AN NATURAL LANGUAGE PROCESSED WEB APPLICATION THAT INTERPRET AND CONVERT ENGLISH TO PYTHON CODE

Sunny Zhao[1], Ang Li[2]

[1]St. Margaret Episcopal School, 31641 La Novia Ave,
San Juan Capistrano, CA 92675
[2]Computer Science Department, California State Polytechnic University,
Pomona, CA 91768

ABSTRACT

As the exchange between natural language and program code gradually becomes the need of industry, more and more interpreters and translators are required. Such natural language interpreters and converters can benefit society in a variety of fields, such as service industry, communication industry, and engineering industry [6]. Concise and accurate language processors will greatly boost the productivity of bottom repetitive works, provide examples and inspirations for students and industry workers, and become the tendency of the future. This paper introduces an application using natural language processing and neural network to effectively interpret and translate English to Python code, and detailly present the structural flow of the application [7]. This paper will also introduce the structure of the neural network, its validity, and how the python torch was applied and integrated. Furthermore, it will demonstrate the application and limitation of this model as well as its future improvements. We applied our application to educational needs and conducted qualitative evaluation of the approach. The result shows a beneficial and potential effect that is applicable to a greater field.

KEYWORDS

Natural language Processing , English to Python , Web Application

1. INTRODUCTION

As computers and the internet boomed and flourished in our modern world, humans greatly benefited from its conveniences. While more people start to learn programming and choose to enter the industry of Computer Science, it is easy to get lost in the enormous informational internet. Students may find confusion and even being falsely directed either by the search engine or the clickbaited websites [1]. Such troubles may waste students' time and effort, and even cost students' interest in the subject of Computer Science [8]. Hence, this subject is created to solve this problem: to save students' time and effort by providing a quick access to programming and algorithmic coding examples. Though the implication of the project is for the students and those who are interested in the field of programming, it can also be used by industry workers to refresh their memory and help on some repetitive jobs, such as handling file strings. Industry workers often forget the exact details of an algorithm but remember its general structure as they haven't used them for years. However searching online would be an inefficient method because they merely need the details of the code, not even to mention the inconsistency of the qualities of these websites. An application that can quickly provide access to it will perfectly suffice their needs. Considering the needs of the public and industry in present and future, the conversion between natural language and code has a huge market and potential. Not only will it instantly improve the efficiency of learning, but it will also become crucial to the future Artificial Development (AI) [9]. The interaction between human and AI will require the ability to convert between natural language and code of AI.

An existing related tool that has been proposed is ChatGPT [10]. ChatGPT is an artificial intelligence chatbot that provides a conversational interface that allows users to input natural language and output answers. Unlike traditional chatbots, ChatGPT records user's earlier requests for the follow-up questions, rejecting inappropriate requests, and challenging incorrect responses. Because ChatGPT was trained in a diverse range of text data including books, articles, and online conversations, it has shown a remarkable level of accuracy, sensitivity, and sophistication[2]. Though ChatGPT has wider applicability and stronger performance, its issues are blatant. The first problem is its potential disruption of academic honesty and education in general. As the COVID-19 shifts our world online, a rising trend of online education has emerged. Among educators, it is believed that academic misconduct is on the rise and the online assessment is particularly conductive of cheating. OpenGPT, in this case, is capable of doing such work [11]. Its sophistication in dialogue and on a variety of topics as well as its ability of generating compelling and accurate answers to difficult questions are extremely prone to being exploited and misused. However, our application will not cause such problems since it only provides users examples of code. Our application strives for educational purposes while maintaining and protecting the academic sphere. Although OpenGPT is likely to exacerbate academic dishonesty, its characteristic of close source is also a potential problem. Since it is closed source, users' data may be illegally recorded and used. However, our application will not cause such a problem since it's open source. Though OpenGPT demonstrated unprecedented sophistication, it is built on the expense of an enormous amount of dataset and scientists' efforts, and it is likely inexplicable. However, our application uses a simple and relatively effective approach that is highly replicable. We proposed and built a simple, replicable but effective neural network model on interpreting natural language and converting natural language to code, the English to Python converter, or ETP. Our goal is inspired by the OpenAI projects which includes ChatGPT and InstructGPT. Our application integrates backend programs and frontend websites to provide a smooth and consistent experience to the users. There are some good features of our application. First, the dataset we used to train the neural network is highly reliable because of its coverage on a wide range of coding examples. The training dataset consisted of approximately 5,000 classic examples that covered the majority of the common problems. This dataset is what enables our neural network model to maintain high accuracy and flexibility. The second feature is the neural network model we used.

