in terms of decreasing mental effort and time spent. In this study, a layered approach to generate Cascading Styles Sheets rules is proposed. The abstract data is obtained using a large-scale language model from natural utterances. Then the information is encoded into Abstract Syntax Tree. Finally, Abstract Syntax Tree structure is decoded in order to generate the Cascading Styles Sheets rules. In order to measure the performance of the proposed method an experimental procedure is constructed. Using pre-trained transformers and generated training data for Cascading Styles Sheets rules, different tests are applied to different datasets and the accuracies are obtained. Promising results for Cascading Styles Sheets code generation tasks using structural and natural prompt design are achieved. 46.98% and 66.07% overall accuracies are obtained for structural and natural prompt designs, respectively.

Keywords: *Natural Language Processing, Transformers, Abstract Syntax Tree, Code Generation, Deep Learning*

1. INTRODUCTION

The styles of web pages are defined with Cascading Styles Sheets (CSS) rules which are groups of CSS properties. HTML elements can be customized with these CSS properties. Therefore, modifying the style of a web page needs to have a better understanding of the schema of CSS and familiarity with computer programming languages which requires a significant mental effort. Generating CSS rules with natural utterances might reduce this effort and increase the productivity of designers. However, converting natural utterances to logical forms such as CSS codes is a challenging task and requires semantic parsing practices with natural language processing.

Prior to the description of the transformer model architecture [1], it should be noted that RNN models are commonly utilized for code generation tasks where current state is determined by the past information [2-5]. The earlier models were based on the sequence-to-sequence methodology [5, 6]. However, such purely sequence oriented methods could cause loss of structural information [7]. Therefore, subsequent studies depend on a grammar tree structure that improves the generalization and reliability capabilities of code generation models [8-10]. In recent reports, it is presented that transformers can also compete with the state-of-the-art models for semantic parsing tasks, such as code generation [7, 11-13]. Hence, transformer-based code generation and completion models have been advanced by the industry and integrated into the commercial products including integrated development environments (IDEs) [10, 14].

Recently, general purpose autoregressive language models have provided favorable results on natural language generation tasks [15]. The recent auto regressive transformer model of OpenAI, Generative Pre-trained Transformer-3 (GPT-3), scaled up the transformer based autoregressive models to 175 billion parameters and presented impressive results on multiple tasks [15]. Despite the semantic parsing tasks does not naturally fit into the paradigm of these models, a recently reported work shows that auto-regressive language models achieve reasonable results with few examples [12]. Inspired from the low data demand and existing capabilities of GPT-3, we propose a three-layer approach for generating CSS rules with given natural utterances.

First, a verbal expression is converted into text if the input is given in speech format. Then, the information needed for generating a CSS rule is extracted from natural utterance, after which the extracted information encoded into an Abstract Syntax Tree (AST). Lastly, the AST structure is decoded, and the corresponding CSS Rule is obtained for the given natural utterance.

The flow of the context is as follows; a summary of Natural Language Processing (NLP) concepts that is relevant to this work is provided in the second section. Later, the details of three-level approach are described in order to utilize an autoregressive language model for code generation tasks. In results section, different prompts for GPT-3 are experimented and the results are compared.

2. METHODS

While developing a system that modifies web page styles dynamically, using a natural language interface, an input from the user should be construed clearly, and a compliant CSS rule should be generated. The operators should be able to decorate any web page by describing their request in natural language (e.g. "Enlarge the headings."). In the following subsections, the background needed to introduce this work is summarized.

2.1. Transformers in Natural Language Processing

Natural Language Processing is generating abstract data from a language either in speech or text. By it is nature NLP is a hard problem to solve since most of the tasks contain messy data. In 2017, the paper "Attention is all you need" introduced a novel model architecture called transformers for language models [1]. The transformers are quickly embraced by the industry and became state-of-the-art architecture for seq2seq language models [11].

In transformers, the input is not processed sequentially. However, sequence of the words or positions comprises valuable information of the semantic meanings of a text given in natural language [7].

