Research Article

Applying Code Transform Model to Newly Generated Program for Improving Execution Performance

Bao Rong Chang ¹, Hsiu-Fen Tsai ², and Po-Wen Su ¹

¹Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan

²Department of Fragrance and Cosmetic Science, Kaohsiung Medical University, Kaohsiung, Taiwan

Correspondence should be addressed to Hsiu-Fen Tsai; sftsai@kmu.edu.tw

Received 20 November 2020; Revised 17 January 2021; Accepted 20 January 2021; Published 2 February 2021

Academic Editor: Sikandar Ali

Copyright © 2021 Bao Rong Chang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The existing programs inside the voice assistant machine prompt human-machine interaction in response to a request from a user. However, the crucial problem is that the machine often may not give a proper answer to the user or cannot work out the existing program execution efficiently. Therefore, this study proposes a novel transform method to replace the existing programs (called sample programs in this paper) inside the machine with newly generated programs through code transform model GPT-2 that can reasonably solve the problem mentioned above. In essence, this paper introduces a theoretical estimation in statistics to infer at least a number of generated programs as required so as to guarantee that the best one can be found within them. In addition, the proposed approach not only imitates a voice assistant system with filtering redundant keywords or adding new keywords to complete keyword retrieval in semantic database but also checks code similarity and verifies the conformity of the executive outputs between sample programs and newly generated programs. According to code checking and program output verification, the processes can expedite transform operations efficiently by removing the redundant generated programs and finding the best-performing generated program. As a result, the newly generated programs outperform the sample programs because the proposed approach reduces the number of code lines by 32.71% and lowers the program execution time by 24.34%, which is of great significance.

1. Introduction

Alpha Go was developed by Google DeepMind in London in 2014, and it defeated all other Go masters. Since then, research in artificial intelligence has been increasing again. Accordingly, research about human-computer interaction is meant to imitate human behavior, especially natural language representation and interpretation in the voice assistant machine [1]. The use of artificial intelligence for human-computer interaction in voice assistant machines-related tools have flourished, such as Tesla's NoA, Apple's Siri, Amazon Echo and Alexa, and Google Home. These tools not only function as the basis of human language imitation but also play a key role for API offerings in AI applications. Nowadays, there are quite a lot of brands and types of voice assistant machines in the world, and their

existing programs are used for human-computer interaction in response to user requests. However, some voice assistant machines have encountered certain problems, that is, the machine may not be able to answer the questions correctly [2], or the existing programs have low execution efficiency [3]. It is worth exploring as to how to solve the above problems, and thus we have formulated the following idea. Can the machine transform an existing program to a newly generated program that can runs programs with higher efficiency and produce correct results?

Even though the deep learning model is relatively mature, imitation of human natural language behavior through deep learning is still a difficult task. Language model is a technology that allows machines to understand and predict human language. The use of recently developed language models with large-scale data and huge computing power can help solve

various applications of human natural language. Famous open-source pretrained language models, such as ELMo, BERT, and GPT-2 [4], can implement the best level of Natural Language Processing (NLP) tasks, and they have the ability required for huge hierarchical models. Through transfer learning and few-shot learning on such powerful natural language models, the solution to the complex NLP problem can be obtained.

The objective of this study was to explore ways to solve the following two problems. First, how to construct a complete semantic database that can implement data retrieval quickly and respond correctly. Second, how to generate high-performance programs through the code transform model to replace the existing programs inside the machine to improve execution performance. The natural language processing involves understanding and generating. Regarding the technology involved in the improvement method, a complete and quickly searchable semantic database using MariaDB is constructed with the Natural Language Toolkit (NLTK) model of sentence segmentation [5] to provide correct answers to users. At present, the most fluent open source natural language model is the Generative Pre-trained Transformer 2 (GPT-2) [6], which is a natural language simulation machine developed by using OpenAI. It is currently being used to generate fake news [7]. In this study, Python [8] provides the run-time environment for the aforementioned tools, namely, MariaDB, NLTK, GPT-2, and Tensorflow. Python combined with Hadoop Streaming to provide big data processing and distributed storage in Hadoop, and it can also be used with PySpark to provide big data analytics in Spark.

There is so far no way to infer how many programs generated from GPT-2 can guarantee a pass ratio over 90% without which we cannot find the best one among them. Nevertheless, this paper introduces a theoretical estimation in statistics to infer at least a number of generated programs produced by GPT-2, which guarantee that the best one can be found within them. This approach refers to the pre-determined the number of generated programs statistically. In order to improve the efficiency of keyword retrieval, both the parameters of filtering redundant keywords and adding new keywords are used to optimize keyword-search in the semantic database in order to improve the hit rate of keywords from the database. In addition, we present cosine text similarity algorithm Simhash [9] to check the code similarity of generated programs and example programs and use the longest common subsequence (LCS) [10] to verify the conformity between the execution results of generated programs and example programs.

The following paragraphs of this paper are arranged as follows. In Section 2, related work in word segmentation processing and language generation model will be described. The way to system implementation is given in Section 3. The experimental results and discussion is given in Section 4. Finally, we drew a brief conclusion in Section 5.

2. Related Work

A particular voice assistant machine, Amazon Alexa [11], is a smart assistant and Echo smart speaker developed in recent years. The audio system relies on a set of complex AI

technologies that use automatic speech recognition (ASR) to receive sound waves and to convert them into words. Then, natural language understanding (NLU) is used to translate these words into meanings. In order to respond to people, smart speakers and the technology of natural language generation (NLG) [12] for suggestions, content determination, conversation planning, sentence aggregation, vocabulary, citation expression generation, language experiment, and descriptions of basic NLG task have been developed. While exploring various emerging issues of AI, some researchers are pondering the possibility of machines automatically producing programs [13]. In reality, people are often pressurized to complete programming or modify the codes of a program before the deadline or the challenge of improving the execution of a program. As a result, the program execution speed may be too slow to get the job done, and there may still be deficiencies when running the program in real-time. In contrast, a high-performing new program can be generated through the code transform model, and it will operate faster than the original one [14]. This would be a great progress in the application of AI, and the audio-text conversion technology would further apply in the sound-controlled related products, such as drones, robots, and flying iron men.

There is not much literature on source-code to source-code transform model. First, Li et al. (2011) used XML and XSLT technology to generate a web page code [15]. They will be automatically converted into target code after being exported via XML files. However, they did not provide any data to show how the results were generated. Next, Li et al. (2020) employed Java code transform models, Java-Codetool, and CodeGeneration to produce newly generated java programs and evaluate their performance [16]. They took the Binary tree traversal program as an example and divided the program into 4 parts to produce newly generated source-code programs individually. The experimental results show that the average time to generate a code line in a program using Java-Codetool is about 0.17 seconds, and CodeGeneration takes 0.15 seconds. No matter whether it is using Java-Codetool or CodeGeneration, the number of code lines is not reduced. In fact, the compilation of all generated programs can be done successfully. However, it did not show further information about the execution results of their generated programs. In this paper, the proposed approach will present the code similarity check and make sure the conformity of the execution results between generated programs and sample programs that resulted in the credibility and validity of the findings in this study.

Uncertainties will exist depending on the situation. Despite this, the system can generate similar programs instantly to greatly reduce the possibility of errors or incomplete programs with GPT-2. In order to realize the “applying code transform model to newly generated program for improving execution performance”, this study will use the following key technologies: Anaconda (Data Science Platform with Virtual Environment Conda), Tensorflow (Dataflow and Differentiable Programming), NLTK (English Text Segmentation), GPT-2 (Text Generating Model), and

Simhash (Cosine Text Similarity Algorithm), etc., to achieve the goal of this research.

2.1. Word Segmentation Processing-NLTK. Natural Language Processing (NLP) [17] is regarded as a branch of AI and linguistics. This field discusses how to process and use natural language, including multiple steps, basically for cognition, understanding, generation; cognition and understanding are to let the computer turn the natural language input into interesting symbols and relationships. Natural Language Toolkit (NLTK) is a Python library commonly used in NLP research. This is an open source project, including data sets, Python modules, tutorials, etc.

The main functions of NLTK are English word segmentation processing, part-of-speech tagging [18] called pos, font restoration called lemmatization, stopword, named entity recognition called ner, etc. In NLTK, the text is usually stored as a list, that is, a text is a huge list. If additional information such as part-of-speech is attached, it can be converted into a dictionary. The Latin language system is a little troublesome because they like to add a modifier to the words to describe different tenses, actions, part-of-speech, mood, and quantity. So we will consider all the same words in different tenses or different changes into the same word for processing. Finally, we filter out unnecessary words. and the English word segmentation flowchart is shown in Figure 1.