To prove the capability and test the accuracy of our model, we conducted an experiment consisting of various input prompts and compared them to the standard answers. The comparison test proceeded as follows: after we get the output code from our model, we will give the same input to both our program and the program of the standard program. Then, we compared the output from our program and that of standard program's. Note that before that we fixed and ignored the potential error from the format and index. The reason is that the format and inconsistency do not interrupt the comprehension of the users regarding the coding examples. We collected the experiment data and analyzed its accuracy and structures based on these 25 input prompts. The comparison was focused on the similarity of code structure between the programs generated from our model and those of standard answers and the outputs. Afterward, we randomly generated 50 prompts that are common among beginner programmers and input them into our model to test its accuracy. Lastly, we published our model and selected a number of users to test the applicability of our model. By doing the steps above, we were able to obtain both the general accuracy and applicability of our model. The following steps were repeated with the

ChatGPT as a comparison group such that we can know the efficiency ratio between ChatGPT and our model.

The rest of the paper is organized as follows: Section 2 gives the details of the challenges that we met during the experiment and designing the samples. We will talk about why we encounter these problems and how we solved them. Section 3 focuses on the details of our solutions corresponding to the challenges that we mentioned in Section 2. We will also demonstrate the structural flow of our application through the use of visual diagrams and specific discussion. Section 4 presents the relevant details about the experiment we did, following by presenting the related work in Section 5. Finally, Section 6 gives the conclusion remarks, as well as pointing out the future work of this project.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Handling Training Data

The first challenge we encountered was to handle the training data in the csv file we obtained. In order to decide what ways to handle the dataset, we have to first decide in what ways the data is going to be fed into the model. At first, we thought of a hashmap to store and use the data [12]. However, using hashmap is an inefficient and incomplete way to handle the dataset and achieve what we expect: to filter the strings while preparing for the natural language processing. Then we proposed another way to handle and filter the strings: using the tokenizer library in Python. The tokenizer library provides string tokenization that is easy to implement, store, and reuse. Through using the tokenizer library, we are able to purify the data, differentiate between keywords, and make the following process easier.

2.2. Designing Structures and Choose Tools

The second challenge we encounter is to design the structures and layers of our model. In order to implement our thoughts, we have to carefully choose in what ways and what tools we are going to implement. Luckily Python provides a variety of libraries that are easy to access and understand. After careful consideration, we choose PyTorch, an optimized tensor library for deep learning using GPUs and CPUs, as our main library. PyTorch provides us operations and prebuild models that are highly useful, such as CUDA operations and Normalization Layers [13]. Initially, we struggle on deciding the amount of layers the neural network would have, because we neither want the layers to consume excessive computing resources nor the layers lack accuracy. However, we eventually came to the conclusion that there our model should contain 3 layers: Encoder Layer, Decoder Layer, and Multihead-Attention Layer. We also decided using supervised learning as our training model, simply it's the most suitable in our case.

3. SOLUTION

English To Python Converter (ETP) is a web application that uses neural network and natural language processing to interpret and translate English to Python code. The purpose of ETP is to provide a platform where students and industry workers can easily access and learn. It is strived to solve the problem of inconsistent qualities of the resources of the internet and the time consumption resulting from such attempts. ETP has two parts, the backend neural network and the frontend website. The frontend website provides basic instructions and information on the project, while the backend is the actual neural network model application. The backend of the project is built entirely based on Python. It utilizes Python libraries such as numpy, tokenizer, and PyTorch. The construction of ETP contains many steps. The first step is data cleaning and natural language processing on the training dataset that contains relevant high frequency data. After the data is cleaned and the keywords removed, the tokenization takes place. Tokenization will categorize and prepare the data to be fed into the neural network model. The neural network model is composed of 3 layers: Encoder Layer (EL), Decoder Layer (DL), and Multihead-Attention Layer (MAL). As its name suggests, the Encoder layer is composed of multiple Encoder nodes and takes a tokenized string as input. The Encoder class is created using PyTorch, same as MAL and DL. In the Multihead Layer, linear regression is used to model the relationship between the scalar response and the explanatory variables. The output computed from MAL will then be passed to the DL, where the untokenization will occur. In the DL, the output will also be evaluated and backward propagated if necessary.

