

國立高雄大學資訊工程學系

碩士論文

應用元強化學習於雲端應用服務快速在線異常檢測

Applying Meta-Reinforcement Learning to Cloud Application Services for Fast Online Anomaly Detection

|  |  |
| --- | --- |
| 研 究 生： | 陳冠儒 |
| 指導教授： | 張保榮 教授 |

中華民國一百一十二年一月

**應用元強化學習於雲端應用服務快速在線異常檢測**

指導教授：張保榮 教授

國立高雄大學資訊工程學系

研究生：陳冠儒

國立高雄大學資訊工程學系碩士班

摘要

雲端運算與網路服務已成為現在科技服務的主流，據增的用戶對雲端負載增加壓力，導致資料中心發生非預期故障。雖然已有監測工具即時檢測，但大多是以被動式方法判定故障。檢測雲端服務的異常並不容易，使用者的行為會隨時變動，導致故障檢測模型在短時間內失去作用。因此，檢測模型除了具備準確性，也需要快速適應當下的資料分佈。本研究使用Deep Reinforcement Learning結合Model-Agnostic Meta-Learning(MAML)訓練框架生成在線預測模型，對伺服器資源使用率之時間序列進行異常檢測。MAML演算法建立多個子任務學習不同異常行為間的隱含表徵。每個子任務包含一個強化學習環境與一個代理者執行決策，最終損失透過Trust Region Policy Optimization(TRPO)優化並更新參數得到一個初始模型。最後，經過幾次小步長更新即可快速適應當前的使用者模式，減少管理人員更新模型參數的額外成本。

關鍵字：雲端運算、非預期故障、Deep Reinforcement Learning、Meta Learning、MAML、TRPO。

**Applying Deep Learning Approaches to the Fast Verification of Code Transformation**

Advisor: Dr. Bao-Rong Chang

Department of Computer Science and Information Engineering

National University of Kaohsiung

Student: Guan-Ru Chen

Department of Computer Science and Information Engineering

National University of Kaohsiung

ABSTRACT

This study uses natural language generation models GPT-2, MASS, and BART as code transformation models for code transformation operations. In order to speed up the verification after code transformation, a variational simhash (VSH) algorithm is proposed in this study to compare the similarity between the sample program and the preliminary programs. In addition to improving the accuracy of code similarity, it also reduces the number of qualified programs and compilation time. This study also proposes a piecewise longest common subsequence (PLCS) algorithm to check the consistency of execution results. Not only the number of string comparisons is reduced, but also less memory space is used to temporarily store the test results so that there is more memory for the string comparison operation, which can speed up checking the consistency of the execution results. Finally, this study uses local interpretable model-agnostic explanations (LIME) to explain the model's decision to make inferences. The experimental results show that using the VSH algorithm can reduce the qualified program by 22.11%. Using the PLCS algorithm, the number of string comparisons can be reduced by 21.18%, and the time to check the consistency of the execution results can be reduced by 23.01%. The entire code transformation process has improved the execution speed by 1.27 times. In addition, this study also built a graphical user interface so that users can easily operate the code transformation operation.

Keywords: Code Transformation Model, Variational Simhash, Piecewise Longest Common Subsequence, Explainable AI, and LIME.

# 誌謝

本論文能夠順利完成，首要感謝指導教授張保榮老師，張老師有耐心地指導研究方法以及實驗設計的部分，且對於論文的內文缺失部分也提出許多相當具有建設性的建議。

在研究所求學生涯中，在張老師的指導下，不僅學到了做研究應有的態度，也學到了做人處事的方法及許多研究外的種種事物，更讓我利用實驗室豐富的資源得到很多實作上的進步。在畢業之際，要感謝的是在我研究所的學長姐們郁傑、炯霖、函霖，在我初來乍到時不厭煩的教我電腦裝修與課業知識，讓我能在短時間內適應研究所的生活。也感謝學弟翔宇、易儒、富洋能夠接替我的工作，讓我能在撰寫論文期間無後顧之憂。再來，我由衷的感謝和我從大學到研究所共同打拼的同學佳衛，有你的管理實驗室帳務與報帳工作，讓我很放心處理我份內的事情。當我們忙碌到深夜時你也可以陪我打屁聊天，有事情時你也會主動詢問幫忙和提供建議，很感謝這三年有你在的研究所生活。

最後，很感謝讀研究所期間家裡的支持，不會限制我的選擇。當我壓力大時和你們聊天讓我身心得到釋放，周末回家時也讓我好好充電與鼓勵。有了你們的支持，才會有今天順利畢業的我。

**Directory**

[1 摘要 i](#_Toc105581002)

[2 ABSTRACT ii](#_Toc105581003)

[3 誌謝 iii](#_Toc105581004)

[**Directory** iv](#_Toc105581005)

[**List of Tables** vi](#_Toc105581006)

[**List of Figures** vii](#_Toc105581007)

[Chapter 1. Introduction 1](#_Toc105581008)

[Chapter 2. Related Work 4](#_Toc105581009)

[2.1 Literature review 4](#_Toc105581010)

[2.2 Anaconda 5](#_Toc105581011)

[2.3 CUDA 7](#_Toc105581012)

[2.4 Tensorflow 7](#_Toc105581013)

[2.5 Natural Language Toolkit (NLTK) 8](#_Toc105581014)

[2.6 Text transform model 9](#_Toc105581015)

[**(1) Generative Pre-trained Transformer 2 (GPT-2)** 9](#_Toc105581016)

[**(2) Masked Sequence to Sequence Pre-training (MASS)** 11](#_Toc105581017)

[**(3) BART** 12](#_Toc105581018)

[2.7 Simhash 13](#_Toc105581019)

[2.8 XAMPP 15](#_Toc105581020)

[2.9 Longest Common Subsequence (LCS) 16](#_Toc105581021)

[2.10 Multimedia information comparison 18](#_Toc105581022)

[2.11 Predetermined generative programs 19](#_Toc105581023)

[2.12 Explainable AI Technique 21](#_Toc105581024)

[Chapter 3. Research Method 23](#_Toc105581025)

[3.1 Code transformation process 23](#_Toc105581026)

[3.2 Deploy GPU Cluster Computing 25](#_Toc105581027)

[3.3 Create an Anaconda virtualized server environment 27](#_Toc105581028)

[3.4 Variational simhash algorithm (VSH) 27](#_Toc105581029)

[3.5 Piecewise longest common subsequence (PLCS) 31](#_Toc105581030)

[Chapter 4. Experimental Results and Discussion 36](#_Toc105581031)

[4.1 Experimental environment 36](#_Toc105581032)

[4.2 Experimental design 37](#_Toc105581033)

[4.3 Experimental results 38](#_Toc105581034)

[**4.3.1 Experiment 1** 38](#_Toc105581035)

[**4.3.2 Experiment 2** 39](#_Toc105581036)

[**4.3.3 Experiment 3** 43](#_Toc105581037)

[**4.3.4 Experiment 4** 45](#_Toc105581038)

[**4.5 Discussion** 46](#_Toc105581039)

[Chapter 5. Conclusion 48](#_Toc105581040)

[References 49](#_Toc105581041)

**List of Tables**

[Table 1. System Hardware Specifications 26](#_Toc105581042)

[Table 2. The Length of a Segment (Unit: bit) 34](#_Toc105581043)

[Table 3. Performance evaluation between LCS and PLCS 35](#_Toc105581044)

[Table 4. Open-source package 36](#_Toc105581045)

[Table 5. Example sentences 38](#_Toc105581046)

[Table 6. The list of example programs in Experiment 1 39](#_Toc105581047)

[Table 7. PLCS conformity according to the number of identical codes (unit: %) 41](#_Toc105581048)

[Table 8. Number of code lines reduction 42](#_Toc105581049)

[Table 9. Program execution time reduction (unit: second) 42](#_Toc105581050)

[Table 10. Comparison of the number of qualified programs produced 43](#_Toc105581051)

[Table 11. Comparison of the conformity check 44](#_Toc105581052)

[Table 12. Execution time of the entire process 44](#_Toc105581053)

**List of Figures**

[Figure 1. Anaconda control panel 6](#_Toc105581054)

[Figure 2. Jupyter web interface 7](#_Toc105581055)

[Figure 3. NLTK word segmentation flow 9](#_Toc105581056)

[Figure 4. NLTK word segmentation flow 10](#_Toc105581057)

[Figure 5. GPT-2 architecture 11](#_Toc105581058)

[Figure 6. MASS architecture 12](#_Toc105581059)

[Figure 7. BART architecture 13](#_Toc105581060)

[Figure 8. Simhash algorithm flow 14](#_Toc105581061)

[Figure 9. Calculate fingerprint 14](#_Toc105581062)

[Figure 10. Longest common subsequence (LCS) 17](#_Toc105581063)

[Figure 11. LIME processing flow 22](#_Toc105581064)

[Figure 12. Natural language generating program process 23](#_Toc105581065)

[Figure 13. Model generation stage – training phase architecture diagram 25](#_Toc105581066)

[Figure 14. Model generation stage 25](#_Toc105581067)

[Figure 15. Model use stage 25](#_Toc105581068)

[Figure 16. Anaconda Prompt control interface 27](#_Toc105581069)

[Figure 17. Variational simhash algorithm 29](#_Toc105581070)

[Figure 18. word2vec architecture 31](#_Toc105581071)

[Figure 19. Variational AutoEncoder (VAE) architecture 31](#_Toc105581072)

[Figure 20. PLCS algorithm 33](#_Toc105581073)

[Figure 21. DNN estimate the length of a segment 34](#_Toc105581074)

[Figure 22. Loss curve during DNN training phase 35](#_Toc105581075)

[Figure 23. GPU workstation cluster 36](#_Toc105581076)

[Figure 24. Table of four sample programs 37](#_Toc105581077)

[Figure 25. Screenshot of NLTK word segmentation 38](#_Toc105581078)

[Figure 26. The pass ratio of the preliminary programs in Example 1 40](#_Toc105581079)

[Figure 27. The pass ratio of the preliminary programs in Example 2 40](#_Toc105581080)

[Figure 28. The pass ratio of the preliminary programs in Example 3 40](#_Toc105581081)

[Figure 29. The pass ratio of the preliminary programs in Example 4 40](#_Toc105581082)

[Figure 30. Execution result of Example 1 41](#_Toc105581083)

[Figure 31. Execution result of Example 2 41](#_Toc105581084)

[Figure 32. Execution result of Example 3 41](#_Toc105581085)

[Figure 33. Execution result of Example 4 41](#_Toc105581086)

[Figure 34. LIME explains the results produced by GPT-2 45](#_Toc105581087)

[Figure 35. LIME explains the results produced by MASS 45](#_Toc105581088)

[Figure 36. LIME explains the results produced by BART 45](#_Toc105581089)

[Figure 37. LIME explains the results produced by simhash 46](#_Toc105581090)

[Figure 38. LIME explains the results produced by variational simhash 46](#_Toc105581091)

[Figure 39. LIME explains the results produced by LCS 46](#_Toc105581092)

[Figure 40. LIME explains the results produced by PLCS 46](#_Toc105581093)

# Chapter 1. Introduction

雲端運算技術整合了大量運算資源並將其虛擬化，成為了現在應用程式提供服務的媒介，使用者可以透過網際網路按需(on-demand)即取的運算。資源集成管理減少維護成本，整合了多個異質性系統並在特殊任務上提供強大的運算能力支持。隨著集群的複雜度增加，發生崩潰或停機時總會造成困擾。資料中心全天候監控著數台伺服器的運行狀態，包含處理器、記憶體、磁碟、網路設備等日誌，並在異常狀態發生時發出警告。但雲端服務的動態性導致故障難以預測，許多正常行為不斷的被重新定義，正需要一個主動性的方法來減少故障帶來的影響。

深度學習在許多方面上有很好的成就，但需要大量人力生成標籤化的資料集訓練維持準確度，當資料分佈發生變化會導致模型失效。其中，時間序列的異常檢測任務是一大挑戰。首先，異常樣本並不常見，時常會導致樣本比例失衡，其次，時常會發生連續性的異常導致異常模式難以辨認。在雲端伺服器系統的資源使用率序列中，可以得知當使用者行為發生變化時，導致資料分布變化，使得模型無法即時適應。若是在執行高風險應用，發生故障會造成很大的影響。

近年來強化學習演算法的快速發展，很大程度的改善了異常模式難以辨認的問題。深度強化學習可以透過與環境互動學習，並透過經驗調整模型參數形成最優策略。然而，強化學習與環境的相依性強，當環境發生變化時也會導致此前的最優策略無效，對於雲端複雜的環境變化會難以適應。針對此問題，我們結合了Meta Learning。Meta Learning演算法致力於少量樣本的快速學習，以過去經驗來對新的、少數的資料做出快速而準確的學習。其中，MAML(Model-Agnostic Meta-Learning)為Meta Learning系列的演算法之一，它的特色在於與模型無關，專注在學習各個任務之間的共通表徵。我們研究此演算法應用於快速適應異常模式的應用，解決以上問題。

本文研究對象為台灣恩智浦半導體股份公司雲端應用服務，服務包含多項虛擬主機共同運作維持。該公司使用Zabbix Server監測各項服務的數值，並設定超過預設閥值發出警告通知管理者。然而，此方案僅能告知使用率過高的裝置，無法體現整體的故障及實際異常原因。因此本研究首先分析Zabbix Server監控資料，使用滑動窗口生成時間序列資料以及異常標記，以TSFEL、PyOD套件擷取可轉移的元特徵。再來，我們使用MAML-RL演算法在訓練階段產生不同的子任務進行訓練，記錄策略損失和網路參數。外部的循環我們使用TRPO(Trust Region Policy Optimization)尋找出最優的策略最大化決策的獎勵，以此生成一個初始模型。最後，使用目標資料集進行小步長訓練快速得到一個在線預測模型。

# Chapter 2. Related Work

## 2.1 Literature review

Natural language processing has also expanded in various industries in recent years. For example, intelligent personal assistants can interact with people through natural speech-language and then assist users in handling personal affairs or connecting to smart home appliances and other application levels [12] (Using intelligent personal assistants to assist the elderlies. An evaluation of Amazon Alexa, Google Assistant, Microsoft Cortana, and Apple Siri). Chatbots use natural language to communicate with humans. And can provide real-time services around the clock and provide more accurate product information and personalized services [13] (A Smart Chatbot Architecture based NLP and Machine Learning for Health Care Assistance). As well as sentiment analysis is a method of mining words or discourse opinions. Rules are established to quantify vocabulary to know the emotions, opinions, or intentions behind the words, which can be used for analysis or classification [14] (Deep Sentiment Classification and Topic Discovery on Novel Coronavirus or COVID-19 Online Discussions: NLP Using LSTM Recurrent Neural Network Approach).

People are in the process of writing programs. Often due to deadlines, difficulties in program debugging, or temporary requirements, the program cannot be completed immediately or there are defects. Suppose the program is automatically generated by the machine [15] (Automatic Source Code Documentation using Code Summarization Technique of NLP). You can save a lot of time and effort. However, program after code transformation can only be used after verification because the program after code transformation may have some code missing or too outrageous results. So it must be verified before it can be used. In the verification process can take a lot of time.

Therefore, this study aims to realize the " Applying Deep Learning Approaches to the Fast Verification of Code Transformation " and will build and integrate the following systems: Anaconda, Tensorflow, CUDA, etc. This study will use the following key technologies: NLTK, GPT-2, MASS, BART, Simhash, LCS, LIME, etc., to achieve the goal of this paper. This chapter will introduce the critical content of these technologies in order.

## 2.2 Anaconda

Anaconda [16] is currently the most popular virtual environment management system in the Python development platform. In addition to many users and enterprise users, there are currently more than 1000 Data Science Packages can available, which are suitable for conda systems under different operating system environments of Windows, Linux, and MacOS. Anaconda is mainly used in data science [17], machine learning [18], big data processing, and predictive analytics [19]. This study uses the Anaconda platform for environmental construction. The virtual environment set up by Anaconda can support various versions of python. It can also install many Python Packages for science, mathematics, engineering, and data analysis in the Anaconda virtual container [20], making the system have high performance, high fault tolerance, etc. Effect. The Anaconda control panel is shown in Figure 1.