2.2. Language Generation Model-Generative Pre-Training 2. The second generation of Generative Pre-Training (GPT-2) is an unsupervised [19] transformer language generation model [20], released by OpenAI in 2019. Researchers believe that the language model of unsupervised learning is a general language model. Furthermore, GPT-2 proves that the model is not meant for any specific task to predict the next word as the training target. It is used to adopt the sentence database WebTex [21] for data training, which contains 8 million web pages as the training data. These web pages are part of the data from Reddit [22] and are more than 40 GB. Compared with other text-generating models, such as ELMo and Bert for producing texts, its main advantage is that the code is in English and the one-way language model is easier to train and understand. The traditional Transformer model is composed of Encoder and Decoder, called the Transformer architecture stack. The transformer architecture stack is shown in Figure 2. This model solves the problem of machine translation.

In many subsequent studies, the Transformer architecture removes either the Encoder or the Decoder, uses only one Transformer stack, and provides a large amount of training text and machine equipment. GPT-2 is composed of the Decoder architecture according to the Transformer model. As shown in Figure 3, the stacking height is the size of various GPT-2 models. Currently, there are four sizes of models: GPT-2 Small, GPT-2 Medium, GPT-2 Large, and GPT-2 Extra Large [23].

2.3. Cosine Text Similarity Algorithm-Simhash. The traditional Hash [9] algorithm is only responsible for mapping the original content into a signature value equally and randomly. The principle is only equivalent to a pseudo-random number generating algorithm. Even if the two original contents differ in only one byte, the generated signature value may be very different. Therefore, traditional Hash cannot measure the similarity of the original content in the dimension of signature. Simhash algorithm [24] is a locally sensitive Hash algorithm, and its main idea is to reduce the dimensionality of feature vector. That is, Simhash algorithm is used to convert the high-dimensional feature vector into f-bit fingerprint [25] and to determine the similarity of two f-bit fingerprints by calculating the Hamming distance [26] of these fingerprints. The smaller the Hamming distance, the higher the similarity. The overall process is shown in Figure 4, which includes word segmentation, hash calculation, weighting, merging, and dimensionality reduction. Word segmentation is used to obtain N-dimensional feature vectors (64-dimensional default) for the word segmentation of the text; Hash is to perform the Hash calculation on all the obtained feature vectors. Weighting refers to weighing all the obtained feature vectors. Merging refers to accumulation of all the obtained vectors. Dimensional reduction changes the accumulated result greater than zero to one and less than zero to zero, obtains a text Fingerprint as shown in Figure 5, and finally calculates the Hamming distance between the two text fingerprints.

In the Information Theory, the Hamming distance between two equal-length character strings is the number of different characters at the positions corresponding to the two characters. The Hamming distance is the number of characters that need to be replaced to convert one string into another for a fixed length. Moreover, Hamming distance is a distance measure for the character vector space, and it maintains the measured distance with nonnegative, unique, and symmetrical. In Hamming distance formula equation (1) [27], d^{HAD} is the Hamming distance between objects i and j , and k is the index of the corresponding variable reading y in the total number of variables n . In equation (2) and equation (3), $[y_{i,k} \neq y_{j,k}]$ is the value of 1 or 0 given by the logical value True or False determined according to internal conditions $y_{i,k} \neq y_{j,k}$. The Hamming distance itself gives the number of mismatches between variables paired by k .

$$d^{\text{HAD}} = \sum_{k=0}^{n-1} [y_{i,k} \neq y_{j,k}], \quad (1)$$

$$[y_{i,k} \neq y_{j,k}] = 1 \text{ if } y_{i,k} \neq y_{j,k} \text{ is True,} \quad (2)$$

$$[y_{i,k} \neq y_{j,k}] = 0 \text{ if } y_{i,k} \neq y_{j,k} \text{ is False.} \quad (3)$$

If the Hamming distance is used to measure the similarity of the original content, the similarity can be converted into a pass ratio as a measure that tests the object based on

FIGURE 1: NLTK word segmentation process.

FIGURE 2: Transformer architecture stack.

FIGURE 3: GPT-2 Decoder architecture.

the original standard. According to the Hamming distance d^{HAD} and the total number of variables n , the qualification ratio formula equation (4) can be obtained.

$$q = \left(1 - \frac{d^{\text{HAD}}}{n} \right) \times 100\%. \quad (4)$$

2.4. Longest Common Subsequence (LCS). The Longest Common Subsequence [10], abbreviated LCS, is the problem of finding the longest common subsequence in all sequences in a sequence set (usually two sequences). This is different from the problem of finding the longest common substring (Longest Common Substring) in that the subsequence does not need to occupy consecutive positions in the original sequence. To solve the LCS problem, we cannot use the brute force search method. We need to use dynamic programming to find the length of the LCS and backtracking strategy to find the actual sequence of the LCS.

We assume that $z = \langle z_1, z_2, \dots, z_k \rangle$ is the LCS of x and y , and we observe that if $x_m = y_n$, then $z_k = x_m = y_n$, and z_{k-1} is the LCS of x_{m-1} and y_{n-1} ; If $x_m \neq y_n$, then z_k is the LCS of x_{m-1} and y_{n-1} , or the LCS of x_{m-1} and y_n . Therefore, the problem of solving LCS becomes two sub-problems of a recursive solution. However, in the abovementioned recursive solution method, there are many repeated sub-problems and the efficiency is low. The improved method uses space instead of time and uses an array to store intermediate states to facilitate subsequent calculations. Therefore, using the two-dimensional array $c[i, j]$ to record the LCS lengths of the strings $x_1, x_2, \dots, x_i$ and $y_1, y_2, \dots, y_j$, the state transition equation can be obtained in equation (5).

$$c[i, j] = \begin{cases} 0, & i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1, & i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j-1], c[i-1, j]), & i, j > 0 \text{ and } x_i \neq y_j. \end{cases} \quad (5)$$

The longest common subsequence is used to measure the similarity of the execution results of two programs [28]. In order to illustrate the degree of conformity of the respective output results of the two programs, the similarity is renamed as text conformity. First, convert the output results of individual programs into ASCII code, and then store them into arrays a and b individually, and then calculate the c array according to the longest common subsequence. Here, the lengths of the a , b , and c arrays are recorded as $|a|$, $|b|$ and $|c|$, and the length of the above array is substituted into the formula equation (6) to obtain the text conformity. Here, the lengths of arrays a , b , and c are denoted as $|a|$, $|b|$, and $|c|$, and the length of the above array is substituted into equation (6) to obtain LCS conformity, denote as f .

FIGURE 4: Simhash algorithm process flow.

FIGURE 5: Calculate fingerprint.

$$f = \frac{2 \cdot |c|}{|a| + |b|} \times 100\%. \quad (6)$$

3. Research Method

The purpose of this study was to produce newly generated programs for improving program execution speed using the code transform mode. In other words, in order to increase the efficiency of human-machine interaction like voice assistant machine, the existing programs (called sample programs in this paper) inside the machine is replaced with a new high-performance program using GPT-2. Nevertheless, the proposed approach, as shown in Figure 6, consists of three parts: prior keyword retrieval optimization, code transform model, and posterior verification of generated program. The first part is to imitate a voice assistant machine to segment a sentence, select keywords, and find sample programs. Next, a transform model is used to produce a number of new candidate programs according to the corresponding sample programs. The last one is to test and verify the new candidate programs and choose the one with the best quality as a pocket program. A pocket program is the one with the best performance among new candidate programs.

3.1. Newly Generated Program System. The objective of this paper is to explore the use of the second generation of Generative Pre-Training (GPT-2) as a code transform model to produce newly generated high-performance programs. In this way, the sample program in the semantic database can be replaced with a new program called the pocket program to implement high-performance execution due to reduced code lines and less execution time. With GPT-2 transform model, the proposed approach has to imitate a voice assistant system with prior optimization of keyword retrieval associated with the semantic database, and to build a content checker using posterior verification of program execution results between a sample program and the newly generated

program. The system includes model generation stage and model use stage, as shown in Figure 7. In the model generation stage, the user sends spoken sentences using text or voice and then proceeds with word segmentation to select keywords and find corresponding sample programs. After that, the system trains the model and generates preliminary programs in which some of the programs are chosen as qualified programs. Finally, a generative program model is confirmed and saved in the semantic database. During the model use stage, the user sends a spoken sentence in text or voice, and searches the semantic database for pocket programs that could be made earlier in the model generation stage. If not, it will go back to the step of generating preliminary programs, and then go through test and verification steps to get a new pocket program for execution.