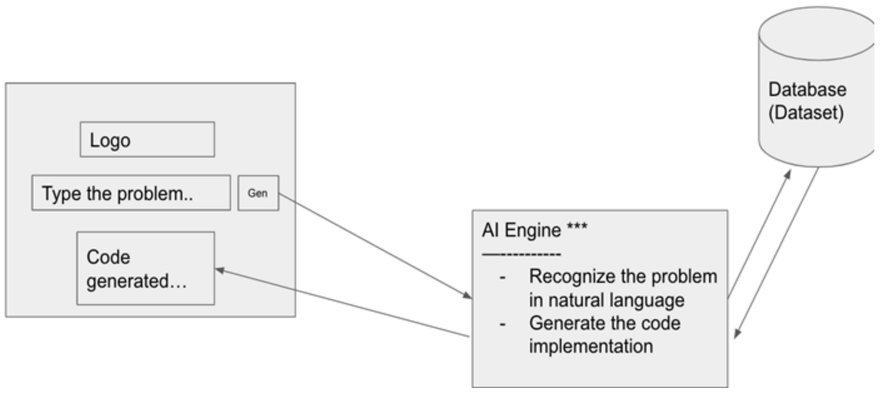
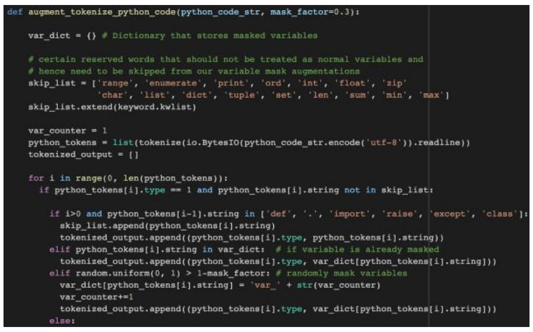

Logo
Type the problem..
Gen
Code generated…
AI Engine ***
- Recognize the problem in natural language
- Generate the code implementation
Database (Dataset)

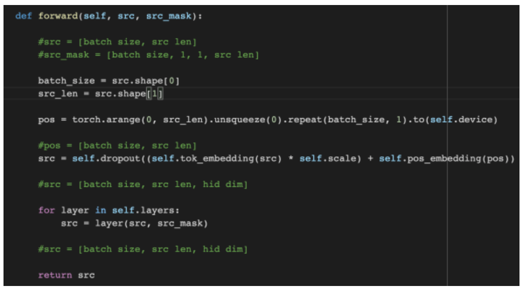
Figure 1. Overview of the solution

The first step before building our neural network is data cleaning and preparation. After we obtain our training dataset file, we first read the data and store them into a dictionary. Then we utilized the Python Tokenizer library to tokenize and mask the string while filtering out the keywords (Figure 2). The dataset was split into two parts: 80% of them would be used to train the model, and the rest would be used to test the model. After the dataset was prepared, the next step was to construct the Encoder class and the Encoder Layer class. The Encoder class was composed of its attributes and the forward function, and it includes the process of tokenizing input strings. The Dropout method was used throughout the model for regularization and preventing the co-adaptation of feature detectors. Note that the scale attributed used the sqrt function and FloatTensor to process the input (Figure 3). The forward function then integrates the variables and passes it to the Multihead Attention Layer. The MA nodes processed all the input again with the sqrt function in a multidimensional matrix and passed the output into the decoder nodes. We decided to use linear regression in the MAL to adjust the weights of the MA nodes, simply because the feature that linear regression would minimize the discrepancy of predicted and actual output numbers was the most suitable in this case. The purpose of the decoder layer is to compute the final input from the hidden layer and unmask the output into the corresponding strings according to the natural language processing. The Decoder class had a similar structure compared with the Encoder class, which used the Embedding function and FloatTensor model.