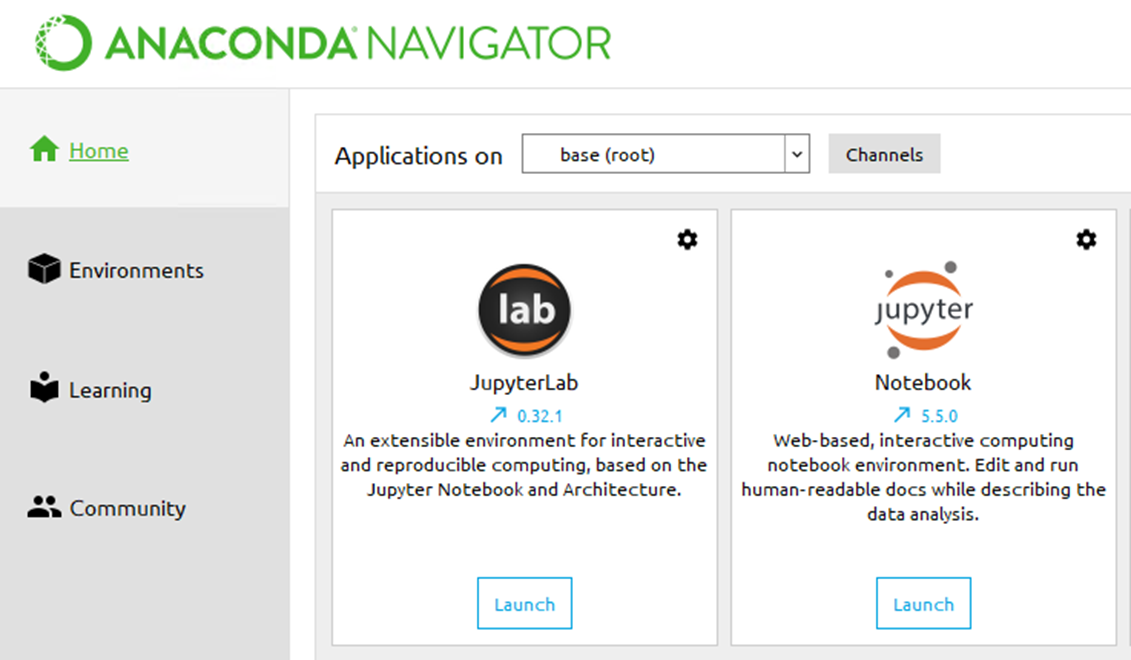


Figure 1. Anaconda control panel

In addition to installing the spyder compiler, Anaconda can also install Jupyter Notebook as shown in Figure 2. Jupyter Notebook allows users to write and execute programs remotely through the network and execute and debug code in sections. It can improve the efficiency of debugging in writing programs and allow users to understand some programs' operation results quickly. The programming language used in the programming of this study is Python. Python can be used in data analysis and processing, web development applications and artificial intelligence applications, etc., and supports most open source software and suites. For example, Tensorflow for deep learning, MatplotLib [21] for data science, MySQLdb for database access, NLTK for English natural language segmentation model, and an English natural language generation model GPT-2, MASS, and BART, etc. Compared to C++ or Java, Python enables users to express ideas with less code and is more widely and more multi-tasking.

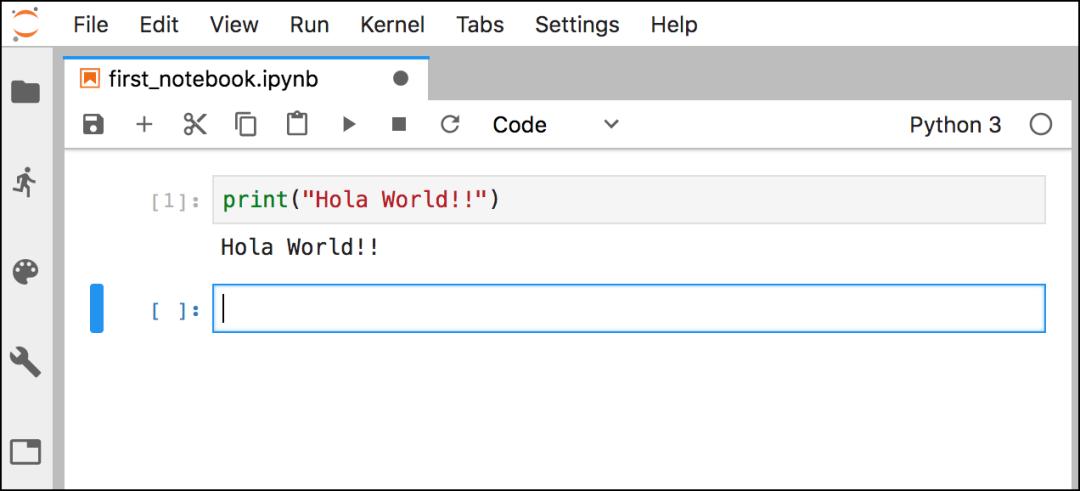


Figure 2. Jupyter web interface

## 2.3 CUDA

NVIDIA [22] was established in 1993. There is a fabless IC semiconductor design company. The main products are graphics processors, PC platform (motherboard logic core) chipsets, and software for digital media players. In recent years, the integration technology CUDA (Compute Unified Device Architecture) [23] has been launched, NVIDIA's official name for GPGPU. CUDA is a computing environment that can perform parallel operations on NVIDIA graphics processors. Users can use the C language extension of CUDA to write programs directly in the C language. It can design data distribution and program flow distribute computing work through thousands of threads and hundreds of cores in graphics processors. CUDA is also compatible with OpenCL.

## 2.4 Tensorflow

Google Brian developed Tensorflow for various perception and language understanding tasks in machine learning. In 2015, Google open-sourced it, making it one of the most important deep learning frameworks today. Tensorflow supports a variety of deep learning algorithms and is used in significant enterprise services. It also supports running deep learning on a variety of different devices. Ex: Tensorflow Lite, Tensorflow.js, etc.TensorFlow can run on one or more CPUs and GPUs, and can also run on embedded systems

## 2.5 Natural Language Toolkit (NLTK)

Natural Language Processing (NLP) is regarded as a sub-discipline in artificial intelligence and linguistics. This field discusses how to process and use natural language, including multiple aspects of steps. Cognition, understanding, generation, etc.. Cognition and understanding: Let the computer turn the input natural language into interesting symbols and relationships.

Natural Language Toolkit (NLTK) is a natural language processing toolkit. It is from Steven Bird and Edward Loper of the University of Pennsylvania developed a module based on Python. It is used in the field of NLP research. With more than 100,000 lines of code to date. This open-source project includes datasets, Python modules, tutorials, and more.

The main functions of NLTK are English word segmentation, part-of-speech tagging, font restoration, stop words, etc. In NLTK, the text is usually stored as a list. That is, a text is a vast list. If additional information such as part of speech is attached, it can be converted to the form of a dictionary. Because the Latin family is a little troublesome, they like to add a modifier after the word to describe different tenses, actions, parts of speech, moods, and quantities. So we will turn all the same words in different tenses or inflections into the same word for processing. Finally, filter out unwanted words. The flow chart of English word segmentation is shown in Figure.3.

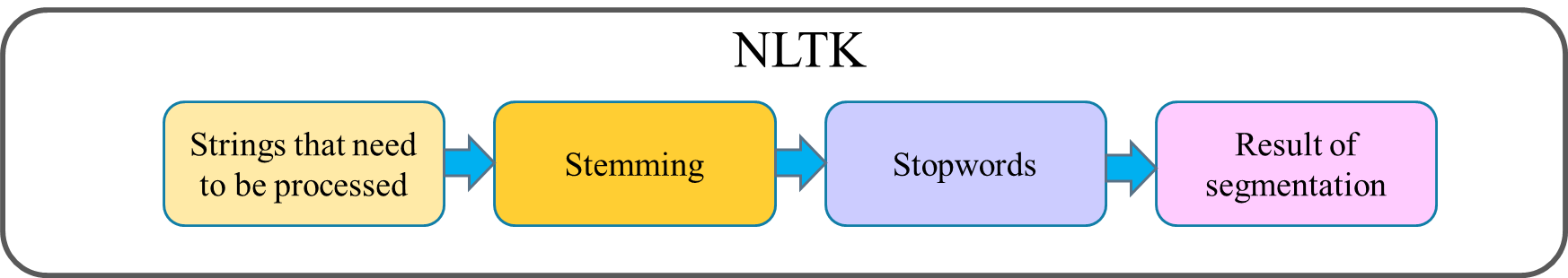


Figure 3. NLTK word segmentation flow

## 2.6 Text transform model

**(1) Generative Pre-trained Transformer 2 (GPT-2)**

Language generation aims to convey information by predicting the next word in a sentence. Among millions of possibilities, people use language models to predict a word. Language models can be built at the character level, n-gram level, sentence level, or even paragraph level. In the early days, language models trained by recurrent neural networks (RNNs) had the problem of vanishing gradients. As the length of the sequence increases, the RNN cannot store words encountered far later in the sentence and can only make predictions based on the most recent words. Later, although the language model trained by the long short-term memory model (LSTM) can solve the problem of gradient disappearance. However, since there is still a complex sequential path from the previous unit to the current unit, there is a limit to the amount of information saved. In 2017, Google proposed Transformer [24], which can solve the above problems using a self-attention mechanism. Transformers are currently widely used in various natural language processing tasks, such as language modeling, machine translation, and text generation. The Transformer consists of an Encoder and a Decoder, called a stack of Transformer architectures. The former handles inputs of arbitrary length, and the latter outputs the generated sentences. As shown in Figure 4. In many subsequent studies, the researchers tried to remove either the Encoder or the Decoder, use only one Transformer stack, stack as much as possible, and provide a large amount of training text and machine equipment

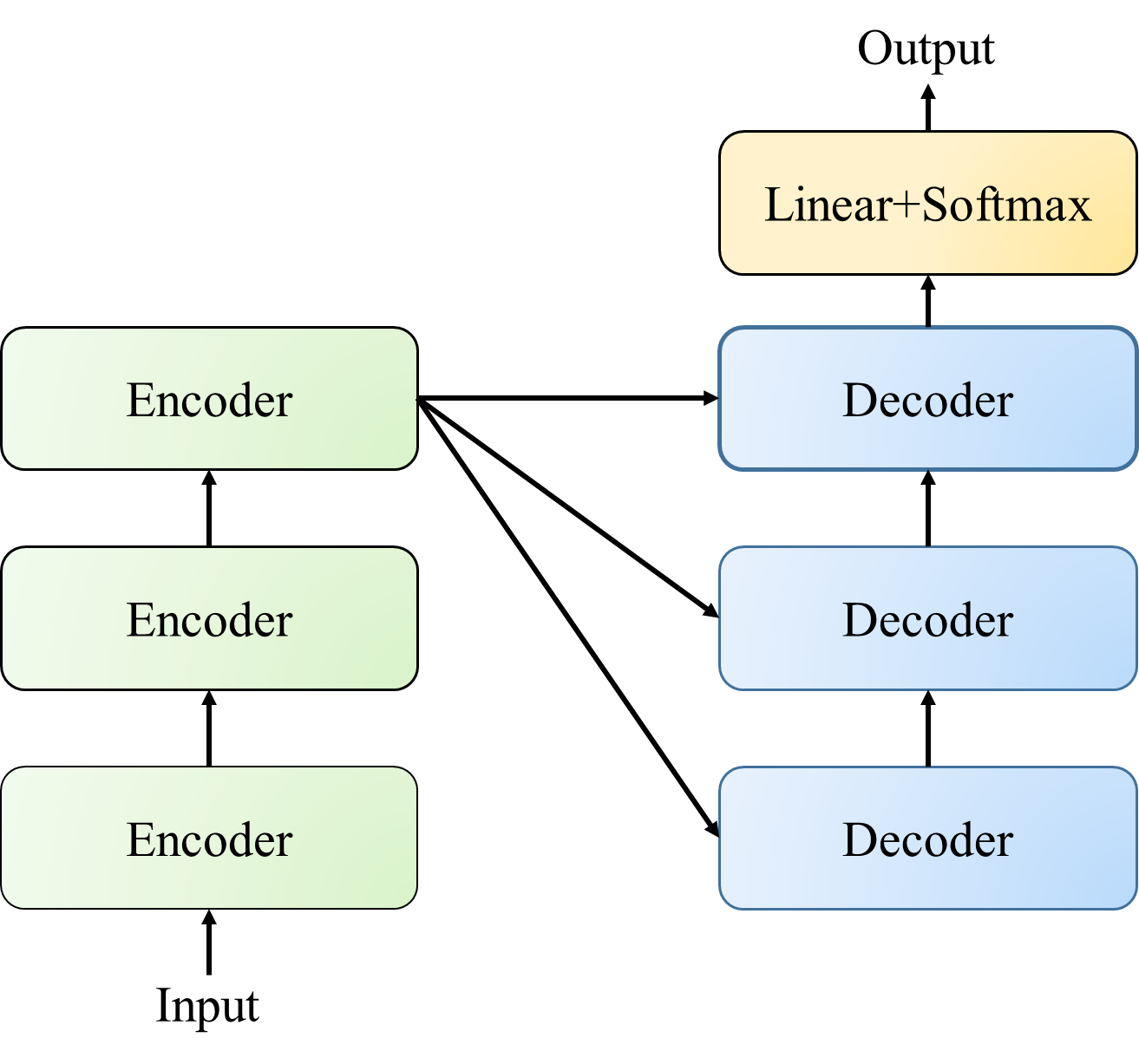


Figure 4. Traditional text transformer model

The second generation of Generative Pre-Training referred to as GPT-2, is an unsupervised Transformer language generation model released by OpenAI in 2019. GPT-2 is composed of the Decoder architecture based on the Transformer model, as shown in Figure 5. It crawls 8 million web pages and a 40G large dataset "WebText" [25] from the Internet as the training data for the language model and trains multiple models of different sizes. The stacking height is the difference in the size of different GPT-2 models. Currently, there are four sizes of models, namely GPT-2 Small, GPT-2 Medium, GPT-2 Large, and GPT-2 Extra Large [26].

