International Journal of Aquatic Science
ISSN: 2008-8019
Vol 15, Issue 01, 2024

Code Generation Using NLP and AI Based Techniques

Nirali M. Kamble[1], Purva M. Khode[2], Vaishnavi S. Dhabekar[3], Sahil S. Gode[4], Suraj S. Kumbhare[5], Yash G. Wagh[6], Dr. S. W. Mohod[7]

[1,2,3,4,5,6,7]Computer Engineering Department, Rastrasant Tukdoji Maharaj Nagpur University

Email: [1]niralikamble@gmail.com, [2]purvakhode@gmail.com, [3]vaishnavidhabekar99@gmail.com, [4]sahilgode267@gmail.com, [5]surajkumbhare480@gmail.com, [6]yashwagh12001@gmail.com
Guide Email: [7]sudhirwamanrao@gmail.com

Abstract: The project described focuses on the development of a system aimed at enhancing human-machine interaction by translating natural language commands into executable code, particularly focusing on arithmetic and mathematical operations. The overarching objective is to simplify the process of code generation. Motivated by the increasing success of AI and automation across various fields, automatic code generation has the potential to significantly enhance software engineering and development projects, making them more accessible to a wider audience. Addressing a key concern in software engineering, the proposed system employs novel techniques using NLP and AI to generate relevant code fragments based on provided natural language descriptions into programming language. The ultimate goal is to create a system capable of understanding given description and convert it into a plain code or programming language.

Keywords: Machine Learning, Natural Language Processing (NLP), Codellama-7b, Deep Learning, Automatic Programming.

1. INTRODUCTION

In today's technology-driven world, software development plays a pivotal role in fostering innovation and solving problems across diverse domains. However, writing code remains a complex and time-consuming endeavor, often requiring specialized knowledge in programming languages and development practices, posing a significant entry barrier for individuals and organizations alike. Code Generation emerges as a crucial field aimed at predicting explicit code or program structures from various data sources, including incomplete code, programs in different programming languages, natural language descriptions, or execution examples. These tools hold promise in facilitating the development of automatic programming tools to enhance programming productivity. To tackle this challenge, the convergence of Artificial Intelligence (AI) and Natural Language Processing (NLP) offers exciting prospects for automating code generation. Through the utilization of AI/NLP techniques, we can bridge the gap between human-readable natural language and machine-executable code, thereby rendering software development more accessible and efficient. This project delves into the captivating realm of "Code Generation Using AI/NLP Techniques," aiming to explore and exploit the synergies between these cuting edge technologies.

2. RESEARCH ELABORATION

According to paper — “Code prediction by feeding trees to transformers” [1] by S. Kim, J. Zhao, Y. Tian and S. Chandra. It presented ways to using the Transformer for code prediction and showed that the Transformer outperforms existing models for code prediction, and when supplied with code’ s structural information they are able to get even better predictive power. This paper focused on predicting the next token, as it is already a challenging task. In future, they intend to explore predicting multiple tokens at a time, i.e., autocompleting entire expressions. Enrique Dehaerne, Bappaditya Dey, Stefan De Gendt, Sandip Halder and Wannes Meert stated in this paper— “Code Generation Using Machine Learning: A Systematic Review”. [2] This paper provides overview studies of code generation using ML. The research in this paper comes to a conclusion that there is three paradigms of code generation that are description-to code, code-to description and code-to-code. The most popular applications that work in these paradigms were found to be code generation from natural language descriptions, documentation generation, and automatic program repair, respectively. In this paper — “Predicting Code Coverage without Execution” [3] by Michele Tufano, Shubham Chandel, Anisha Agarwal, Neel Sundaresan, Colin Clement. It introduced the task of Code Coverage Prediction, which aims to assess the capabilities of Large Language Models (LLM) in understanding code execution by accurately predicting the lines of code that are executed based on given test cases. “Towards Code Generation from BDD Test Case Specifications: A Vision” [4] by Leon Chemnitz; David Reichenbach; Hani Aldebes; Mariam Naveed; Krishna Narasimhan; Mira Mezini, proposed the solution based on ML and AI which are being used in this paper. Introducing a novel approach to generating frontend component code for the Angular framework using behaviour-driven development test specifications as input to a transformer-based ML model.

