Privacy-preserving clustering of single-cell RNA Sequencing data on Intel SGX

by

Qingyuan Cao

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday August 25, 2021 at 4:00 PM.

Student number: 5065232
Project duration: November 2020 – August 2021
Thesis committee: Dr.ir. Z. Al-Ars, TU Delft, supervisor
Dr.ir. Z. Erkin, TU Delft

An electronic version of this thesis is available at https://repository.tudelft.nl/.

TU Delft

Acknowledgments

This thesis work could not have been accomplished without help from others.

First, I would like to thank my supervisors Dr. Zaid Al-Ars and Tanveer Ahmad for providing this wonderful and challenging project topic and continued support. I would also like to thank Mr. Ryo Asakura for his help during the development of the system. I would also like to thank Dr. Zekeriya Erkin for being a member of my graduation committee.

During this special pandemic period, I would like to thank my parents Mr. Cao Zhong and Ms. Lian Qian for their selfless care and support. Furthermore, I would like to thank all my colleagues and friends for their company and encouragement.

Contents

List of Figures

List of Tables

1 Introduction

1.1. Context

Cells are the basic units of life, enabling growth and development of organisms. Each cell has its own unique biological function. Investigations into human organs development, function, and disease depend upon accurate identification and categorization of cell types [5].

Bulk RNA-seq technologies have been widely used to study gene expression patterns at population level in the past decade [6]. The advent of single-cell RNA sequencing (scRNA-seq) provides unprecedented opportunities for exploring gene expression profile at the single-cell level [7]. In recent years, scRNA-seq has been applied to various species, especially to diverse human tissues including normal and cancer, and these studies revealed meaningful cell-to-cell gene expression variability [8]. A crucial step in analyzing scRNA-seq data is to cluster cells into sub-populations to facilitate subsequent downstream analysis [9].

Unsupervised learning is very important in the processing of multi-modal content since clustering or partitioning of data in the absence of class labels is often a requirement [10]. Unsupervised learning methods such as dimension reduction and clustering algorithms are now widely used to group cells of the same type or sub-types based on the gene expression profiles of hundreds to thousands of single cells from tumor or normal tissue [11].

Genomics privacy has been a major concern for data sharing and analysis, which becomes a barrier for precision medicine applications [12]. Lack of sufficient protection on sensitive human genome data can put individuals privacy at risk. Generally, for this type of private data, we can ensure security by encryption methods such as searchable symmetric encryption (SSE) before we transfer data to the cloud for storage and calculation [13]. However, typical practical SSE systems trade-off some aspects of security for performance. Because providing complete security in the SSE setup has large overhead. Similarly, using trusted hardware to directly implement algorithm execution and data storage is also a feasible solution. Intel includes a security module called Software Guard Extension (SGX) in its 6th generation and later CPUs. In short, SGX allows users to create secure isolation compartments (called enclaves) within the process to efficiently run algorithms and store sensitive data.

Therefore, a lot of attention has been paid on how to securely load a complex single-cell data machine learning network into a hardware platform. A series of workshops and competitions like the iDASH-Workshop has called for practical and rigorous solutions to address emerging genomics privacy challenges in biomedical data analysis for many years [14].

This project is dedicated to implementing an unsupervised learning clustering method system for processing big data applied in Intel SGX. Practical data sets are applied for the analysis of efficiency, accuracy and resource allocation of the system.

1.2. Challenges

The problems faced by the development of this project are mainly divided into two parts, the realization of the algorithm and the its application to the hardware. Whether it is supervised learning or unsupervised learning, deep learning algorithms have been widely studied and applied in clustering problems. However, deep learning algorithms often require high memory resources (computing unit resources)

due to its large number of model parameters. Moreover, user scenarios are mostly focused on pure algorithm development because deep learning algorithms development in underlying hardware often requires a large development cost. On the one hand, the lack of relevant third-party support libraries makes it necessary for developers to write algorithms from the lowest level. On the other hand, the versatility of the hardware foundation and the limitation of computing resources often make it difficult to fully implement the relevant algorithms.

In the process of developing this project, the above mentioned two challenges are also important. The unsupervised learning algorithm used in this project is the sczidesk algorithm [9]. How to develop this complex algorithm on SGX hardware is one of the biggest challenges addressed by this project. At the same time, because SGX has a unique computer memory unit. How to use this hardware unit and ensure its complete operation is also a challenge. Last but not least, how to balance accuracy, resource utilization and efficiency is a fundamental challenge throughout the implementation.

1.3. Problem statement and research questions

We formulate the research questions of our work as follows.

1. How to enable Intel SGX hardware in a specific trusted system?
2. How to develop deep learning algorithms in SGX?
3. How to ensure the security of data and the normal transmission of data?
4. How do such SGX compute pipelines compare with non-SGX state-of-the-art work?

With help of the official instruction files, both Windows and Linux based SGX enabled processes are tested in the project. For the algorithm part, by implementing some up to date supporting libraries and tools, the whole algorithm is developed from the scratch. Data security and data transmission are also solved using custom defined TCP and AES protocol. Balance between accuracy, resource utilization and efficiency are considered during the overall development process.

1.4. Thesis contributions

The contributions of the thesis can be summarized as follows:

- **Study of Intel SGX** — We have extensively analyzed the characteristics of Intel SGX in various scenarios, including deep learning, data encryption and transmission. This would serve as a reference for those interested in using Intel SGX to improve security level in their applications.
- **Development of deep learning algorithm** — The main contribution of our work is that we develop a working deep learning system using secure hardware. Moreover, several practical data sets are tested to verify its performance and accuracy. The results are comparable with state-of-the-art solutions such as CIDR, SIMD and RaceID.
- **Contribution to open-source project** — We have also made some contributions to several open-source projects by reporting issues that we encountered in during the implementation of this project. The details of this contribution are documented in Appendix A.

1.5. Thesis outline

The rest of the thesis is organized as follows:

Chapter 2 introduces the relevant background knowledge for our work. We first describe the RNA-sequence data, and then we move to the introduction of Intel SGX including its structure and workflow. AES encryption and decryption principles are also introduced.

Chapter 3 introduces some famous state-of-the-art solutions such as CIDR, SIMD, RaceID in details. The sczidesk algorithm is then introduced in principle to show how that unsupervised machine learning algorithm works.

Chapter 4 introduces the implementation of sczidesk with AES encryption and decryption in Intel SGX. Architectural details and components implementation are described for every step of the development of the system.

Chapter 5 introduces the measurement results of the algorithm in practical environment with several publicly available large single-cell data sets. First, the experimental setup will be discussed both in the Windows and Linux environment. Then, multiple sections with results and discussion are presented in order to analyze the functionality and scalability of the whole system.

Our conclusion is drawn and some possible directions for future work are proposed in **Chapter 6**.

2

Background

2.1. Single-cell analysis

In the field of cellular biology, single-cell analysis is the study of genomics, transcriptomics, proteomics, metabolomics and cell–cell interactions at the single cell level [15]. Multiple steps are involved in the pipeline of single-cell analysis, as we will introduce in the following subsections.

2.1.1. RNA sequencing

Single cell sequencing examines the sequence information from individual cells with optimized next-generation sequencing (NGS) technologies, providing a higher resolution of cellular differences and a better understanding of the function of an individual cell in the context of its micro-environment [16]. For example, in cancer, sequencing the DNA of individual cells can give information about mutations carried by small populations of cells. In development process, sequencing the RNAs expressed by individual cells can give insight into the existence and behavior of different cell types [17].

The results of single-cell RNA sequencing can be stored in an H5 file. A typical single-cell RNA sequencing H5 data file has three main groups: var, obs and uns. Figure 2.1 shows an example of how this file format looks like. The core of the single-cell sequencing data is a two-dimensional table of cell×gene. In this table, the information related to the cell of the object of X is recorded in obs, the information of attribute gene is recorded in var, and the other information is in uns.

The cell×gene two-dimensional table is the raw data for this project. An example of such tables is shown below. That slice comes from one data set that meets the 10X standard [1].

	RP11-34P13.3	FAM138A	...	AC213203.1	FAM231B
AAACCCAAGCGTATGG-1	0.0	0.0	...	0.0	0.0
AAACCCAGTCCTACAA-1	0.0	0.0	...	0.0	0.0
AAACCCATCACCTCAC-1	0.0	0.0	...	0.0	0.0
...					

The raw name represents the genes and the column names are the cells names. Numbers in this matrix are mostly zero, which means that cell has no expression at that gene point. When the value is one, it means that cell has expression at that gene point. In H5 file, data is stored as a sparse matrix. CSR (compressed sparse row format) can be used to significantly reduce storage pressure.

[1] This data is from `https://support.10xgenomics.com/single-cell-gene-expression/datasets/pbmc3k`.

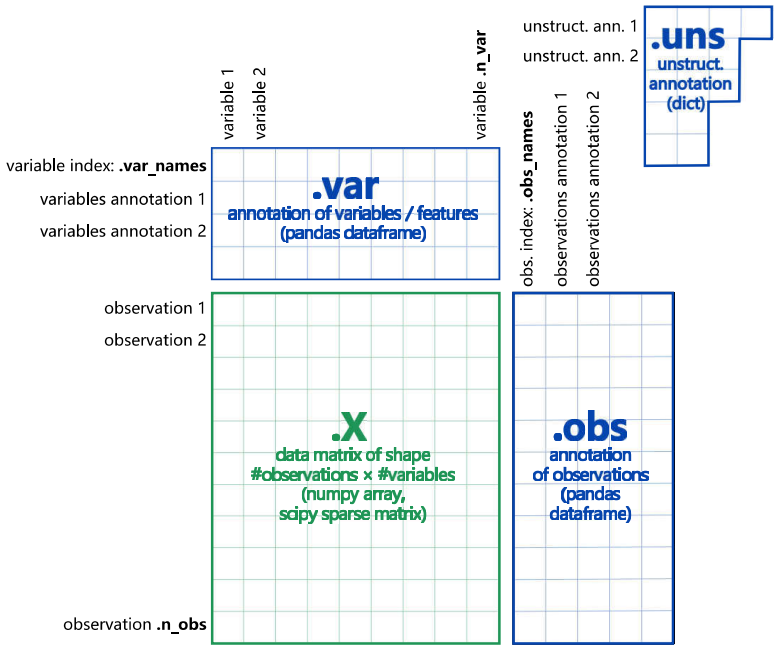


Figure 2.1: A schematic diagram of a single-cell sequencing file [1]

2.1.2. Data Prepossessing

Before useful information can be retrieved from the raw data, i.e. the output of an RNA sequencing machine, it has to be preprocessed. The standard pre-processing workflow for scRNA-seq data includes the selection and filtration of cells, data normalization and scaling, and the detection of highly variable features.

Data Filtration

One of the key challenges of single-cell RNA sequencing analysis is to ensure that only single, live cells are included in downstream analysis, as the inclusion of compromised cells inevitably affects data interpretation. A few QC (quality control) metrics commonly used by the community include the number of unique genes detected in each cell. Low-quality cells or empty droplets will often have very few genes [18]. Therefore, in our project, we filter out genes that have not been expressed by any cell and then filter out the cells without gene expression.

Data Normalization and Scaling

Considering neural network numerical optimization stability, we need to transform discrete data into continuous smooth data. Therefore, after removing unwanted cells from the data set, the next step is to normalize the data. By default, we employ a global-scaling normalization method “LogNormalize” that normalizes the feature expression measurements for each cell by the total expression and log-transforms the result.

Highly Variable Features

Studies show that focusing on several important genes in downstream analysis helps to highlight biological signal in single-cell data sets [19]. This kind of data are called highly variable features who exhibit high cell-to-cell variation in the data set. In order to balance the calculation accuracy and the resource required for the calculation, we select part of the top high variable features by the help of *Scanpy* package [1]. In that package, highly variable genes are sorted by their normalized dispersion values’ ranking.

The last step for data-prepossessing is another normalization for the output before. It scales data to unit variance and zero mean. That provides a standard data distribution for further calculation and development.

2.2. Intel SGX

Intel SGX is a technology developed to meet the needs of the trusted computing industry similarly to *ARM TrustZone*, but this time for desktop and server platforms. It is a set of safety-related instruction codes built into some modern Intel central processing units (CPUs). They allow user-level and operating system code to define private areas of memory, called enclaves. The contents of enclaves are protected and cannot be read or saved by any process other than the enclave itself, including processes running at higher privilege levels [20]. In general, an application is split into two parts when implementing Intel SGX: a trusted one and an untrusted one.

Figure 2.2 shows an ideal about how Intel SGX works. During the running of a program, the code in the unsafe part can call the SGX space through the cell gate. The protected memory space in SGX is called an enclave. Execution results in the enclave will be returned to the application. The data and code in the enclave will only be visible and used by the code in it. All requests for external direct access from outside will be rejected.