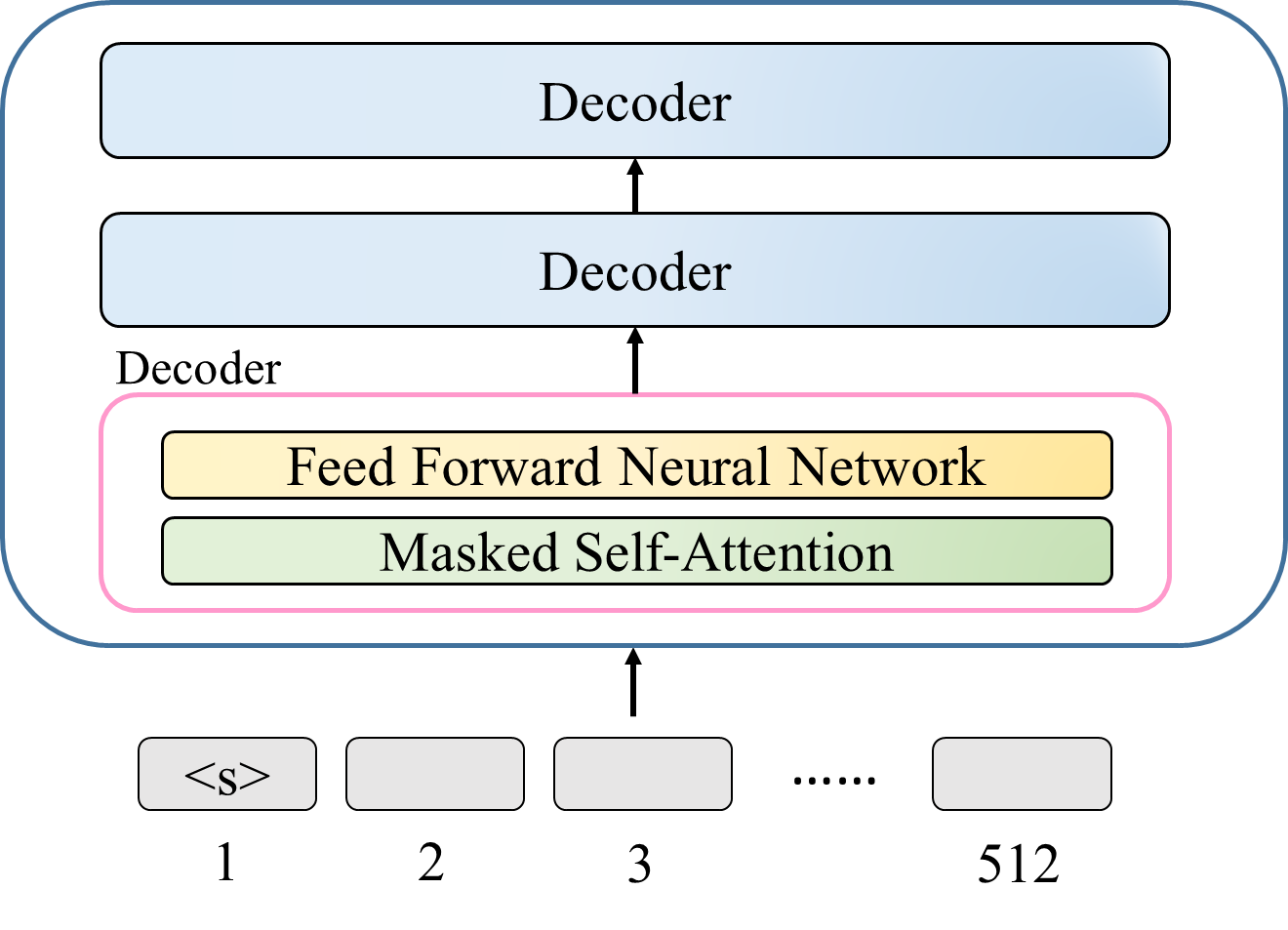


Figure 5. GPT-2 architecture

**(2) Masked Sequence to Sequence Pre-training (MASS)**

Masked Sequence to Sequence Pre-training [27], referred to as MASS, is a new pre-training method proposed by Microsoft in June 2019. Sequence-to-sequence natural language generation tasks consist of Encoder, Decoder, and Attention. The architecture is shown in Figure 6. MASS randomly masks a continuous segment of length k on the sentence and then predicts and generates the segment through the Encoder-Attention-Decoder model. MASS pre-training has three significant advantages. First, to encourage the Decoder to extract information from the Encoder to help predict consecutive segments, the words masked on the Decoder side are the words that are not masked on the Encoder side. This enables joint training of the Encoder-Attention-Decoder structure. Second, to provide the Decoder with more useful information, the Encoder is forced to extract the semantics of unmasked words to improve the Encoder's ability to understand the original sequence text. Third, let the Decoder predict continuous sequence fragments to improve the language modeling ability of the Decoder.

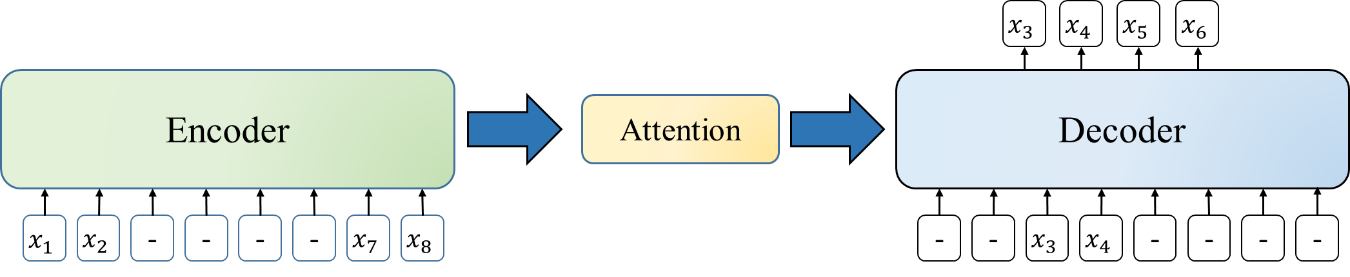


Figure 6. MASS architecture

**(3) BART**

Facebook AI proposed BART [28] in October 2019. It is a pre-training model of denoising autoencoder seq2seq [29] structure based on the Transformer. It can select the input text to be masked with an arbitrary noise function. In extreme cases, the original information can be completely missing. Like MASS, the Encoder input is masked, but the difference is that BART replaces the excitation function ReLU with GeLUs, and BART does not change the Decoder. The architecture is shown in Figure 7.

BART encodes, calculates, and extracts features from the input masked text in the Encoder section. The Decoder will use Cross Attention [30] and the hidden state results of the last layer of the Encoder to calculate. At this time, the output of the Decoder and the original text, before being destroyed by the noise function, calculate the cross-entropy loss (cross-entropy loss) and then use this cross-entropy loss to optimize the entire model.

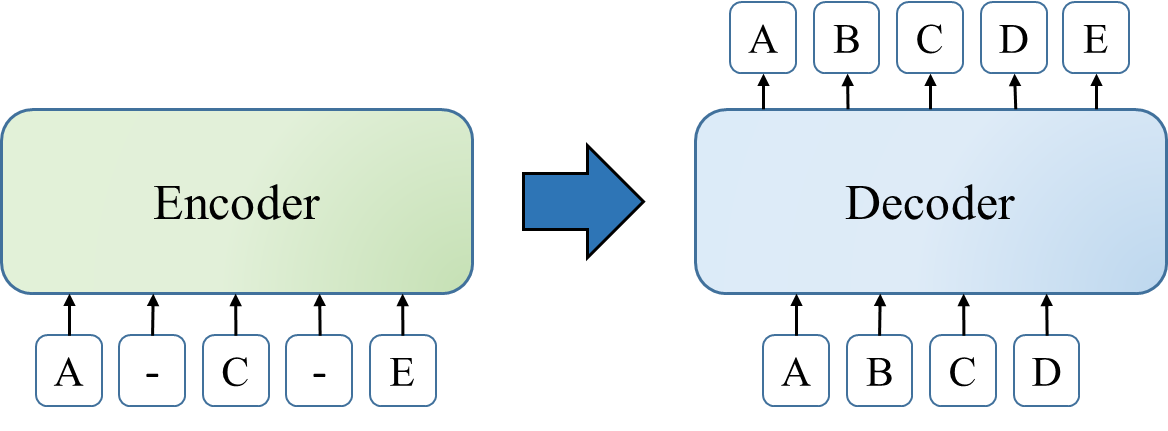


Figure 7. BART architecture

## 2.7 Simhash

The traditional Hash algorithm is a way to any data through hash to establish a fingerprint[31]. In an ideal hash function, the similarity of the hash values ​​should directly reflect the similarity of the input content. In other words, if the input content changes slightly, even if the two original content differ by only one byte, the hash value will change a lot. Therefore, the traditional Hash [32] cannot measure the similarity of the original content. Simhash [33] is a locality-sensitive hash whose main idea is to reduce dimensionality. The high-dimensional feature vector is mapped to a low-dimensional feature vector, and the similarity of the article is determined by the Hamming Distance of the two vectors. The smaller the Hamming distance, the higher the similarity. The overall process is shown in Figure 8. sentence segmentation, Hash calculation, weighting, merging, and dimensionality reduction. Segment the sentences in the text to get the feature vector, and then perform the Hash calculation. Then, weighting is given to the feature vector that has just been calculated, and then all weighted vectors are accumulated to be combined. Dimension reduction takes the accumulated result greater than zero as one and less than zero as zero to obtain the text fingerprint (Fingerprint) as shown in Figure 9. Finally, it calculates the Hamming distance between the two text fingerprints.

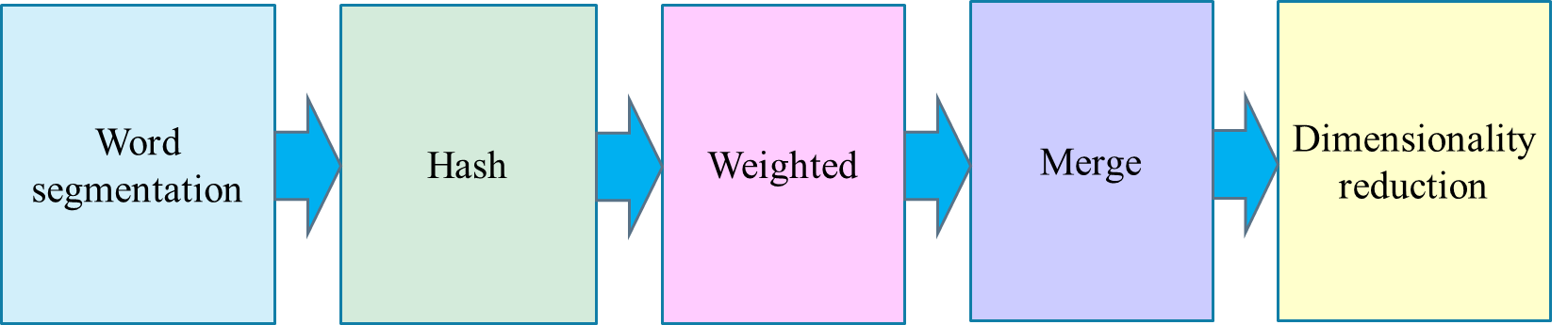


Figure 8. Simhash algorithm flow

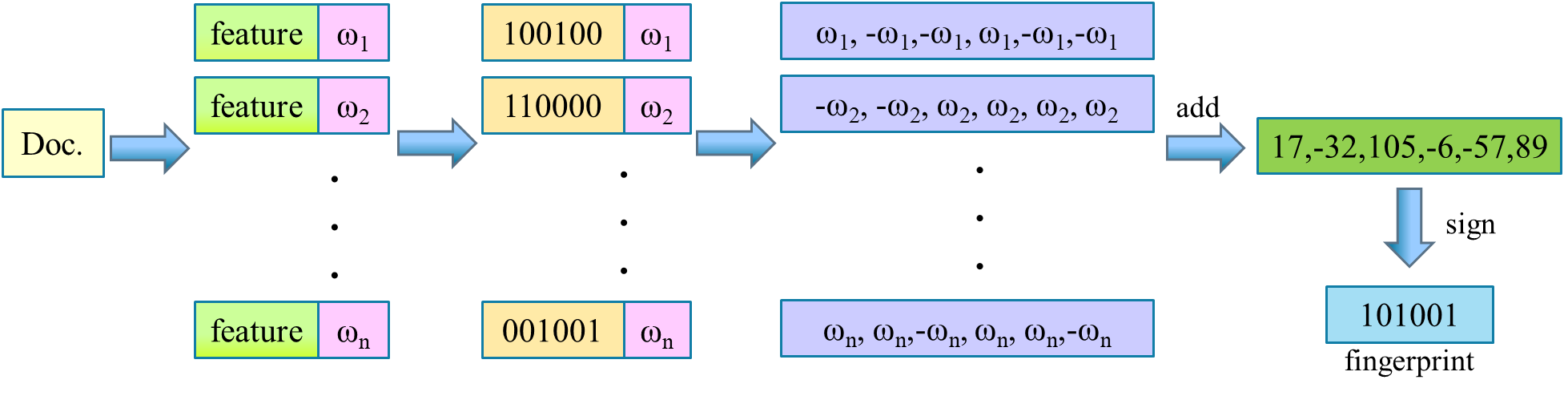


Figure 9. Calculate fingerprint

The Hamming distance [34] of two equal-length strings is the number of different characters in the same position in the information domain. Hamming distance measures its character vector space in a fixed length, which satisfies non-negative, unique, and symmetric.In the Hamming distance formula Eq. (1) [35]. *dHAD*is the Hamming distance between objects *i*, *j*, and *k*, the corresponding variable in the total number of variables *n*. In Eq. (2) and Eq. (3), [*yi,k≠yj,k*] is a value of 1 or 0 to judge True or False based on *yi,k≠yj,*k.

(1)

(2)

(3)

If use the Hamming distance to measure the similarity, the similarity can be converted into a pass rate as the degree of similarity of the tested object in line with the original content. According to the Hamming distance *dHAD* and the total number of variables *n*, we can obtain the qualification ratio from Eq. (4).

(4)

Generally speaking, the simhash algorithm uses the formula of tf-idf to estimate the corresponding weight value for each keyword. In Eq. (5), is referred to as inverse document frequency, which refers to a measure of the universal importance of a word. Furthermore, represents the number of files in which the word *i* appears in the text *j*, and is the total number of all texts. In Eq. (6), represents term frequency meaning the average frequency of the word *i* in the text *j*, and stands for the frequency of the word *i* appearing in the text *j*. is the sum of the frequency of each word appearing in the text *j*, which is the length of the text. In Eq. (7), is the estimated weight value of the specific word, is the term frequency meaning the frequency of a word given in the file, and is the inverse document frequency, a measure of the universal importance of a word given in the file.

(5)

(6)

(7)

## 2.8 XAMPP

The Web Server, also known as the web service, is transmitted to the client through the Hypertext Transfer Protocol (i.e., HTTP). The more common web servers are Apache HTTP Server (referred to as Apache) and Microsoft's Internet Information Server (IIS). In the early days, to build a complete web server on each platform, it was necessary to download Apache[36], PHP[37], and MySQL[38], respectively. Additional installation of the PHPMyAdmin database is required if necessary. After installing the software, users often set the parameters incorrectly and cannot execute the software. Later, integrated installation packages such as AppServ or The Uniform Server were developed, allowing users to save a lot of installation time, set steps and debugging time, and research and edit website work.

XAMPP [39] has been widely used in recent years and is an integrated service that can quickly set up stations. X of XAMPP supports cross-platform, A of XAMPP is Apache, M of XAMPP is MySQL or MariaDB, P of XAMPP is PHP, and P of XAMPP is Perl. It supports various operating systems such as Windows, MacOS, and Linux. It enables general web developers and web designers to quickly set up servers on their computers and then operate tests. In addition, when installing XAMPP, you can also choose to install utilities such as FileZilla FTP server and Mercury Mail Server.

## 2.9 Longest Common Subsequence (LCS)

Longest Common Subsequence [40], abbreviated LCS. It means that in multiple sequences, it appears each sequence and has the longest length. Unlike searching for the longest common substring, the position of the subsequence in the original sequence does not need to be continuous, as shown in Figure 10. The yellow part in Figure 10 indicates the completion of the comparison. We use dynamic programming to find the length of the LCS.