Therefore, in order to capture the positional relations, a positional embedding mechanism applied to each token [1]. Additional to the positional relations, the self-attention-mechanism estimates the relations between each token of the input [16]. With self-attention mechanism the model is able to capture the dependencies between tokens more reliably [7].

Unlike RNN based models, transformers do not suffer from vanishing gradient problem as the input sequence gets longer. The transformers are also advantageous to RNN based models in training time since it allows to process input sequence in parallel. Therefore, training process is much faster than RNN based models [1].

2.2. GPT-3

GPT-3 is the third generation natural language deep neural network model from OpenAI's Generative Pretrained Transformer model family. Technically, GPT-3 is a seq2seq language model which generates a completion for a given token sequence by calculating the probabilities of the next tokens [17]. These tokens are the smallest units of a text data that the model can process. The tokens can represent sentences, words, or a small fraction of words. The architecture of GPT-3 consists of two main parts: encoder and decoder. The input words are encoded, and a vector representation is obtained which is used to predict the next words. Then the probabilities of possible words are decoded from the input and the generated vector by the encoder.

GPT-3 remained as the biggest natural language model with 175 billion parameters until another transformer language model with 1 trillion parameters is announced in 2021 [18]. GPT-3 is trained with a very large corpus that contains different languages, domains, or grammar structures [1]. As the work proposed, language models yield to learn multiple tasks for different domains since the model is capable of capturing the information about different task domains. Therefore, GPT-3 shows strong performance on various tasks without fine tuning. In some cases, GPT-3 performs close or outperforms to the domain specific state-of-the-art fine-tuned models [1].

GPT-3 is able to carry out various tasks by designing a prompt that provides a brief explanation of the task with few examples [1]. For any given prompt, GPT-3 generates a completion which maximizes the likelihood with the corpora that is given to the model while training [13]. Thus, the design of the prompt is crucial for the accuracy of the obtained results.

3. PROPOSED APPROACH

A three-level pipeline is designed, in a high-level view, as an Artificial Intelligent (AI) agent. The AI agent accepts user request in text or speech format and generates a CSS rule that satisfies the user request. Such an approach provides a good level of abstraction for the internal structure. Abstraction of internal details makes the agent more reliable and useful when it is used within a complex application.

In a more detailed view, each input is processed in a pipeline of three components. These components, also called as layers, takes the input from previous component, and generates the output for the next component. The layered structure of the CSS code generation and connections are shown in Figure 1.

JSR A
Journal of Scientific Reports

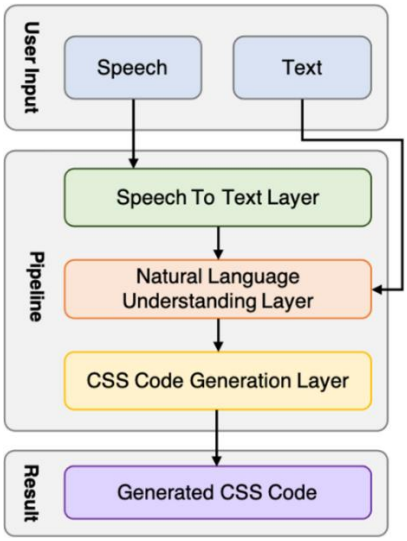

Figure 1. Layered Structure of CSS Code Generation.

3.1. Pipeline Layers

Each of the layers in the pipeline takes on different responsibilities in order to generate CSS code from user requests. When a request is received with a verbal expression: Speech to Text (STT) layer converts user speech to the text format. Then Natural Language Understanding (NLU) layer extracts the required information for generating a CSS rule by processing the user request. Finally, CSS Code Generation Layer parses the information extracted by NLU layer and generates a syntactically valid CSS rule that can be injected into the HTML code of the web site.

Pipeline layers, abstracts their internal implementation from other layers. Layered approach makes the development of software easier, more reliable and makes possible to replace or improve a layer independently. For example, in STT layer any third-party software that converts voice data to text data can be used or STT engine can be implemented without concerning about other components.