The model generation stage is divided into the training phase and the test phase. Phase diagrams are shown in Figures 8–10. In the training stage, there are four units: word segmentation unit, sample program unit, generative program model unit, and the generated program unit. The inputs/outputs during training phase are natural language sentences, keywords, sample programs, generative program models, and preliminary programs. Natural language sentences are initially sent by a user. The word segmentation model NLTK is used to perform word segmentation. The system is used to select the keywords and then search the semantic database for the sample program that is corresponding to the keywords. Semantic database is built in the XAMPP [29] cloud server. System puts the sample program into GPT-2 for the first pass. GPT-2 uses a sample program to train and generate the generative program model. After modeling is completed, the model will be provided as feedback to GPT-2 again for the second pass on which a number of preliminary programs are produced. In the test phase, there are three units: test unit, verification unit, and the storage unit. Simhash algorithm is used to compare the similarity between the preliminary program and the sample program. Qualified programs are obtained as the preliminary program with the similarity pass ratio higher than the

FIGURE 6: Applying the code transform model to the newly generated high-performance programs.

FIGURE 7: Diagram of newly generated program system.

FIGURE 8: Model generation stage-training phase diagram.

predetermined one, and they are compiled with Python. Next, in the verification unit, we select the qualified program with the highest similarity pass ratio to compare with the corresponding sample program using LCS conformity that measures the conformance of their execution results, and chooses a qualified program that meets the predetermined conformity ratio as a pocket program. Finally, in the storage unit, the keywords, pocket programs, and generated program models will be stored in the semantic database.

With the trained generative program model found in semantic database, the corresponding selected keyword will be directed to the process of model use stage after the word segmentation. The stage diagram is shown in Figure 10. In the model use stage, there are five units: word segmentation unit, generative program model unit, generated program unit, test and verification unit, and storage unit. Likewise,

the word segmentation unit initially allows the user to send natural language sentences and use NLTK algorithm to segment a sentence and select keywords. Then, it will use keywords to search the semantic database for generative program models or pocket programs. If pocket programs are found, they will be executed directly. If not, it will go back to the step of generating preliminary programs, and then go through test and verification steps to get a new pocket program for execution.

3.2. System Execution Flow. Figure 11 shows the overall execution flow of the proposed approach. In the model generation stage, the user sends sentences or articles into the word segmentation model NLTK to perform word segmentation and select keywords from all separated words.

FIGURE 9: Model generation stage-test phase diagram.

FIGURE 10: Model use stage diagram.

Execution flow of model generation stage

Execution flow of model use stage

FIGURE 11: The overall execution flow of the program generation system.

Then, the selected keywords are checked against the semantic database to check if the same keywords exist. If not, the keywords are added to the semantic database. After the keywords are presented, the sample program path in the same row of the keywords in a table is obtained and judged. If there is a sample program path, we output the sample program according to the path to the next step. If there is no sample program, it is expected that a new sample program can be obtained through the web crawler to collect the proper sample program and store it into the semantic database. The procedure is then going to search for whether there is already a trained generative program model. If not, the sample program is put into GPT-2 as the first pass to produce the generative program model. If yes, the second pass is taken to feed the generative program model to GPT-2 so as to decode the model and generate 100 preliminary programs from GPT-2. The preliminary programs have been brought to the next test and verification steps. In the beginning, the 100 preliminary programs are sent individually to compare with the corresponding sample program and both are calculated with the Simhash signature value. Moreover, they are compared using Hamming Distance, and the similarity ratio is judged by the distance. After checking the code similarity between any one preliminary program and the sample program, this preliminary program will be viewed as the qualified program if its similarity exceeds the predetermined pass ratio (e.g., $\geq 90\%$). If so, it then will be sent to the next step for the verification of its execution result. However, if no preliminary program exceeds the predetermined pass ratio, it has to get back to the previous step to retrain a new generative program model and then try to regenerate 100 preliminary programs. Once all qualified programs have been produced, we need to check whether the number of qualified programs produced is enough. If there are only fewer qualified programs produced, it goes back to the previous step to re-generate a model. If the ratio of the number of qualified programs to 100 preliminary programs is big enough (e.g., $\geq 80\%$), the qualified programs are a majority and they are naturally compiled with Python. After the compilation is successful, LCS conformity between the execution result of the qualified program with the highest similarity pass ratio and the execution result of the corresponding sample program is computed to check whether their conformance meets the predetermined conformity (e.g., $\geq 95\%$). If so, such a qualified program serves as a pocket program. Finally, the pocket program, the generative program model, the keyword, and sample program are stored into the same row in a specific table in the semantic database.

In the model use stage, NLTK not only applies to segment words but also implements keyword drop-out and addin to optimize keyword retrieval. After the word segmentation, the useless keywords are not selected and the accuracy of the keyword hit is improved. Similarly, new keywords are added to the semantic database to improve the accuracy of the keyword hit. Next, we will check if a corresponding pocket program in the semantic database has been executed before. If the corresponding pocket program exists, the pocket program will be provided to the user for

execution. On the other hand, if the corresponding pocket program does not exist, it moves on to the step of training a new generative program model. After that, the subsequent step is to pick up several pocket programs corresponding to the respective keywords in the database and then merge them together to form a complete final program. "Evaluation and execution unit" is a further step to carry out a final program. It is expected that we can evaluate the execution performance of the final program to see how long it will take. In this way, the user may allow it to be taken or aborted.

3.3. Hardware Specification and Recipe of Software Tools.

In the model training phase, a high-level GPU cluster used for GPT-2 execution is used for rapid model training to reduce the processing time spent on traditional CPU. In Table 1, the operation would be carried out by the following tools: (1) NLTK word segmentation model, (2) unsupervised Generative Pre-Training second-generation transformer language model, (3) Simhash algorithm as an advanced version of Hash algorithm, and (4) LCS algorithm as an advanced version of DP algorithm.

The hardware equipment for running the program is based on two Nvidia-brand GPU P100 and two RTX2080Ti. Four GPU cluster workstations are connected through a high-speed local network to accelerate the calculation [30]. Cluster workstations have higher availability, reliability, and scalability than a workstation. Each workstation server transmits data through the high-speed network QPI, and uses a hardware interface PCIe x16 channel to connect both CPU and GPU. The GPU link uses NVLink [31] developed by Nvidia to allow four GPUs to share memory by using a point-to-point structure and serial transmission. Not only between CPU and GPU, but the connection between multiple Nvidia GPUs are also established. Under multiple GPUs, SLI, Surround, and PhysX options will show in the Nvidia system panel. Turning on the SLI, the users can share the graphics card memory for more data calculation. The detail hardware specifications are shown in Table 2. The overall architecture diagram is shown in Figure 12.

3.4. Evaluation of the Performance of Keyword Retrieval.

An experiment of the feasibility of generating the program model was conducted in two parts: keyword selection from a sentence and keyword retrieval optimization [32], and the estimation of the number of programs generated from code transform model GPT-2 and the calculation of the pass ratio of similarity checking between a generated program and a sample program. The first part is used to optimize the keyword-searching in a semantic database in order to improve the success of hit keywords from the database. There are two ways to optimize keyword-searching. The first one is to filter the redundant keywords in a sentence after word segmentation, and the second one is to add the new keywords into the semantic database so as to increase the success of hit keyword hits from the database. The first one filters keywords to select the keywords that were separated from the NLTK word segmentation. The nonrelated connectives or auxiliary words are deleted to improve the

TABLE 1: Open-source software tools.

Package	Version
Anaconda2	5.2.0
Python	3.7.5
Tensorflow	1.14
CUDA	10
XAMPP	3.2.4
NLTK	3.5
GPT-2	0.6
SimHash	2.0.0
LCS	Unknown

TABLE 2: Hardware specification.

Hardware	Specification	Amount
Server	HP Z8 G4 workstation	2
	HP Z4 G4 workstation	2
CPU	Xeon silver 4108	4
	I9-7900X	2
Ram	DDR4-2666 8G	16
	DDR4-2666 16G	12
Disk	MDFDDAK512TBN-1AR1ZABHA	2
	SAMSUNG-MZVPV256HEGL	2
	TOSHIBA-DT01ACA200	2
GPU	NVIDIA Quardro GP100	2
	NVIDIA GeForce RTX 2080 Ti 11G	2
Network	Intel ethernet connection X722 for 1GbE	4

accuracy of hit keywords form the semantic database. The second one is to add keywords to check if the necessary keywords exist in the semantic database, to add keywords to the semantic database, and to notify the users to find and add the corresponding sample programs in the database as well. To improve the hit ratio of keyword retrieval if the user submits the same keyword again, it is necessary to check whether the hit ratio of keyword retrieval in the semantic database is increased. Because F1-Score [33] is often used as a measure of accuracy in pattern recognition, sample survey, and information retrieval, this study used F1-Score as an Evaluation Metric to measure the accuracy. F1-Score is a harmonic average calculated using Precision and Recall. To find the F1-Score, the user must firstly define the positive class, negative class, and consider whether it is retrievable. The definition of the Confusion Matrix [34] is shown in Table 3.