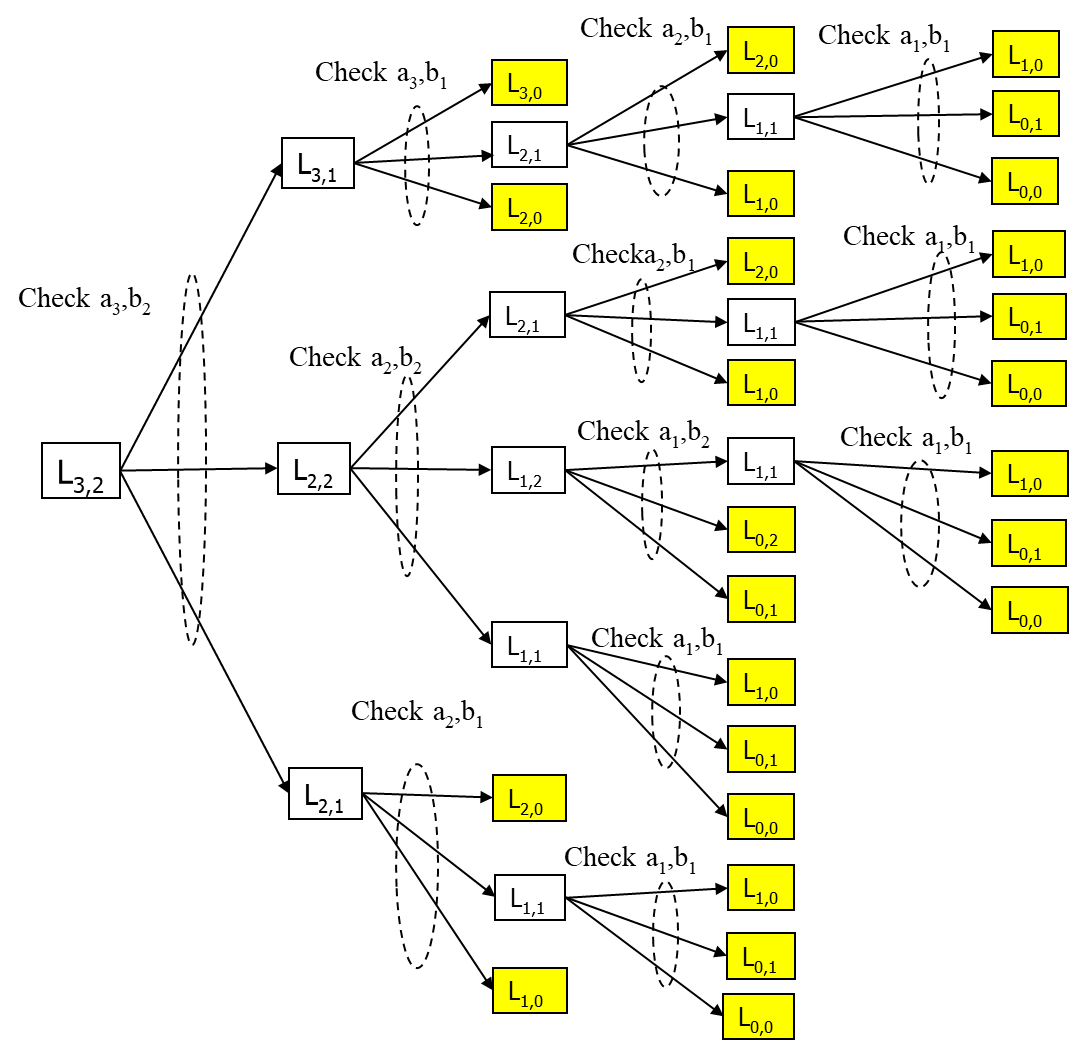


Figure 10. Longest common subsequence (LCS)

We assume that z(z1,z2,⋯,zk) is the LCS of x and y. If xm=yn, then zk=xm=yn. zk−1 is the LCS of xm−1 and yn−1. However, xm≠yn, zk is the LCS of xm and yn−1. Or the LCS of xm−1 and yn. Then the problem of solving LCS becomes two sub-problems of recursive solution. However, there are many repeated sub-problems, resulting in low efficiency. To calculate later, the space is replaced by time, and the two-dimensional array c[i,j] is used to record the LCS lengths of the word strings x1x2⋯xi and y1y2⋯yj. The state transition equation can be obtained in Eq. (8).

(8)

## 2.10 Multimedia information comparison

Multimedia information includes text, pictures, sound, video, animation, and other forms. Different media types have different content and formats, corresponding content management and processing methods are also different, and the storage capacity is also different.

Sentence similarity will use some distance for text comparison. For example, Euclidean distance, Manhattan distance, etc. The smaller the distance, the greater the similarity.

Generally speaking, the image similarity will use the Hash algorithm. By obtaining the hash value of the image, compare the Hamming distance of the hash value of the two images to measure whether the two images are similar. The more similar the two images are, the smaller the Hamming distance between the two images.

After extracting these features, the comparison of sounds is usually based on some features, such as frequency, tone, etc., and then comparing [42].

As for the comparison of the films, the films are cut into pictures one by one for picture comparison. Or grab the object in the picture, track the object's position in each picture, draw a displacement map and compare it [43].

Although there are many methods to compare different forms of media information, this study uses LCS as a single effective method to detect the consistency of multimedia information, which is suitable for text, pictures, sounds, and movies.

In this study, we convert the execution result of each qualified program and the execution result of the corresponding sample program into ASCII code or binary code and use LCS to check the conformity. To check whether the execution result of the generated program is consistent with the execution result of the sample program, please confirm that the generated program is available.

## 2.11 Predetermined generative programs

This study theoretically introduces a statistical estimation of the number of predetermined generated programs. This estimation implies how many preliminary programs are needed to ensure that the code transformation process can find the best-performance generated program. Therefore, users can first calculate the quantity of the preliminary programs generated by a sample program with a pass ratio exceeding 90% and obtain the ratio , as shown in Eq. (9). In Eq. (9), represents the total number of primary programs, and stands for the number of primary programs whose pass ratio exceeds 90%. After the previous calculation, Eq. (10) gets the average ratio , where is the total number of sample programs. Then users can find out the number of misjudgments in, and calculate the ratio of misjudgments , as shown in Eq. (11). In Eq. (11), represents the number of misjudgments among the number of the preliminary programs with a pass ratio of more than 90%. Then Eq. (12) can obtain the average ratio of misjudgments .

(9)

(10)

(11)

(12)

After that, users can calculate the number of the preliminary programs whose pass ratio is less than 90% to get the ratio , as shown in Eq. (13). In Eq. (13), Represents the number of the preliminary programs whose pass rate is less than 90%. Next, Eq. (14) finds out the average ratio . Then users can find out the number of misjudgments and then calculate the proportion of misjudgments , as shown in Eq. (15). In Eq. (15), represents the number of misjudgments. The preliminary programs have many misjudgments with less than a 90% pass ratio. Finally, Eq. (16) gives the average proportion of misjudgments .

(13)

(14)

(15)

(16)

Eq. (17) counts the total of preliminary programs generated by all the sample programs to get . According to the statistics such as , ,, , and , Eq. (18) obtains the average pass ratio of over a 90% probability [44]. Assuming that the pass ratio of programs exceeds 90%, means that for these j programs, the probability that the pass rate of the similarity check exceeds 90% is valid, such as Eq. In Eq. (19), represents programs with a similarity check pass rate of 90% or more among the generated programs. According to the above statistics, the probability of the similarity check with the pass ratio of programs exceeding 90% is. Therefore, Eq. (20) can infer the minimum number of preliminary programs generated from the code transformation process to ensure that at least programs have a similarity check with the pass ratio of more than 90%, where N is the total number of preliminary programs to be generated.

(17)

(18)

(19)

(20)

## 2.12 Explainable AI Technique

Most data scientists tend to prefer high-precision metrics when using models to solve the problems, and high-precision models are too complex to understand the decision-making of their algorithms. Because the algorithms are too complex, the designers can consistently not explain why artificial intelligence can achieve specific effects. Today, researchers propose explainable artificial intelligence so that people can understand the inference of the results from AI-related algorithms. There are three classification criteria for interpretability methods: (1) Essential or ex-post interpretability, (2) Model-specific or general, and (3) Local or global interpretability. In 2016, Marco Ribeiro, Sameer Singh, and Carlos Guestrin proposed Local Interpretable Model-Agnostic Explanations (LIME) [45], a post-analytical approach to interpretation after model establishment, as shown in Figure 11. The first is to randomly select attributes from the data as data perturbations and label the results further. Next, in LIME, we calculate the distance between the result and the attributes to get the weights of the attributes and then filter the attributes with small weights. After that, we can explain the result according to the remaining attributes. Therefore, people can use explainable AI techniques to supervise the rules discovered by AI systems, and judging those rules can explain the outcomes of their decisions.

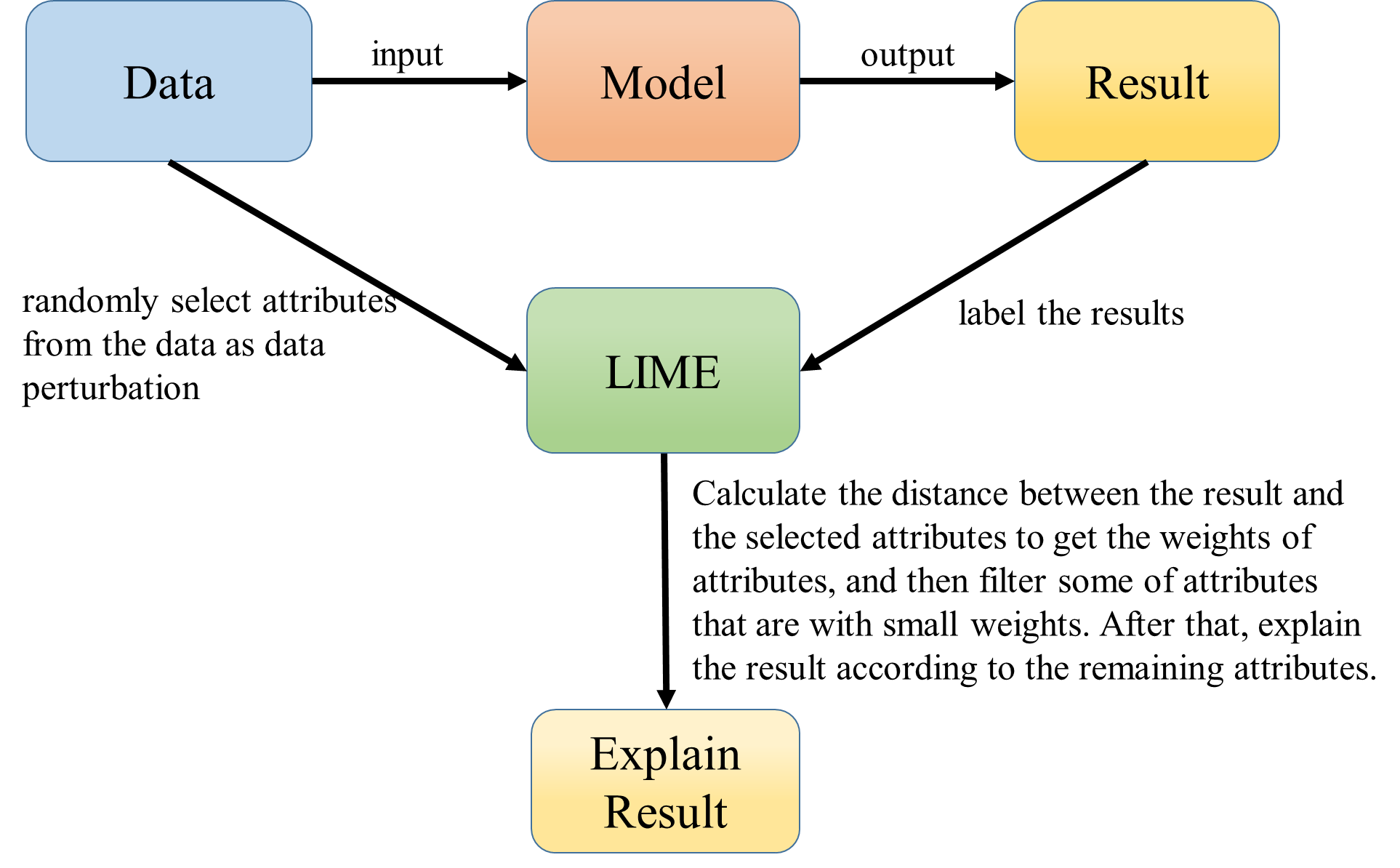


Figure 11. LIME processing flow

# Chapter 3. Research Method

本研究結合台灣恩智浦半導體公司集群建立異常檢測系統，該集群使用Hadoop與Spark框架，資料來源皆是透過HDFS擷取串流資料。本系統首先擷取過去的硬體資源使用率資料進行滑動窗口標記，以PyOD、TSFEL擷取序列的重要特徵後進行Meta-Reinforcement Learning訓練與測試得到初始模型。針對檢測對象我們進行小步長參數更新後佈署至集群。最後，警報系統會隨時詢問模型的預測結果是否有異常，當系統發現有即將到來的故障時會即時通知管理人員處置，減少非預期意外發生，系統運行流程如Figure 12所示。

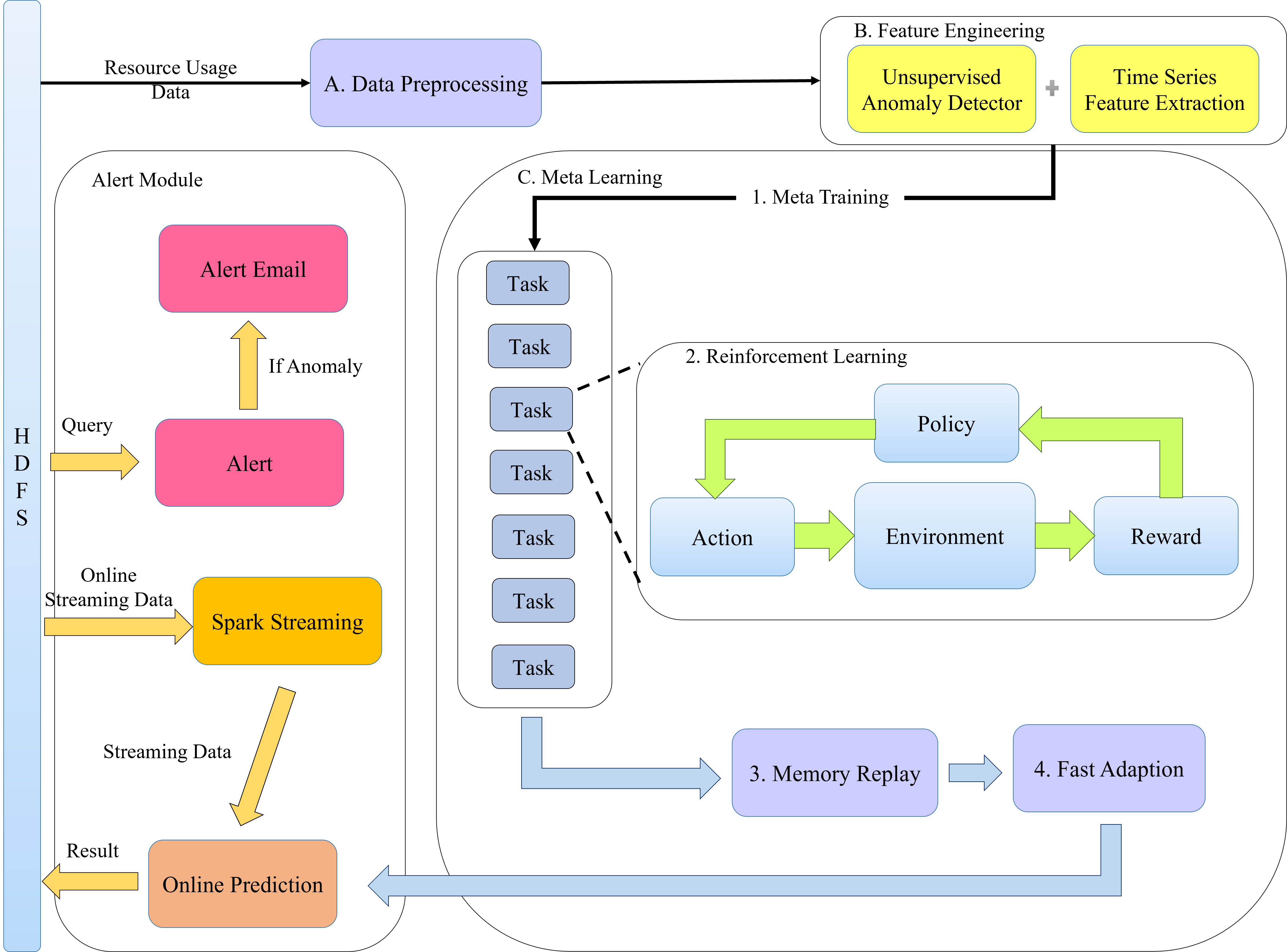


Figure 12.