3.1.1. Speech to text layer

Speech to text layer is responsible for converting user speech into the text. This layer gets user input in audio signals and transcribes user speech. Thus, the NLU layer can process the user request and extract the properties of CSS rule which is going to be generated. If the input is already given in text format, this layer is skipped, and the user request directly passed to the NLU layer. This layer is ignored when measuring the accuracy of our NLU implementation since the speech to text transformation process is not relevant to the success of our NLU implementation.

3.1.2. Natural language understanding layer

Natural Language Understanding layer processes the user request in text format and extracts the required features of the CSS rule that is compliant with the user request. The NLU layer utilizes the GPT-3 for processing and extracting features from the text. These extracted features are the CSS selector, CSS property, and the value of the property. NLU layer encodes these features in a data structure called Abstract Syntax Tree (AST) for further processing by CSS Code Generation layer.

The abstract syntax tree structure represents the syntactical structure and content related features of a CSS rule such as CSS selectors, CSS property, the value for the extracted property, keywords like important etc.

CSS AST only includes the required syntactical properties of CSS in this study. In Figure 2, the structure of CSS AST is shown. According to the AST structure, a CSS document consists of many CSS rules, where each CSS rule consists of many selectors and declarations. The declarations consist of property, value and can also have an important keyword and so on.

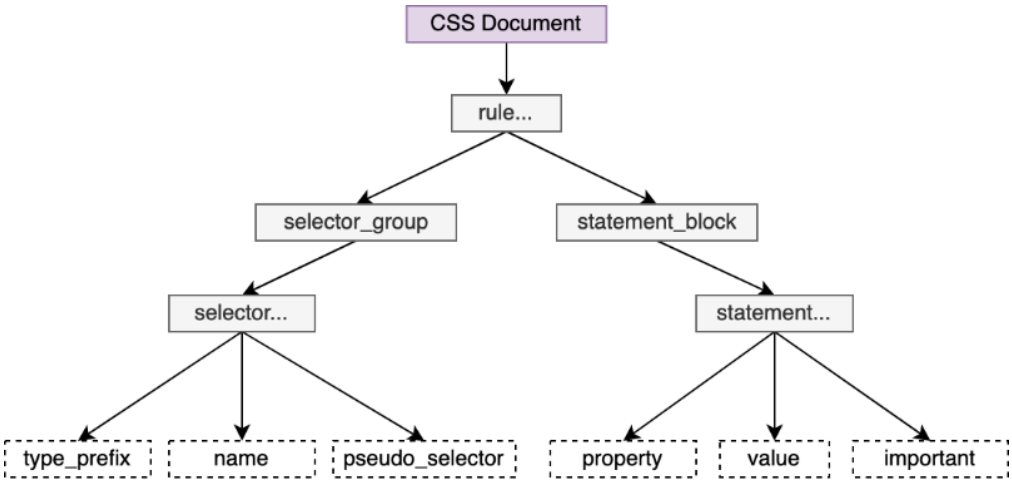

Figure 2. CSS AST Model.

In Figure 3, the CSS rule body {background: black}, which indicates the background color of the body element is black, is shown in AST format. The AST structure provides a general and compact representation of CSS documents. Currently, in this work only one CSS rule is generated per request. Therefore, the AST approach might be simplified to the simple entity mappings by extracting the selectors, properties and the values. However, the entity mapping approach would not represent the complete syntactical structure of CSS and cannot be generalized enough to decouple CSS code generation layer with NLU layer.

Using the generalization capability of the AST approach, the CSS code generation implementation does not need to be modified whenever NLU capabilities are decided to be extended. It is also not needed to be modified existing AST structure and AST decoding procedure of CSS code generation layer even though the AST can be generated in a better way and more accurate in the future, since the syntactical structure of CSS does not change.

The AST approach also ensures the consistency of generated CSS rules by eliminating the invalid CSS statements that could be generated. If GPT-3 generates an invalid output, it is not possible to build a valid AST. Therefore, the inconsistent outputs can be detected and eliminated without effecting the style of the website.