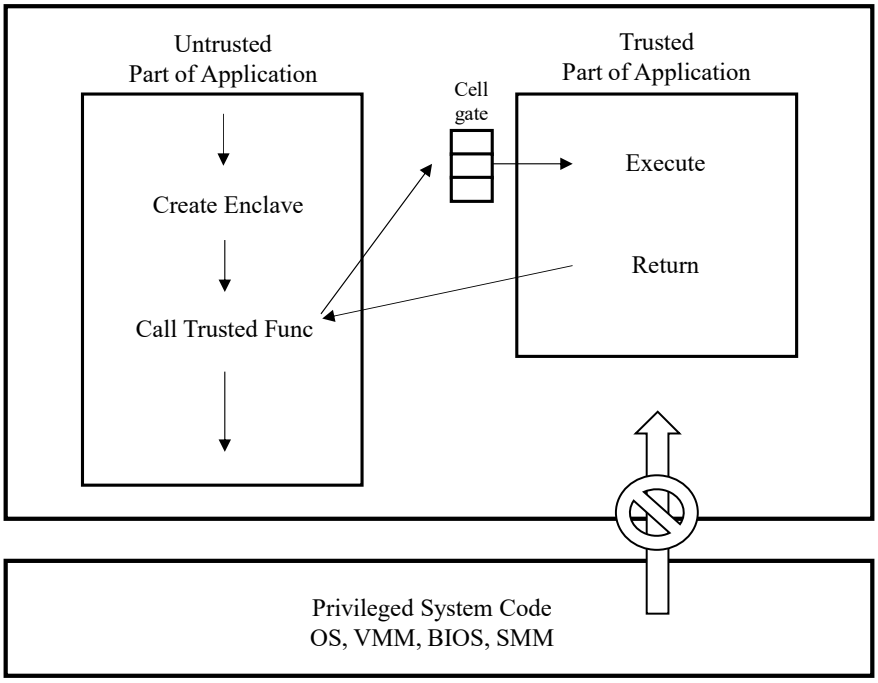


Figure 2.2: An overview for Intel SGX implementation.

2.2.1. Memory

SGX defines several new data structures to maintain the information needed of SGX security properties. The most important memory area is called EPC (Enclave Page Cache). The EPC is divided into 4 KB chunks called EPC page, and every EPC page can either be valid or invalid to store the enclave data or the SGX structure data. Enclaves are stored in the Enclave Page Cache with a specific SGX structures, including data used to security check and manage the enclave entry points.

Figure 2.3 shows an overview of how Intel SGX EPC looks like. In this storage space, the SECS (SGX Enclave Control Structure) is responsible for controlling the metadata like hash and size of the storage space. Each enclave can have one or more than one TCS (Thread Control Structure) and its corresponding SSA (Save State Area) where TCS indicates an execution point into the enclave. As SGX supports multi-threading, an enclave can have as many active threads as TCS number. Each TCS can have at least one SSA structure which is used to be responsible for the processor's state storage. At the same time, each enclave also has its own sigstruct and version array pages which provide identical information of the enclave. The whole EPC memory area is encrypted using the MEE (Memory Encryption Engine), a new and dedicated chip to ensure that all external reads outside can

only see the encrypted message.

Another important structure used in SGX memory is EPCM (Enclave Page Cache Map). It is the security metadata attached to each EPC page to store EPC page state including configuration, permissions and types. The mapping between each EPC page and EPCM entry is 1:1 and EPCM size limits the size of the EPC. Normally the maximum size of EPC is 128 MB maximum for the desktop CPU.

Figure 2.5 shows an example of a typical EPC memory structure. The EPC is located inside the PRM (Processor Reserved Memory) of the DRAM, the EPCM is a look-up table inside the CPU with enclave related data. With the help of MEE, data and code stored in PRM is encrypted to all other applications outside SGX and CPU is the only place of system to read and operate date in the enclave. Figure 2.4 shows the communication between EPC in DRAM with EPCM and MEE in CPU.

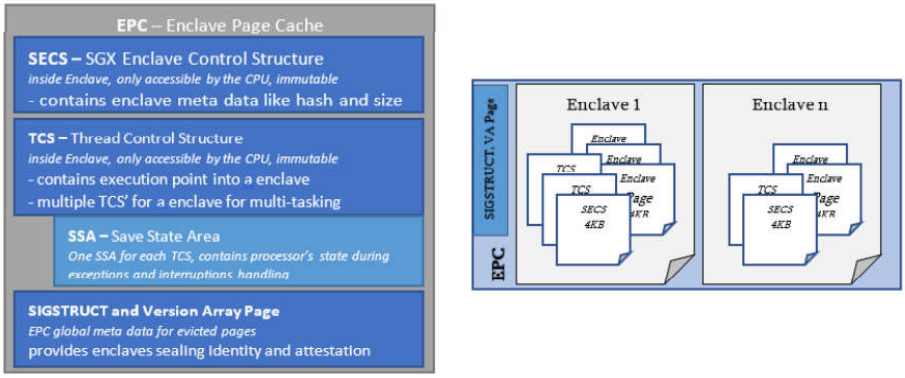


Figure 2.3: An overview for Intel SGX Enclave Page Cache [2].

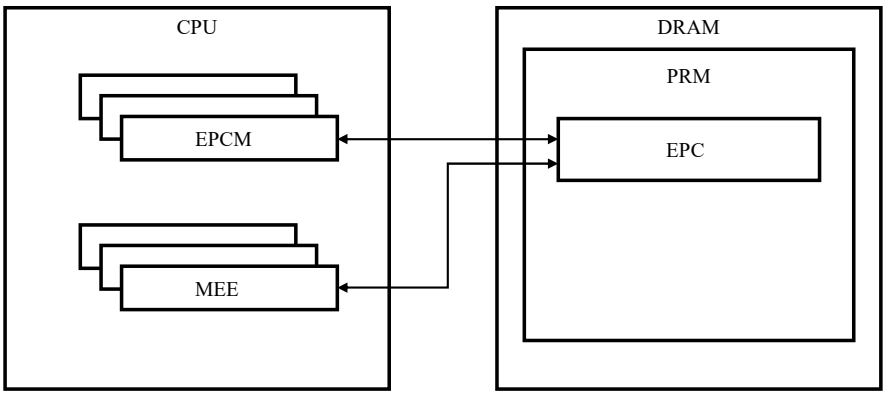


Figure 2.4: The communication between EPC in DRAM with EPCM and MEE in CPU.

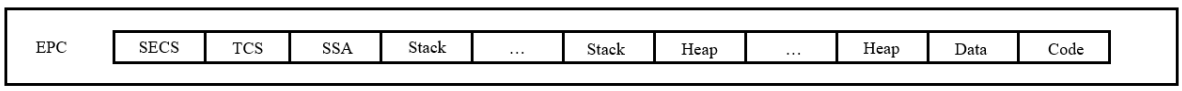


Figure 2.5: An example of a typical EPC memory structure.

2.2.2. Processor

Intel SGX defines 18 new instructions: 13 to be used by the supervisor and 5 by the user. Table 2.1 shows the list of all those instructions. They are all implemented in micro-code so that their behavior can be modified. Those instructions construct important processor operations for SGX including enclave creation, enclave entry/exit and interrupt handling.

Table 2.1: Intel SGX new instructions

Supervisor	*Description*	*User*	*Description*
EADD	Add 4k page	EENTER	Enter an enclave
EBLOCK	Block an EPC page	EEXIT	Exit an enclave
ECREATE	Create an enclave	EGETKEY	Create a cryptographic key
EDBGRD	Read data by debugger	EREPORT	Create a cryptographic report
EBDGWR	Write data by debugger	ERESUME	Re-enter an enclave
EINIT	Initialize en enclave		
ELDB	Load an EPC page as blocked		
ELDU	Load an EPC page as unblocked		
EPA	Add a version array		
EREMOVE	Remove a page from EPC		
ETRACE	Activate EBLOCK checks		
EWB	Write back/invalidate an EPC page		
EEXTEND	Measure 256 bytes		

Enclave creation and page adding play important roles in SGX working process. Figure 2.6 shows the workflow of the enclave creation and page adding. The enclave creation process starts with ECREATE. After this instruction is passed to the EPC through the CPU, the EPC page is converted to an SECS page and the structure is initialized. As part of ECREATE, the system software chooses which EPC page to use as SECS and specifies several attributes of the enclave, including the range of protected addresses that the enclave can access, operating modes (32-bit and 64-bit), the enclave, and finally whether to allow debugging access [21]. After the SECS is created, the enclave page can be added to the enclave through EADD. This involves converting blank EPC pages to REG or TCS. When EADD is called, the EPC page to be written will be associated with the SECS page provided by the input. The hardware uses the EPCM entry information to provide SGX access control for the page, and then EADD records the EPCM information in an encrypted log stored in SECS and copies 4 K bytes of data from unprotected memory to the allocated EPC page.

After adding the page to the enclave, each page is added to the measure of the enclave using the EEXTEND instruction. Each call to EEXTEND adds a header to the encrypted log indicating which area it is in. At the same time, the information of this area will be recorded after this header. After the creation and addition are complete, the initialization of enclave ends.

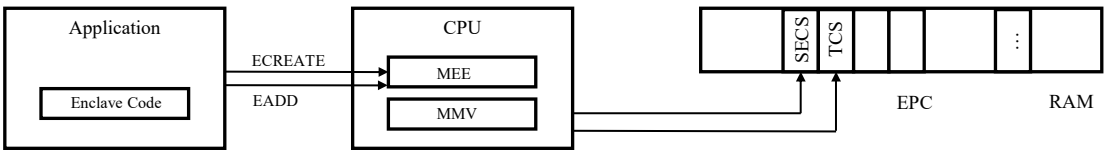


Figure 2.6: Enclave creation and pages adding process.

The Entry/Exit and interrupt operations follow similar workflow between application, CPU and enclave area. The complex processing flow and memory operation mechanism of SGX effectively protect the security of code and data, but at the same time it also gives overhead of running computations to the programs running in the enclave. Performance overhead may mainly come from two aspects: the first is the actual overhead of executing CPU instructions and accessing encrypted memory in the enclave. The second is the overhead associated with entering and exiting the enclave [22]. The specific test results will be carried out in the Chapter 5.

2.2.3. Fortanix-EDP

In this project, Fortanix EDP is used as the SGX development platform. Fortanix EDP can completely encapsulate the Rust code in the program and provide an interface for the enclave and OS system. Figure 2.7 shows the workflow of Fortanix EDP between the OS system and enclave. It provides all the necessary abstractions and interfaces to launch the enclave and take the output from the enclave and send it to the OS.

The usercall interface is implemented inside the enclave directly in Rust's standard library. Outside the enclave, the enclave-runner crate takes care of loading the enclave. Once the enclave is loaded, it provides a shim layer between the usercalls coming from the enclave and the system calls needed to talk to the outside world. Ftxsgx-runner is an executable file based on enclave-runner which should be suitable for most enclave applications [3].

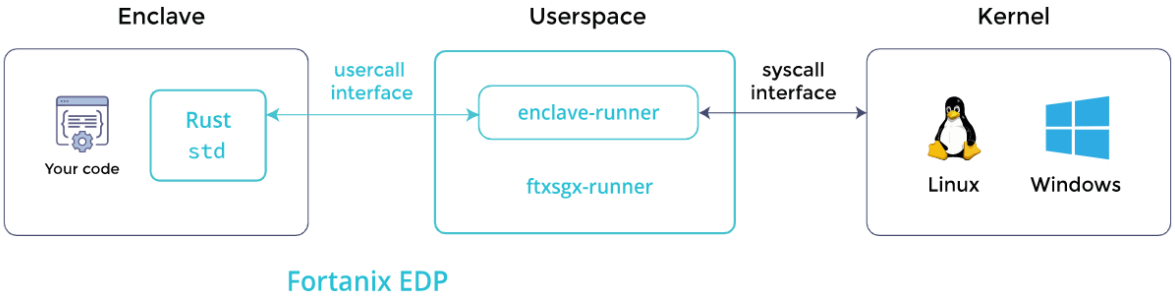


Figure 2.7: Workflow of Fortanix EDP between OS system and enclave [3].

Using Fortanix EDP with rust for project development will greatly reduce the developer's development time. The native rust code can be directly applied to Fortanix EDP to be put in the enclave. Developers can place the complete algorithm in a safe and trusted space. At the same time, The Fortanix EDP is fully integrated with the Rust compiler. Therefore, project development with Rust can become relatively convenient.

2.3. AES128 Encryption

In this project, the security of sensitive cell data is our major concern. In addition to implementing the algorithm and storing data in SGX to ensure operational security, AES128 is also used for encryption during data transmission. AES (Advanced Encryption Standard) is a commonly used symmetric key encryption method.

AES is a block cipher. The block cipher is to divide the plain text into several groups with the same length. Each time a group of data is encrypted until the entire plain text is encrypted. In the AES standard specification, the packet length can only be 128 bits, that is, each packet is 16 bytes (8 bits per byte). The length of the key can be 128 bits, 192 bits, or 256 bits. The length of the key is different, and the number of recommended encryption rounds is also different.