3. PROPOSED METHOD

This project focused on developing a system that facilitates human-machine interaction by enabling the conversion of natural language commands into executable code, with a specific focus on arithmetic and mathematical operations. The overarching goal is to streamline the process of code generation, making it more accessible to a broader audience.

A. Background

The primary quest was to enable direct programming of machines based on natural language commands. While this goal initially seemed ambitious, recent advancements in Natural Language Processing (NLP) and machine learning models have opened new possibilities. Leveraging these advancements, our approach has shifted from direct machine programming to predictive code generation, a stepping stone towards achieving the broader objective.

This system utilizes a sophisticated transformer architecture, a departure from conventional predictive modelling in text-to-code applications. Drawing inspiration from the successes of transformer-based models in natural language processing, our core methodology involves fine-tuning this architecture using a meticulously curated dataset. This dataset comprises paired samples that associate natural language commands with their corresponding code segments, allowing the model to grasp the intricate relationships between linguistic expressions and executable code. The training process is iterative, refining the model’s accuracy and adaptability through multiple stages. We meticulously optimize hyperparameters and rigorously conduct validation exercises to ensure the model's proficiency in accurately generating code from diverse natural language inputs.

In addition, while our current focus remains on arithmetic and mathematics for computational feasibility, our framework is inherently designed to expand its functionality into various domains in subsequent iterations. This innovative approach represents a pivotal step towards enabling a direct and intuitive conversation between humans and machines, marking a transformative path where programming becomes increasingly accessible and conversational.

B. Primary Goal

The primary goal of this project is to explore and implement AI and NLP methodologies to generate code snippets and solutions based on high-level natural language descriptions, pseudocode, or other human-readable inputs. Our project will encompass a range of code-related tasks, such as code summarization, code completion, code translation, refactoring, and code generation from domain-specific languages.

C. Objective

The objectives of creating a code generation system using NLP and AI techniques could include:

1. Automating Routine Tasks: One of the main objectives is to automate routine coding tasks, reducing the time and effort required by developers.
2. Improving Code Quality: By using AI and NLP, the system can generate high-quality code that follows best practices and reduces the likelihood of errors.
3. Enhancing Developer Productivity: Such a system can enhance developer productivity by providing code suggestions, completing code snippets, and even writing whole functions or modules.

D. Technical Architecture:

Code Llama: - The Code Llama models constitute foundation models for code generation. They come in three model sizes: 7B, 13B and 34B parameters. The 7B and 13B models are trained using an infilling objective (Section 2.3), and are appropriate to be used in an IDE to complete code in the middle of a file. All Code Llama models are initialized with Llama 2 model weights and trained on 500B tokens from a code-heavy dataset.

Code Llama - Instruct :- The Code Llama - Instruct models are based on Code Llama and fine-tuned with an additional approx. 5B tokens to better follow human instructions. Our instruction fine-tuned models Code Llama Instruct are based on Code Llama and trained to answer questions appropriately.

CodeLlama Instruct by Hugging Face: - Trained on large datasets of code snippets and programming language syntax. They utilize Transformer architectures to understand and generate code based on user input.