JSR A
Journal of Scientific Reports

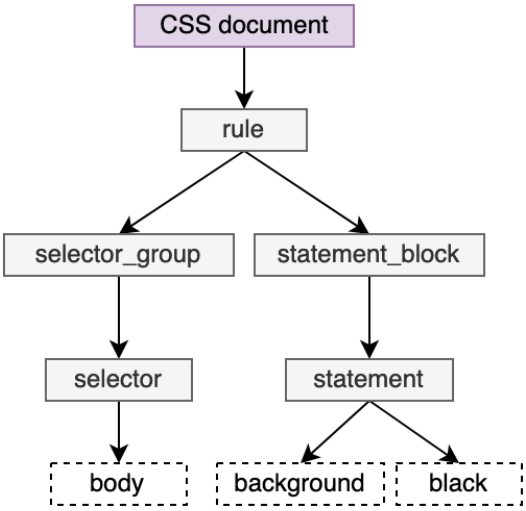

Figure 3. AST Example for the CSS Rule: body {background: black}.

3.1.3. CSS code generation layer

CSS Code Generation layer decodes the AST that is generated by the NLU layer and generates the CSS code. Since the AST contains syntactical hierarchy and content related features of the CSS code, it is easy to convert AST data into a syntactically valid CSS code. Since the AST schema is decoded with a deterministic procedure, it is possible to guarantee that the output of the CSS code layer is always syntactically valid. However, the content related features such as CSS selector, property and values might be inconsistent. For example: the property can be extracted as "color" and the value might be extracted as "12px" which are not compliant with each other. This issue can be achieved by checking if the AST attributes are valid. However, this is not concerned because the most browser engines ignore attribute errors covered in this way.

4. EXPERIMENTAL RESULTS

Meaningful representation of CSS code highly depends on the utilization of GPT-3 to construe the requests and generation of the AST as mentioned before. Since GPT-3 captures the context of the problem within the prompt, prompt design is crucial for the accuracy [11]. To achieve better accuracy and discovering the nature of auto-regressive language models, the three-level approach in this study is tested with different prompt designs.

4.1. Dataset

For discovering the capabilities of our NLU approach, a dataset which contains hand annotated user request examples with their expected CSS equivalents is created.

The natural utterances that NLU layer is going to handle, have differences in terms of how the features are explained and how much of the information is given. Therefore, it would be more beneficial to examine natural utterances in different categories. Hence, test data is grouped into three categories which are property update requests, relative update requests and decoration requests.

JSR A
Journal of Scientific Reports

Property update requests contains clear and distinct information about the style rule. The utterance contains all features of AST: selector, property, and value. (e.g., "*Make heading one color blue*").

Relative update requests clearly describe the target and the property of the rule. However, the value is not clear and described with a comparison to its older value. (e.g., "*Make headings bigger*").

Decoration requests describes a new design feature that will modify the webpage. These examples do not clearly describe the property and value unlike the target which is clearly described. (e.g., "*Add text shadows to the hyperlinks*").

4.2. Experimental Procedure

In order to measure the performance of the layered approach in this work, the tests are applied separately to all categories of the dataset to find out the capabilities of our NLU layer on different categories of requests which are property update requests, relative update requests and decoration requests. The accuracy is calculated by comparing the model output with the ground truth for each of the example in dataset. However, the nature of CSS allows different properties and values to behave effectively same. For example, the rules using background-color property and background property behaves identical when the value is a color. As a result, there exists more than one compliant CSS rule for a particular natural utterance. Therefore, counting only the exact matches lead incorrect measurements since the outputs that are compliant with the user request. In order to overcome this situation, the generated CSS rules that behaves compliant with the user requests are accepted as accurate outcomes. The equation for output accuracy is shown in Eqn. 1.

$$OutputAccuracy = \frac{TrueOutputs}{NumberOfGeneratedAST} \times 100 \quad (1)$$

where *OutputAccuracy* represents the ratio of the accurate outputs that is generated. The term *TrueOutputs* is the number of the outputs that is compliant with the user request and behaves effectively same with the ground truth. The term *NumberOfGeneratedAST* is the number of successfully generated ASTs. Note that generation of a valid AST only guarantees that the generated output can be converted into a syntactically valid CSS rule. The generated AST may include wrong entities and not compliant with the true CSS rule.