This project uses a 128-bit key with 10 rounds encryption. The encryption formula of AES is C = E(K,P). In the encryption function E, a round function will be executed, and this round function will be executed 10 times. The first nine times of this round function are the same, only the tenth is different. In other words, a plain text packet will be encrypted for 10 rounds. Figure 2.8 shows the workflow of AES. AES includes four operations: byte substitution, row shift, column mixing, and round key addition. Here is a brief idea of how each operation works:

- **SubBytes** — Through a non-linear replacement function, each byte is replaced with the corresponding byte using a look-up table.
- **ShiftRows** — Shift each row in the matrix circularly.
- **MixColumns** — In order to fully mix the operations of each straight row in the matrix. This step uses linear conversion to mix each inline four bytes. In the last encryption cycle, the MixColumns step is omitted and replaced by another AddRoundKey.
- **AddRoundKey** — Each byte in the matrix is XORed with the round key; each subkey is generated by the key generation scheme.

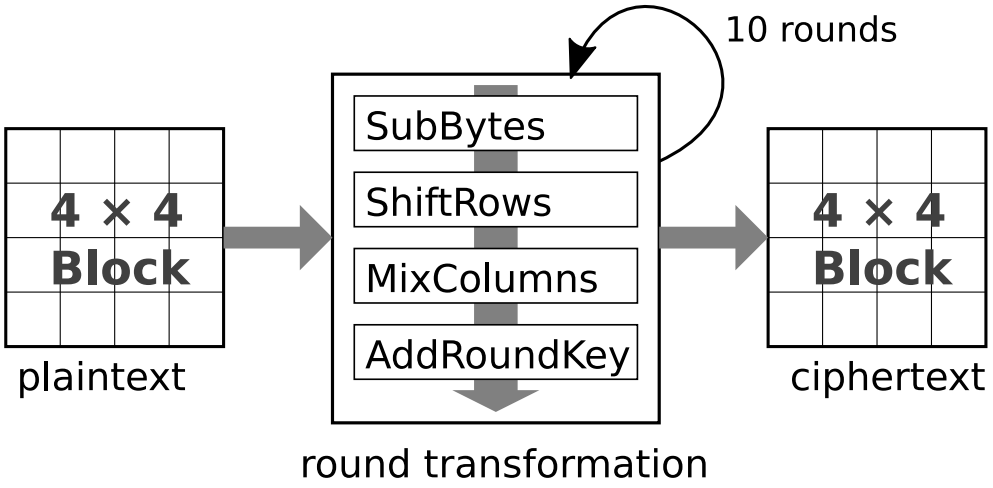


Figure 2.8: The workflow of AES encryption [4].

3

Clustering Solutions

3.1. Overview

The research and identification of scRNA-seq cell data is an important part of genetic data analysis [23]. A key step in research and identification is the classification of cells. The classification of cells can provide essential information for follow-up studies, including identification of marker genes [24], constructing gene expression regulation network [25], and cell cycle stage [26].

For the clustering problem of cell data, there are many classic algorithms including CIDR [27], SIMLR [28] and RaceID [29]. At the same time, with the wide application of deep learning, there are also some classification algorithms based on deep learning including DCA [30], scVAE [31], scziDesk, etc. This project uses scziDesk as the clustering algorithm applied to SGX. In this chapter, the basic algorithm principles of CIDR, SIMLR and RaceID will be introduced. Similarly, the principle of the scziDesk classification algorithm and the optimization based on rust/SGX will be discussed in detail.

3.2. CIDR

CIDR stands for clustering through imputation and dimensionality reduction. This is a very fast optimized clustering algorithm. The main focus of CIDR is on the improvement of principal component analysis. The results show that this algorithm achieves higher accuracy in less computing time [27]. A standard CIDR process can be divided into the following steps:

1. Find the minimum point in the two kernels of R function density and the Epanechnikov kernel to determine the dropout candidate threshold.
2. The direct dissimilarity matrix between the estimated loss rate and gene expression is calculated through a specific probability fitting function.
3. Perform PCoA (principal coordinate analysis) on the CIDR dissimilarity matrix, then select the first few principal coordinates.
4. Perform Hierarchical clustering on the selected principal coordinates.

CIDR related libraries can be directly called in R. The biggest advantage of this algorithm is its extremely fast calculation speed. For data units of thousands of cells, this algorithm can calculate results with higher accuracy in just a few minutes [27].

3.3. SIMLR

SIMLR stands for single-cell interpretation via multi-kernel learning. This is a similarity learning framework through multi-kernel learning to learn an appropriate distance metric for data. The results show that this algorithm achieves higher accuracy and sensitivity [28]. A standard SIMLR process can be divided into the following steps:

1. Construct kernels using Gaussian kernels with various hyper parameters.

2. Compute the distances between pairs of cells incorporating multiple kernel learning.
3. Iterates parameter updating steps until model convergence.
4. Cluster through the learned latent representation.

SIMLR related libraries can be directly called in R. The biggest advantage of this algorithm is its accuracy and sensitivity. Moreover, the cell-to-cell similarity values calculated through SIMLR can be used to create an embedding of the data in 2-D or 3-D for visualization [28].

3.4. RaceID

RaceID stands for rare cell type identification. RaceID is a clustering algorithm for the identification of cell types from single-cell RNA-sequencing data. It was specifically designed for the detection of rare cells which correspond to outliers in conventional clustering methods [32]. A standard RaceID process can be divided into the following steps:

1. K-means clustering of single cell expression data.
2. Identification of outlier cells by calibration of background model, identification of outlier cells and cells merging based on their similarity.
3. Inference of the final clusters that represent distinct cell types or cell states and using t-SNE maps for cluster examination.

RaceID related libraries can be directly called in R. The biggest advantage of this algorithm is its ability to cluster rare cells which normally hard to be detected in other algorithm. Researches also shows that by replacing the k-means clustering with k-medoids clustering, the cluster performance can be further optimized [29].

3.5. ScziDesk

SczíDesk is a machine learning based clustering algorithm. It can achieve high accuracy with excellent compatibility and robustness [9]. One of the most important target of our project is to implement SczíDesk inside SGX enclave. The original algorithm of SczíDesk is developed with python and tensorflow, however neither of those two is supported in SGX. Therefore, the whole algorithm needed to be redesigned by using Rust inside enclave. Details of the design will be introduced in chapter implementation. For this part, concepts and adaptive changes of SczíDesk including architecture, loss function and training strategy will be discussed.

3.5.1. Architecture

Figure 3.1 shows the architecture of SczíDesk clustering algorithm. The network part is an autoencoder with symmetrical encoder and decoder structures. The outputs of that autoencoder are three sets of parameter: dropout rate, mean value and dispersion value coming from ZINB modeling. Kmeans is used inside the latent space, and Adam optimizer is implemented for the parameter updating.

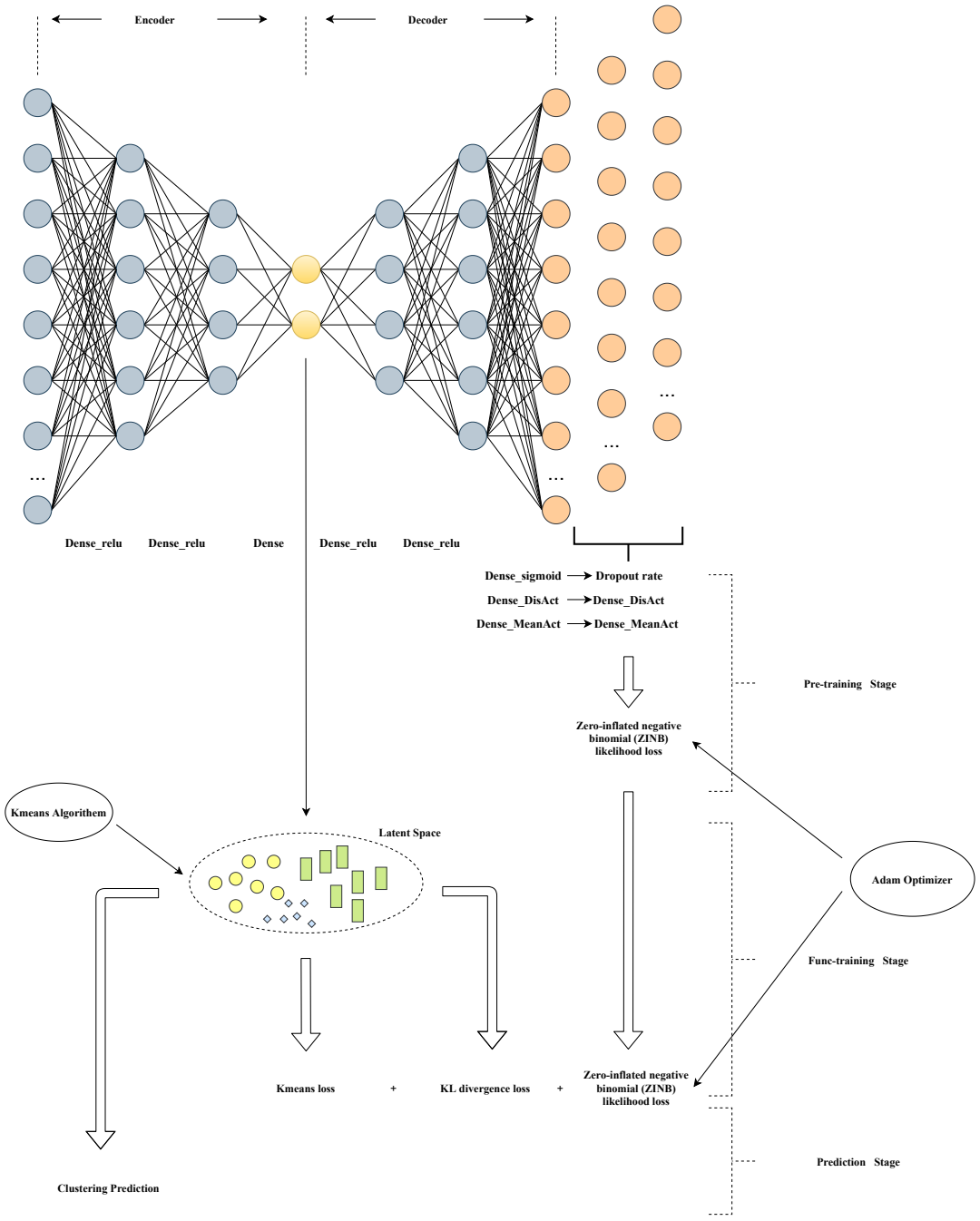


Figure 3.1: Architecture of SczíDesk algorithm.

Autoencoder

Autoencoder is one type of unsupervised learning artificial neural network. The purpose of an autoencoder is to learn a representation for a set of data, which is usually used for dimensionality reduction. The autoencoder has two main parts: encoder and decoder. Both the encoder part and the decoder part can be composed of multiple hidden layers, and the activation function of the hidden layer can also have multiple choices such as relu function or ordinary dense layer.

In the simplest case, given one hidden layer, the encoder stage of an autoencoder takes Equation (3.1) to generate a middle matrix h called latent representation. W in the equation is the weight matrix and b is the bias matrix. After that, the decoder stage of the autoencoder maps h to reconstruct x' with a same size of x using Equation (3.2). Figure 3.2 shows the schematic of the simplest autoencoder. Weight matrix and biases are initialized randomly and then updated in every iteration of the

training using back-propagation. The ultimate goal of autoencoder is to minimize the loss function of these model parameters. The data in the latent space can have a higher dimension than the original data, which is called over-complete, or it can have a lower latitude which is called under-complete. The autoencoder used in this project adopts under-complete approach and reduces the dimension of input data.

$$h = \sigma(W * x + b) \tag{3.1}$$

$$X' = \sigma'(W' * h + b') \tag{3.2}$$

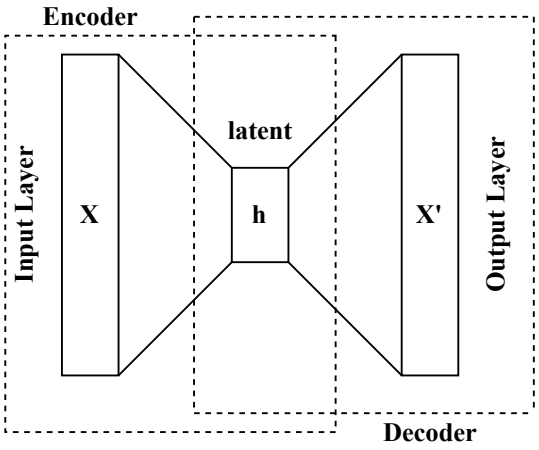


Figure 3.2: Schema of a basic Autoencoder with one layer encoder and one layer decoder.

The autoencoder has two important roles in this project. The first is that the dimensionality of the original data can be greatly reduced by using autoencoder, so that the relevant algorithm can be effectively applied to the data in the low-latitude latent space. The second is to restore the original data as much as possible after reducing the dimensionality, which provides an important foundation for subsequent training.