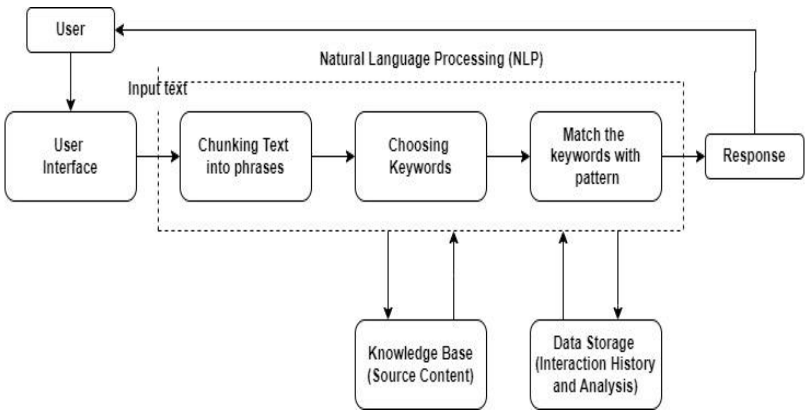

User
Natural Language Processing (NLP)
Input text
User Interface
Chunking Text into phrases
Choosing Keywords
Match the keywords with pattern
Response
Knowledge Base (Source Content)
Data Storage (Interaction History and Analysis)

Fig.1: Architecture of Proposed System.

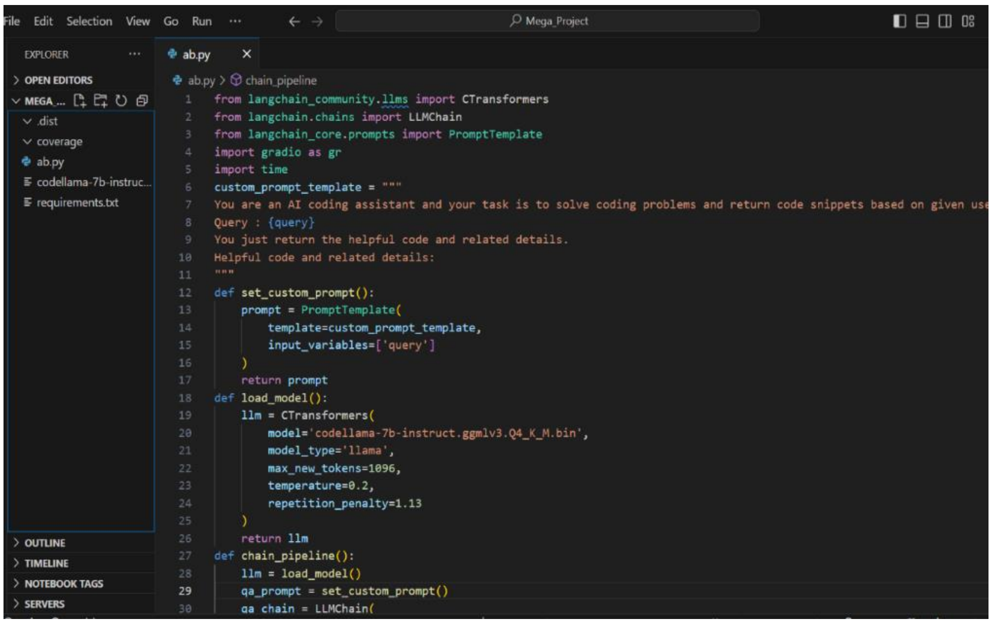
E. Coding:

Requirements -

- C Transformers.
- Gradio.
- LangChain.

