DESIGNING OF A VOICE – BASED PROGRAMMING IDE FOR SOURCE CODE GENERATION

Dr Anil A R
AI & ML Department
Sree Buddha College of Engineering,
Alappuzha, India
cs.anilar@sbcemail.in

Amit Sankar Arun
CSE Department
Sree Buddha College of Engineering,
Alappuzha, India
2020cs.amit@sbcemail.in

Anandhu Anilkumar
CSE Department
Sree Buddha College of Engineering,
Alappuzha,India
2020cs.anandhu@sbcemail.in

Anandu S Sivan
CSE Department
Sree Buddha College of Engineering,
Alappuzha, India
2020cs.anandu@sbcemail.in

Anoop Manoharan
CSE Department
Sree Buddha College of Engineering,
Alappuzha, India
2020cs.anoop@sbcemail.in

***Abstract*— Nowadays coding is not a complex thing to do, by the advancement in technology and AI gives a crucial role in the easiness to the day to day life of human beings. Traditional type of coding is complex and not everyone is flexible with that, by using the voice coding we can make coding easier. Here we are integrating the gpt model to find the required code they asked for, this is done with the help of Natural Language Processing and Speech Recognition. We are integrating python libraries with the open AI model gpt 3.5 to get the answers in response to the speech input that is given by the user. Python libraries are used for these functions : converting audio to text format and searching the text in the gpt model and response that is given by the model.**

Keywords*— deep learning, natural language processing, source code generation, voice to source code, voice-based ID

I. INTRODUCTION

In an era marked by the convergence of artificial intelligence and human-computer interaction, the VoiceCode IDE project stands as a pioneering venture at the intersection of natural language processing (NLP) and programming. Recognizing the profound nature of human communication through voice, this project sets out to develop a revolutionary voice-basedIntegrated Development Environment (IDE) that enables users to design and develop applications through spoken commands. Human beings possess an innate ability to comprehend and express thoughts seamlessly through text and speech, utilizing the nuances of contextual interpretation, understanding, and manipulation.

II. Literature Review

The literature review section of the VoiceCode IDE project delves into the rich tapestry of existing research, exploring key themes at the intersection of voice-based programming, natural language processing (NLP), and artificial intelligence (AI). Beginning with an examination of foundational works in NLP, the review scrutinizes how language models, especially transformer models, have evolved to comprehend and generate human-like language.Insights from studies on transfer learning methodologies within the AI domain provide a theoretical foundation for VoiceCode IDE's innovative approach in adapting a pre-trained transformer model for voice-to-code translation. Furthermore, the section investigates prior works on developing custom IDEs and explores how these interfaces have evolved to accommodate diverse user interactions.

10.5281/zenodo.12526366

Comparative analyses of voice-enabled IDEs and their successes and limitations provide valuable insights into the design considerations and potential pitfalls that the VoiceCode IDE project may encounter. The literature review culminates with a synthesis of existing knowledge, identifying gaps in current research and laying the groundwork for the unique contributions of the Voice Code IDE project. By synthesizing insights from studies in NLP, voice recognition technologies, and custom IDE development, this section provides a comprehensive understanding of the project's context, positioning

III. Methodology

The proposed system represents a pioneering endeavor at the intersection of Natural Language Processing (NLP), Automatic Speech Recognition (ASR), and code generation. At its core, the system aims to seamlessly translate spoken language into syntactically correct Python source code through the integration of advanced technologies and methodologies. Leveraging the power of Deep Learning (DL), the system harnesses a pre-trained Transformer model with a transfer learning approach, utilizing the English to Python dataset for training. This strategic combination not only enables the model to comprehend and process spoken commands but also facilitates the generation of code that adheres to Python's syntactical norms

The incorporation of the Word2Vec model enhances the voice-to-text conversion process, enriching the input for the Transformer model. To showcase the system's functionality, a custom Python Integrated Development Environment (IDE), known as PyVoice IDE, has been meticulously crafted. PyVoice IDE acts as the interface between users and the underlying technology, providing a user-friendly platform for generating source code through voice input. The top-level architecture seamlessly orchestrates the system's components, initiated by voice commands through PyVoice IDE

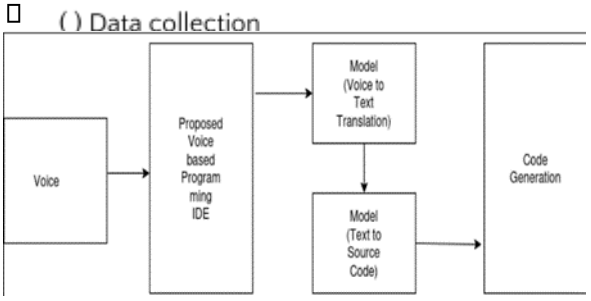
The system effectively listens to and processes voice commands, converting them into text using the pre-trained

(*sponsors).*

model. This text serves as input for the text-to-source code generation model within the Transformer architecture, producing syntactically accurate Python code.