## 3.1 系統異常狀態分析

本研究對象為台灣恩智浦半導體股份公司用於半導體封測生產線資料的上傳、搜尋與運算的集群，該集群使用Zabbix Server工具監控集群內的各項數值。Zabbix Server是一種伺服器的監控工具，可以監測資料中心內各項服務和資源的使用率，並將監測資料即時寫入CSV檔案儲存。本步驟首先觀察恩智浦公司Oplus應用服務在2021/2/8~2021/3/10之資源使用率資料。Oplus為半導體生產線上的重點服務，該應用結合多台主機的運算資源。我們發現該應用服務的記憶體會在使用率達到高峰時發生一次切換，切換當下 CPU使用率也會達到最高峰，切換後記憶體會維持約20天的上升趨勢，如Figure 13(a)、(b)所示。

|  |
| --- |
| (a) |
| (b) |

Figure 13. Resource usage plot

硬碟佇列(Disk Queue)長度也是一項重要指標，當硬碟佇列長度大於2時代表硬碟此時等待處理的任務過多。當Oplus應用服務的CPU使用率在高使用率與低使用率時有較長的硬碟佇列長度，如Figure 14所示。因此可知，複雜的走勢難以用閥值來判定異常。

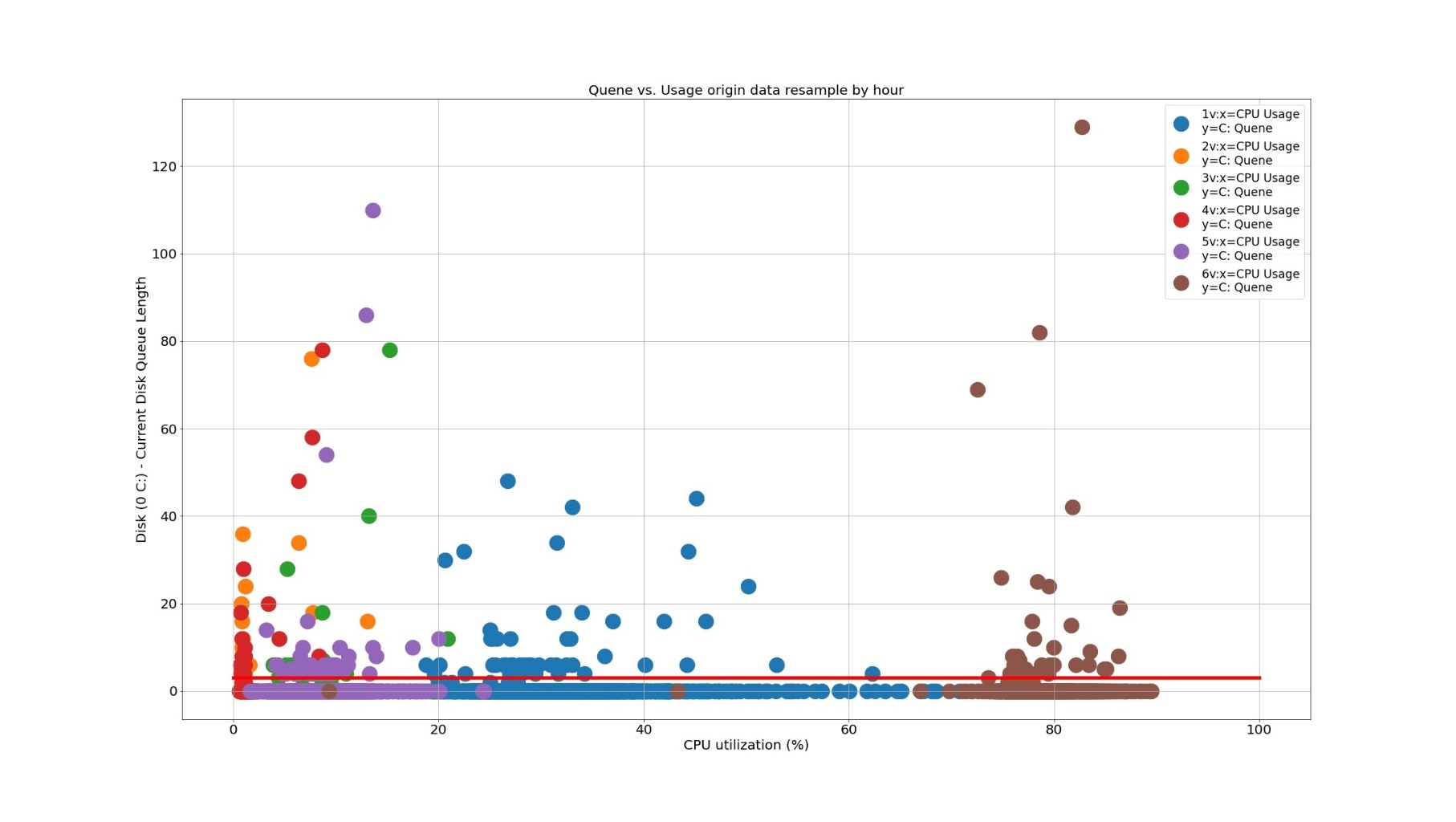


Figure 14. Disk queue and usage plot

## 3.2 異常資料標記

資料集為每三分鐘接收一次資料。其中，包含了CPU、Memory、Disk Queue等資料以及不同編號的虛擬機(代號1V、2V、3V、4V......等)。我們主要觀察系統的CPU與Memory使用率資料作為主要特徵，如Figure 15所示。

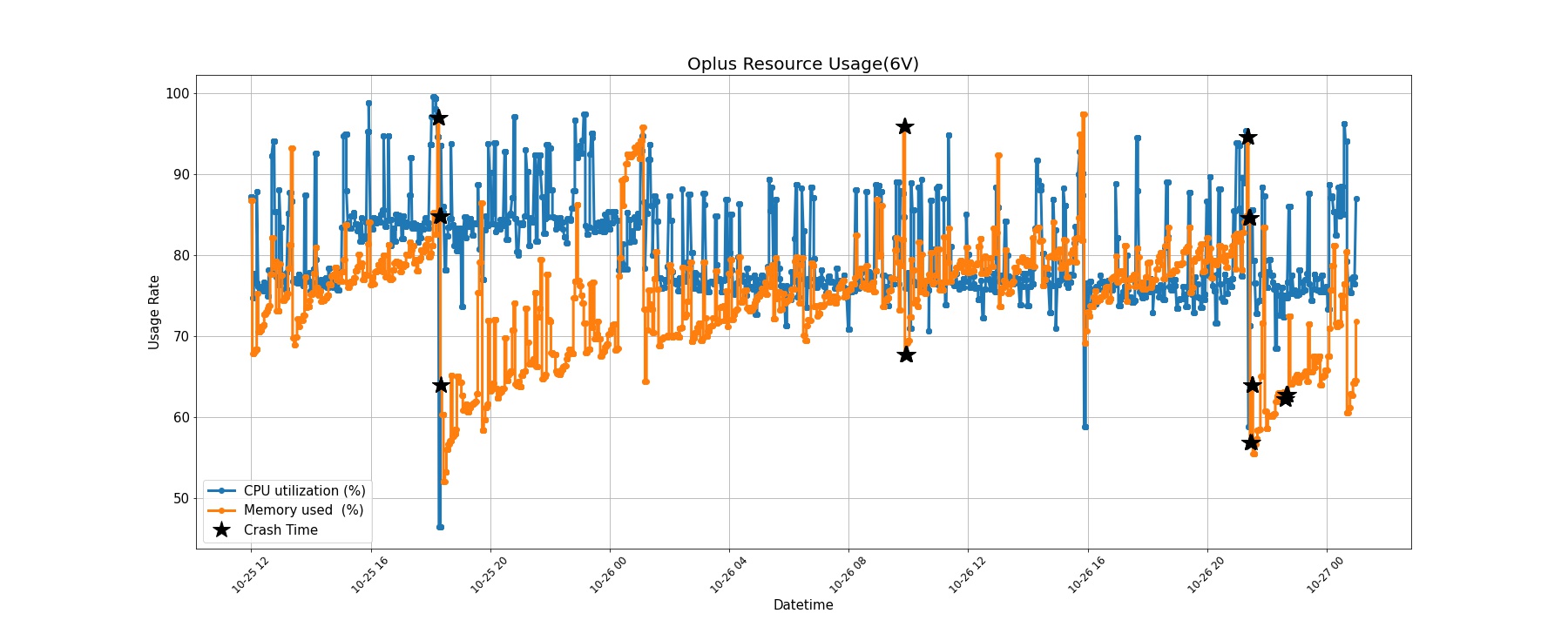


Figure 15. Oplus資源使用率與異常時間標記

對於真實的異常時間是以一固定寬度的窗口標記，滑動窗口內的資料我們採用local normalization化方法[6]，以此方法來顯現出突如其來的資料變化，如所示。

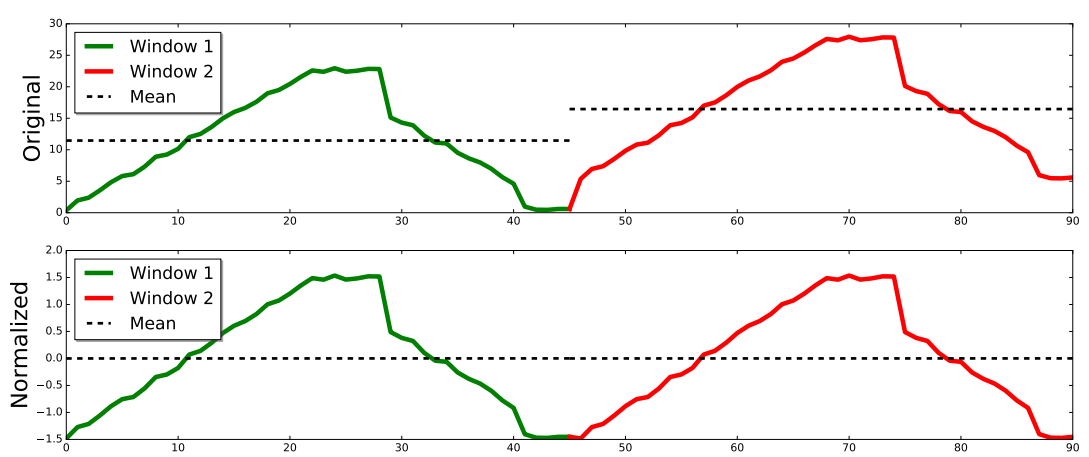


Figure 16. 平均漂移的影響圖[6]

在實際的二元數字標記上我們希望當檢測模型遇到異常的前兆時發出警告。因此，資料集在異常區間內都標記為1，其餘正常資料我們標記為0，如此一來窗口內只要出現一個異常點的話即可認定此窗口可能會發生異常。標記狀況如Figure 17所示。

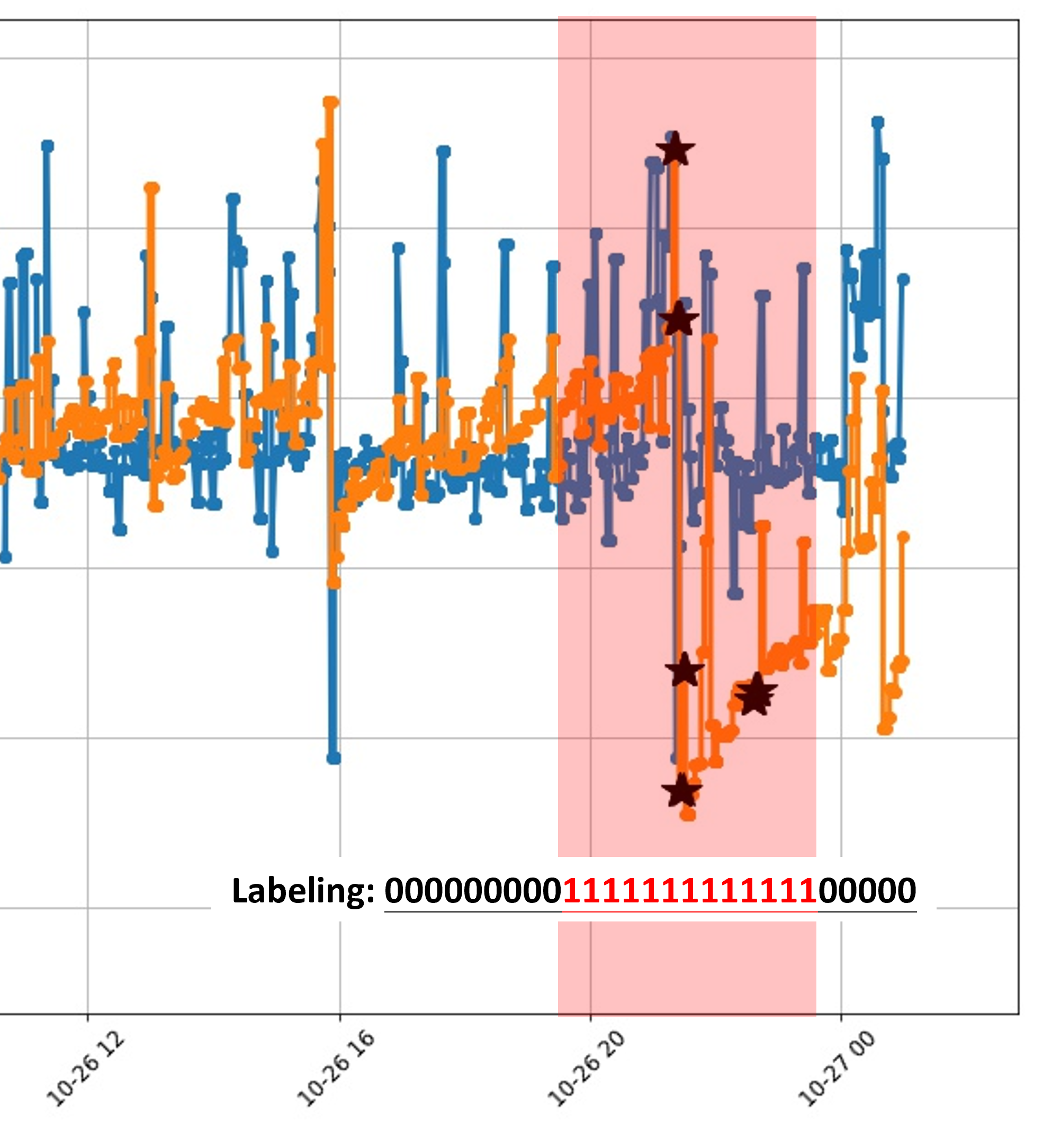


Figure 17. 異常窗口的標記方法

## 3.3 建立深度學習訓練環境

本文使用Anaconda[19]建立虛擬環境，使用CUDA[20]以及cudnn[21]硬體加速訓練流程。我們使用Pytorch作為主要的深度學習框架

## 3.4 Variational simhash algorithm (VSH)

Generally speaking, the weighting method of simhash uses the TF-IDF algorithm as its weighting value. Its method is to count the frequency of word occurrence, suitable for text comparison. But in code, unreserved words appear very frequently but are not very important. Suppose we obtain the weights according to the original method. In that case, the weight of the non-reserved words will be greater than the weight of the reserved words, which will make the code similarity inaccurate, so the original method is not suitable for the code. Therefore, this study proposed a variational autoencoder [49] model suitable for code similarity comparison, called the variational simhash algorithm, and Figure 17 gives its algorithm. This study will first train a set of variational autoencoder (VAE) to give weights that are reserved words greater than symbols and symbols greater than non-reserved words. After that, convert all reserved words, symbols, and non-reserved words in the code to be compared to an n-bit vector via word2vec [50]. The weights are then directly mapped to an m-bit vector via the VAE. Compared with user-defined weights used in the simhash algorithm, the proposed variational simhash algorithm can provide weights much closer to the normal distribution. Since the user-defined weights don't follow any protocol or regulation, the traditional simhash algorithm cannot provide the appropriate weight for each corresponding word. Suppose that a word doesn't exist in the list of defined words before. The traditional algorithm cannot give it appropriate weight. On the contrary, the proposed approach can assign a proper weight based on the VAE's inference.