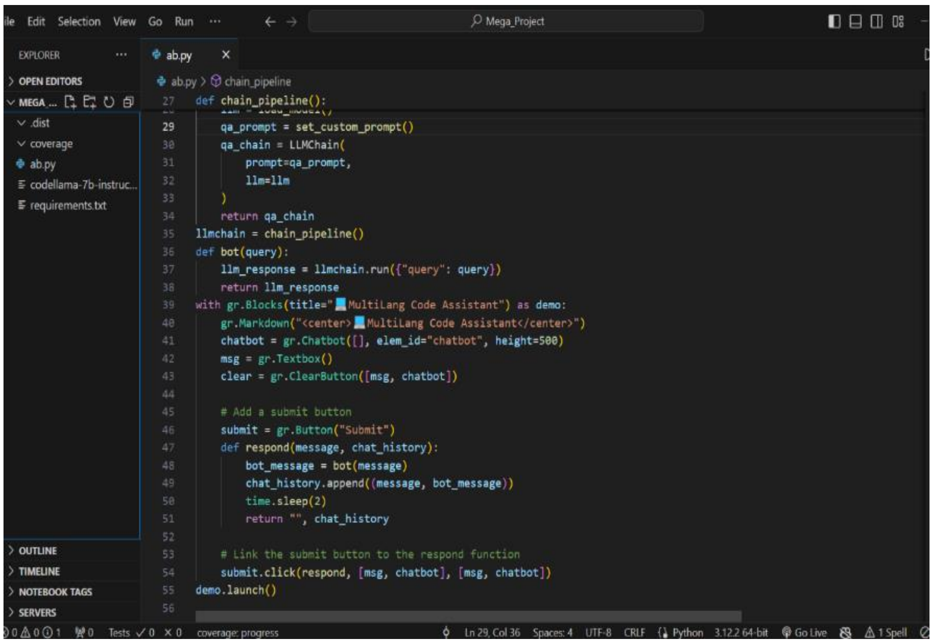
Use Cases -

- Rapid prototyping and experimentation
- Boilerplate code generation
- Legacy code modernization
- Time-saving and productivity boost
- Accessibility for non-technical users (low-code/no-code development)
- Learning resource for programming languages
- Cost-effectiveness compared to hiring multi-lingual developers



```python
from langchain_community.llms import CTransformers
from langchain.chains import LLMChain
from langchain_core.prompts import PromptTemplate
import gradio as gr
import time
custom_prompt_template = """
You are an AI coding assistant and your task is to solve coding problems and return code snippets based on given use
Query : {query}
You just return the helpful code and related details.
Helpful code and related details:
"""
def set_custom_prompt():
    prompt = PromptTemplate(
        template=custom_prompt_template,
        input_variables=['query']
    )
    return prompt
def load_model():
    llm = CTransformers(
        model='codellama-7b-instruct.ggmlv3.Q4_K_M.bin',
        model_type='llama',
        max_new_tokens=1096,
        temperature=0.2,
        repetition_penalty=1.13
    )
    return llm
def chain_pipeline():
    llm = load_model()
    qa_prompt = set_custom_prompt()
    qa_chain = LLMChain(
```

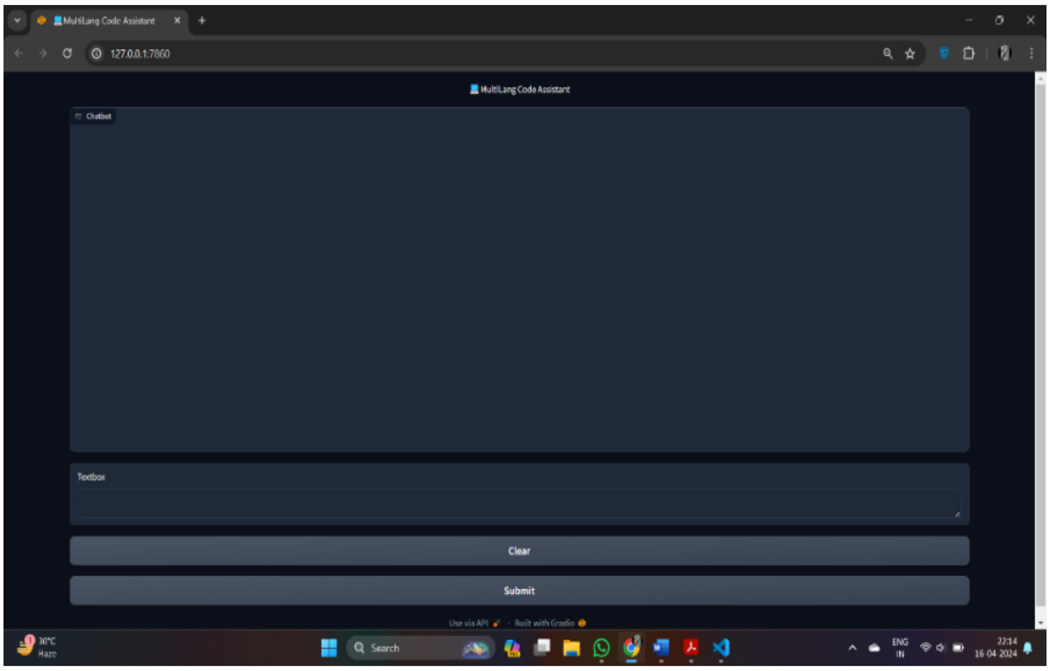
Snapshot 1: Coding of Proposed System.



```python
def chain_pipeline():
    llm = load_model()
    qa_prompt = set_custom_prompt()
    qa_chain = LLMChain(
        prompt=qa_prompt,
        llm=llm
    )
    return qa_chain
llmchain = chain_pipeline()
def bot(query):
    llm_response = llmchain.run({"query": query})
    return llm_response
with gr.Blocks(title="💻MultiLang Code Assistant") as demo:
    gr.Markdown("<center>💻MultiLang Code Assistant</center>")
    chatbot = gr.Chatbot([], elem_id="chatbot", height=500)
    msg = gr.Textbox()
    clear = gr.ClearButton([msg, chatbot])

    # Add a submit button
    submit = gr.Button("Submit")
    def respond(message, chat_history):
        bot_message = bot(message)
        chat_history.append((message, bot_message))
        time.sleep(2)
        return "", chat_history

    # Link the submit button to the respond function
    submit.click(respond, [msg, chatbot], [msg, chatbot])
demo.launch()
```