In the confusion matrix about the keyword-searching issue, there are four terms. True positive (TP) is positive and judged to be positive. False positive (FP) is negative but wrongly judged to be positive. False negative (FN) is positive but wrongly judged to be negative. True negative (TN) is negative and judged to be negative. The accuracy is defined in equation (7), which is to calculate the ratio of TP to TP+FP. The recall is defined in equation (8), which is to calculate the ratio of TP to TP+FN. The recall is defined in equation (8), which meant is to calculate the proportion of all results (TP + FN) retrieved by the positive results (TP). After finding the accuracy and recall, F1-Score is the harmonic average of accuracy and recall. It is defined in equation (9). After adjusting equation (9), F1-Score is

concise in equation (10). Corresponding to the F1-Score evaluated after word segmentation, the default keywords are selected to check the performance of keyword retrieval from the semantic database. TP refers to those keywords that are retrieved and positively related. The nonrelevant retrieved keywords are FP. The nonretrieved but related keywords are FN. For evaluation metrics, the process of calculating accuracy, recall, and F1-Score is conducted.

$$P = \frac{TP}{TP + FP}, \quad (7)$$

$$R = \frac{TP}{TP + FN}, \quad (8)$$

$$\frac{2}{F1} = \frac{1}{P} + \frac{1}{R}, \quad (9)$$

$$F1 = \frac{2 \cdot P \cdot R}{P + R} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}. \quad (10)$$

3.5. Performance Evaluation of Program Execution. This section evaluates the performance improvement of newly generated programs produced by GPT-2. In other words, we are going to compare the execution performance between sample program and each generated program individually. The performance evaluation includes (1) comparing the number of code lines of the sample program and the average number of code lines of generated programs and (2) comparing the execution time between them as well. Two indicators are used to explain how much performance would be improved for any program execution where the first one is to measure the ratio of average code lines of the generated program to the code lines of the sample program, and the second one is to evaluate the ratio of average execution time of the generated program to the execution time of the sample program. Reducing the percentage of the program lines r_l is shown in equation (11). The l_o and l_g represent the number of code lines of the sample program and the average number of code lines of the generated programs, respectively. The reduction in program execution time percentage r_t is shown in equation (12). The t_o and t_g stand for the execution time of the sample program and the average execution time of the generated programs, respectively.

$$r_l = \left(1 - \frac{l_g}{l_o}\right) \times 100\%, \quad (11)$$

$$r_t = \left(1 - \frac{t_g}{t_o}\right) \times 100\%, \quad (12)$$

3.6. Predetermining the Number of Generated Programs Statistically. This section is to explore how many of the generated programs produced by GPT-2 can guarantee at least a few predetermined programs existing and having the pass ratio of code similarity checking with sample program over 90%. Therefore, we first take a count of code the generated programs that have been generated from a single

FIGURE 12: GPU cluster workstation.

TABLE 3: Keyword search confusion matrix.

	Relevant, positive class	Nonrelevant, negative class
Retrieved	True positives (TP)	False positives (FP)
Not retrieved	False negatives (FN)	True negatives (TN)

sample program, having the pass ratios of code similarity checking over 90%, and then make it possible to calculate the percentage q_i as shown in equation (13). Among them, N_{si} is the total number of generated programs produced from a single sample program in which x_{si} is the number of generated programs whose pass ratio of similarity checking is more than 90%. After every percentage q_i mentioned above has been obtained, equation (14) represents the average percentage of q_i where t stands for the total number of sample programs in a single example sentence. Then, we make a judgment to determine how many misjudgments are there in x_{si} , and then calculate the percentage of misjudgments q_{mi} , as shown in equation (15), where x_{msi} is the number of misjudgments within the generated programs having a pass ratio of code similarity checking over 90%. Then, the average percentage of misjudgments q_m can be obtained as shown in equation (16).

$$q_i = \frac{x_{si}}{N_{si}}, \quad i = 1, 2, \dots, t, \quad (13)$$

$$q = \frac{\sum_{i=1}^t q_i}{t}, \quad (14)$$

$$q_{mi} = \frac{x_{msi}}{x_{si}}, \quad i = 1, 2, \dots, t, \quad (15)$$

$$q_m = \frac{\sum_{i=1}^t q_{mi}}{t}. \quad (16)$$

Next, we count how many of the generated programs y_{si} , which are generated from a single sample program, have pass

ratios of below 90%, and then make it possible to calculate the percentage u_i , as shown in equation (17). After all of the existing percentages mentioned above have been calculated, equation (18) can give an average percentage u of the generated programs having the pass ratio of code similarity checking less than 90%. Then, we make a judgment to determine how many misjudgments are there in y_{si} and then calculate the percentage of misjudgments u_{mi} , as shown in equation (19), where y_{msi} is the number of misjudgments within the generated programs having the pass ratio of code similarity checking below 90%. Then, the average percentage of misjudgments u_m can be obtained as shown in equation (20).

$$u_i = \frac{y_{si}}{N_{si}}, \quad i = 1, 2, \dots, t, \quad (17)$$

$$u = \frac{\sum_{i=1}^t u_i}{t}, \quad (18)$$

$$u_{mi} = \frac{y_{msi}}{y_{si}}, \quad i = 1, 2, \dots, t, \quad (19)$$

$$u_m = \frac{\sum_{i=1}^t u_{mi}}{t}. \quad (20)$$

Then, we add up the number of programs generated from all the sample programs to get as shown in equation (21). After obtaining N_g , q , q_m , u , and u_m through the above calculations, equation (22) can calculate an average probability P_{gq} [35] of the generated programs having the pass ratio over 90%, and they are generated from the sample programs. We assume that there are j programs with a pass ratio of code more than 90%, so $P(K_j|P_{gt})$ represents the probability of the pass ratio of similarity checking more than 90% for these j programs is real, as shown in equation (23) [36], where $P(K_j \cap P_{gt})$ means the probability of there are at most j programs having the pass ratio of code similarity

checking over 90% in the generated programs, and K_j indicates there are j programs having the pass ratio of code similarity checking over 90% within the generated programs produced by the code transform model at a time. According to the abovementioned statistics, we know that the probability of j programs having the pass ratio of code similarity checking over 90% is $P(K_j|P_{gt})$. We can deduce how many programs must be generated to guarantee that there are j programs having the pass ratio of code similarity checking more than 90%, as shown in equation (24), where N is the total number of programs to be generated.

$$N_g = \sum_{i=1}^t N_{si}, \quad (21)$$

$$P_{gq} = \frac{N_g \cdot q \cdot (1 - q_m) + N_g \cdot u \cdot u_m}{N_g}, \quad (22)$$

$$P(K_j|P_{gt}) = \frac{P(K_j \cap P_{gt})}{P_{gt}}, \quad (23)$$

$$N \cdot P(K_j|P_{gt}) \geq K_j. \quad (24)$$

Let's take 4 sample programs as an example. Each sample program generates 500 programs individually and then counts how many programs have the pass ratio of code similarity checking more than 90% among the 500 programs, and then calculates the pass ratio of each generated program to have more than 90%, the average percentage can be calculated to be 3%. Then, we judge the programs whose pass ratio of code similarity checking is over 90%. After judging the number of misjudgments, we calculate the individual percentages, and finally the average percentage of misjudgments is 1%.

Moreover, we judge how many of these 500 programs have the pass ratio of code similarity checking not more than 90%, and then calculate their percentages. After calculation, we can find that the pass ratio of code similarity checking for 4 sample programs does not have an average of more than 90%. The percentage is 97%, and then we judge the programs whose pass ratio of code similarity checking is not more than 90%, judge the number of misjudgments, and then calculate the percentage. Finally, we can get the average misjudgment ratio of 2%.

Based on the above statistics, we can find that the pass ratio of code similarity checking in the generated program is really over 90% and the average probability is 4.91%. Then we want to know how many programs need to be generated to guarantee 5 programs having the pass ratio of code similarity checking over 90%. The above related values can be substituted into equation (24) to obtain the answer. As a result, at least 100 generated programs must be produced to guarantee 5 of them having the pass ratio of code similarity checking more than 90%.