ZINB Model

Noticing that variance of single-cell gene expression data is often larger than its mean value, SczïDesk algorithm takes NB (negative binomial) distribution instead of Poisson distribution to model the data set. Negative binomial distribution is a statistical description of the discrete probability distribution of the number of failures when the number of successes reaches a specified number (denoted as r) in a series of independent and identically distributed Bernoulli trials. Mean parameter μ and dispersion parameter θ are two main parameters in the NB distribution following Equation (3.3). I and j in formula represents the expression of jth gene in the ith cell[9].

$$\begin{aligned} P_{\mathrm{NB}}\left(X_{ij} \mid \mu_{ij}, \theta_{ij}\right) = & \frac{\Gamma\left(X_{ij}+\theta_{ij}\right)}{\Gamma\left(X_{ij}+1\right)\Gamma\left(\theta_{ij}\right)} \times \left(\frac{\theta_{ij}}{\theta_{ij}+\mu_{ij}}\right)^{\theta_{ij}} \\ & \times \left(\frac{\mu_{ij}}{\theta_{ij}+\mu_{ij}}\right)^{X_{ij}} \end{aligned} \tag{3.3}$$

It can also be found that in the count matrix for single-cell gene expression data, zeros dominate the value matrix. Therefore, the whole data can be modeled by the combination of zero components and NB distribution, namely ZINB(Zero-inflated negative binomial). The whole model follows the Equation (3.4) where π is the weight coefficient of the point mass at zero [9].

$$\begin{aligned} P_{\mathrm{ZINB}}\left(X_{ij} \mid \pi_{ij}, \mu_{ij}, \theta_{ij}\right) = & \pi_{ij}\delta_0\left(X_{ij}\right) + \left(1-\pi_{ij}\right) \\ & \times P_{\mathrm{NB}}\left(X_{ij} \mid \mu_{ij}, \theta_{ij}\right) \end{aligned} \tag{3.4}$$

Therefore, mean parameter μ, dispersion parameter θ and weight coefficient π together form the representative parameters of the autoencoder in this project. Those parameters are trained to recover the undistorted data matrix using decoder outputs.

K-means
K-means algorithm is one of the simplest and most popular unsupervised machine learning algorithms. It is an iterative algorithm that tries to partition the data set into K pre-defined distinct non-overlapping sub-clusters where each data point belongs to only one group. K-means algorithm is very popular and used in a variety of applications such as market segmentation, document clustering, image compression.

In this project, the K-means algorithm will be applied to the data that has been reduced in dimensionality in the latent space . Different from directly classifying the original data, operating on the data in the learned embedding space can greatly reduce the computational complexity and improve the accuracy.

In order to help the K-means algorithm to achieve higher accuracy, this project also applies a weighted soft K-means model to realize the clustering in the latent space for K-means. In soft clustering, data points may belong to more than one cluster, and these clusters and data points pass through a membership level (actually similar to the concept of membership in fuzzy sets) to connect. The membership level shows how strong the connection between the data point and a cluster is. Fuzzy clustering is the process of calculating these membership levels and determining which cluster or clusters a data point belongs to according to the membership level. This application can effectively improve the classification accuracy [33].

Adam Optimizer
Adam algorithm is an algorithm that performs a step optimization on a random objective function. The algorithm is based on adaptive low-order moment estimation. Adam is an effective stochastic optimization method. It only requires a first-order gradient and requires only a small amount of memory. This method calculates the adaptive learning rate of different parameters through the estimation of the first and second gradients. It is very effective in the calculation of the sparse gradient, so it was selected as the optimizer of this project [34]. A standard Adam optimizer process can be divided into the following steps:

1. Determine the parameters α (step size), β1, β2 (the exponential decay rate estimated by the moment, between [0, 1)) and the random objective function f(θ).

2. Initialize the parameter vector θ0, the first-order moment vector m0, the second-order moment vector v0 and the time step t.

3. Loop, when the parameter θ does not converge, the loop updates each part iterative. That is, the time step t plus 1, update the gradient gt obtained by the objective function on the parameter θ at this time step, update the first-order moment estimate mt of the deviation and the second-order original moment estimate vt, and then calculate the first-order moment estimate and the deviation correction. The second-order moment of the deviation correction is estimated, and then the parameter θt of the model is updated with the value calculated above.

Adam algorithm does not guarantee to find extreme points. When the Adam algorithm converges to a sub-optimal solution, it is observed that some small batches of samples contribute a large and effective information gradient. After exponential averaging, their influence is reduced, resulting in poor model convergence [34]. Although the Adam algorithm cannot guarantee to find extreme points, it has high computational efficiency and low memory requirements.

3.5.2. Loss function

The loss function refers to the optimization object of the optimizer in the deep learning algorithm. Often we need to find the minimum or maximum value of this loss function. The optimization goal in this project is to find the minimum value of loss function to complete the training of the entire model. The scziDesk algorithm uses an integration of three different loss functions L1, L2 and L3.

L1

$$L_1(\pi, \mu, \theta \mid X) = -\sum_{i=1}^{n}\sum_{j=1}^{m} \log\left(P_{\text{ZINB}}\left(X_{ij} \mid \pi_{ij}, \mu_{ij}, \theta_{ij}\right)\right) \tag{3.5}$$

Equation (3.5) is the formula of the first loss function. It represents the objective function for data reconstruction and parameter estimation in the autoencoder. It is a simple negative log-likelihood of ZINB distribution function and the training target is to minimum that loss function.

L2

$$L_2(v, Z) = \sum_{i=1}^{n}\sum_{r=1}^{K} w_{ir} \left\|Z_i - v_r\right\|^2 \tag{3.6}$$

$$\tilde{w}_{ir} = \frac{\exp\left(-\left\|Z_i - v_r\right\|^2\right)}{\sum_{k=1}^{K}\exp\left(-\left\|Z_i - v_k\right\|^2\right)} \tag{3.7}$$

$$w_{ir} = \frac{\tilde{w}_{ir}^{\alpha}}{\sum_{j=1}^{K}\left(\tilde{w}_{ij}^{\alpha}\right)} \tag{3.8}$$

Equation (3.6) is the formula of the second loss function. It represents the objective function for weighted soft K-means clustering approach. In the formula v_r stands for the centers and w_{ir} stands for the weight matrix. In SczciDesk algorithm, the Gaussian kernel function is utilized as weight measure since the exponential function can smooth the gradient descent optimization process. It follows Equation (3.7). Inflation operation is also utilized to speed up the convergence of the function following Equation (3.8).

L3

$$p_{j|i} = \frac{\left(1 + \left\|Z_i - Z_j\right\|^2 / t\right)^{-(t+1)/2}}{\sum_{k \neq i}\left(1 + \left\|Z_i - Z_k\right\|^2 / t\right)^{-(t+1)/2}} \tag{3.9}$$

$$q_{j|i} = \frac{p_{j|i}^2 / \sum_{i \neq j} p_{j|i}}{\sum_{k \neq i}\left(p_{j|i}^2 / \sum_{i \neq j} p_{j|i}\right)} \tag{3.10}$$

$$L_3(Z) = KL(q \| p) = \sum_{i}\sum_{j} q_{j|i} \log \frac{q_{j|i}}{p_{j|i}} \tag{3.11}$$

Equation (3.11) is the formula of the third loss function. It represents the objective function for KL divergence approach. KL divergence is a measure of the asymmetry of the difference between two probability distributions P and Q. In this project, this algorithm is applied to preserve and strengthen the association between similar cells. P uses the t-distribution kernel function to describe the pairwise similarity like in t-SNE[35]. Q works as an auxiliary target distribution to strengthen the affinity between similar data points and put less emphasis on those pairwise data points with low similarity like in DEC [36].

In SczciDesk algorithm, the total loss function uses the combination of the three loss functions together with coefficients γ and λ to control the contribution of L2 and L3. The total loss function is show in Equation (3.12)

$$L(\pi, \mu, \theta, v, Z \mid X) = L_1 + \gamma L_2 + \lambda L_3 \tag{3.12}$$

3.5.3. Training strategy and optimization

The overall network training is divided into three parts.

- **Pretaining** — Minimize L1 using Adam optimizer. The purpose is to update the network parameters of the autoencoder network so that it can better restore the input data.
- **K-means clustering** — Perform cluster analysis of K-means algorithm on the low-dimensional data in latent space to obtain the initial clustering results.
- **Funetraining** — Minimize total loss using Adam optimizer. The purpose is to consider the important impact of soft-kmeans and KL divergence on the clustering results.

There are many important parameters that can be adjusted throughout the training process. These parameters not only affect the accuracy of the training results, but also have an important impact on the required training time and memory allocation.

The first is the input data size. After prepossessing, there will be a certain number of top highly variable genes work as input data for the entire deep learning network. In general, 300-500 cells are a reasonable data usage interval. Too little gene input will affect the matrix's restoration of the original data, and too many genes will cause a high occupation of computing resources.

The second is the size of the network model. The size of the autoencoder model includes two aspects: layer size and layer number. Larger layer size and layer number can reflect the association relationship of the data in more detail, but at the same time, this will bring a huge training burden to the overall training network. Because the increase of each layer of the training network will bring a huge increase in training parameters, which obviously will greatly increase the consumption of computing resources.

The third is the training parameters. In the entire training network, batch training is used in the training method. The choice of minibatch size has a great impact on training accuracy. A large minibatch size can bring more stable training results and faster convergence speed. But at the same time, a large batch size will put higher requirements on the memory size, and this will cause problems in the optimization and generalization of deep learning (large generalization gap). Although a small minibatch size can provide better accuracy, it has high requirements for training time. On the other hand, the training epoch number also affects the training results and training resource consumption. Fortunately, in the case of Adam optimizer, you do not need a high epoch number to obtain acceptable accuracy results quickly.

Finally, there are possible changes in training strategies. As mentioned earlier, the complete training consists of three parts, the parameter training of the model, the K-means classification and the final overall training. The choice of each part will also affect the results a lot. Specifically, how to allocate resources for different training steps will greatly affect the final training results. At the same time, in the K-means clustering algorithm, the optimization and selection of the algorithm, including the initial centroid points and the training method, will also have an impact on the training accuracy. The specific results will be discussed in the Chapter 4

4

Implementation

4.1. Overview

The implementation target of this project is to apply a deep learning clustering algorithm combined with data pre-processing and AES128 encrypted transmission on the Intel SGX platform. This project develops the algorithm using the Rust language and the Fortanix-EDP platform from scratch. The training and prediction parts of the deep learning network are all implemented in the SGX secure hardware.

For the implementation of the deep learning network in Intel SGX, some studies have provided a solution to separate the complete algorithm into network model and results prediction like tvm or sgntx [37]. These schemes generally use C or python to run the training part of the network outside SGX, transfer the network files into the secure hardware through a specific interface, and then predict the encrypted data inside the secure hardware. Because python/C are used in most of the current mainstream deep learning algorithms, this solution can save a lot of development time. However, given that the development of relevant third-party software is not fully mature, the application of this kind of approaches is quite difficult with lack of necessary updates and maintenance. At the same time, these schemes cannot achieve full security protection of high-value algorithms since the deep learning network model is placed outside a safe environment. Therefore in our project we decide to implement both training and prediction program inside SGX without the help of third-party frameworks.

Intel SGX provides related operation support for both Rust and C. This project chooses Rust as the algorithm development language. On the one hand, Fortanix-EDP's unique support for the Rust language on the SGX platform gives a lot of convenience for development. On the other hand, Rust has high-performance characteristics and memory security support which are all required in this project. But at the same time, as a relatively new language system, one of the biggest shortcomings of Rust now is the lack of mature third-party deep learning libraries. At present, many mainstream Rust deep learning libraries often need to be developed based on libc such as Rustml, and these libraries basically adopt a tensorflow-like system. Unfortunately, the dependent library libc is currently not supported in SGX compilation. Therefore, this project will rewrite many important parts of the deep learning algorithm from scratch to achieve tensorflow-like functions with the support of limited dependent libraries. Specific concepts and rewriting methods will be discussed in various parts of this chapter. Important dependency libraries used in this project include *ndarray*, *autograd*, *statrs*, *byteorder*, *aes* and *block-models* which are all purely developed in Rust.

In this chapter, first the architecture of the entire project will be introduced, and then the implementation methods of each part will be discussed in detail including pre-processing, AES encryption and the core neural network algorithm.

4.2. Architecture

The architecture of the whole project is shown in Figure 4.1. The whole project is divided into two parts, the data owner part and the SGX enclave hardware part. The data owner part includes the pre-processing of single-cell RNA-seq data, the encryption of data and the data feed to establish the TCP transmission stream. The SGX enclave part includes data reception, data decryption, neural network processing (including SczíDesk clustering network training and result prediction), and result feed. Those two parts are running and working at the same time, and during the operation of the SGX security hardware, other systems of the task will not be able to access the running memory space.