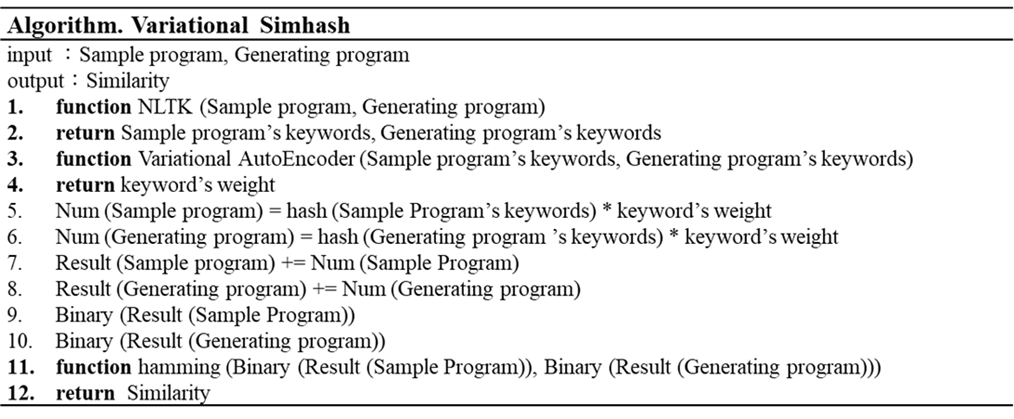


Figure 19. Variational simhash algorithm

This study uses VAE to map weights directly. First, convert a word into a 9-dimensional vector through word2vec, as shown in Figure 18. The input layer is this vector. The Encoder has two layers, the first layer has seven neurons, and the second layer has five neurons. We then reduce the vector to a 3-dimensional vector by the mean and standard deviation, and the Decoder is responsible for restoring the data dimension to the original dimension. Then go back to the word via word2vec. This study found 550 codes from GitHub, of which 500 are training data, 40 are validation data, and 10 are test data. The loss function is MSE, the activation function is ReLU, and the optimizer is Adam. Figure 19 shows the architecture diagram.

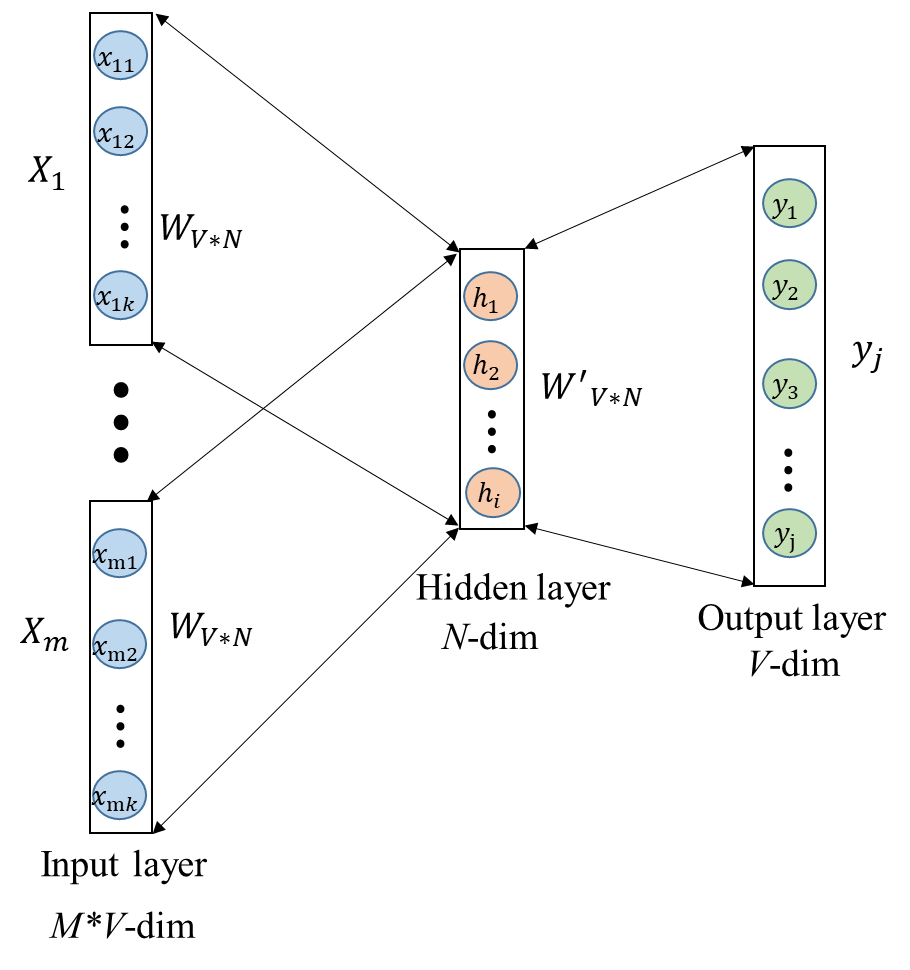


Figure 20. word2vec architecture

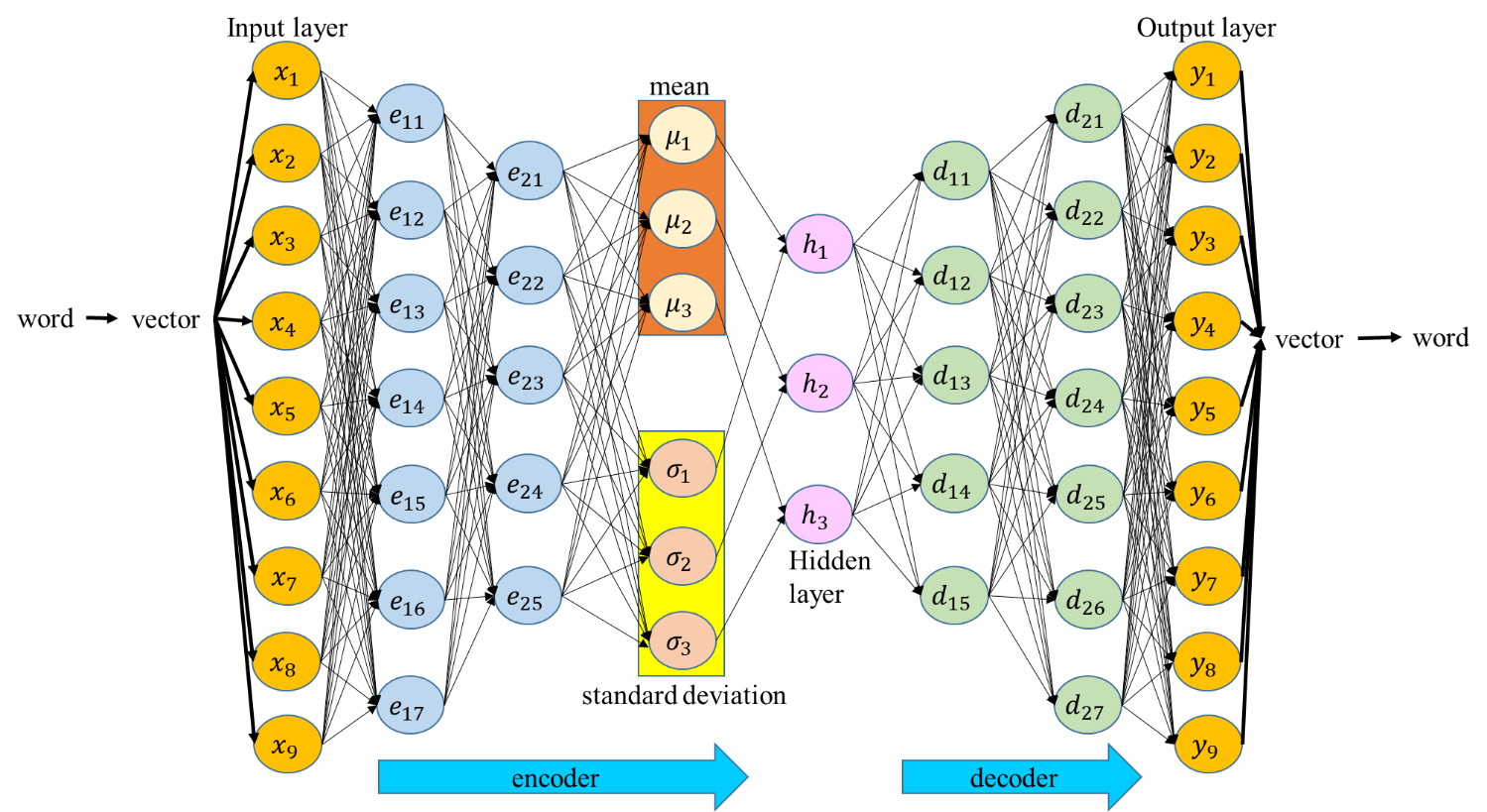


Figure 21. Variational AutoEncoder (VAE) architecture

After reducing to a three-dimensional vector, VAE substitutes the data into Eq. (21) to obtain the weight value, where *i* is the number of neurons in the hidden layer, represents the value of the i-th neuron in the hidden layer, and stands for the weight value. This study has carried out an example with two simple program codes, Test1 [51] and Test2 [52]. It turned out that the simhash algorithm gave 68% of the code similarity between them. The variational simhash algorithm, by contrast, inferred it to be 90%.

(21)

## 3.5 Piecewise longest common subsequence (PLCS)

This study proposed a new effective LCS-like method to check multimedia information's consistency rapidly. After converting the execution result of each program into a string of ASCII code or binary code, we use LCS to compute the conformity of the sample program and the generated program execution results. This study found that when the length of a string of ASCII or binary code is very long, it takes a long time to finish the conformity check. Technically speaking, supposed two strings with the length of n individually, LCS will spend times of comparisons to check the conformity between them. This study has proposed an improved LCS algorithm called the piecewise longest common subsequence (PLCS) to shorten the conformity check, as shown in Figure 20. PLCS can use a deep neural network (DNN) [53] to predict the appropriate segmented length of a string of ASCII or binary code. First, it converts the execution result into a string ASCII or binary code, breaking it into several segments where a segment has a fixed length. After that, it uses the LCS algorithm to perform a conformity check segment by segment. After the algorithm completes the LCS operation on each segment, it will empty the memory allocated for the calculation. The algorithm will return only the LCS result of the segment as well. Finally, we add the LCS results of the segments to get the final LCS result. Supposed the length of the two strings is n, the algorithm derives every segment with k characters to perform PLCS. The PLCS will spend times of the comparison. The number of comparisons used in the proposed approach, PLCS, is much less than the traditional method LCS. The operation of PLCS is faster than that of LCS because a small amount of memory is allocated for a single segment computing to speed up the conformity operation. In theory, the data type or the length of the string affects how long the segment length set in the string should be. Therefore, this study employs a DNN model to predict how many characters combine a segment to infer segment lengths for long sequences as the most suitable way to compute PLCS quickly.

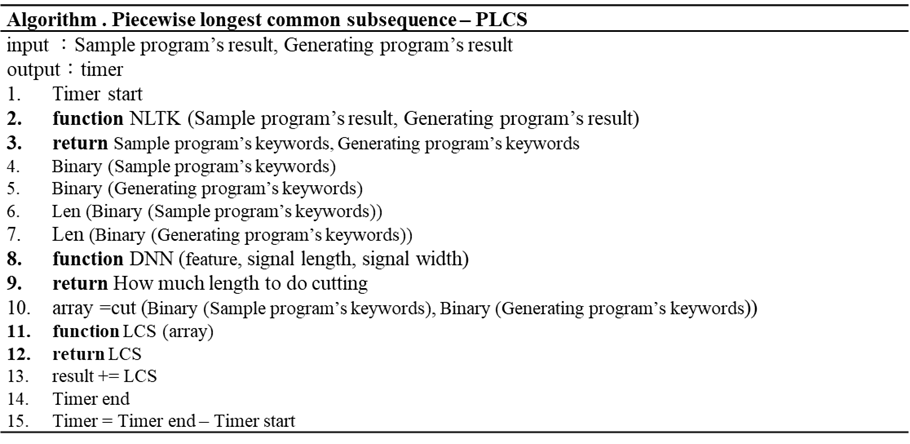


Figure 22. PLCS algorithm

This study employs a deep neural network (DNN) model with a softmax function to predict the length of a segment, as shown in Figure 21. In Figure 21, the symbol from O1 to O5 represents the different lengths of a segment, and we specify these symbols to different lengths of a segment, as listed in Table 2. The input layer contains three parts: feature and the length and width of the signal. The loss function is the sum of squared errors (SSE), the activation function is the rectified linear unit (ReLU), and the optimizer is the adaptative gradient (AdaGrad). The feature input is a one-dimensional vector with a length of 784 elements. The data set for training a DNN model consists of 100 input vectors and their corresponding 100 output labels where the model performs segmentation results of different lengths.

Regarding data allocation for training a DNN model, there are 90 vectors as training data, five validation data, and five test data. There are two hidden layers in DNN. The number of neurons in the first hidden layer is 30 and in the second 40. If the input signal is text, the feature value of the input layer adopts Doc2Vec [54] to convert the text into a vector, and the signal length is the original length of the sentence in which the signal width is 1. If the input is an image, users can use VGG16 [55] to capture image features as an input signal. The signal length is the original length of the image, and the signal width is the image's original width. If the input is a voice signal, users can use the Python package librosa.display.waveplot to convert it into the waveform of an image as an input signal. The training process of a voice input will do the same task as the image input process mentioned above. If the input is a movie, users can use the Yolo v3 [56] model to track the object's motion and convert the track of motion into a displacement image as an input signal. The training process of a move input will do the same task as the image input process as mentioned above. Figure 22 shows the loss curve during the training phase of a DNN model.

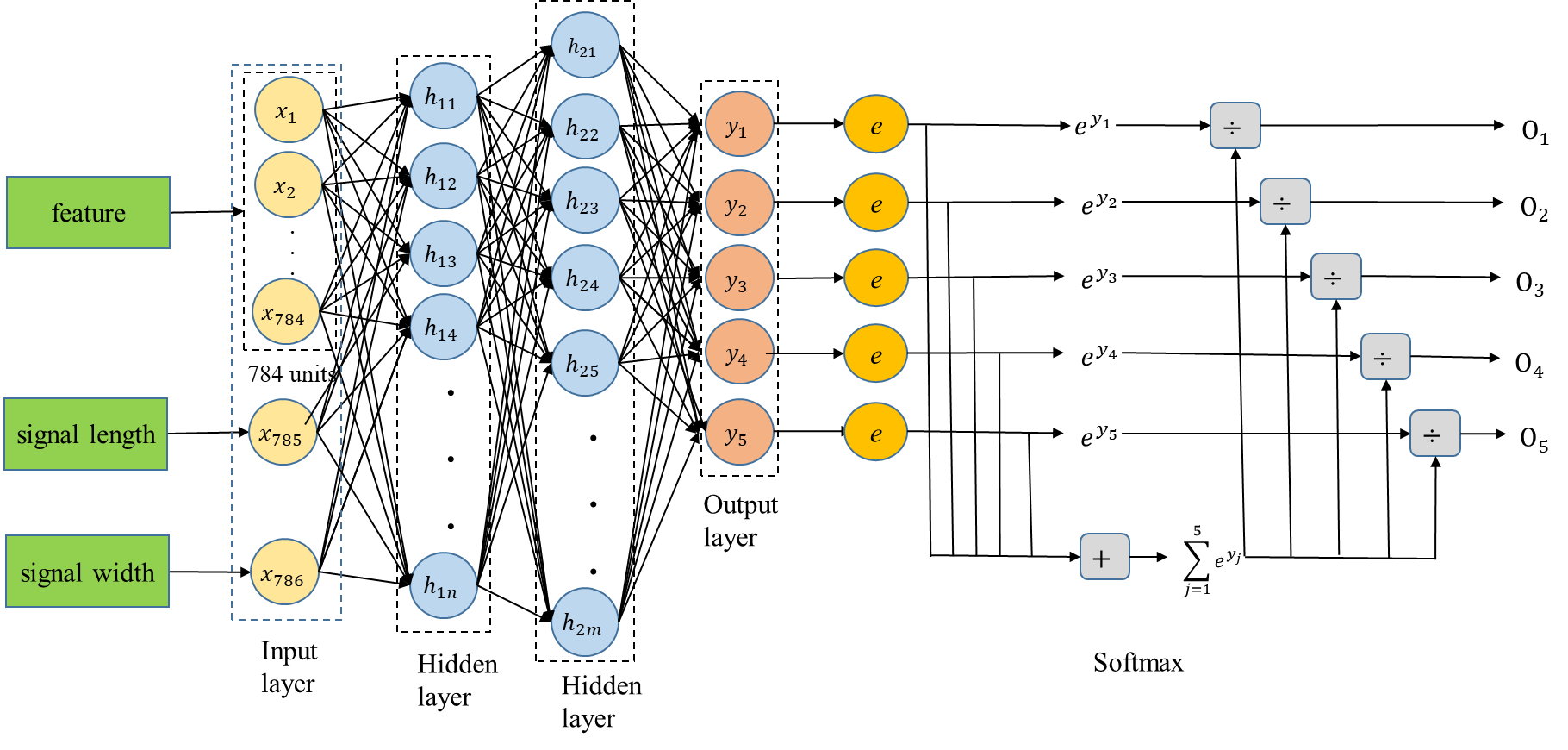


Figure 23. DNN estimate the length of a segment

Table 2. The Length of a Segment (Unit: bit)

|  |  |  |
| --- | --- | --- |
| Symbol | One hot encoding | Length of a segment |
| O1 | 000012 | 500 |
| O2 | 000102 | 1000 |
| O3 | 001002 | 5000 |
| O4 | 010002 | 10000 |
| O5 | 100002 | 50000 |