4. Experimental Results and Discussion

4.1. Experimental Design. Four experiments are carried out in the following. The first experiment is to make word segmentation to select keywords and to optimize keyword retrieval. The second experiment is to search for sample programs and generate a number of preliminary programs based on the predetermined number of generated programs statistically. The third experiment is to analyze the pass ratio of code similarity checking and classify few preliminary programs as qualified programs. Finally, applying LCS conformity checking between each qualified program and sample program intents to find out the one with the highest LCS conformity, and this qualified program denotes a pocket program in the last experiment.

The experimental setting has exemplified four sample sentences into practice for all experiments. After word segmentation, the keyword retrieval optimization was implemented in two aspects. The first one is to filter the redundant keywords, and the second is to add the required keywords. Evaluation metrics, such as accuracy, recall, and F1-Score, are used to measure the performance of keyword retrieval. The sample programs associated with keywords are obtained from GitHub [37] and both of them are stored in a semantic database. In the XAMPP server, the correlation table of a semantic database consists of the several fields: keyword, sample program names, sample program paths, generated model paths, and pocket program path, as shown in Figure 13. The objective of this paper is to improve the performance of sample programs by transforming them into newly generated programs produced by GPT-2 based on two indicators: (1) to reduce the number of program code lines and (2) to decrease the program execution time.

The sample program of example 1 is related to a web crawler [38], and the corresponding keywords were "weather, traffic". The purpose of the sample program 1 was to crawl the corresponding data on the Internet to get weather forecast on that day from the Weather Center and automatically allocate the traffic congestion spots on Google Map. Next, in the sample program of example 2, the corresponding keywords are "stock, currency" that are related to exchange rate [39]. The main purpose of the sample program was to display the current currency exchange rate or the current index of the stock to the users. Third, in the sample program of example 3, the corresponding keyword is a "pets" and it's related to the web camera [40]. The objective of the web camera programming was to have the camera installed on the desktop computer at home for the video of the pet. Finally, the corresponding keywords of the sample program of example 4 are "invest, don't, insure". The system would give the user the market data analysis as per the request from investment exploration [41].

In the experimental settings, four example sentences are used to optimize keyword retrieval by filtering redundancy and adding new keywords. The example sentences are shown in Table 4.

				id	keyword	program_name	location	checkpoint
☐				1	the weather	Web-Crawler	D:\	D:\checkpoint
☐				2	exchange rate	Exchange-Rate	D:\	D:\checkpoint
☐				3	photography	Web Camera	D:\	D:\checkpoint
☐				4	investment	Market-Analysis	D:\	D:\checkpoint

FIGURE 13: Correlation table of four sample programs.

TABLE 4: Example sentences.

Example	Sentence context
Example 1	The weather is very good today, I want to know the traffic flow.
Example 2	Recently, the stock has continued to fall and is not stable, looking for currency trading or gold trading.
Example 3	How are your pets at home? I don't know what's going on in the shop.
Example 4	If you want to invest in financial management, don't insure, but deposit.

TABLE 5: NLTK word segmentation.

Example	Word segmentation
Example 1	["The", "weather", "is", "very", "good", "today", ",", "I", "want", "to", "know", "the", "traffic", "flow", "."]
Example 2	["Recently", ",", "the", "stock", "has", "continued", "to", "fall", "and", "is", "not", "stable", ",", "looking", "for", "currency", "trading", "or", "gold", "trading", "."]
Example 3	["How", "are", "your", "pets", "at", "home", "?", "I", "do", "n't", "know", "what", "s", "going", "on", "in", "the", "shop", "."]
Example 4	["If", "you", "want", "to", "invest", "in", "financial", "management", ",", "do", "n't", "insure", ",", "but", "deposit", "."]

```

In [1]: runfile('D:/eng1.py', wdir='D:')
['The', 'weather', 'is', 'very', 'good', 'today', ',', 'I', 'want', 'to', 'know', 'the', 'traffic', 'flow', '.']

['Recently', ',', 'the', 'stock', 'has', 'continued', 'to', 'fall', 'and', 'is', 'not', 'stable', ',', 'looking', 'for', 'currency', 'trading', 'or', 'gold', 'trading', '.']

['How', 'are', 'your', 'pets', 'at', 'home', '?', 'I', 'do', 'n't', 'know', 'what', 's', 'going', 'on', 'in', 'the', 'shop', '.']

['If', 'you', 'want', 'to', 'invest', 'in', 'financial', 'management', ',', 'do', 'n't', 'insure', ',', 'but', 'deposit', '.']

```

FIGURE 14: Screenshot of NLTK word segmentation.

4.2. Experimental Results

4.2.1. The Experiment #1. The above four sentences were used as the word segmentation model NLTK to select the keywords. The results of keyword selection are shown in Table 5. The screenshot is shown in Figure 14.

As mentioned above, NLTK operation that carried out word segmentation was not optimized in key words retrieval. All segmented words except for punctuation marks were selected as keywords. The selected keywords are shown in Table 6, and the screenshot is shown in Figure 15.

The initial selected keywords were consistent with the existing keywords in the semantic database as shown in Table 7. Hit keywords were “weather, traffic” in example 1, “stock, currency” in example 2, “pets” in example 3, and “invest, don’t, insure” in example 4.

After selecting keywords out of the sample sentences, the accuracy of keyword-searching was on the basis of the number of hit keywords in the semantic database. Those related keywords were retrieved and denoted as true positive

(TP), and those retrieved but unrelated keywords were denoted as false positive (FP). The confusion matrix for keyword retrieval from example 1 to example 4 is shown in Tables 8–11.

After initial keyword-searching using NLTK, the first optimization method is to filter out the unrelated auxiliary words or conjunctions in the sentences. The detailed results of hit keywords in the semantic database are shown in Table 12. The related screenshot is shown in Figure 16. After the screening process, the confusion matrices after optimization for keyword retrieval from Example 1 to Example 4 are shown in Tables 13–16.

The second keyword search optimization method was to add related keywords and sample program of the semantic database. Furthermore, the number of keywords that reflected the semantic database was newly optimized. Added keywords are “today, very, good, know, flow” in Example 1, “Recently, continue, fall, gold, trading” in Example 2, “home, How, shop, know, what” in Example 3, and “want, but, deposit” in Example 4. The details of keywords in the

TABLE 6: Keywords selection out of the example sentences.

Example	Keyword
Example 1	["The", "weather", "is", "very", "good", "today", "I", "want", "to", "know", "the", "traffic", "flow"]
Example 2	["Recently", "the", "stock", "has", "continued", "to", "fall", "and", "is", "not", "stable", "looking", "for", "currency", "trading", "or", "gold", "trading"]
Example 3	["How", "are", "your", "pets", "at", "home", "I", "don", "t", "know", "what", "s", "going", "on", "in", "the", "shop"]
Example 4	["If", "you", "want", "to", "invest", "in", "financial", "management", "do", "n't", "insure", "but", "deposit"]

```
In [1]: runfile('D:/eng1.py', wdir='D:')
['The', 'weather', 'is', 'very', 'good', 'today', 'I', 'want', 'to', 'know', 'the', 'traffic', 'flow']

['Recently', 'the', 'stock', 'has', 'continued', 'to', 'fall', 'and', 'is', 'not', 'stable', 'looking', 'for', 'currency', 'trading', 'or', 'gold', 'trading']

['How', 'are', 'your', 'pets', 'at', 'home', 'I', 'don', 't', 'know', 'what', 's', 'going', 'on', 'in', 'the', 'shop']

['If', 'you', 'want', 'to', 'invest', 'in', 'financial', 'management', 'do', 'n't', 'insure', 'but', 'deposit']
```

FIGURE 15: Screenshot of selected keywords.

TABLE 7: Hit keyword in semantic database.

Example	Hit keyword in semantic database
Example 1	Weather, traffic
Example 2	Stock, currency
Example 3	Pets
Example 4	Invest, don't, insure

TABLE 12: The result of filter keywords in the semantic database in Experiment 1.

Example	Hit keyword in semantic database
Example 1	Weather, traffic
Example 2	Stock, currency
Example 3	Pets
Example 4	Invest, don't, insure

TABLE 8: Example 1: Confusion matrix for keyword retrieval in Experiment 1.

	Related	Unrelated
Retrieved	2	6
Not retrieved	0	0

TABLE 9: Example 2: Confusion matrix for keyword retrieval in Experiment 1.

	Related	Unrelated
Retrieved	2	7
Not retrieved	0	0

TABLE 10: Example 3: Confusion matrix for keyword retrieval in Experiment 1.

	Related	Unrelated
Retrieved	1	7
Not retrieved	0	0

TABLE 11: Example 4: Confusion matrix for keyword retrieval in Experiment 1.