As mentioned before, in order to create an AST from a natural utterance, it is required to extract selector, property and value entities. Therefore, performance of detecting these entities is also essential for understanding the strengths and weaknesses of the method. Hence, accuracies of detecting each entity are also calculated.

$$SelectorAccuracy = \frac{TrueSelectors}{NumberOfGeneratedAST} \times 100 \quad (2)$$

$$PropertyAccuracy = \frac{TrueProperties}{NumberOfGeneratedAST} \times 100 \quad (3)$$

$$ValueAccuracy = \frac{TrueValues}{NumberOfGeneratedAST} \times 100 \quad (4)$$

The accuracies of extracted entities are calculated with the equations given in Eqns. 2, 3 and 4. The terms: *TrueSelector*, *TrueProperties* and *TrueValues* represent the number of the entities that semantically match the ground truth.

4.3. Prompt Design

In NLU layer, the behavior of the GPT-3 with different prompt designs is investigated to find out how to create a promising prompt design. We proposed two different prompt designs that have structural and contextual differences.

4.3.1. Structural representation

First approach to the prompt design requires syntactical rules that GPT-3 requires to follow. In order to retrieve the information that GPT-3 produced; a meaningful representation of AST is proposed. Using this meaningful representation, it is possible to build an AST thereby, a syntactically valid CSS rule can be generated.

The prompt contains a set of examples to encourage GPT-3 for generating outputs in form of our meaningful representation. Each line represents a single example, which consists of a natural utterance and its meaningful representation.

Make body background black => [selector:"body", property:"background", value:"black"];

An example from the prompt is shown above. The example is separated into natural utterance and meaningful representation sections with the characters: '=>'. The natural utterance section represents an example user request in natural language. The meaningful representation section represents the AST which is compliant with this natural utterance. The meaningful representation is given between the square brackets, and contains the entities required for a CSS rule: selector, property, and value. Finally, a semi-colon character ';' is used for limiting the output of the GPT-3.

4.3.2. Natural representation

The second approach for the prompt design aims to provide data in more convenient structure for GPT-3. Despite GPT-3 mostly focused on natural language, the corpora given to the model contains considerable amount of CSS code [15]. Therefore, the model is more familiar with the rules of CSS language than particular syntactic design in this work.

Similar to the previous prompt design, each line in the prompt contains an example of user request and the meaningful representation of this request. However, in this approach, the meaningful representation follows the original CSS syntax, which is in the same format of HTML style attributes.

Make body background black => body {background: black}

The example of this representation is shown above. As shown in the example, the meaningful representation section fallows the original CSS syntax. Since the CSS rules are already structural, it is possible to create an AST for validating the output of GPT-3.

JSR A
Journal of Scientific Reports

4.4. Test Results

For discovering the capabilities of both prompt designs: structural representation and natural representation are tested on the request categories: property updates, relative updates and decorations. The prompts consist of 4 property update, 4 relative update and 3 decoration request examples that are randomly picked from the data set.

4.4.1. Results with structural representation prompt design

As mentioned before, the first approach to the meaningful representation was a custom syntactical representation proposed within this research. The results obtained with this representation is given in Table 1.

Table 1. Results of Structural Representation Prompt Design.

Data Set	Selector Accuracy (%)	Property Accuracy (%)	Value Accuracy (%)	Output Accuracy (%)
Property Updates	83.54	82.90	82.59	68.28
Relative Updates	77.25	72.39	-	46.32
Decorations	63.04	54.86	55.29	26.35
Overall	74.61	70.05	68.94	46.98

The test results show that the accuracy in property update requests, clearly outperforms other categories. Additionally, the output accuracy show that the model produces promising results on property update requests with a unique syntactical data format that the model is not familiar with.