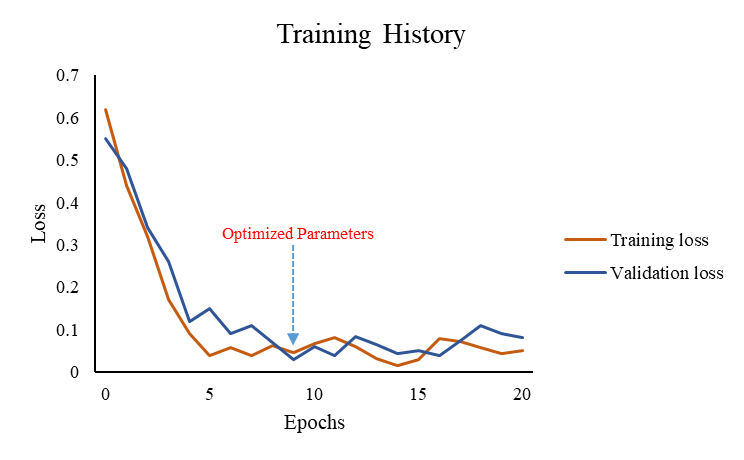


Figure 24. Loss curve during DNN training phase

This study uses the execution results of four sample programs and the programs they generate for testing. The execution results are article text [57], graphic image [58], speech signal [59] and video signal [60], respectively. Table 3 shows the performance evaluation between LCS and PLCS. The LCS algorithm takes 659.50 seconds on average for the conformity check of execution results. In contrast, the PLCS algorithm is 173.25 seconds.

Table 3. Performance evaluation between LCS and PLCS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Case | Predicted segment length (bit) | Number of comparisons using LCS | LCS execution time (sec.) | Number of comparisons using PLCS | PLCS execution time (sec.) |
| Article text | 500 | 485,809 | 0.006 | 288,809 | 0.0017 |
| Graphic image | 1000 | 1,227,241,024 | 298 | 35,001,024 | 78 |
| Voice signal | 10000 | 21,278,640,384 | 1,397 | 1,434,480,384 | 431 |
| Video signal | 5000 | 6,430,917,249 | 943 | 400,037,249 | 184 |
| Average | 4125 | 7,234,321,117 | 659.50 | 467,451,866.5 | 173.25 |

## 3.6 Graphical User Interface

Graphical User Interface, referred to as GUI. GUI is a kind of human-machine interface. It is an interface display format for human-computer communication. It allows users to manipulate icons or menu options on the screen using an input device such as a mouse. Used to select commands, call files, start programs, or perform some other daily tasks. Compared to a character interface where you enter text or character commands through the keyboard to complete routine tasks. Graphical user interfaces have many advantages. GUI is composed of windows, drop-down menus, dialog boxes and their corresponding control mechanisms. It is standardized across modern applications that the same operation is always done the same way. In GUI, what the user sees and operates are all graphical objects, and the technology of computer graphics is applied.

This study uses PyQt5 to build the interface, the main screen as shown in Figure 23. In Figure 23, it can be seen that it is divided into 3 blocks, namely Keyword Input, Code Transformation, and Check. After the user input a sentence in the first block, the keyword and the corresponding sample program can be obtained. In the second block, users can choose to use GPT-2, MASS or BART for code transformation. In the third block, users can choose to use Simhash or Variational Simhash for code similarity check. Then, the user can choose to use LCS or PLCS to check the consistency of the execution results.

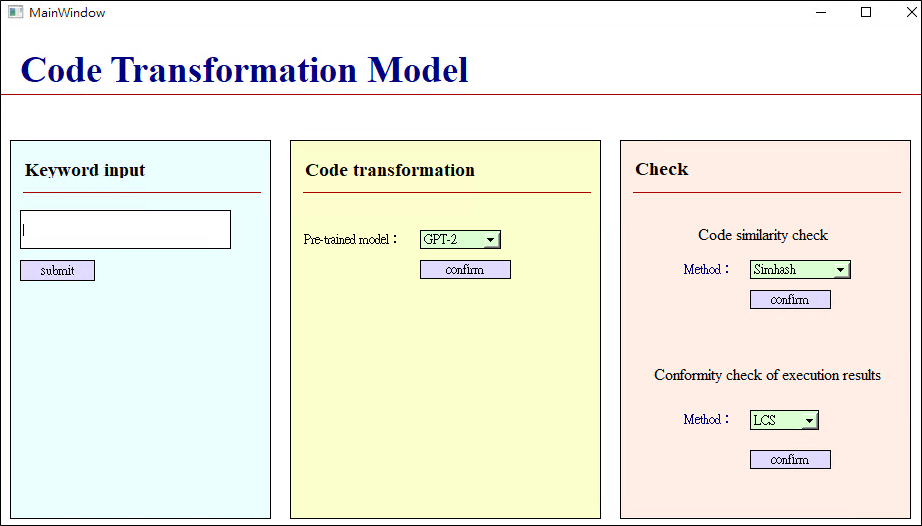


Figure 25. Main screen of interface

# Chapter 4. Experimental Results and Discussion

## 4.1 Experimental environment

This study uses fast model training on an advanced GPU cluster architecture to reduce the processing time spent on traditional CPU training models, as shown in Figure 24. NLTK performed sentence segmentation and keyword searches to find the corresponding sample programs, and then users fed those programs into code transformation models GPT-2, MASS, and BART. The variational simhash algorithm checks for code similarity. The Piecewise Longest Common Subsequence algorithm checks the consistency of the execution results of two different programs. Use LIME to interpret the model. Users can build all of the above tools in a cloud environment to execute most applications and generate programs. Therefore, this study uses open source packages to establish the operating environment, as listed in Table 4.



Figure 26. GPU workstation cluster

Table 4. Open-source package

|  |  |
| --- | --- |
| 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 | − |

## 4.2 Experimental design

We performed four experiments in this section. Experiment 1 has 4 example sentences, and each sentence will select the keyword and then use the keyword to retrieve the sample program from the semantic database. The second experiment was to generate 100 programs separately from each sample program. Then check the code similarity between the sample program and the generated program, and verify whether the execution results of the generated program and the sample program are consistent. And analyze the performance of the generated program. The third experiment is to analyze the execution speed of the whole system. Experiment 4 explains the model.

This study established a semantic database for the experiment. The keywords, example program names, example program paths, generated model paths, and other tables in the database created by XAMPP are shown in Figure 25.

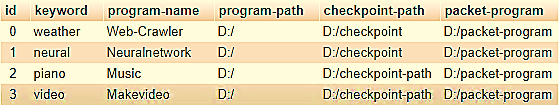


Figure 27. Table of four sample programs

## 4.3 Experimental results

**4.3.1 Experiment 1**

In Experiment 1, NLTK will be used to segment words from four input example sentences and select the appropriate keywords accordingly. Experiment 1 adopted four example sentences, as listed in Table 5. The results of word segmentation using NLTK have shown in Figure 26.

Table 5. Example sentences

|  |  |
| --- | --- |
| Case | Sentence content |
| Example 1 | The weather is very good today, I want to know the traffic flow. |
| Example 2 | Fit approximate equations through neural network. |
| Example 3 | I want to listen to piano music and relax. |
| Example 4 | I want to turn the photo into a video for viewing, and recall it. |

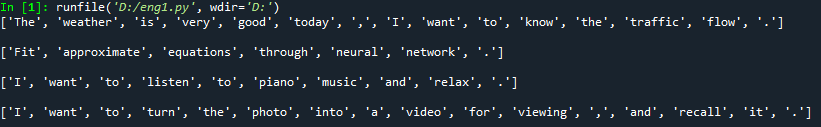


Figure 28. Screenshot of NLTK word segmentation

The keywords have the corresponding sample programs precisely found and pick-up from the semantic database where the corresponding sample programs have entitled Web-Crawler, Neuralnetwork, Music, and Makevideo, as listed in Table 6. The sample programs in this study are all obtained from Github [61]. The sample program in Example 1 is related to web crawlers [62], and the corresponding keywords are weather and traffic. Sample program 1 is to grab the corresponding data on the Internet, get the weather forecast from the weather center, and automatically assign the traffic jam spots on Google Maps. Next, in the example program of Example 2, the corresponding keyword is "equation, neural, network" [63] related to neural network applications. The primary purpose of the example program is to find an approximate equation by training a neural network. Third, in the sample program of Example 3, the corresponding keywords are piano and music, which is related to the program that generates music [64]. The webcam programming goal is to generate a short piece of piano music automatically. Finally, in Example 4, the keywords corresponding to the sample program are photo and video. This program can convert photos into videos for users to watch [65].

Table 6. The list of example programs in Experiment 1

|  |  |  |
| --- | --- | --- |
| Case | Extracted keywords from example sentence | Sample program |
| Example 1 | weather, traffic | Web-Crawler |
| Example 2 | equations, neural, network | Neuralnetwork |
| Example 3 | piano, music | Music |
| Example 4 | photo, video | Makevideo |

**4.3.2 Experiment 2**

Experiment 2 with four sample programs implements in a single GPU workstation. This experiment first imported four sample programs into GPT-2, MASS, and BART to generate the preliminary programs. In Appendix, we have demonstrated a few samples of the generated preliminary programs. After each sample program generates 100 preliminary programs, the next is to check the code similarity using the variational simhash (VSH) algorithm. We set a qualification level with the pass ratio of code similarity greater than or equal to 90%. Figures. 27 to 30 show the pass ratio of the generated preliminary programs. We have selected some of them with a higher pass ratio (≥ 90%) called qualified programs.

|  |  |
| --- | --- |
|  |  |
| Figure 29. The pass ratio of the preliminary programs in Example 1 | Figure 30. The pass ratio of the preliminary programs in Example 2 |

|  |  |
| --- | --- |
|  |  |
| Figure 31. The pass ratio of the preliminary programs in Example 3 | Figure 32. The pass ratio of the preliminary programs in Example 4 |

|  |  |
| --- | --- |
| 範例程式1 結果.PNG | 範例2 |
| (a) Execution result of sample program 1 | (a) Execution result of sample program 2 |
| 英文生成程式1 結果.PNG | 生成2 |
| (b) Execution result of newly generated program 1 | (b) Execution result of newly generated program 2 |
| Figure 33. Execution result of Example 1 | Figure 34. Execution result of Example 2 |

|  |  |
| --- | --- |
| ee2 |  |
| (a) Execution result of sample program 3 | (a) Execution result of sample program 4 |
| 33333333333333333333333333 |  |
| (b) Execution result of newly generated program 3 | (b) Execution result of newly generated program 4 |
| Figure 35. Execution result of Example 3 | Figure 36. Execution result of Example 4 |

The third is to compile every qualified program. Once any program has complied successfully, we execute that program immediately. As shown in Figures 31 to 34, PLCS will check the results of the successfully executed programs, as listed in Table 7. This PLCS is to find the conformity between the execution result of the sample program and the qualified program. The one with the highest compliance has been chosen and called the pocket program.

Table 7. PLCS conformity according to the number of identical codes (unit: %)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Case  Subject | Example 1 | | | Example 2 | | | Example 3 | | | Example 4 | | | |
| GPT-2 | MASS | BART | GPT-2 | MASS | BART | GPT-2 | MASS | BART | GPT-2 | MASS | BART |
| Sample program | 62 | 62 | 62 | 35218 | 35218 | 35218 | 162964 | 162964 | 162964 | 68087 | 68087 | 68087 | |
| Generated program | 62 | 63 | 62 | 36513 | 36102 | 35783 | 188218 | 179341 | 179376 | 66537 | 66983 | 67210 | |
| Identical codes | 61 | 61 | 61 | 35218 | 35195 | 34218 | 161998 | 160301 | 163911 | 65894 | 65912 | 66548 | |
| PLCS conformity | 98.38 | 97.60 | 98.38 | 98.19 | 98.69 | 96.3 | 92.25 | 93.65 | 95.75 | 97.89 | 97.59 | 98.37 | |

Finally, we have evaluated the performance of the proposed approaches, including VSH and PLCS algorithms, according to the execution result of sample programs and their respective pocket programs, as listed in Tables 8 and 9. Table 8 shows the number of code lines reduced between the sample program and the pocket program in four cases where the minimum number of code lines either in the sample program or the pocket program could be out of GPT-2, MASS, or BART. The proposed approach can reduce the number of code lines by 28.22% and program execution time by 30.98% on average. As a result, the proposed approach in this study can outperform the previous method published in 2021 [11].

Table 8. Number of code lines reduction

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case  Subject | Example 1 | Example 2 | Example 3 | Example 4 |
| Sample program | 291 | 152 | 174 | 147 |
| Pocket program | 169\* (174^) | 123\* (128^) | 137# (146^) | 102\* (111^) |
| Reduction ratio (%) | 41.92\* (40.34^) | 19.08\* (15.78^) | 21.26# (16.09^) | 30.61\* (24.48^) |
| Average reduction ratio (%) | 28.22 (27.21^) | | | |

p.s. abbreviated symbol ^: GPT-2, #: MASS, and \*: BART and parenthesis () indicating the minimum number of code lines of a sample program

Table 9. Program execution time reduction (unit: second)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case  Subject | Example 1 | Example 2 | Example 3 | Example 4 |
| Sample program | 8.35 | 10.57 | 14.58 | 12.43 |
| Pocket program | 6.04\* (6.97^) | 6.89\* (7.13^) | 10.56# (11.71^) | 7.97\* (8.92^) |
| Reduction ratio (%) | 27.66\* (16.59^) | 34.81\* (32.54^) | 27.57# (19.68^) | 35.88\* (28.23^) |
| Average reduction ratio (%) | 30.98 (24.62^) | | | |

p.s. abbreviated symbol ^: GPT-2, #: MASS, and \*: BART and parenthesis () indicating the minimum execution time of a sample program

**4.3.3 Experiment 3**

In this study, the proposed method makes the generated programs produced more efficient and increases the system's speed. The first part is to improve code similarity comparison using the variational simhash (VSH) algorithm that can reduce the number of qualified programs. Reducing the number of qualified programs deducts the time required to compile all qualified programs. Compared with simhash (SH) algorithm, the proposed one obtained fewer qualified programs, as shown in Table 10. Table 10 shows that the reduction ratio of the number of qualified programs is up to 22.08%.

Table 10. Comparison of the number of qualified programs produced

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Method | Example 1 | | | Example 2 | | | Example 3 | | | Example 4 | | | |
| GPT-2 | MASS | BART | GPT-2 | MASS | BART | GPT-2 | MASS | BART | GPT-2 | MASS | BART |
| SH | 23 | 34 | 41 | 25 | 29 | 32 | 28 | 37 | 42 | 31 | 36 | 40 | |
| VSH | 19 | 27 | 33 | 19 | 21 | 23 | 22 | 29 | 32 | 25 | 29 | 31 | |
| Reduction ratio (%) | 17.39 | 20.58 | 19.51 | 24.00 | 27.58 | 28.12 | 21.43 | 21.62 | 23.81 | 19.35 | 19.44 | 22.50 | |
| Average reduction ratio (%) | 22.11 | | | | | | | | | | | | |

Next, the proposed PLCS can perform the conformity check of the program execution results faster than the traditional LCS, as shown in Table 11. Therefore, it makes the system run rapidly. In Table 11, users pick up the best-performing pocket programs generated from GPT-2, MASS, or BART in Examples 1, 2, 3, and 4. Then users compare the conformity check using LCS and PLCS according to the number of string comparisons and its execution time. As a result, PLCS can reduce the number of character comparisons by 21.18% and the execution time shortened by 23.01%.