	Related	Unrelated
Retrieved	3	4
Not retrieved	0	0

```
In [1]: runfile('D:/eng1.py', wdir='D:')
['weather', 'traffic']
['stock', 'currency']
['pets']
['invest', 'do', 'n't', 'insure']
```

FIGURE 16: The screenshot of keywords retrieval with filter optimization.

TABLE 13: Example 1: keyword retrieval confusion matrix with filter optimization in Experiment 1.

	Related	Unrelated
Retrieved	2	1
Not retrieved	0	0

TABLE 14: Example 2: keyword retrieval confusion matrix with filter optimization in Experiment 1.

	Related	Unrelated
Retrieved	2	1
Not retrieved	0	0

TABLE 15: Example 3: keyword retrieval confusion matrix with filter optimization in Experiment 1.

	Related	Unrelated
Retrieved	1	0
Not retrieved	0	0

TABLE 16: Example 4: keyword retrieval confusion matrix with filter optimization in Experiment 1.

	Related	Unrelated
Retrieved	3	4
Not retrieved	0	0

TABLE 17: The result of added keywords in the semantic database in Experiment 1.

Example	Hit keyword in semantic database
Example 1	today, weather, very, good, know, traffic, flow
Example 2	recently, continue, stock, fall, currency, gold, trading
Example 3	home, pets, how, shop, know
Example 4	want, invest, don't, insure, but, deposit

TABLE 18: Experiment 1: Example 1: added optimized keyword search confusion matrix.

	Related	Unrelated
Retrieved	7	1
Not retrieved	0	0

TABLE 19: Experiment 1: Example 2: added optimized keyword search confusion matrix.

	Related	Unrelated
Retrieved	7	2
Not retrieved	0	0

TABLE 20: Experiment 1: Example 3: Added optimized keyword search confusion matrix.

	Related	Unrelated
Retrieved	6	2
Not retrieved	0	0

TABLE 21: Experiment 1: Example 4: added optimized keyword search confusion matrix.

	Related	Unrelated
Retrieved	6	1
Not retrieved	0	0

TABLE 22: Evaluation indexes of Example 1 in Experiment 1.

Evaluation index (%)	Initial selection (%)	Filter optimization (%)	Add optimization (%)	Average (%)
Accuracy	25	67	75	94
Recall	100	100	100	100
F1-score	40	80	86	97

TABLE 23: Evaluation indexes of Example 2 in Experiment 1.

Evaluation index (%)	Initial selection (%)	Filter optimization (%)	Add optimization (%)	Average (%)
Accuracy	22	67	78	89
Recall	100	100	100	100
F1-score	36	80	88	94

TABLE 24: Evaluation indexes of example 3 in Experiment 1.

Evaluation index (%)	Initial selection (%)	Filter optimization (%)	Add optimization (%)	Average (%)
Accuracy	13	100	75	88
Recall	100	100	100	100
F1-score	22	100	86	93

TABLE 25: Evaluation indexes of Example 4 in Experiment 1.

Evaluation index	Initial selection (%)	Filter optimization (%)	Add optimization (%)	Average (%)
Accuracy	30	75	86	89
Recall	100	100	100	100
F1-score	46	86	92	94

semantic database are shown in Table 17. The confusion matrices for the keyword search from Example 1 to Example 4 are shown in Tables 18–21.

The Evaluation Metrics such as the accuracy, recall, and F1-Score of the initial keywords retrieval, filter keywords retrieval, and newly added keywords retrieval were evaluated. The results are shown in Tables 22–25. Since all keywords in the segmented words in sample sentences were selected by default, the recall was 100%.

4.2.2. The Experiment #2. The second experiment was based on four sample programs obtained from GitHub. Corresponding to the keywords found in the experiment #1, the corresponding keywords were extracted from the natural language sentences and will be applied to the sample programs. The correspondence of the sample programs and keywords are listed in Table 26.

In order to transform sample programs to the high-performance generated programs, a code transform model GPT-2 generated 100 preliminary programs, and its time consuming was also recorded at the same time. In this experiment, a total of five rounds was performed, and the estimated average time to generate a program in real-time was summarized in Table 27.

4.2.3. The Experiment #3. In experiment #3, the comparison of Simhash similarity checking between the above four sample programs and the programs generated by GPT-2 were performed on a cluster GPU workstation. The aim of this experiment was to find out how many completed programs would have similarity percentage greater than or equal to the default pass ratio set by the user earlier. In this experiment, the diagram of qualified ratio distribution set the X-axis as the similarity percentage, ranging from 0% to 100% with 20% as the separation interval. The Y-axis was set as the number of programs in the percentage ratio interval. For each corresponding sample program, code transform model GPT-2 will generate 100 programs denoted as

TABLE 26: The list of example programs in Experiment 2.

Example	Sample program	Keyword
Example 1	Web crawler	Weather, traffic
Example 2	Exchange rate	Stock, currency
Example 3	Web camera	Pets
Example 4	Market-analysis	Invest, don't, insure

TABLE 27: Estimated time to generate one hundred programs (unit: second).

Sample program	First round	Second round	Third round	Fourth round	Fifth round	Average
Sample program 1	170	185	163	183	147	170
Sample program 2	142	188	176	150	178	167
Sample program 3	171	144	168	196	183	172
Sample program 4	162	186	175	144	152	164

```

File Edit Format View Help
from bs4 import BeautifulSoup
import requests

winning_Numbers_Sort_lotto = ['Lotto649Control_history_d1Query_No1_', 'Lotto649Control_hist

def search_winning_numbers(css_class):
    global winning_Numbers_Sort_lotto
    if(css_class != None):
        for i in range(len(winning_Numbers_Sort_lotto)):
            if winning_Numbers_Sort_lotto[i] in css_class:
                return css_class
    def parse_tw_lotto_html(data_Info,number_count):
        data_Info_List = []
        data_Info_Dict = {}
        tmp_index = 0
        for index in range(len(data_Info)):
            if (index == 0):
                data_Info_List.append(data_Info[index].text)
            else:
                if(index % number_count != 0):
                    data_Info_List.append(data_Info[index].text)
                else:
                    data_Info_Dict[str(tmp_index)] = list(data_Info_List)
                    data_Info_List = []
                    tmp_index += 1
    
```

FIGURE 17: Sampled preliminary program associated with sample program 1 in Experiment 3.

```

File Edit Format View Help
import pandas
dfs = pandas.read_html( "http://rate.bot.com.tw/xrt?Lang=zh-TW")
currency = dfs[0]
currency = currency.ix[:,0:5]
currency.columns = ['u' type', 'u' buy', 'u' sell', 'u' buy', 'u' sell' ]
currency['u' type'] = currency['u' type'].str.extract( '\\((\\w+)\\)' )
currency.to_excel( 'currency.xlsx' )
import pandas
dfs = pandas.read_html( "http://rate.bot.com.tw/xrt?Lang=zh-TW")
currency = dfs[0]
currency = currency.ix[:,0:5]
currency.columns = ['u' type', 'u' buy', 'u' sell', 'u' buy', 'u' sell' ]
currency['u' type'] = currency['u' type'].str.extract( '\\((\\w+)\\)' )
currency.to_excel( 'currency.xlsx' )
import pandas
dfs = pandas.read_html( "http://rate.bot.com.tw/xrt?Lang=zh-TW")
currency = dfs[0]
currency = currency.ix[:,0:5]
currency.columns = ['u' type', 'u' buy', 'u' sell', 'u' buy', 'u' sell' ]
currency['u' type'] = currency['u' type'].str.extract( '\\((\\w+)\\)' )
currency.to_excel( 'currency.xlsx' )
import pandas
dfs = pandas.read_html( "http://rate.bot.com.tw/xrt?Lang=zh-TW")
currency = dfs[0]

```

FIGURE 18: Sampled preliminary program associated with sample program 2 in Experiment 3.

```

Videostream.py - Notepad
File Edit Format View Help
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('//140.127.218.63//64//video//output_64.avi',fourcc, 20.0, (640,480))

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
        #frame = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
        frame = cv2.flip(frame,0)
        frame1 = cv2.flip(frame,0)
        # write the flipped frame
        out.write(frame1)

    cv2.imshow('frame1',frame1)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    else:
        break

```

FIGURE 19: Sampled preliminary program associated with sample program 3 in Experiment 3.

```

Market-Analysis - Notepad
File Edit Format View Help
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def customer_segmentation(df2,frequency_cut,recency_cut,product ):

    df2['customer'] = np.where( (df2['frequency'] >=frequency_cut[4]) & (df2['recency'] <=r
        np.where( (df2['frequency'] >=frequency_cut[4]) & ( df2['recency'
            np.where( (df2['frequency'] < frequency_cut[4]) & ( df2[
                'new customers' )))

    recom_list = []
    for i in range(len(df2[product])):
        aa = df2[product].iloc[i,:].rank().sort_values(ascending = False)
        recom_list.append(' '.join(aa.index))

    df2['recommend_product'] =recom_list

```

FIGURE 20: Sampled preliminary program associated with sample program 4 in Experiment 3.

preliminary programs. Samples of the preliminary programs are shown in Figures 17–20. The pass ratios of the preliminary programs are shown in Figures 21–24. We define the pass ratio as how many programs out of 100 preliminary programs would have the similarity falling within the range of 80%–100%. In other words, the pass ratios of this experiment associated with sample programs 1, 2, 3, and 4 were 40%, 30%, 38%, and 37%, respectively, based on 100 generated preliminary programs, and those preliminary programs are referred to as qualified programs instead.