For relative update requests, the accuracy of the model significantly decreases. Despite the selector accuracy and property accuracy are promising, the output accuracy is not as good as property update requests output accuracy. However, the model still produces reasonable results with very little amount of data.

The model had difficulties on decoration requests, which is the most challenging kind of requests, and has the lowest accuracy as a result. Even though the model captures the selector correctly in general, since the property and value are not given clear in the natural utterance, model performance is not satisfying for property and value detection.

In general, it is seen that the model has a higher accuracy on selector detection on all kinds of requests, followed by property and value detections respectively.

4.4.2. Results with natural representation prompt design

As mentioned before, the second approach to the prompt design was the standard CSS syntax, which is expected to GPT-3 is already familiar with. The results obtained with this representation is given in Table 2.

Similar to the results of the structural representation prompt design, the accuracy on the property update requests is significantly higher than other categories. Additionally, it is seen that the model performs slightly better with the natural representation than the structural representation.

Using a natural representation also increases the accuracy on relative update requests. Even though the property detection accuracy is slightly decreased, the output accuracy on relative update requests is significantly higher when compared to the structural representation.

Table 2. Results of Natural Representation Prompt Design.

Data Set	Selector Accuracy (%)	Property Accuracy (%)	Value Accuracy (%)	Output Accuracy (%)
Property Updates	85.38	84.40	84.19	70.75
Relative Updates	85.00	68.75	-	60.42
Decorations	78.38	81.08	70.00	70.27
Overall	84.71	77.13	82.37	66.07

The results show the performance on the decoration requests dramatically increased with the natural representation prompt design. When compared to the previous output accuracy for decorations, which is 26.35%, using natural representation prompt design interestingly leads to a dramatic increase on output accuracy: 70.27%.

For all categories of requests in the dataset, the natural representation prompt design clearly outperforms the structural representation that is originally defined within this research.

Therefore, it is seen that prompt design is a major factor for the accuracy and using a meaningful representation that GPT-3 is already consumed before, significantly increases the performance of the model.

In the literature, there are different research works reporting generating codes using pretrained language methods [19-21]. In [19], current methods are analyzed and compared, however there is no clear accuracy information that is provided, and these methods do not mention about the CSS codes. However, in [20], a code generation from natural language queries using pretrained models to generate SQL queries is proposed. It is mentioned that with the developed decoding method, the accuracy is improved from 49% to 72%. In [21], GPT-3 is fine tuned to generate python codes with several models. These models improved the percentage of solved problems to a best value of 72.5%. Hence, these accuracy results, although not generated for CSS codes, are comparable to our results.

5. CONCLUSIONS and DISCUSSIONS

In this study, a layered code generation structure is proposed using Natural Language Processing based on transformers. The model provides promising results for the internal structure. The reliability and the accuracy of the results depend on the details of the representations and the prompt design. The natural prompt design results with an overall accuracy of 66.07% are better than the structural prompt design which has an overall accuracy of 46.98%.

The prompt design is crucial for the accuracy of auto-regressive language models such as GPT-3. Therefore, focusing on a better prompt design strategy is an opportunity for significant improvement of the accuracy. In the next paragraphs, some interesting future work ideas which are mostly about the prompt design, are summarized.

Dynamic example selection for prompts might be an opportunity for improving the accuracy. According to the structure and existing CSS classes of the target web page, a prompt could be generated by picking up the examples that is more appreciate with the content of the web page. Recent studies show such an approach can improve accuracy of the autoregressive language models for tasks of semantic parsing [12]. However, this approach strongly relies to the quality and the size of the example dataset. Therefore, extending the dataset could be a significant improvement.

Generating prompt with in-context examples could also improve the accuracy. For each user request, a prompt could be generated by picking the examples that semantically similar to the user request. Additionally, fine tuning could reduce the cost by each request with smaller prompt sizes and improve the accuracy of generated ASTs.