Snapshot 2: Coding of Proposed System.

4. RESULTS AND FINDINGS



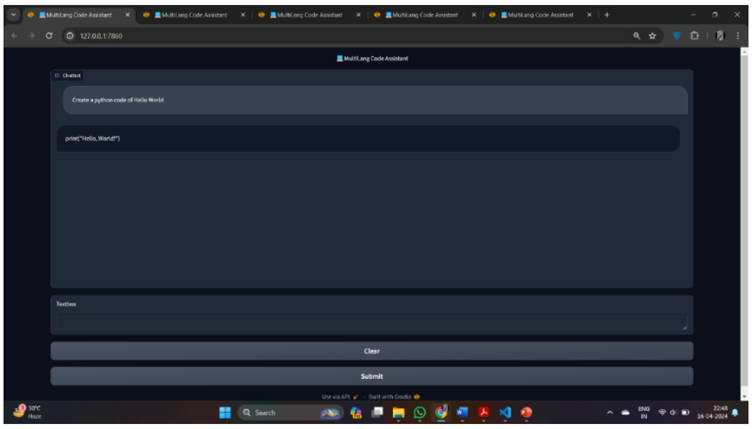
Snapshot 3: Interface of Code Generation System.

5. INTERFACE OF CODE GENERATION

The fastest way to demonstrate any machine learning model with a friendly web interface is with Gradio. So, we have made our interface using gradio framework.