4.2.4. The Experiment #4. According to 4 examples demonstrated in the experimental setting, the fourth experiment first attempts to verify whether the execution result of the qualified program meets a certain proportion of conformity with the sample program. After qualified programs have been compiled successfully, the qualified programs are executed individually, and then the execution result of each qualified program is compared with the execution result of a corresponding sample program using LCS conformity. Their execution results are shown in Figures 25–28. The program execution result is converted into ASCII code through LCS algorithm to compare

FIGURE 21: The pass ratio of the preliminary programs associated with sample program 1 in Experiment 3.

FIGURE 22: The pass ratio of the preliminary programs associated with sample program 2 in Experiment 3.

FIGURE 23: The pass ratio of the preliminary programs associated with sample program 3 in Experiment 3.

the conformance. As a result, the qualified program with the highest pass ratio of LCS conformity is called the best qualified program and is also designated as a pocket program. The experimental result is listed as shown in Table 28.

Next, we have compared the performance of the above four best qualified programs produced by GPT-2 with their corresponding sample programs. The evaluation includes (1) to compare the number of code lines between the sample

FIGURE 24: The pass ratio of the preliminary programs associated with sample program 4 in Experiment 3.

```
In [1]: runfile('C:/Users/user/Desktop/Web-Crawler.py', wdir='C:/Users/user/Desktop')
{'0': ['16', '17', '18', '19', '33', '45', '25'], '1': ['08']}
```

(a)

```
In [1]: runfile('C:/Users/user/Desktop/untitled9.py', wdir='C:/Users/user/Desktop')
{'0': ['16', '17', '18', '19', '30', '41', '25'], '1': ['08']}
```

(b)

FIGURE 25: Program execution result in example 1. (a) Execution result of sample program 1. (b) Execution result of the best qualified program.

```
In [1]: runfile('C:/Users/user/Desktop/Market-Analysis.py', wdir='C:/Users/user/Desktop')
Recommend you to invest in Sinopec Yuanta 02 purchase 01
opening 8.4,increase 0.6%
```

(a)

```
In [1]: runfile('C:/Users/user/Desktop/untitled16.py', wdir='C:/Users/user/Desktop')
Recommend you to invest in Sinopec Yuanta 02 purchase 01
open 8.7 ,increase 0.66%
```

(b)

FIGURE 26: Program execution result in example 2, (a) Execution result of sample program 2. (b) Execution result of the best qualified program.

```
In [1]: runfile('C:/Users/user/Desktop/Videostream.py', wdir='C:/Users/user/Desktop')
Turn on the mini camera
Watch the situation in the store
```

(a)

```
In [1]: runfile('C:/Users/user/Desktop/untitled19.py', wdir='C:/Users/user/Desktop')
Turn on the mini camera
look the situation in the store
```

(b)

FIGURE 27: Program execution result in example 3, (a) Execution result of sample program 3. (b) Execution result of the best qualified program.

program and the corresponding best qualified program and (2) to compare the execution time of the sample program and the corresponding best qualified program. To understand how much improvement was made in the speed of program execution, the average number of code lines of 100

preliminary programs and the average execution time for the best qualified programs were carefully examined. The estimation results are listed in Tables 29 and 30, respectively.

Regarding the credibility and validity of the findings in this study, the comparison of performance evaluation

```
In [1]: runfile('C:/Users/user/Desktop/Exchange-Rate.py', wdir='C:/Users/user/Desktop')
type:USD
Buy 29.1925 dollars
Sell 29.6030 dollars
```

(a)

```
In [1]: runfile('C:/Users/user/Desktop/untitled22.py', wdir='C:/Users/user/Desktop')
type : USD
Buy 29.8964 dollars
Sell 29.1079 dollar
```

(b)

FIGURE 28: Program execution result in example 4, (a) Execution result of sample program 4. (b) Execution result of the best qualified program.

TABLE 28: Comparison of program execution results based on LCS conformity (unit: %).

Number of ASCII code	Example 1	Example 2	Example 3	Example 4
Number of ASCII code counted from the sample program execution output	62	89	57	51
Number of ASCII code counted from the best qualified program execution output	62	87	57	51
Number of ASCII code counted from LCS output	61	86	56	49
LCS conformity	98.38%	97.72%	98.24%	96.07%
Average LCS conformity	97.60%			

TABLE 29: Number of code lines comparison.

Number of code lines or percentage	Example 1	Example 2	Example 3	Example 4
Number of source code lines of the sample program	291	175	138	153
Average of number of code lines of 100 preliminary programs	174	117	99	108
Reduced the percentage of code lines	40.34%	32.96%	28.02%	29.54%
Average percentage reduction	32.71%			

TABLE 30: Program execution time comparison (unit: second).

Time or percentage	Example 1	Example 2	Example 3	Example 4
Sample program execution time	8.35	11.83	9.74	11.86
Best qualified program execution time	6.97	8.05	7.07	9.32
Reduced the percentage of program execution time	16.59%	31.93%	27.40%	21.43%
Average percentage reduction	24.34%			

among source-code to source-code transform models is listed in Table 31. The performance evaluation includes (1) average time of generating a single instruction (s), (2) reduced the percentage of code lines (%), and (3) conformity of program execution results (%). As a result, the proposed approach with GPT-2 outperforms Java-Codetool and CodeGeneration. Java-Codetool and CodeGeneration introduced in the paper [16] are of Java code transform models. However, this article didn't present the execution results of generated programs. Thus, there is no information about the conformity of program execution results for Java-Codetool and CodeGeneration models in Table 31.

5. Discussion

In the first experiment, after NLTK word segmentation, keyword-searching optimization has shown that if there were fewer hit keywords, filtering operation would require screening more irrelevant keywords to improve the

F1-Score. In contrast, the alternative was to add keywords to the semantic database and improving the accuracy and F1-Score of keywords retrieval significantly. In the second experiment, the number of generated program was produced through GPT-2 based on predetermined the number of generated programs statistically and an average of 100 programs were generated for every corresponding sample program and on average, each program was generated for about 1 second. The number of code lines for generating 100 programs was reduced by an average of 32.71%, so that the code review could save about 30% of time due to less number of code lines. After the code similarity checking for the generated preliminary programs was completed in the third experiment, the fewer programs having the highest pass ratio were selected as the qualified programs in this part. After the qualified programs were compiled successfully, the last experiment is the LCS conformity checking between the qualified programs and the sample program where the ratio of conformity in fact exceeds 97.60% on average and

TABLE 31: Performance evaluation of code generation model.

Time or percentage	Java-codetool	CodeGeneration	Proposed approach with GPT-2
Average time of generating a single instruction (s)	0.17	0.15	0.01
Reduced the percentage of code lines (%)	0%	0%	24.34%
Conformity of program execution results (%)	—	—	97.60%

The symbol “—” stands for the information that is not available so far.

the one with the highest ratio of conformity is chosen as a pocket program. Regarding the performance of the qualified programs, the average execution time of the generated program was reduced by 24.34%.

The experiments have shown that the system could not only quickly generate programs but also greatly improve the program execution efficiency. Based on transfer learning and few-shot learning, GPT-2 has implemented the best level of code transform task to produce newly generated programs for significantly improving its execution efficiency. Furthermore, with the powerful cloud platform, Hadoop or Spark, people in the future can realize collective learning to integrate all of the small data sets provided from different units, evolve into sophisticated machine learning applications, create a large enough semantic database, and achieve a great computing power.

6. Conclusion

This study proposes a novel transform method to replace the existing programs inside the voice assistant machine with high-performance generated programs through GPT-2 reasonably. In particular, this paper introduces theoretical estimation in statistics to infer at least a number of generated programs needed so as to guarantee the best one can be founded within them. As a result, in terms of performance evaluation, the average number of newly generated code lines decreased by 32.71%, and the average execution time of the program decreased by 24.34%. This proved that the system could not only quickly generate programs but also greatly improve the performance of program execution. In the future, we are looking for developing a new method to speed up data retrieval in the semantic database and finding the revision of Simhash and Longest Common Subsequence (LCS) to achieve better accuracy of measuring the code similarity and the conformity of program execution results.