CSS document building with natural utterances might be a good practice for styling a webpage. The users could describe their style rules in a text document line by line. For each line, a CSS rule could be generated in AST structure. The generated ASTs could be combined into one general AST that can be decoded into a complete CSS document which comprises all generated rules.

ACKNOWLEDGEMENT

This work is supported by Eskisehir Technical University scientific research projects with the project number of 21GAP084. We acknowledge the support provided by Greg Brockman from OpenAI, for allowing academic access to the GPT-3 beta program.

REFERENCES

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I., (2017), Attention is all you need. Advances in Neural Information Processing Systems, 30.

[2] Yin, P., and Neubig, G., (2019), Reranking for neural semantic parsing. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.

[3] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., and Liu, X., (2019), A novel neural source code representation based on abstract syntax tree. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 783-794, IEEE.

[4] Uma, M., Sneha, V., Sneha, G., Bhuvana, J., and Bharathi, B., (2019), Formation of SQL from natural language query using NLP. In 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), 1-5, IEEE.

[5] Galassi, A., Lippi, M., and Torroni, P., (2020), Attention in natural language processing. IEEE Transactions on Neural Networks and Learning Systems, 32, 4291-4308.

[6] Sun, Z., Zhu, Q., Mou, L., Xiong, Y., Li, G., and Zhang, L., (2019), A grammar-based structural cnn decoder for code generation. In Proceedings of the AAAI Conference on Artificial Intelligence, 7055-7062.

[7] Shiv, V., and Quirk, C., (2019), Novel positional encodings to enable tree-based transformers. Advances in Neural Information Processing Systems, 32.

[8] Sun, Z., Zhu, Q., Xiong, Y., Sun, Y., Mou, L., and Zhang, L., (2020), Treegen: A tree-based transformer architecture for code generation. In Proceedings of the AAAI Conference on Artificial Intelligence, 8984-8991.

[9] Quirk, C., Mooney, R., and Galley, M., (2015), Language to code: Learning semantic parsers for if-this-then-that recipes. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 878-888.

[10] Kim, S., Zhao, J., Tian, Y., and Chandra, S., (2021), Code prediction by feeding trees to transformers. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 150-162.

[11] Ferraro, G., and Suominen, H., (2020), Transformer semantic parsing. In Proceedings of the The 18th Annual Workshop of the Australasian Language Technology Association, 121-126.

[12] Shin, R., Lin, C. H., Thomson, S., Chen, C., Roy, S., Platanios, E. A., Pauls, A., Klein, D., Eisner, J., and Van Durme, B., (2021), Constrained language models yield few-shot semantic parsers. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 7699-7715.

[13] Shah, M., Shenoy, R., and Shankarmani, R., (2021), Natural language to python source code using transformers. In 2021 International Conference on Intelligent Technologies (CONIT), 1-4, IEEE.

[14] Svyatkovskiy, A., Deng, S. K., Fu, S., and Sundaresan, N., (2020), 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, 1433-1443.

[15] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., and Askell, A., (2020), Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877-1901.

[16] Liddy, E. D, (2001), Natural language processing. In Encyclopedia of Library and Information Science, (2nd Ed.) NY. Marcel Decker, Inc.

[17] Egonmwan, E., and Chali, Y., (2019), Transformer and seq2seq model for paraphrase generation. In Proceedings of the 3rd Workshop on Neural Generation and Translation, 249-255.

[18] Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., and Zaharia, M., (2021), Efficient large-scale language model training on GPU clusters using megatron-LM. In

Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p Article 58, Association for Computing Machinery, St. Louis, Missouri.

[19] Xu, F. F., Alon, U., Neubig, G., and Hellendoorn, V. J., (2022), A systematic evaluation of large language models of code. arXiv preprint arXiv:2202.13169.

[20] Poesia, G., Polozov, O., Le, V., Tiwari, A., Soares, G., Meek, C., and Gulwani, S., (2022), Synchromesh: Reliable code generation from pre-trained language models. arXiv preprint arXiv:2201.11227.

[21] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., and Brockman, G., (2021), Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.