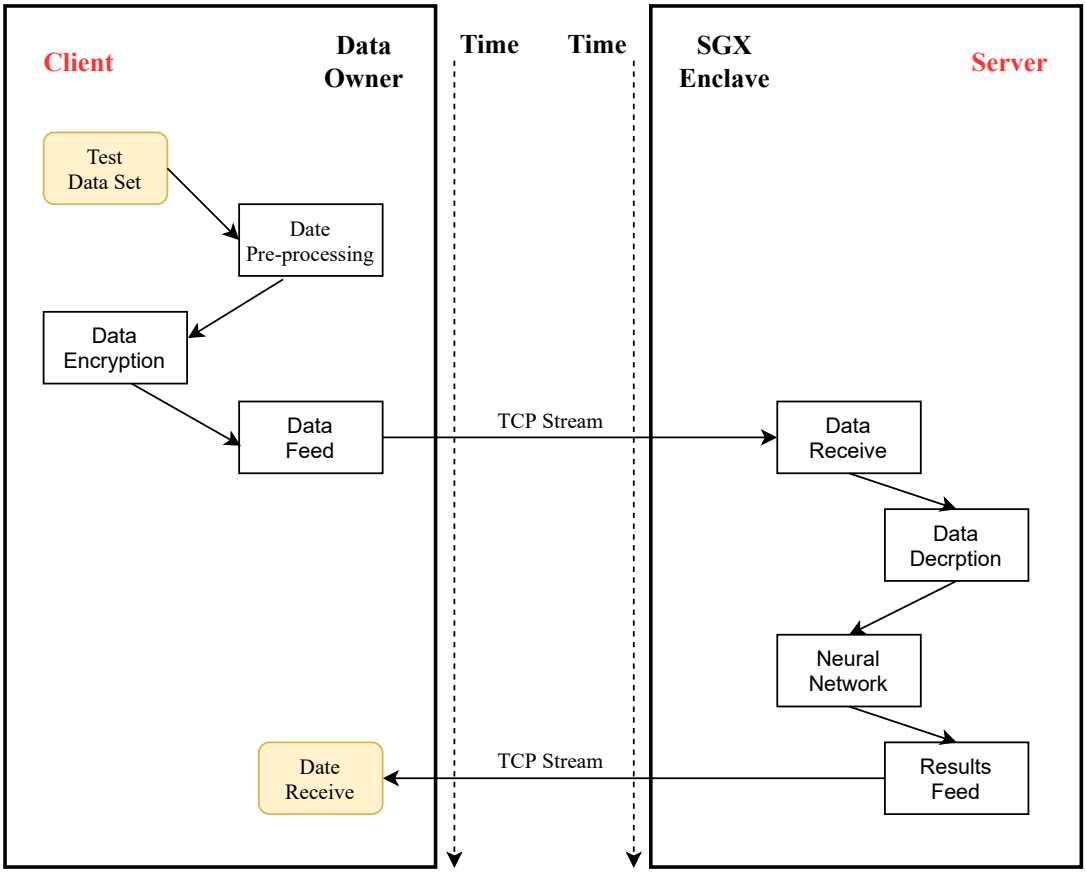


Figure 4.1: Architecture graph of the whole system.

In this project, taking into account the number of useful libraries, the data pre-processing is written in python. Raw data is the H5 data file of single-cell RNA-seq result. All other parts are programmed in Rust. The data owner part will run outside the SGX, and the SGX enclave part will run inside the SGX.

4.3. Pre-Processing

The data sets used in this project are stored in H5 files. This file type stores data information through a sparse matrix. The pre-processing part includes the following steps:

1. Implement the pandas library to read data from the file and convert the sparse matrix to a normal Numpy matrix. The data used in this project are cell×gene data matrix and cell labels.

2. Pre-process the count matrix data using the Scanpy library. It includes the following steps:

   - Filtering — Filter out the part where gene expression and cell expression are zero to clean up invalid data.

- Normalization — Normalize each cell by total counts over all genes, so that every cell has the same total count after normalization.
- Logarithmization — Logarithmize the data matrix and select out the top highly variable genes.
- Scaling — Scale data to unit variance and zero mean.

3. The matrix data corresponding to the top high variable genes processed by Scanpy is used as the input data of the model, and the matrix data corresponding to the top high variable genes that has not been normalized is used as the data modeling. These two matrices are written as csv format data for Rust code to read in.

Figure 4.2a shows a snippet (10x20) of an original single-cell expression count matrix of data set *GSE131907_raw_UMI_matrix_3kcell*. Figure 4.2b shows a snippet (10x20) of the normalized data matrix result after data pre-processing. The matrix value went through the filtering, normalization and scaling processes. Cell number is kept the same and the gene number decrease from more than 20 thousand to 500 high variable genes.

4.4. AES Encryption

For sensitive cell and gene data, this project uses Aes128-CBC as an encryption method. We use the Rust libraries crate aes, hex-literal and block-modes to realize the encryption and decryption. Encryption and decryption include the following processes:

1. Generate key and iv values for encryption.
2. Generate encryption cipher from key and iv values.
3. Create a new buffer to store the original text information and extra padding. Use cipher to encrypt the original text.
4. Obtain the key and iv values for decryption. And use these two values to generate a decryption cipher.
5. Decrypt the encrypted content by decryption cipher and split it into corresponding matrix.

An example of Aes128 encryption and decryption is shown below. The string ”Hello world!” in the example is first encrypted as a string using provided key and iv value and then decrypted using the same value.

```
”Hello world!”
”1b7a4c403124ae2fb52bedc534d82fa8”
”Hello world!”

key: ”000102030405060708090a0b0c0d0e0f”
iv: ”f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff”
```

In this project, the data matrix value are all float64. Each float64 value will be converted into a cipher text of 8 usize values during the encryption process. The normalized matrix and the unprocessed matrix will be encrypted and decrypted as a whole considering the convenience of TCP stream transmission.

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]
```

(a) A snippet of an original single-cell expression count matrix. Matrix value of the first 10 cells with 20 genes are shown in the graph