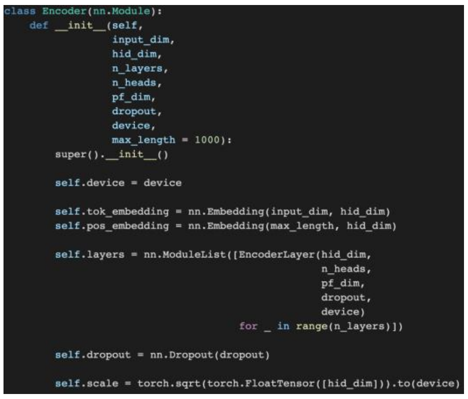
```python
def augment_tokenize_python_code(python_code_str, mask_factor=0.3):

    var_dict = {} # Dictionary that stores masked variables

    # certain reserved words that should not be treated as normal variables and
    # hence need to be skipped from our variable mask augmentations
    skip_list = ['range', 'enumerate', 'print', 'ord', 'int', 'float', 'zip'
                 'char', 'list', 'dict', 'tuple', 'set', 'len', 'sum', 'min', 'max']
    skip_list.extend(keyword.kwlist)

    var_counter = 1
    python_tokens = list(tokenize(io.BytesIO(python_code_str.encode('utf-8')).readline))
    tokenized_output = []

    for i in range(0, len(python_tokens)):
      if python_tokens[i].type == 1 and python_tokens[i].string not in skip_list:

        if i>0 and python_tokens[i-1].string in ['def', '.', 'import', 'raise', 'except', 'class']:
          skip_list.append(python_tokens[i].string)
          tokenized_output.append((python_tokens[i].type, python_tokens[i].string))
        elif python_tokens[i].string in var_dict:  # if variable is already masked
          tokenized_output.append((python_tokens[i].type, var_dict[python_tokens[i].string]))
        elif random.uniform(0, 1) > 1-mask_factor: # randomly mask variables
          var_dict[python_tokens[i].string] = 'var_' + str(var_counter)
          var_counter+=1
          tokenized_output.append((python_tokens[i].type, var_dict[python_tokens[i].string]))
        else:
```

Figure 2. String Filtering and Tokenization



```python
def forward(self, src, src_mask):

    #src = [batch size, src len]
    #src_mask = [batch size, 1, 1, src len]

    batch_size = src.shape[0]
    src_len = src.shape[1]

    pos = torch.arange(0, src_len).unsqueeze(0).repeat(batch_size, 1).to(self.device)

    #pos = [batch size, src len]
    src = self.dropout((self.tok_embedding(src) * self.scale) + self.pos_embedding(pos))

    #src = [batch size, src len, hid dim]

    for layer in self.layers:
        src = layer(src, src_mask)

    #src = [batch size, src len, hid dim]

    return src
```

Figure 3. Forward function of Encoder class



```python
class Encoder(nn.Module):
    def __init__(self,
                 input_dim,
                 hid_dim,
                 n_layers,
                 n_heads,
                 pf_dim,
                 dropout,
                 device,
                 max_length = 1000):
        super().__init__()

        self.device = device

        self.tok_embedding = nn.Embedding(input_dim, hid_dim)
        self.pos_embedding = nn.Embedding(max_length, hid_dim)

        self.layers = nn.ModuleList([EncoderLayer(hid_dim,
                                                  n_heads,
                                                  pf_dim,
                                                  dropout,
                                                  device)
                                     for _ in range(n_layers)])

        self.dropout = nn.Dropout(dropout)

        self.scale = torch.sqrt(torch.FloatTensor([hid_dim])).to(device)
```

Figure 4. Attributes of Encoder class

The frontend web application is rather simple. We created a website integrated with Javascript which acted as an agent to transfer the user input into the model. We used Python Flask as our server. Since we have already created the interface function of the model, we can just connect it to the website.