Table 11. Comparison of the conformity check

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Case | The number of string comparison in LCS | The number of string comparison in PLCS | Reduction ratio (%) | Time for string comparison in LCS (second ) | Time for string comparison in PLCS (second ) | Reduction ratio (%) |
| Example 1 | 3,721 | 3,721 | 0 | 0.0030 | 0.0030 | 0 |
| Example 2 | 1,240,307,524 | 937,040,324 | 24.45 | 392 | 279 | 28.82 |
| Example 3 | 26,557,265,296 | 17,808,785,296 | 32.94 | 7839 | 5,268 | 32.79 |
| Example 4 | 4,635,839,569 | 3,368,007,569 | 27.34 | 894 | 622 | 30.42 |
| Average | 8,108,354,027.5 | 5,528,459,227.5 | 21.18 | 2,281.25 | 1542.25 | 23.01 |

Tables 10 and 11 confirm that the proposed method improves the efficiency of producing the generated programs and the execution speed of the conformity check. Then, users can calculate the entire process's execution time in Eq. (22), where represents the time of selecting keywords after word segmentation using NLTK, stands for the time of searching the corresponding sample program in semantic database, is the time of producing the newly generated program from the code transformation models, denotes the time for checking code similarity, expresses the time for compiling all qualified programs, indicates the time of the consistency check of execution result, and evinces the time to execute the pocket program. Finally, users must use the code transformation model to estimate the time taken for the entire process of code transformation, as listed in Table 12. In contrast, the previous work [11] employed GPT-2 model, simhash algorithm, and LCS algorithm. The proposed one in this study uses GPT-2, MASS, or BART models, variational simhash algorithm, and PLCS algorithm. As a result, the proposed approach outperforms the method mentioned in the previous work, increasing the speed up to 1.27 times.

(22)

Table 12. Execution time of the entire process

|  |  |  |  |
| --- | --- | --- | --- |
| Case | The previous method (second) | The proposed approach (second) | Speedup factor |
| Example 1 | 2635.34 | 2633.62 | 1.00 |
| Example 2 | 3045.50 | 2922.02 | 1.04 |
| Example 3 | 10478.24 | 5655.76 | 1.46 |
| Example 4 | 3570.98 | 3098.50 | 1.13 |
| Average | 4,932.52 | 3,577.48 | 1.27 |

**4.3.4 Experiment 4**

This study uses LIME to explain the decision-making from AI models or algorithms such as GPT-2, MASS, BART, simhash, variational simhash, LCS, and PLCS. First, users applied LIME to interpret the outcomes of the decisions made from three pre-trained code transformation models, GPT-2, MASS, and BART. Given sample program 1, GPT-2, MASS, or BART produced the newly generated programs and then sent them into LIME to obtain the explainable results, as shown in Figures 35 to 37. The results show the effect of each line of the program and its probability of being generated. It shows that pre-trained code transformation models can decide what code should not be generated, thus making the code transformation process more efficient. As a result, there is no difference in the code transformation results among the three models mentioned above.

|  |  |
| --- | --- |
| GPT2 | MASS2 |
| Figure 37. LIME explains the results produced by GPT-2 | Figure 38. LIME explains the results produced by MASS |

|  |  |
| --- | --- |
| B2 |  |
| Figure 39. LIME explains the results produced by BART |  |

Next, users applied LIME to explain the decision-making from the algorithms of code similarity check, both simhash and variational simhash algorithms. Given the preliminary programs and the corresponding sample program, simhash or variational simhash produced the newly qualified programs and then sent them into LIME to obtain the explainable results, as shown in Figures 38 to 39. The weights of the words in a code line affect the result of the code similarity check. Finally, given the qualified programs and the corresponding sample program, LCS and PLCS produced the comparison of the execution results of the sample program and the pocket program. They then sent them into LIME to obtain the explainable results, as shown in Figures 40 and 41. Consequently, the length of ASCII code or binary code affects the execution time significantly.

|  |  |
| --- | --- |
| 111 | 222 |
| Figure 40. LIME explains the results produced by simhash | Figure 41. LIME explains the results produced by variational simhash |

|  |  |
| --- | --- |
| 333 | 444 |
| Figure 42. LIME explains the results produced by LCS | Figure 43. LIME explains the results produced by PLCS |

This study uses LIME to explain the AI model or algorithm decision-making and let people learn how the system works out to optimize the algorithm and increase the overall efficiency. This study complies consistently with the European Parliament-issued Ethics Guidelines for Trustworthy AI, proving that this study is trustworthy.

**4.4 Graphical User Interface**

This study uses PyQt5 to build the main screen interface, as shown in Figure 23. In Figure 23, it can be seen that it is divided into 3 blocks, namely Keyword Input, Code Transformation, and Check. In the first block, the user enters a sentence, then presses confirm, and the results window is displayed, as shown in Figure 42. As can be seen from the result window, the system will display the selected keywords and the corresponding sample programs. Press the return button to return to the main screen.

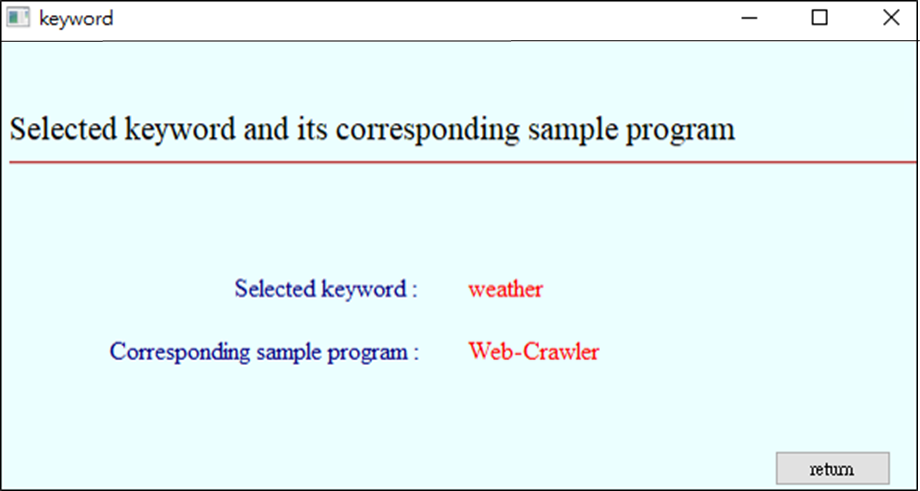


Figure 44. Keyword input result of interface

In the second block, the user can choose which model to use for code transformation. The results are shown in Figures 43-45. In Figures 43-45, the above is a hyperlink to the file of the transformation result. Press it to open the file.

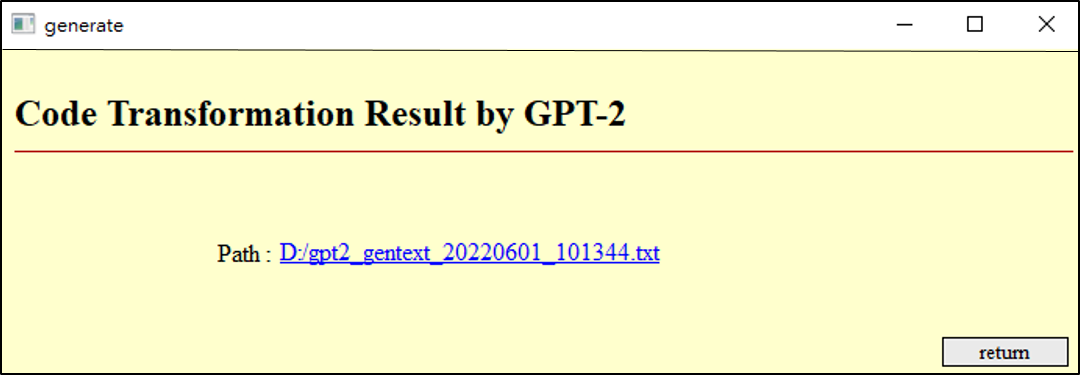
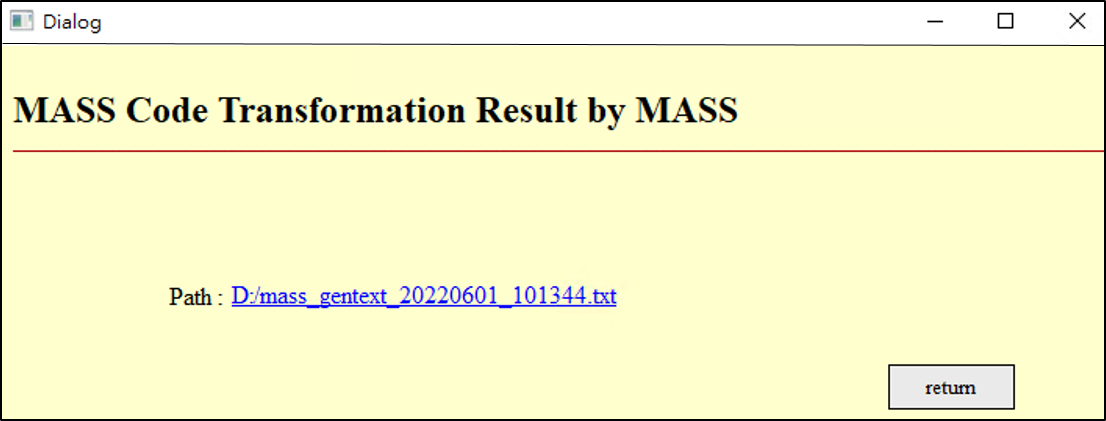


Figure 45. code transformation result by GPT-2 of interface



|  |
| --- |
| Figure 46. code transformation result by MASS of interface |



Figure 47. code transformation result by BART of interface

The third block can be divided into code similarity check and execution result consistency check. Users can choose different check methods. Figures 46-47 show the results of the code similarity check using the simhash algorithm and the variational simhash algorithm. It can be seen from the results that the first row shows the similarity between the average preliminary program and the sample program. Next, the similarity of the individual preliminary programs to the sample programs is displayed.

|  |  |
| --- | --- |
|  |  |
| Figure 48. Check result by simhash | Figure 49. Check result by simhash |

Figures 48-49 show the results of the consistency check of execution results using the LCS algorithm and the PLCS algorithm. The results show that the conformity of the qualified program and the sample program and the time to execute the test will be displayed.

|  |  |
| --- | --- |
|  |  |
| Figure 50. check result by LCS | Figure 51. check result by PLCS |

**4.5 Discussion**

It can be seen from the experimental results that the variational simhash algorithm and the piecewise longest common subsequence algorithm proposed in this study are effective. The qualified program can be reduced by 22.11%. The number of string comparisons is reduced by 21.18%, and the time to check the consistency of execution results can be reduced by 23.01%. The entire code transformation process has improved the execution speed by 1.27 times. Finally, use LIME to explain the model's decisions. In addition, the study also builds a graphical user interface, allowing users to operate the code transformation operation very conveniently.

The variational simhash algorithm in this study changes the way that gives weights in the simhash algorithm and increases the accuracy of code similarity. However, the effect may not be so good because the variational simhash algorithm is against the python code compared to other programming languages. The PLCS algorithm of this study uses way of segment length to let it be fast to compare the execution results consistently. However, the comparison time may not be the least because there are only five kinds of segment lengths in the PLCS algorithm.

# Chapter 5. Conclusion

It can be seen from the experimental results that this study improves the accuracy of code similarity comparison and reduces the time for checking the consistency of execution results. This study makes the entire code transformation process has improved the execution speed by 1.27 times. In addition, explainable AI can also be used to explain the rules by which the model operates. The above experimental results achieved the expected goal of this study. This research also builds a graphical user interface to provide users with more convenient operations.

In this study, the program's programming language is python, so if it is necessary to generate other languages, this system is not necessarily suitable. So the future can be towards generating different programming languages. In addition, this system does not store many keywords and sample programs in the semantic database. As the data of keywords and sample programs stored in the semantic database becomes larger, it must be considered that too many may be added. This data results in slow searches when searching the semantic database, resulting in poor performance. Therefore, in the future, deep learning methods can be used to optimize the search to speed up the search. The above is how the system needs to develop in the future.

# References

1. A. Bousdekis, K. Lepenioti, D. Apostolou, and G. Mentzas, ‘A Review of Data-Driven Decision-Making Methods for Industry 4.0 Maintenance Applications’, *Electronics*, vol. 10, no. 7, 2021.
2. Wu, Tong, and Jorge Ortiz, “Rlad: Time series anomaly detection through reinforcement learning and active learning”, *arXiv preprint arXiv:2104.00543* (2021).
3. D. Zha, K.-H. Lai, M. Wan, and X. Hu, “Meta-AAD: Active Anomaly Detection with Deep Reinforcement Learning”, *CoRR*, vol. abs/2009.07415, 2020.
4. Vanschoren, Joaquin, “Meta-learning”, *Automated machine learning*. Springer, Cham, 2019. 35-61.
5. C. Finn, P. Abbeel, and S. Levine, ‘Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks’, *CoRR*, vol. abs/1703.03400, 2017.
6. S. Saurav *et al.*, ‘Online Anomaly Detection with Concept Drift Adaptation Using Recurrent Neural Networks’, in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, Goa, India, 2018, pp. 78–87.
7. M. Barandas *et al.*, ‘TSFEL: Time Series Feature Extraction Library’, *SoftwareX*, vol. 11, p. 100456, 2020.
8. J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, ‘Trust region policy optimization’, in *International conference on machine learning*, 2015, pp. 1889–1897.
9. S. M. R. Arnold, P. Mahajan, D. Datta, I. Bunner, and K. S. Zarkias, ‘learn2learn: A Library for Meta-Learning Research’, *arXiv [cs.LG]*, Aug. 2020.
10. Y. Zhao, Z. Nasrullah, and Z. Li, ‘PyOD: A Python Toolbox for Scalable Outlier Detection’, *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1–7, 2019.
11. Aygun, R. Can, and A. Gokhan Yavuz. “Network Anomaly Detection with Stochastically Improved Autoencoder Based Models.” *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2017.
12. C. Zhou and R. C. Paffenroth, “Anomaly Detection with Robust Deep Autoencoders,” *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.
13. C.-W. Tien, T.-Y. Huang, P.-C. Chen, and J.-H. Wang, “Using Autoencoders for Anomaly Detection and Transfer Learning in IoT,”*Computers*, vol. 10, no. 7, p. 88, Jul. 2021.
14. H. Xu, Y. Feng, J. Chen, Z. Wang, H. Qiao, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, and D. Pei, “Unsupervised anomaly detection VIA Variational Auto-Encoder for seasonal KPIs in web applications,” *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, 2018.
15. J. Alonso, J. Torres and R. Gavaldà, "Predicting Web Server Crashes: A Case Study in Comparing Prediction Algorithms," 2009 Fifth International Conference on Autonomic and Autonomous Systems, 2009, pp. 264-269.
16. M. Farshchi, J. Schneider, I. Weber, and J. Grundy, “Metric selection and anomaly detection for cloud operations using log and metric correlation analysis,” *Journal of Systems and Software*, 137, 531-549.
17. S. Gupta and D. A. Dinesh, “Resource usage prediction of cloud workloads using deep bidirectional long short term memory networks,” *2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2017.
18. Z. Zou and J. Ai, "Online Prediction of Server Crash Based on Running Data," *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2020, pp. 7-14.
19. ‘Anaconda Software Distribution’, *Anaconda Documentation*. Anaconda Inc., 2020.
20. Nvidia, P. Vingelmann, and F. H. P. Fitzek, ‘CUDA, release: 10.2.89’. 2020.
21. S. Chetlur *et al.*, ‘cuDNN: Efficient Primitives for Deep Learning’, *CoRR*, vol. abs/1410.0759, 2014.