```
[[-0.07194739 -0.01825742 -0.0641837  -0.20778793 -0.08022229 -0.10738294
  -0.14211926 -0.13037442 -0.15126511 -0.17833477 -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]
 [-0.07194739 -0.01825742 -0.0641837  -0.20778793 -0.08022229 -0.10738294
  -0.14211926 -0.13037442 -0.15126511 -0.17833477 -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]
 [-0.07194739 -0.01825742 -0.0641837   2.6493282  -0.08022229 -0.10738294
  -0.14211926 -0.13037442 -0.15126511 -0.17833477 -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]
 [-0.07194739 -0.01825742 -0.0641837  -0.20778793 -0.08022229 -0.10738294
  -0.14211926 -0.13037442 -0.15126511 -0.17833477 -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]
 [-0.07194739 -0.01825742 -0.0641837  -0.20778793 -0.08022229 -0.10738294
  -0.14211926 -0.13037442  5.825563    8.490172   -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]
 [-0.07194739 -0.01825742 -0.0641837  -0.20778793 -0.08022229 -0.10738294
   2.7847393  -0.13037442 -0.15126511 -0.17833477 -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]
 [-0.07194739 -0.01825742 -0.0641837  -0.20778793 -0.08022229 -0.10738294
  -0.14211926 -0.13037442 -0.15126511 -0.17833477 -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]
 [-0.07194739 -0.01825742 -0.0641837  -0.20778793 -0.08022229 -0.10738294
  -0.14211926 -0.13037442 -0.15126511 -0.17833477 -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]
 [-0.07194739 -0.01825742 -0.0641837  -0.20778793 -0.08022229 -0.10738294
  -0.14211926 -0.13037442 -0.15126511 -0.17833477 -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]
 [-0.07194739 -0.01825742 -0.0641837  -0.20778793 -0.08022229 -0.10738294
  -0.14211926 -0.13037442 -0.15126511 -0.17833477 -0.06807479 -0.04729109
  -0.07230155 -0.13450223 -0.11345073 -0.21715245 -0.12004908 -0.16967778
  -0.34381247 -0.05459556]]
```

(b) A snippet of the normalized data matrix result after data pre-processing. Matrix value of the first 10 cells with 20 genes are shown in the graph

Figure 4.2: Snippets of data before and after pre-processing

4.5. Core Algorithm

The core algorithm of this project is divided into four parts, data interface, training network, loss functions and helper functions. The data interface is responsible for opening and closing the TCP stream and the type conversion of the input data. The training network is responsible for the application of the SczíDesk algorithm (which includes K-means clustering and the main function). The loss function works for the special custom defined loss functions and the helper functions are responsible for writing the necessary underlying functions. The autograd library used in the project provide the basic definition of tensor and basic operations like multiplication, division and tensor transpose. Figure 4.3 shows the relationship between core algorithm and other parts in the SGX-SczíDesk system. Compared with original python SczíDesk algorithm, Rust-SczíDesk we implement have several optimizations including helper functions, loss functions, K-means clustering and main functions. The implementation approach of them are discussed in detail below.

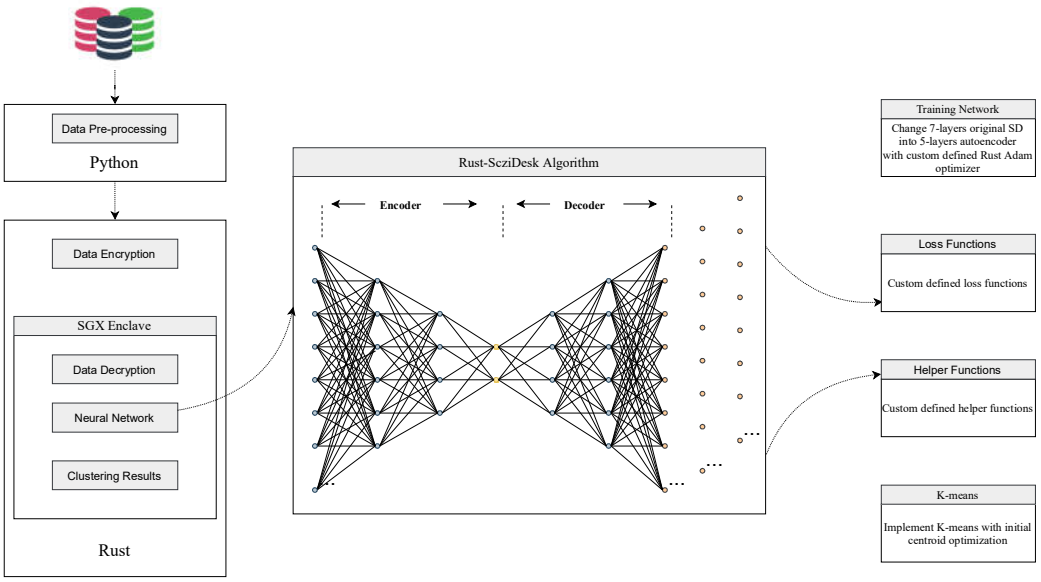


Figure 4.3: Core algorithm of SGX-SczíDesk system.

4.5.1. Helper Functions

Due to the lack of dependent Rust libraries in the SGX environment, this project needs a number of related helper functions. Helper functions are divided into the following two parts in general: basic deep learning network functions and gradient calculation functions for the back propagation process.

Network functions

For the deep learning network of the autoencoder, the most important network function is the Dense layer function. This function is equivalent to matrix multiplication of the input tensor and weighted matrix. Based on this function, different Dense functions such as Dense relu function and Dense sigmoid function can be created. These are the important components in the autoencoder network construction of this project. Algorithm 1 shows an example of writing a Dense layer function with an exponential activation function.

Algorithm 1 Dense Layer Function Pseudo-code

1: Function input data are tensor matrix x and weight tensor w.
2: Take the graph of x as g.
3: Calculate y, the graph matrix multiplication of x and w.
4: Calculate the exponential result of y and clip it between $1e-5$ and $1e-6$.
5: Return the result.

This is a relatively simple example, because all the operations on tensor required in this function are provided in the basic autograd library. However, the basic operations for tensor provided in this library are currently quite limited, and most of the complicated functions need to be written by ourselves using the combination of basic mathematics operations.

Gradient functions

When the deep learning network is updating the parameters, the gradients of the relevant learning parameters will be calculated for the realization of back-propagation. This is a necessary calculation function for the training of network parameters. In this project, in order to realize the normal operations of the network, quite a lot of gradient calculation functions are created. Since the calculation of the gradient involves differential expressions of related expressions, the creation process is more complicated when faced with complex tensor expressions. The development of the lgamma function will be taken as an example to show how the calculation of these gradient functions is applied in this project.

lgamma function is the logarithm of the famous gamma function. The formula of gamma function is defined in Equation (4.1).

$$\Gamma(z) = \int_0^\infty t^{z-1} \mathrm{e}^{-t}\, \mathrm{dt} \tag{4.1}$$

The construction of the lgamma function is divided into two parts, one part is the establishment of the forward propagation expression, and the other part is the backward propagation expression and gradient calculation. The forward propagation expression can be directly constructed using applicable lgamma function to the f64 type variable, and the backward propagation needs to be constructed using a new function - the derivative of lgamma function. For this new helper function, there is no need to define its gradient calculation. Algorithm 2 shows the process of developing lgamma function.

Algorithm 2 lgamma Function Pseudo-code

1: Create struct digamma, which is the derivative of lgamma function. The structure of digamma function includes three parts: struct statement, function impl and function
2: In the impl of struct digamma, fn compute and fn grad are created
- fn compute switch the tensor data to an ndarray view type data, and then implement the derivative calculation for the input array. The result is appended out.
- fn grad is empty and append none as output.
3: In the digamma function, input data are tensor matrix x and graph g. Then a new tensor is built calculating the result of the digamma of input x
4: Create struct lgamma, which is the lgamma function. The structure of lgamma function includes three parts: struct statement, function impl and function itself
5: In the impl of struct lgamma, fn compute and fn grad are created
- fn compute switch the tensor data to an ndarray view type data, and then implement the lgamma calculation for the input array. The result is appended out.
- fn grad first take the gradient input as x, gradient output as gy, compute output as y. Then calculate the gx with formula gx = gy×$digamma(x)$. The result is appended out.
6: In the lgamma function, input data are tensor matrix x and graph g. Then a new tensor is built calculating the result of the lgamma of input x

In the calculation of specific functions, the value of tensor can be extracted as an array value to use the ndarray library for performing related operations. This greatly improves the possibility of potential expansion of the gradient function. A series of related functions, including power calculation between two tensors, assign functions, etc., were written in this project. This is an important foundation for subsequent network development.

4.5.2. Loss Functions

In this project, the loss function is the object of the optimizer iterative update. The purpose of the optimizer is to minimize the loss function in each iteration. In normal deep learning network, there are many commonly used loss functions such as softmax-loss, logistic-loss or cross entropy. A major feature of SzciDesk is the application of complex custom defined loss functions.

The three loss functions will be calculated separately. They are the ZINB loss function that simulates the autoencoder learning network parameters, the KL divergence loss and the K-means loss of self-learning K-means. The specific expressions have been discussed in Chapter 3. Here we choose the calculation of latent q matrix in KL divergence loss to demonstrate how to program in SGX through Rust. The pseudo-code is shown in Algorithm 3.

Algorithm 3 Latent q Function Pseudo-code

1: Function input data are tensor matrix $latentp$ and usize $cellnum$.
2: Take the graph of $latentp$ as g.
3: Calculate the reduce sum of $latentp$ of the first axis as $latredu$.
4: Reshape the $latredu$ as one row matrix and tile it with $cellnum$ times. The result is $latentpsum$
5: Calculate the division between the power of $latentp$ and $latentpsum$. Then transpose the result as $latentq$
6: Calculate the reduced sum of $latentq$ and then reshape it as a one row matrix. Then tile it with $cellnum$ times as $latentqsum$.
7: Calculate the division between the transpose result of $latentq$ and $latentqsum$.
8: Transpose the result and return it.

Basic calculation method of the matrix such as transpose, tile and reshape are heavily used here. It should be noted that the division of matrices of different sizes in Rust will cause the calculation to be impossible. Therefore, the matrix in the operation process needs to be continuously reshaped or tiled to ensure the uniformity of the matrix size. The specific calculation of each function needs to be used in conjunction with the custom defined tensor calculation function introduced earlier to ensure that these loss functions can be calculated for the gradient and back-propagation.

4.5.3. K-means clustering

K-means clustering is a very common and classic unsupervised learning clustering method. The pseudocode of the K-means is shown in Algorithm 4.

Algorithm 4 K-means Algorithm Pseudo-code

1: For each cluster centroid $c_1, c_2, ...c_k$, randomly select a feature vector as initialization.
2: Repeat steps 3 and 4 until convergence
3: For each data point x_i
   - find the nearest centroid $c_1, c_2..c_k$
   - assign the point to that cluster
4: For each cluster, new centroid = mean of all points assigned to that cluster

However, the K-means application of tensor is relatively complicated. This project has developed this algorithm from scratch. First, the latent space data is used as the data point. Then, initial centroids are chosen from the set of data points with the help of annotation data generated from pre-processing. For the next step, element-wise subtraction of points and centroids that are 2D tensors are needed. As the tensors have different shape, points and centroids needed to be expanded into three dimensions, which enables us to use the broadcasting feature of subtraction operation. Then, calculate the distances between points and centroids and determine the cluster assignments. Next, each cluster can be compared with a cluster assignments vector, get points assigned to each cluster, and calculate mean values. These mean values are refined centroids. Finally, the centroids variable is updated with the new values. Assign function and several constant flag array generator are also implemented during the process by using custom defined tensor functions introduced before. The core loop code of the algorithm is shown in Algorithm 5.

Algorithm 5 Centroids Update Loop Pseudo-code

1: For each centroid i, create a representation matrix c which has all i values in the matrix.
2: Calculate the assignment number to the representation matrix. The result is calculated by the reduced sum of the equal number between assignments results and the representation matrix.
3: Calculate the gather matrix for this centroid. The result is calculated by two steps:
- Generate the place matrix for every points where the value is one if the point belongs to this centroid.
- Generate the gather matrix by calculating the multiplication between points matrix and place matrix.
4: Calculate the mean value for the gather matrix.
5: Concat the mean value into the mean matrix.
6: Loop into next centroid i+1.

The location of the initial centroid points can greatly affect the clustering results of the K-means algorithm. For the optimization of this part, the traditional K-means uses a random initialization point method, but we found that this method will lead to a lower accuracy. K-means++ uses the initial classification method so that the initial centroid position is the set of points with the largest distance from each other. We tried to apply K-means++ but failed to develop it successfully. Therefore, in this project, we directly selected the corresponding random points in the different types of cell content in the annotation file as the initial point of the K-means algorithm. This can easily realize the improvement of the accuracy of the clustering results for the K-means algorithm.

4.5.4. Main Function

Given that the relevant basic algorithm development has been completed, the main function of this project is responsible for the integration of all content, as well as the training and prediction of the autoencoder network. The whole main function has four major components: network construction, pre-training, K-means clustering and funetraining.

Network construction

Different from the development of python language which has a high-level library integration, when using Rust language for neural network development, the parameters in network training need to be clearly defined and initialized. For this project, the parameters that the deep learning network needs to learn include the weighted matrix of each Dense layer and the cluster assignment used as the result. Glorot uniform random value generator is used to initialize these parameters.

After the necessary parameters are defined and initialized, the entire autoencoder network consists of three main parts: encoder, decoder and calculation inside the latent space. It can be created using the different Dense layers that we defined before. The three autoencoder parameters we need in the algorithm: pi value, mean value and dispersion value are calculated in sequence. These three values are used as part of the input for the ZINB loss function calculation. At the same time, for the latent space matrix that has been reduced in the dimension, KL divergence and K-means loss are calculated. The results of the calculation are stored in the intermediate tensors.

Training

The development of the training part of the network mainly includes the following parts: optimizer construction, gradient function establishment and batch training.

Adam optimizer is used in this project. This optimizer can adapt to the learning rate, and has a good convergence speed in the case of small training epochs. Using autograd in Rust to call Adam optimizer includes the following steps. First, you need to use the parameters that need to be optimized to make a state of Adam optimizer. Then you need to instantiate an Adam optimizer with default setting. Finally, update ops of ‘params‘ using its gradients and Adam optimizer.

The training part of this project is divided into two blocks, pretraining and funetraining. These two parts correspond to the process of optimizing different loss functions. The pretraining part only iterates on the weight matrix of the overall autoencoder, while the funetraining part includes model weight matrix, KL divergence loss and K-means loss. This also means that different gradient functions need to be

established for different training parameters and training objects. After the loss function and gradient function are established respectively, two different update operations can be constructed.

Batch training technique is also implemented in the training process. A training dataset can be divided into one or more batches. When all training samples are used to create one batch, the learning algorithm is called batch gradient descent. When the batch is the size of one sample, the learning algorithm is called stochastic gradient descent. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent. Different batch sizes will have great impact on accuracy and training time. In the selection of this project, the limitations of the memory capacity and operating speed of the SGX security hardware are taken into account. Measurements and balanced choice will be shown in Chapter 5.

K-means clustering

The K-means algorithm will be placed in the middle of the entire training network between pretraining and funetraining. This algorithm will perform clustering operations on the latent space generated by the pre-trained model. The calculated cluster centroid result will be used as one of the input tensors of funetraining. At the same time, this result will also be used as one of the training objects in the funetraining training process. The entire K-means clustering process is calculated in sequence.

5 Measurements

5.1. Overview

This chapter introduces the measurement results of this project, which is divided into two parts: the construction of the experimental hardware environment and the measurement results. In the construction of the test environment, the construction approach of Windows and Linux environments are discussed so that other users can choose a reasonable development environment. In the measurement results, the benchmark measurement for SGX is first displayed, and then detailed tests and analyses are carried out in terms of accuracy, memory allocation and running time.

5.2. Experimental setup

5.2.1. SGX setup

Whether it is for Windows SGX environment or Linux SGX environment. The Intel SGX development environment needs to meet the following prerequisites:

1. At least 6th series CPUs, preferably 8th series CPUs, to support the safety-related FLC (Flexible Launch Control) instructions added by Intel and better compatible with ECDSA type RA (Remote attestation) services. Whether the specific CPU supports SGX hardware needs to be inquired on the official website.

2. Set the SGX hardware to [enabled] in the BIOS. Usually the location of this service is Intel Advanced Menu -> CPU Configuration Software Guard Extended Instruction Set (SGX) ->. The BIOS will display the following options: Enabled, Software Controlled, Disabled. If the setting is activated, the Intel Software Guard extension becomes an application that is enabled and available for use.

3. If it is an 8th CPU, you need to enable FLC in the BIOS. Note that there are differences in the process of building development environments with and without FLC. The existing official process may be more friendly to the development of SGX development environment on platforms with FLC. Here, only the building of SGX development environment on platforms without FLC will be introduced.

Windows

The Intel SGX construction under Windows environment includes the following steps:

1. Download and install Intel ME. Some related security functions of SGX require Intel Management Engine (ME) to provide monotonic counter and real-time clock RTC, so it is recommended to maintain Internet connection when installing SGX SDK and PSW. Then install the full version of Intel ME software package

2. Install Visual Studio (VS) 2017. This step must be done before installing the SGX SDK. Otherwise, when you start VS2017, you cannot see the SGX plug-in.

3. Install SGX SDK, download PSW (platform environment) and SDK (required). After the downloaded exe file is decompressed, there are documentation and installed exe. After following the instructions for installation, if you can see the Intel SGX Enclave Project when you create a new VS studio project, it means that your environment is ok.

It is recommended to use version 2017 or earlier for the installation of VS. We experimented with using the VS 2019 version for installation, but in that case a new project is not able to detect the Intel SGX Enclave Project.

Linux
The basic hardware enablement process in the Linux environment is more complicated. And the official instruction may have many problems in actual use, here is a brief start-up process used in this project. Details of the instructions are shown in Appendix B.

1. Build and Install the Intel® SGX Driver. It is recommended to run the executable file to install directly by default. The processes of SGX Driver installation includes:
   - File downloading — Download three files in Linux-sgx-repo to downloads folder by default.
   - File authorization — Authorize the execution permission of the driver file and run it.
2. Build and install the Intel® SGX SDK and Intel® SGX PSW Package. The process is shown as follows:
   - Tools installation — Install the tools needed to compile the SGX SDK.
   - Get source code — Get source code of SGX SDK and SGX PSW from the git.
   - SDK and PSW installation — Install Intel SGX SDK and PSW.
3. If you see [ok] after a while, the set-up is successful.

SGX PSW provides three services: Launch, EPID-based attestation and Algorithm agnostic attestation. Starting from version 2.8, PSW has been split into these three small services, which can be installed separately. After installation, you can find a sgx-asem-services directory in the /opt/intel/ directory.

5.2.2. Fortanix-EDP setup
Fortanix-EDP has a very clear installation process on its official website. Therefore, it will not be described in detail here. A simple procedure is shown as follows:

1. Install Rust with Rustup. The Rust nightly toolchain is also needed. Also, install the Fortanix-EDP target.
2. Install SGX driver. Enable the Fortanix APT repository and install the intel-sgx-dkms package under Ubuntu environment
3. Install AESM service. Enable the Intel SGX APT repository and install the sgx-aesm-service and the libsgx-aesm-launch-plugin packages.
4. Install Fortanix-EDP utilities. You will need to install the OpenSSL development package and the Protobuf compiler. Then, you can use cargo to install the utilities from source.

After the installation is complete, you can use sgx-detect for set-up check. If the result is positive, then the platform is ready to use.

5.2.3. Measurement Environment
For the measurement of the SGX-SzciDesk system in our project, the heap size is set to 90 MB which is the maximum heap size of the testing system. The CPU used for accuracy, memory allocation and running time measurement is Intel Core i7-6700HQ. Graphics card model is GTX965M, memory type is 8GB DDR3L.

For the measurement of the original SzciDesk algorithm in our project. AMD Ryzen 5 3600X 6-Core processor is used with 16GB DDR4 memory. Graphics card model is GTX1660Super.

5.3. Benchmark Measurement

5.3.1. SGX Hardware

As a technology that is completely different with ordinary unsecured memory space operation, Intel SGX brings security to the program running inside the enclave, but it also significantly reduces the running speed. This part will show the difference of running time between outside SGX and inside SGX of basic operation, and will try to analyze the possible reason.

In order to test the time loss in the SGX enclave, we mainly measured two types of operations-mathematical operations and data writing. These two types of operations are important parts required by the algorithms implemented inside SGX. When testing the time required for mathematical operations, we used a simple calculation loop code, and each mathematical operation was looped 100,000 times to measure the total time required. The time shown in the Table 5.1 is the average single operation time after calculation. When measuring the time required for data writing operations, we create a new array, and then use a loop to continuously write one byte of data into that array. The time shown in the Table 5.2 is the total writing time of different size of data.

Table 5.1: Running time for mathematics operations outside and inside SGX

Operation Type	*Time outside SGX (µs)*	*Time inside SGX (µs)*	*Ratio*
Addition	0.03	0.55	18.3
Subtraction	0.03	0.55	18.3
Multiplication	0.03	0.54	18
Division	0.03	0.57	19