4. EXPERIMENT

4.1. Experiment 1

Our trained model is able to receive the input and compute the output through its neural network. It does so through the Sequence to Seqence neural network model. We create a eng_to_python that accepts string input parameter. The rest of the experiment would be using eng_to_python function as a interface. The first experiment that we proposed is a small dataset only contains 25 experiment data. These 25 data had been standardized and they were designed specifically for the model. These 25 testing data all start with the command of "Write a program/function that.." or "a program/ function that…". Because of our model is train using such similar structured data, other structured commands might not be interpreted thoroughly by the model. The purpose of the first experiment is merely to test if the model is performing as expected under the predetermined constraints. The process of the evaluation proceeds as followed:

1. ETP model receive the input
2. Output generated from ETP model being collected
3. Output program and Control program take in the same input
4. Comparing results and collecting data

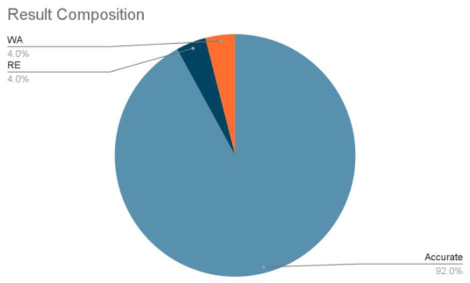
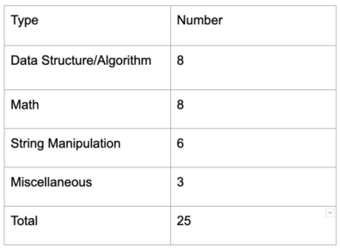


Figure 5. Result composition



Type	Number
Data Structure/Algorithm	8
Math	8
String Manipulation	6
Miscellaneous	3
Total	25

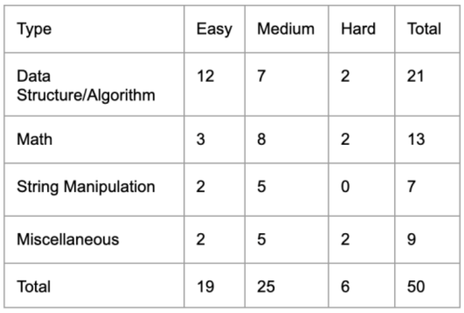
Figure 6. Table of experiment 1

The experiment data collected demonstrates a positive and optimistic result. According to the Result Composition, the wrong answer and runtime error each occupy 4.0% of the 25 data test, and the accurate answer occupies 92.0% of the 25 data test. Despite the result being certainly exhilarating and affirmative to our effort, there are multiple limitations and concerns in this experiment. The first problem and limitation is the bias of the dataset. Due to the limited size of the dataset, the result collected from this experiment is certainly not representative of all. In order to truly test out the capability and margin of error of the ETP model, more and more distinct structured data is inevitable. Besides the limited data size, the ways these data are formatted can also cause bias and deviation. According to the experiment process that we designed, all of the inputs are formatted in a specific way that was used in the ETP model. However, the effect of more generalized input on the output remains unknown.

4.2. Experiment 2

In order to test out the true capability and effectiveness of the ETP model, we proposed a more random and authentic experiment. We found 10 subjects and let each enter 5 commands into the ETP model. The occupation of the experiment subject ranging from Software Engineers to Teachers and to students, and each of them have backgrounds on programming in some extent.

Their proposed commands were classified and rank by the participants themselves based on 3 difficulties, easy, medium or hard. At the end of the experiment, we required testing subjects each to rate their satisfaction level to the ETP model out of 3, with 3 being "this is exactly what I want", 2 being "this somewhat helpful", and 1 being "this is of no use". The overall experimental data was collected and presented below.



Type	Easy	Medium	Hard	Total
Data Structure/Algorithm	12	7	2	21
Math	3	8	2	13
String Manipulation	2	5	0	7
Miscellaneous	2	5	2	9
Total	19	25	6	50