6. RESULTS OF EXECUTED INPUTS

- For Python Code

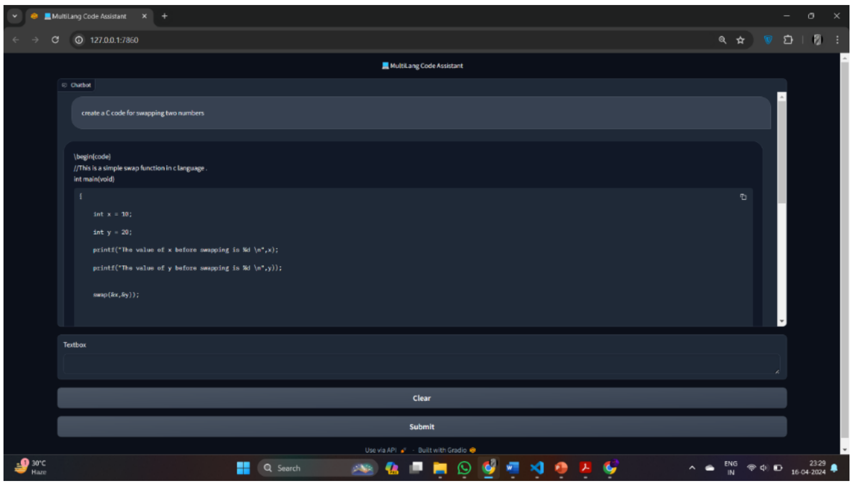


Create a python code of Hello World

```
print("Hello, World!")
```

Snapshot 4: Execution of query "Create a python code of Hello World"

- For C code

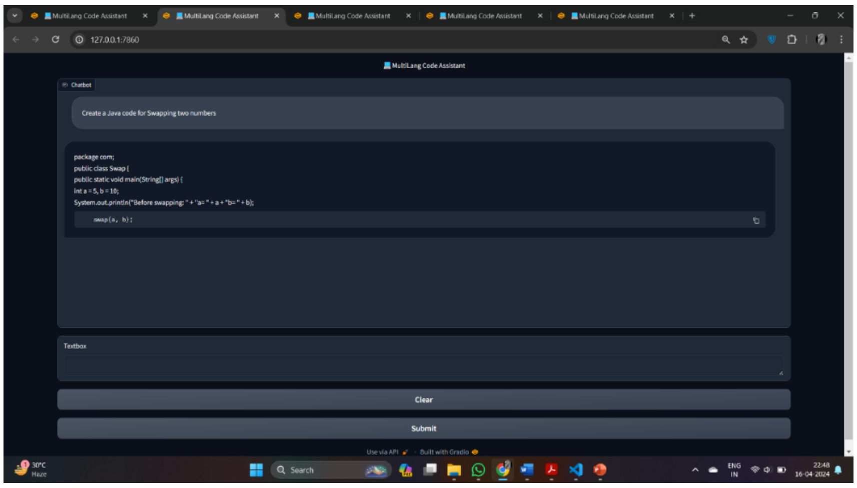


create a C code for swapping two numbers

```
\begin{code}
//This is a simple swap function in c language .
int main(void)
{

    int x = 10;

    int y = 20;

    printf("The value of x before swapping is %d \n",x);

    printf("The value of y before swapping is %d \n",y));


    swap(&x,&y));
```

Snapshot 5: Execution of query "Create a C code for swapping two numbers"

- For Java code

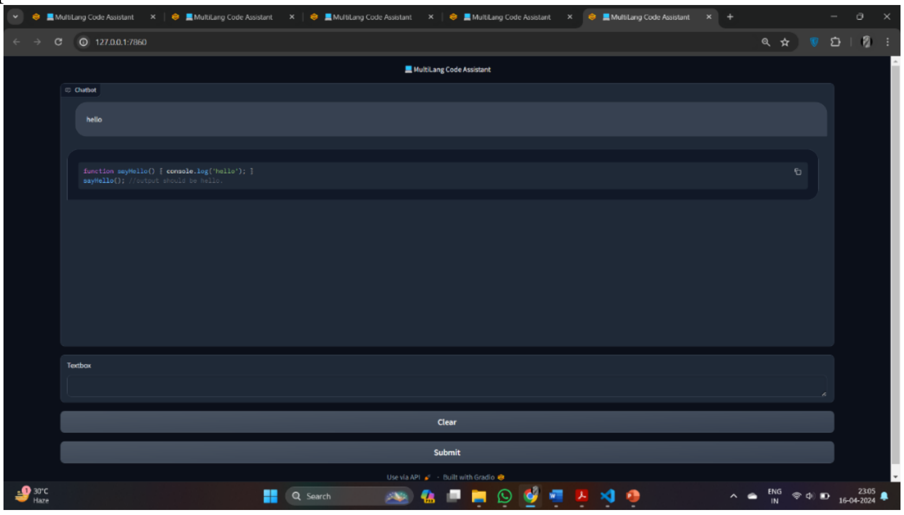


Create a Java code for Swapping two numbers

```
package com;
public class Swap {
public static void main(String[] args) {
int a = 5, b = 10;
System.out.println("Before swapping: " + "a= " + a + "b= " + b);
    swap(a, b);
```