The proposed system not only marks a significant advancement in voice-based code generation but also holds the promise of democratizing programming by offering an intuitive interface for users to articulate their coding needs verbally. By amalgamating cutting-edge DL techniques with user-friendly tools like PyVoice IDE, the system positions itself at the forefront of human-computer interaction, paving the way for future research to broaden its language capabilities and transcend the limitations of English-only datasets. This system exemplifies a transformative approach to coding, where the spoken word becomes a powerful catalyst for generating syntactically accurate code, ushering in a new era of accessibility and efficiency in software development.

IV. Proposed System



In the endeavor to train an effective voice-to-text model and a code generation model, the selection of appropriate datasets plays a pivotal role in shaping the system's capabilities. The Voice to Text DataSet constitutes a critical component, and the chosen Common Voice dataset proves to be an invaluable resource. Specifically focusing on the English language, this dataset offers a vast repository of audio files, each meticulously paired with corresponding text files and In essence, the thoughtful selection of these datasets underscores a commitment to inclusivity and diversity in training the voice-to-text and code generation models.

The datasets serve as the bedrock upon which the models cultivate their understanding of language nuances and coding intricacies. As the system endeavors to revolutionize the interaction between human voice and machine-generated code, the judicious choice of these datasets ensures that the models are well-equipped to handle the complexities inherent in both spoken language and programming syntax. Moreover, the extensive nature of the datasets provides a robust foundation for training, allowing the models to generalize effectively and deliver accurate and contextually relevant outcomes across a diverse range of scenarios in both voice- to-text conversion and code generation.

Model selection

The heart of our project beats with the robust and sophisticated Transformer architecture, chosen as the primary model for our sequence-to-sequence learning endeavors. This architectural marvel, renowned for its triumphs in Natural Language Processing (NLP) applications, aligns seamlessly with the core objectives of our project—transforming voice commands into syntactically correct Python source code. This choice is akin to selecting a masterful conductor to orchestrate the intricate symphony between spoken language and coding syntax. The Transformer architecture stands out in the AI realm, particularly for its prowess in understanding the complexities of human language

Our project necessitates not merely the translation of words but a deep comprehension of context, intent, and sequence. The Transformer excels precisely in this arena, akin to a language virtuoso capable of deciphering the subtleties of spoken instructions and transmuting them into coherent lines of Python code. At its essence, the Transformer acts as a linguistic magician, leveraging attention mechanisms to discern and prioritize specific elements within the spoken input while crafting the output Python code. This attention to detail is paramount, especially in the nuanced realm of coding syntax and logic.

By embracing the Transformer's capabilities, our project ensures a profound understanding of the relationships within spoken phrases, ultimately generating Python code that adheres not just to language rules but also to the logicalconstructs essential in coding paradigms. In documenting our project, it becomes imperative to underscore the transformative role that the Transformer architecture plays. It serves as the linchpin, the intellectual powerhouse that elevates our system's proficiency in translating voice commands into accurate, meaningful, and syntactically precise Python source code. This choice, rooted in the Transformer's proven success in NLP applications, exemplifies a strategic and informed decision, propelling our project toward the forefront of voice-based coding innovations.

C. Training the Transfer Model

To harness the capabilities of the Transformer model for our project, we've strategically employed Google Colab, a cloud-based Jupyter notebook environment, as our training ground. The rationale behind this choice is twofold: accessibility and computational power. Given our constraints with limited access to high-performance Graphics Processing Units (GPUs), Google Colab emerges as a fitting solution, offering a cloud- based platform equipped with GPU support. This proves to be a game changer for our project, as training deep learning models, especially as intricate as the Transformer, demands substantial computational resources. Google Colab provides us with the necessary firepower to execute computationally intensive tasks without the burden of high- end hardware requirements. This technique involves leveraging knowledge gained from a pre-trained model to boost the performance of our specific task, in this case, generating syntactically correct Python source code. The pre- trained Transformer model, initially acquainted with a wealth of linguistic nuances and patterns, serves as the foundation. We fine-tune this pre-existing knowledge by exposing the model to our English to Python dataset, a tailor-made compilation crafted to acquaint the model with the intricacies of converting spoken language into Python code.