Figure 7. types of commands

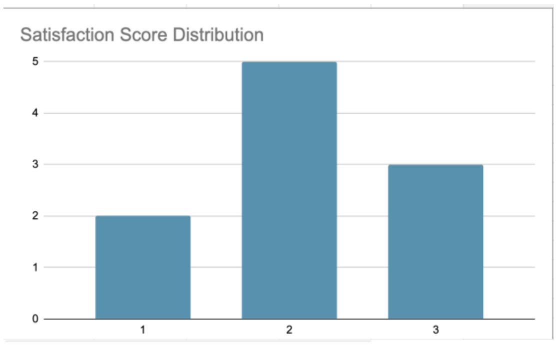


Figure 8. Satisfaction Score Distribution of the users

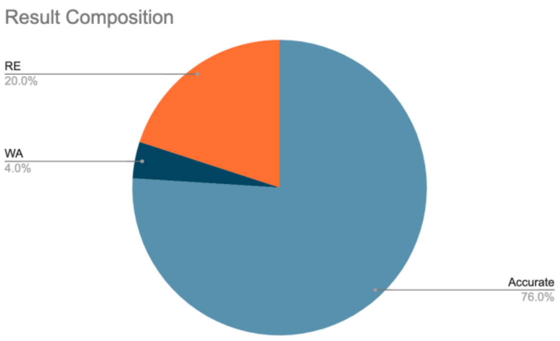


Figure 9. Result Composition

The result is from the second experiment is somewhat expected to us. ETP shows a high level successful convert rate on the easy and medium commands. Owing to the fact that Data Structure can be implemented through in-build functions, they are the most capable question type for ETP. At the same time, data structure may be a challenging part for programming beginners because their lack of experience. In this sense, ETP has fulfill its job to provide a easy-to-access platform for students. According to the data in figure 8, the people who rated a score of 3 occupied the largest percentage, which indicate that ETP has relative usefulness and value. However, that fact that 20% of participant gave a score of 1 suggest that there are still many insufficiency in the model. Problems such as inability to accurately process hard or abstract questions are still need to be considered in the future.

5. RELATED WORK

This study illustrates the definition and new methodology of Natural Language Understanding. It explores the steps and components of a Natural Language Model, which includes syntactic and semantic aspects [3]. We borrow inspiration from this work into our work. Especially its redefinition of the fundamental goals of Natural Language Understanding, which individual strings should not be interpreted exactly. Compared to our work, the work by Bates merely proposed a general model of Natural Language Understanding. The generic NL system he proposed, however, did influence our model since we borrowed the concept of Parser, Semantic Processor, and Reasoner. Though the work done by Bates had great influence on those who were involved in the industry, he did not dive into the technical details. In comparison, we propose an actual and sophisticated model with technical details which can be employed in the market.

The work by Carter and Rayner focused on the translation between spoken languages, such as spoken English to spoken Swedish [4]. The paper focused on the integration of speech translator and language translator. Our work has some similarities with the work by Carter and Rayner. Both contain recognition of finitestate grammar. This finite-state model has its advantages of being fast, robust, and easy to train. Though it is insufficiently expressive to capture many important types of linguistic regularities, we compensate for that through some natural language processing optimizations. What's different between our research and theirs is that they apply a conventional pipeline N-best interface, which integrates their translator and understanding. Carter and Rayner's work demonstrates values primarily on communications. This robust and new recognition model will most likely solve problems of trans-cultural communication. However, the value of our work is shown in greater areas, such as education and engineering.

This study used natural language processing in analyzing contend of COVID-19. A model improved on the base of the BERT model was proposed, which followed the principle of bidirectional transformer models on unlabeled text corpuses[5]. Our work has many aspects that are similar to their work. For instance, we both used the traditional methods of mask language modeling and sentence order prediction. However, their model was trained in a unsupervised manner while we chose supervised training. Besidesa, the variations on the ways of modeling and training, the Covid-Twitter-Bert model was able to detect sentiments and provide interesting perspectives and data to the pandemic. The model used Stanford Sentiment Treebank (SST-2) as its sentiment corpuses, which effectively provided sentiment differentiation by classified keywords as positive and negative.

6. CONCLUSIONS

To give a general overview of our study, we summarized all of the above sections and reexplained them here. Essentially, we proposed an application called The English to Python Converter, or ETP, to solve the problem of lacking an integrated and consistent tool that provides qualitative code examples. We separate our application into frontend website and backend natural language processing and neural network model. We used Python library Pytorch for modeling the neural network system, and Python library Tokenizer for modeling the natural language processing. We first filtered and parsed the data using Python Numpy and Python Pandas. Then, we masked and tokenized the strings for natural language processing. In regard to the layout of the neural network model, we proposed a three-layer structure : encoder layer, multihead-attention layer, and the decoder layer. The whole model can be regarded as a conventional Sequence to Sequence integrated with optimizations techniques such as counting entropy lost and dropout. We applied our application to 2 experiments and concluded with the claim that ETP has its effectiveness. In the first experiment, we created 25 experiment samples each with answers and classified by type and input them into our model to test its effectiveness. The result of the first experiment demonstrated that ETP has a high accuracy rate on easy commands. In the second experiment, we invited 10 participants to each propose 5 commands to ETP and rate their satisfaction. The result collected showed that though ETP hasn't demonstrated sophistication on high level or abstract commands, it is fully capable of intermediate and low level commands or even some high level commands specified with special words. In addition, participants have shown relatively positive comments on ETP, which is a positive sign to our goal for publication and that of helping more people.