Data Availability

The Sample Program.rar data used to support the findings of this study have been deposited in the <https://drive.google.com/file/d/1KYDeoO9s8kA94U9-CW1AsdB0ZZNr4Hcy/view?usp=sharing> repository. The sample sentence data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

B.R.C. and P.-W.S. conceived and designed the experiments; H.-F.T. collected the experimental dataset and proofread the paper; and B.R.C. wrote the paper.

Acknowledgments

This work was fully supported by the Ministry of Science and Technology, Taiwan, Republic of China, under grant numbers MOST 105-2221-E-390-013-MY3 and MOST 109-2622-E-390-002-CC3.

References

- [1] F. Nasirian, M. Ahmadian, and O. K. D. Lee, “Ai-based voice assistant systems: evaluating from the interaction and trust perspectives,” in *Proceedings of the Twenty-third Americas Conference on Information Systems*, pp. 1–10, Boston, MA, USA, 2017.
- [2] H. Shah, K. Warwick, J. Vallverdú, and D. Wu, “Can machines talk? Comparison of Eliza with modern dialogue systems,” *Computers in Human Behavior*, vol. 58, pp. 278–295, 2016.
- [3] S. Arora, V. A. Athavale, H. Maggu, and A. Agarwal, “Artificial intelligence and virtual assistant—working model,” *Mobile Radio Communications*, vol. 140, pp. 163–171, 2020.
- [4] K. Ethayarajh, “How contextual are contextualized word representations? comparing the geometry of BERT, ELMo, and GPT-2 embeddings,” 2019, <http://arxiv.org/abs/1909.00512>.
- [5] NLTK, <https://github.com/nltk/nltk>.
- [6] GPT-2: 1.5B Release, <https://openai.com/blog/gpt-2-1-5b-release/>.
- [7] D. M. J. Lazer, M. A. Baum, Y. Benkler et al., “The science of fake news,” *Science*, vol. 359, no. 1360, pp. 1094–1096, 2018.
- [8] W. Wagner, S. Bird, E. Klein, and E. Loper, “Natural language processing with Python, analyzing text with the natural language toolkit,” *Language Resources and Evaluation*, vol. 44, no. 4, pp. 421–424, 2010.
- [9] J. Park, M.-S. Chen, and P. S. Yu, “Using a hash-based method with transaction trimming for mining association rules,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, pp. 813–825, 1997.
- [10] Longest common subsequence problem, https://en.wikipedia.org/wiki/Longest_common_subsequence_problem.
- [11] H. Chung, J. Park, and S. Lee, “Digital forensic approaches for Amazon Alexa ecosystem,” *Digital Investigation*, vol. 22, pp. S15–S25, 2017.
- [12] E. Reiter and R. Dale, “Building applied natural language generation systems,” *Natural Language Engineering*, vol. 3, no. 1, pp. 57–87, 1997.
- [13] M. Geittle and R. Olsson, “Using automatic programming of design improved variants of differential evolution,” in *Proceedings of the 2017 21st Asia pacific symposium on intelligent*

- and evolutionary systems(IES), Hanoi, Vietnam, December 2017.
- [14] Á. Beszédes, R. Ferenc, T. Gyimóthy, A. Dolenc, and K. Karsisto, "Survey of code-size reduction methods," *ACM Computing Surveys (CSUR)*, vol. 35, no. 3, pp. 223–267, 2003.
 - [15] L. Li, J. Yang, Z. Liu, and L. Bao, "The research and application of web page code automatic generation technology," in *Proceedings of the IEEE 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce*, pp. 5246–5249, Zhengzhou, China, August 2011.
 - [16] Z. Li, Y. Jiang, X. J. Zhang, and H. Y. Xu, "The metric for automatic code generation," *Procedia Computer Science*, vol. 166, pp. 279–286, 2020.
 - [17] R. Collobert, J. Weston, B. Leon, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "natural language processing (almost) from scratch," *Journal of Machine Learning Research*, 2011.
 - [18] K. Gimpel, N. Schneider, B. O'Connor et al., "Part-of-speech tagging for twitter: annotation, features, and experiments," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Shortpapers*, pp. 42–47, Portland, OR, USA, 2011.
 - [19] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *Neural Computation*, vol. 28, pp. 2474–2504, 2016.
 - [20] B. Myagmar, J. Li, and S. Kimura, "Cross-domain sentiment classification with bidirectional contextualized transformer language models," *IEEE Access*, vol. 7, pp. 163219–163230, 2019.
 - [21] S. Schoenmackers, J. Davis, O. Etzioni, and D. Weld, "Learning first-order horn clauses from web text," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, vol. 10, pp. 1088–1098, EMNLP, Brussels, Belgium, 2010.
 - [22] E. Gilbert, "Widespread underprovision on reddit," in *Proceedings of the 2013 Conference of Computer Support Cooperative Work (CSCW'13)*, pp. 803–808, San Antonio, TX, USA, 2013.
 - [23] GPT-2 size, <https://knews.cc/zh-tw/tech/5rlolbk.html>.
 - [24] C. Sadowski and G. Levin, "Simhash: hash-based similarity detection," 2007, <https://www.webrankinfo.com/dossiers/wp-content/uploads/simhash.pdf>.
 - [25] K. Anil, "Jain and jian-jiang feng, "latent fingerprint matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 88–100, 2010.
 - [26] L. Zhang, Y. Zhang, J. Tang, Ke Lu, and T. Qi, "Binary code ranking with weighted hamming distance," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1586–1593, Portland, OR, USA, 2013.
 - [27] Hamming distance, http://www.code10.info/index.php%3Foption%3Dcom_content%26view%3Darticle%26id%3D59:hamming-distance%26catid%3D38:cat_coding_algorithms_data-similarity%26Itemid%3D57.
 - [28] J. Tiedemann, "Automatic construction of weighted string similarity measures," in *Proceedings of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pp. 213–219, 1999.
 - [29] D. D. Dvorski, *Installing, Configuring, and Developing with XAMPP*, "Skills Canada, Skills, Waterloo, Canada, 2007.
 - [30] M. Bouache, J. L. Glover, and J. Boukhobza, "Analysis of memory performance: mixed rank performance across microarchitectures," in *Proceedings of the International Conference on High Performance Computing*, pp. 579–590, Innsbruck, Austria, 2016.
 - [31] D. Foley and J. Danskin, "Ultra-performance pascal GPU and NVLink interconnect," *IEEE Micro*, vol. 37, pp. 7–17, 2017.
 - [32] M. Abdullahi, M. A. Ngadi, and S. i. M. Abdulhamid, "Symbiotic organism search optimization based task scheduling in cloud computing environment," *Future Generation Computer Systems*, vol. 56, pp. 640–650, 2016.
 - [33] H. Huang, H. Xu, X. Wang, and W. Silamu, "Maximum F1-score discriminative training criterion for automatic mispronunciation detection," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, pp. 787–797, March 06, 2015.
 - [34] X. Deng, Q. Liu, Y. Deng, and S. Mahadevan, "An improved method to construct basic probability assignment based on the confusion matrix for classification problem," *Information Sciences*, vol. 340, pp. 250–261, 2016.
 - [35] D. E. Over, C. Hadjichristidis, J. St. B. T. Evans, S. J. Handley, and S. A. Sloman, "The probability of causal conditionals," *Cognitive Psychology*, vol. 54, no. 1, pp. 62–97, 2007.
 - [36] T. Flaminio, L. Godo, and H. Hosni, "Boolean algebras of conditionals, probability and logic," *Artificial Intelligence*, vol. 286, Article ID 103347, 2020.
 - [37] E. Kalliamvakou, G. Gousios, B. Kelly, L. Singer, and M. Daniel, "German, and daniela damian, "the promises and perils of mining GitHub," MSR," in *Proceedings of the 11th Working Conference on Mining Software Repositories 2014*, pp. 92–101, Hyderabad, India, 2014.
 - [38] Web-crawler, <https://github.com/jwlin/web-crawler-tutorial>.
 - [39] Exchange-rate, <https://github.com/wert30678/Repo/blob/master/exchange-rate>.
 - [40] Web camera, <https://github.com/chinaev/python-video-streaming>.
 - [41] Market-analysis, <https://github.com/HowardNTUST/Marketing-Data-Science-Application/blob/master/Python-RFM-RF-basics/>.