Snapshot 6: of Execution of query "Create a Java code for swapping two numbers"

- For simple word



hello

```
function sayHello() [ console.log('hello'); ]
sayHello(); //output should be hello.
```

Snapshot 7: Execution of query “Hello”

7. CONCLUSIONS

In conclusion, the integration of AI and NLP techniques for code generation signifies a frontier where human understanding converges with digital capabilities, revolutionizing our interaction with computers and code. This project actively engages with this frontier, aiming to unlock its potential and enhance the accessibility and efficiency of software development. Through the utilization of Natural Language Processing techniques, significant strides have been made in code generation, promising further advancements. Additionally, recent breakthroughs in attention and pointer mechanisms have bolstered code generation from natural language, particularly in addressing long-range dependencies. Looking ahead, this intersection holds vast potential for reshaping the software development landscape, opening doors to new possibilities and applications.

8. ACKNOWLEDGMENT

We would like to express our sincere gratitude to all those who contributed to the completion of this project. We are also grateful to the participants who provided feedback and insights that enriched our understanding. Additionally, we extend our appreciation to the developers of the AI and NLP tools and libraries that were instrumental in implementing our approach.

9. REFERENCES

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

[2] Enrique Dehaerne; Bappaditya Dey; Stefan De Gendt; Sandip Halder; Wannes Meert, “Code Generation Using Machine Learning: A Systematic Review”. IEEE ACCESS, 10 August 2022.

[3] Michele Tufano, Shubham Chandel, Anisha Agarwal, Neel Sundaresan, Colin Clement, “Predicting Code Coverage without Execution”. Microsoft Redmond WA, USA 25 Jul 2023.

[4] Leon Chemnitz, David Reichenbach, Hani Aldebes, Mariam Naveed, Krishna Narasimhan, Mira Mezini, “Towards Code Generation from BDD Test Case.

[5] Specifications: A Vision”. Melbourne, Australia IEEE Xplore: 04 July 2023.

[6] Peter Teufl1, Udo Payer2, and Guenter Lackner3, “From NLP (Natural Language Processing) to MLP (Machine Language Processing)”. Institute for Applied Information Processing and Communications (IAIK), 2010.

[7] Maggie Johnson, Julie Zelenski, “Intermediate Representation”. Handout July 23, 2010.

[8] Hendrig Sellik, “Natural Language Processing Techniques for Code Generation”. Delft University of Technology 2019. 8. Muhammad Asaduzzaman Chanchal K. Roy Kevin A. Schneider Daqing Hou, “CSCC: Simple, Efficient, Context Sensitive Code Completion”. 2014 IEEE International Conference on Software Maintenance and Evolution.

[9] Nadezhda Chirkova, Sergey Troshin, “Empirical Study of Transformers for Source Code” ESEC/FSE ’21, August 23–28, 2021, Athens, Greece.

[10] Man Fai Wong, Shangxin Guo, Ching Nam Hang, Siu Wai Ho, Chee Wei Tan, “Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review”. 4 Jul 2023.

[11] F. F. Xu, B. Vasilescu, and G. Neubig, “In-ide code generation from Natural language: Promise and challenges,” ACM Trans. Softw. Eng. Methodol., vol. 31, no. 2, pp. 29:1–29:47, 2022.

[12] H. Le, Y. Wang, A. D. Gotmare, S. Savarese, and S. C. Hoi, “Coderl: Mastering code generation through pretrained models and deep Reinforcement learning,” arXiv preprint arXiv: 2207.01780, 2022.