Some limitations of ETP are relatively obvious. First of all, the lack of a more comprehensive dataset limited its ability to produce problems that are out of range. In other words, ETP maintains high accuracy only on those commands that are similar or identical in its dataset. The second limitation may be the natural language processing component. There still exists a lot of space for optimization and improvement. Some optimization or even reconstruction, such as Bayesian Network and Hidden Markov Model, can be applied to our model [14][15]. The problem witha basic natural language processing component is the inability to comprehend complex commands. Commands that are out of the vocabulary range of ETP, such as “provide, write me”, are of none zone to ETP.

The limitations mentioned above can be solved through various means. The limitation of lacking a comprehensive dataset, for instance, can be solved by acquiring such a dataset and feeding it into the model. One approach to do it is to collect open data from the internet. Since there are various platforms that provide coding examples, a web scraper program can easily access such resources. Another approach to solve this is through the voluntary feedback of the users. Users can evaluate the result they got and give ETP feedback that can improve the model, such as the example program that should be generated. Such an approach will surely need some modifications. In regard to the second limitation, such optimizations are fairly simple to implement and thus should not be a big issue.

REFERENCES

[1] Zeng, Eric, Tadayoshi Kohno, and Franziska Roesner. "Bad news: Clickbait and deceptive ads on news and misinformation websites." Workshop on Technology and Consumer Protection. 2020.

[2] Lehmann, Donald R., Leigh McAlister, and Richard Staelin. "Sophistication in research in marketing." Journal of Marketing 75.4 (2011): 155-165.

[3] Dong, Li, et al. "Unified language model pre-training for natural language understanding and generation." Advances in neural information processing systems 32 (2019).

[4] Jiang, Zifan, et al. "Machine Translation between Spoken Languages and Signed Languages Represented in SignWriting." arXiv preprint arXiv:2210.05404 (2022).

[5] Clifton, Chris, Robert Cooley, and Jason Rennie. "Topcat: Data mining for topic identification in a text corpus." IEEE transactions on knowledge and data engineering 16.8 (2004): 949-964.

[6] Frost, Richard, and John Launchbury. "Constructing natural language interpreters in a lazy functional language." The Computer Journal 32.2 (1989): 108-121.

[7] Clement, Colin B., et al. "PyMT5: multi-mode translation of natural language and Python code with transformers." arXiv preprint arXiv:2010.03150 (2020).

[8] Weiser, Mark. "Some computer science issues in ubiquitous computing." Communications of the ACM 36.7 (1993): 75-84.

[9] Paul, Debleena, et al. "Artificial intelligence in drug discovery and development." Drug discovery today 26.1 (2021): 80.

[10] Aydın, Ömer, and Enis Karaarslan. "OpenAI ChatGPT generated literature review: Digital twin in healthcare." Available at SSRN 4308687 (2022).
[11] Jiao, Wenxiang, et al. "Is ChatGPT a good translator? A preliminary study." arXiv preprint arXiv:2301.08745 (2023).
[12] Schwalb, David, et al. "NVC-hashmap: A persistent and concurrent hashmap for non-volatile memories." Proceedings of the 3rd VLDB Workshop on In-Memory Data Mangement and Analytics. 2015.
[13] Bakkum, Peter, and Kevin Skadron. "Accelerating SQL database operations on a GPU with CUDA." Proceedings of the 3rd workshop on general-purpose computation on graphics processing units. 2010.
[14] Rabiner, Lawrence, and Biinghwang Juang. "An introduction to hidden Markov models." ieee assp magazine 3.1 (1986): 4-16.
[15] Marcot, Bruce G., and Trent D. Penman. "Advances in Bayesian network modelling: Integration of modelling technologies." Environmental modelling & software 111 (2019): 386-393.