The beauty of transfer learning lies in its efficiency. By building upon an already adept model, we circumvent the need for exhaustive training from scratch, significantly reducing the computational burden and time required. This process allows our model to adapt and specialize, honing its ability to generate Python code that not only follows syntactic norms but is also contextually relevant. It's akin to giving our Transformer model a head start in understanding the nuances of converting English instructions into Python code, a strategic advantage that aligns perfectly with our project's objectives of accuracy and efficiency. In essence, Google Colab and transfer learning emerge as the dynamic duo, equipping our project with the tools and methodologies essential for training a proficient and context-aware Transformer model.

Development of PyVoice IDE

PyVoice IDE serves as the bridge between users and the underlying technology, offering an intuitive platform where spoken instructions seamlessly transform into syntactically correct Python source code.The design and development of PyVoice IDE prioritize user experience, aiming for a fluid and natural interaction between human voice and machine- generated code. Its user-friendly interface provides a canvas where users can articulate their coding needs verbally, free from the constraints of traditional keyboard input. The integration of voice-based commands into the coding work flow marks a paradigm shift, unlocking a more dynamic and expressive mode of communication with the code generation system. The true magic happens when the trained Transformer model, our linguistic virtuoso, takes center stage within PyVoice IDE. By seamlessly integrating this model into the IDE, we enable PyVoice to comprehend and interpret spoken words with remarkable accuracy. The transformation from voice commands to text is orchestrated flawlessly by the trained Transformer model embedded within PyVoice IDE. This converted text becomes the input, the linguistic blueprint, for the subsequent generation of Python source code. The synergy between PyVoice IDE and the Transformer model is where the real alchemy occurs. Users articulate their coding intentions through voice, PyVoice IDE translates these spoken words into text with the aid of the Transformer model, and finally, this text serves as the creative canvas for generating syntactically precise Python code. The integration ensures that the generated code not only adheres to language rules but also encapsulates the contextual nuances inherent in the spoken instructions. In essence, PyVoice IDE is more than just a coding environment; it's a facilitator of a new, voice-centric coding experience. It brings together cutting-edge technology and user-centric design to redefine the boundaries of how we interact with code, making coding more accessible, dynamic, and attuned to the natural cadence of human speech.

Validation and Testing

The validation and testing phase of our project is a critical juncture where the efficacy and reliability of our voice-based code generation system undergo meticulous scrutiny. Our primary objective during this phase is to ensure that the system not only comprehends a diverse range of voice commands

accurately but also consistently generates syntactically correct Python source code in response to these inputs. The validation process commences with a comprehensive examination of the trained Transformer model. We subject the model to a battery of test cases, encompassing a spectrum of spoken instructions that simulate real-world usage scenarios. This enables us to assess the model's ability to generalize its learning and accurately interpret a variety of spoken commands. Validation also involves checking the model's response to variations in accents, pacing, and linguistic styles, ensuring that it remains robust and adaptable to the diverse ways users may articulate their coding intentions. Following the model validation, we transition to the PyVoice IDE, our user interface, where the real-world interaction between users and the system unfolds. Users are invited to articulate voice commands representing common coding tasks, ranging from variable assignments to control flow structures. This live testing within the PyVoice IDE allows us to evaluate the system's responsiveness, accuracy, and user- friendliness.

It also serves as an opportunity to gather valuable feedback from users, facilitating iterative improvements to enhance the overall user experience. Furthermore, we conduct extensive testing on the system's code generation capabilities, ensuring that the generated Python code not only adheres to syntactic norms but is also contextually relevant and functionally accurate. This phase involves the creation of a diverse set of test cases, covering various programming constructs and language intricacies, to validate the system's proficiency in transforming spoken language into precise and meaningful code.

References

[1] J. Rownicka, S. Renals and P. Bell, "Simplifying very deep convolutional neural network architectures for robust speech recognition," in IEEE Automatic Speech Recognition and Understanding Workshop (ASRU),Okinawa, 2017

[2] Henderig,Sellik NLP techniques for code generation driving pattern recognition, IEEE International Conference on Big Data 2017

[3] P. J. Rani, J. Bakthakumar, B. P. Kumaar, U. P. Kumaar and S.Kumar, "Voice controlled home automation system using Natural Language Processing (NLP) and Internet of Things (IoT)," in Third International Conference onScience Technology Engineering &Management (ICONSTEM), 2017.

[4] Vashishta, J. P. Singh, P. Jain and J. Kumar, "Raspberry PI based voiceoperated personal assistant," in International Conference on Electronics And Communication and Aerospace Technology, 2019.

[5] Deep Learning for Source Code Modeling and Generation: Models- H. Chen., T. Le and M. Babar.

[6] X. Li, K. Li, D. Qiao, Y. Ding and D. Wei, "Application Research of Machine Learning Method Based on Distributed Cluster in Information Retrieval," in International Conference on Communications, Information System and Computer Engineering,Haikou, China, 2019.