Table 5.2: Running time for write operations outside and inside SGX

Data Size	*Time outside SGX (s)*	*Time inside SGX (s)*	*Ratio*
20KB	0.001	0.018	18
50KB	0.003	0.046	15.3
100KB	0.006	0.109	18.2
1MB	0.067	0.949	14.2
10MB	0.507	9.258	18.2

It can be observed that there is a significant loss of time in the SGX enclave, whether it is the calculation of mathematical operations or the writing of data. The operating time multiples between inside SGX and outside SGX are basically maintained at 15-20. In general, the performance overheads can result from two main aspects: first is the actual overhead of executing CPU instructions and accessing the encrypted memory in an enclave. The second is the overhead associated with entering and exiting an enclave. In this project, because Fortanix-EDP is used as the auxiliary development platform, where the algorithm is completely embedded in the enclave, the data entering and leaving the enclave will not constitute a significant negative impact on the time of this project. Therefore, it can be concluded that the using SGX instructions on CPU and access to the enclave induces the most time overhead in our results.

5.3.2. SczíDesk Algorithm

The clustering algorithm of this project is the SczíDesk algorithm. The performance of this algorithm has a very important impact on the performance of the project, so here will be a performance test of the original algorithm developed based on python. This test will serve as a benchmark test for the complete development of this project. The tested data sets are Adam, Quake_10x_Bladder, Quake_10x_Limb_Muscle, Quake_seq2_Diaphram, Quake_seq2_Lung and Romanov. Important parameters for those datasets are shown in Table 5.3. All these data sets can be downloaded from the 10x_genomics website [38]. CIDR, SIMR and RaceID were used as comparison methods in the test.

Table 5.3: Important parameters for data sets used in measurement

Data set Name	*Organism*	*Cell number*	*H5 file size (KB)*
Adam	Mus musculus	3660	13002
Quake_10x_Bladder	Mus musculus	2500	15560
Quake_10x_Limb_Muscle	Mus musculus	3909	12026
Quake_seq2_Diaphram	Mus musculus	870	5663
Quake_seq2_Lung	Mus musculus	1716	14076
Romanov	Mus musculus	2882	17291

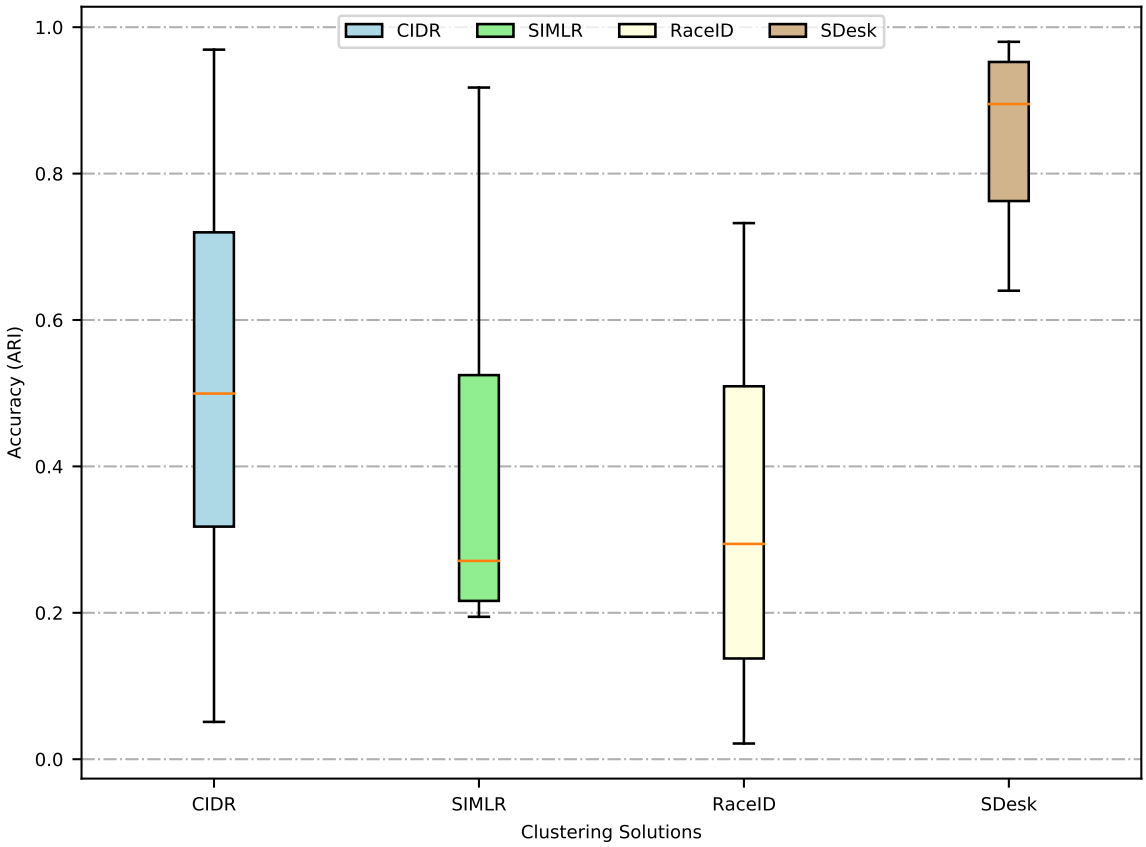


Figure 5.1: Performance (ARI) of CIDR, SIMR, RaceID and SczíDesk on different data sets.

Results of the measurement are shown in Figure 5.1 and Figure 5.2. ARI (adjusted rand index) is used to measure the accuracy of measurement results. It is a measure of the similarity between two data clustering [39]. The value of ARI is between 0-1 and the larger the value, the more similar the results of the two clusters are, which in our case the more accurate the result is.

In the test, the high-varaiable gene of SczíDesk is 500, the batch size is 256, and the pretraining and funetraining are 1000 and 2000 respectively to ensure the convergence of the results. It can

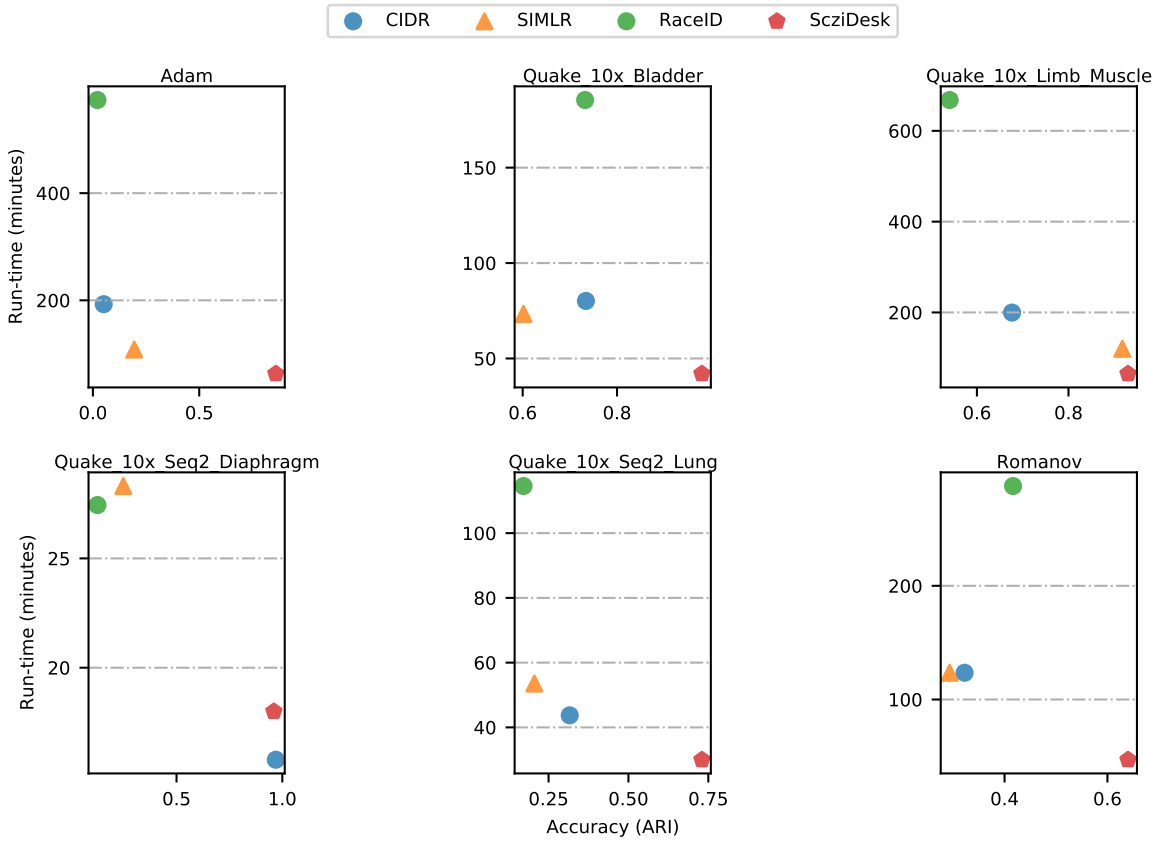


Figure 5.2: Running time and ARI of CIDR, SIMR, RaceID and SzciDesk on different data sets.

be observed in the figure that for all data sets, compared with the other three clustering methods, SzciDesk has shown very high accuracy. At the same time, SzciDesk is far stronger than the other three algorithms in terms of stability, the ARI of these nine data sets are all above 0.7.

SzciDesk algorithm also shows great advantages in the reduction of computing time. For all data sets, the computing time required by this algorithm is the shortest. This huge advantage comes from the effective application of batch training. Therefore, we can conclude that this algorithm is a very effective, powerful and applicable clustering algorithm.

5.3.3. High-variable genes

For the analysis of the number of high-variable genes, the original SczíDesk algorithm is used for measurement. Because according to previous studies, after a certain value, the number of high-variable genes will have a very small effect on the final accuracy [9]. Therefore, we can directly use the original algorithm to determine this limit. In the measurement process, the number of network layers and the number of units of each layer of the autoencoder are set as the minimum within a reasonable range. That is, two layers of the encoder and two layers of the decoder, the magnification and reduction ratio of each layer is 2. In view of the consideration of memory space management and running time, the limit value of the number of high-variable genes will be determined as the application value for further measurement. During the test, the batch size is selected as 128, and the training epoch is selected as 500 to achieve fast convergence.

Measurement results are shown in Figure 5.3. Five high-variable gene number from 200 to 1000 is chosen to find the minimum limitation of that parameter. It can be observed that for Bladder data set, the turning point is around 300 and for Limb_Muscle the point is around 500. Therefore, in subsequent tests, we will always use 300 high-variable genes for the bladder data set, and 500 high-variable genes for the limb_muscle data set.

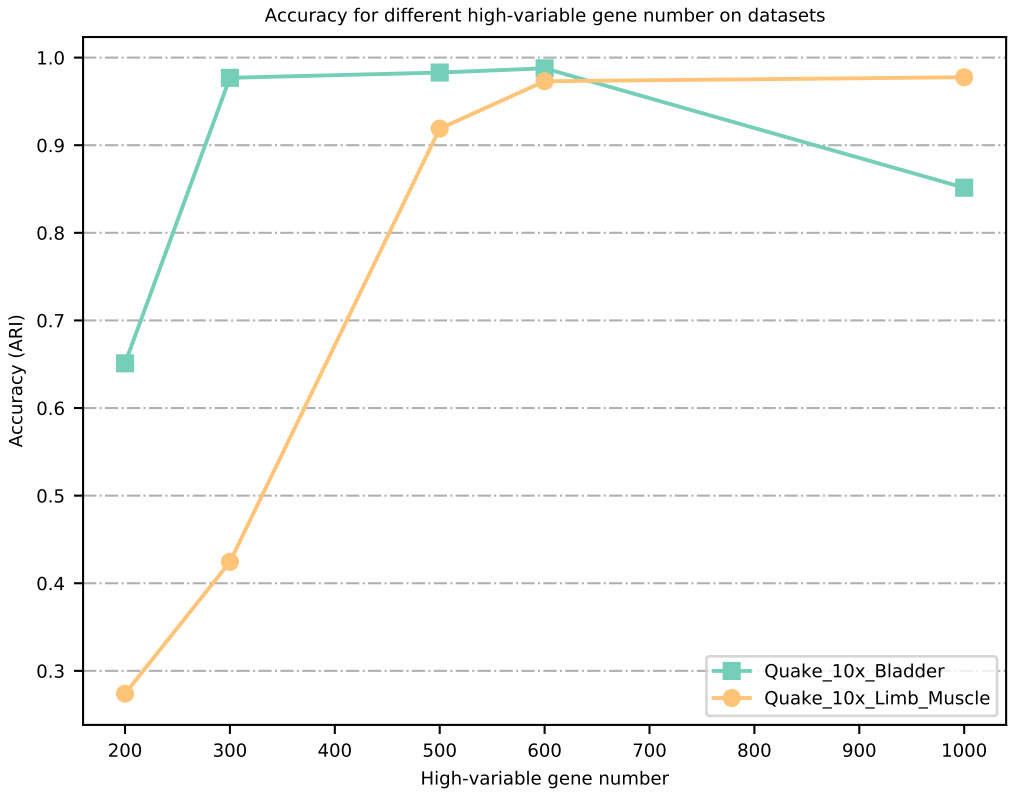


Figure 5.3: Accuracy of SczíDesk clustering method of two data sets in different high-variable gene number.

5.4. Accuracy

The accuracy of the SGX-SczíDesk system in this project is tested here. According to the previous measurement results, the data sets bladder and limb_muscle have the most outstanding performance in the SczíDesk algorithm. Therefore, these two data sets are used as data sets for system testing. Bladder and limb_muscle are composed of 2500 cells and 3909 cells respectively. The parameters involved in the test are divided into three major parts: network size parameters, training strategy and batch size. Here we will introduce and analyze the impact of different parts on the accuracy of the system in detail.

5.4.1. Network Size

The parameters of network size mainly include the number of network layers, and the number of units of each dense layer of the autoencoder learning network.

In the process of testing the network layer size of the autoencoder learning network, the selected value of the batch size is 1/100 of the cell number. All training epochs are selected for five times. The testing values of the network are all numbers based on two for better data compression and decompression.

The value of the row axis is the network layer size. The three number stands for the number of unit in three layers of the SGX-SczíDesk system neural network system. For example, 300-64-16 means that the first layer of encoder and decoder has 300 units, the second layer of encoder and decoder has 64 units and the latent space has 16 units.

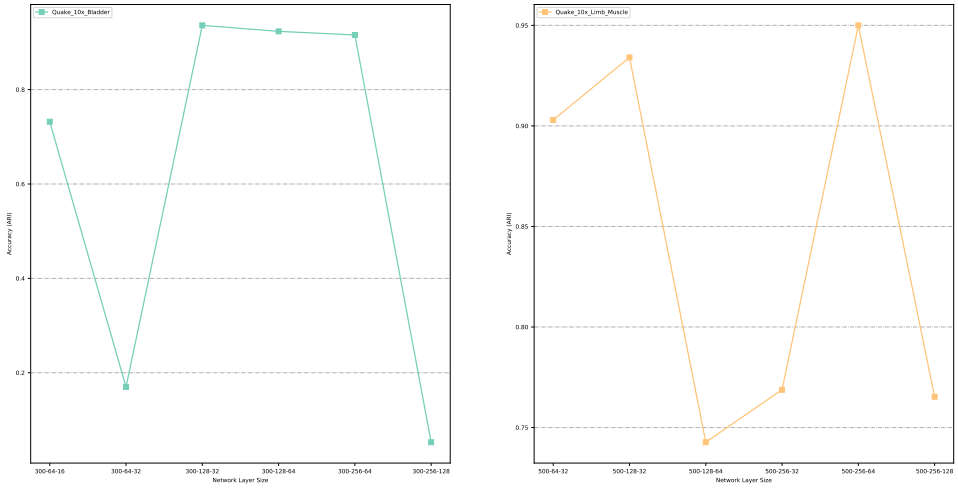


Figure 5.4: Measurement results of accuracy (ARI) for different network size.

The measurement results for two data set are shown in Figure 5.4. It can be clearly observed that the network size of the autoencoder has a great influence on the accuracy of the final result. If the compression/decompression ratio between each layer of network is too small, the original input will not be able to effectively reduce the dimensionality of the data, which will cause great confusion to the clustering method in the Latent space, making the final result unable to perform accurate clustering. And if the compression/decompression ratio between each layer of network is too large, the input matrix will be over-compressed, resulting in too much information loss during network transmission. Then the results cannot be effectively clustered. Similarly, it can be inferred that when the number of layers of the network increases, for example, from five to seven layers, the overall accuracy will also increase, because the increase in the training parameters of the network will restore the original data more accurate. However, considering that the increase in the number of network layers will substantially increase the burden on the training resources for the entire network, this project will maintain the overall number of networks with a five-layer network.

We can also find that for different data model sizes, there are different local optimal network model value. For the data set bladder, in the case of a five-layer network, it can be observed that the network model size of 300-128-32 is its local optimal value with ARI 0.94. For the data set Limb muscle, in the case of a five-layer network, it can be observed that 500-128-32 and 500-256-64 are two possible local optimal solutions with ARI 0.93 and 0.95. Therefore, considering the saving of computing resources, 300-128-32 and 500-128-32 will be used as the network size values for further tests.

5.4.2. Training Strategy

The choice of training strategy includes the number of training epochs of pretraining and funetraining. This part of the test is based on five layers learning network with network layers size, respectively 300-128-32 and 500-128-32 for two test data sets. The selected value of the batch size is 1/100 of the cell number. All training epochs are selected for five times.

Ten tests were performed on the bladder and limb muscle data sets. These ten tests use different combinations of pretraining epoch and funetraining epoch. The value of the row axis is the training strategy. For example, 4-2 means that the pretraining epoch is four and the funetraining epoch is two.

The measurement results for the two data set are shown in Figure 5.5. This graph clearly reflects the impact of training epoch number on the accuracy of the overall algorithm. The impact is divided in two aspects. One is the size of the training epoch number, and the other is the scale of two training parts. Regarding the impact of training epoch number on accuracy, it can be seen that as the number of training epoch increases, the accuracy of the clustering results increases rapidly. This is due to the use of the Adam optimizer in this project, which makes the results converge very quickly.

For the two data sets, it can be observed that when the number of pretraining and funetraining reaches five times, the optimal value of the clustering result can be basically obtained with ARI around 0.9. Regarding the relationship between the two training parts, it can be observed that the asymmetrical training component has a negative impact on the training results. The reason for that is the mutual dependence of the loss function of the two training objects.

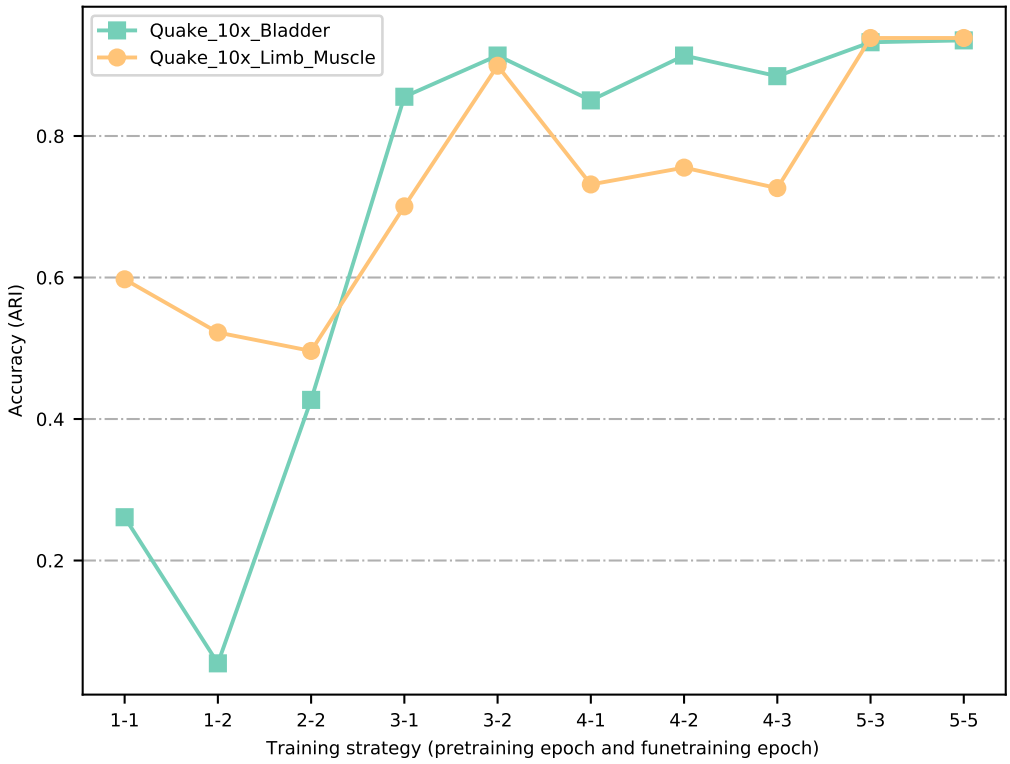


Figure 5.5: Accuracy of of the system output with different training strategy (pretraining epoch and funetraining epoch).

The test results prove the effectiveness and good convergence speed of the deep learning network of this project. In a limited number of training times, a fast convergence value can be achieved. According to the test results in this part, in the subsequent experiments, the default value of the pretraining epoch and funetraining epoch will be five.

5.4.3. Batch Size

The choice of Batch size will also have a great impact on the results. A large minibatch size can bring more stable training results and faster convergence speed. But at the same time, a large batch size will put higher requirements on the memory size, and this will cause problems in the optimization and generalization of deep learning (large generalization gap). Although a small minibatch size can provide better accuracy, it has high requirements for training time.

In order to measure the influence of the choice of different batch size on the accuracy results, we conduct experiments to change the value of batch size from small to large. The smaller batch size will be closer to online learning and the larger batch size will be closer to full batch learning. For the network size, we use 300-128-32 and 500-128-32 for the two test data sets used in the experiment. For the training strategy, we use a value of 5-5 for both pretraining and funetraining epochs.

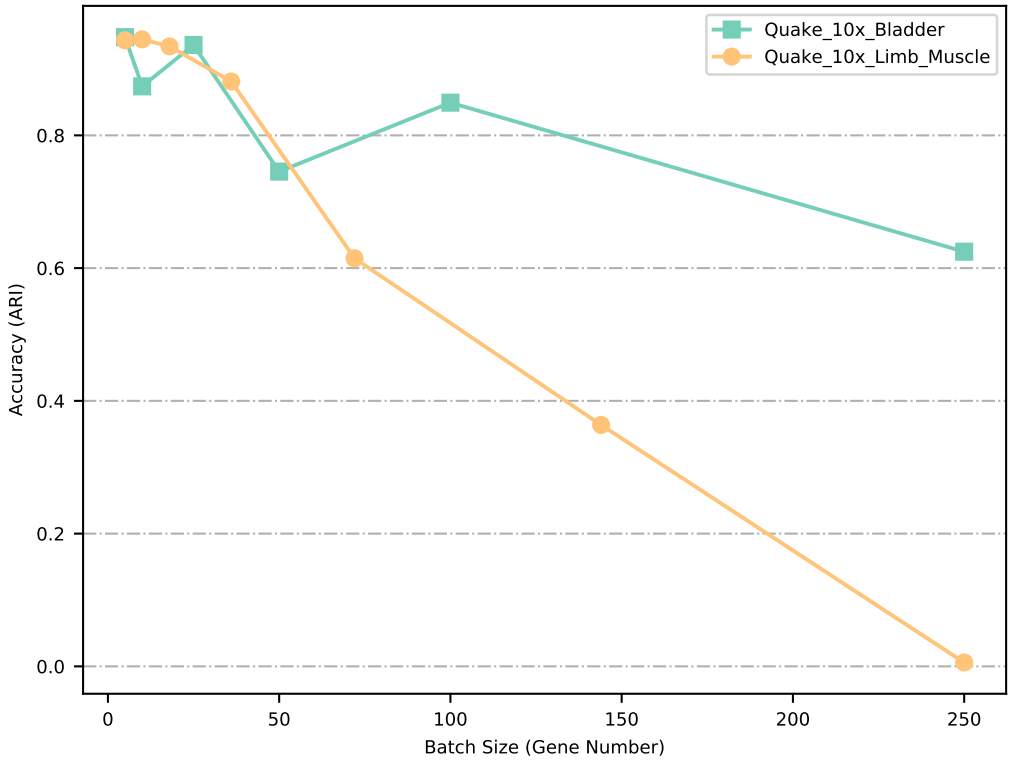


Figure 5.6: Measurement results of accuracy (ARI) for different batch size.

The measurement results for the two data sets are shown in Figure 5.6. ARI is around 0.9 when batch size is less than 10 for the two data set. The measurement result shows that the batch size does have a huge impact on the accuracy results. The displayed results are consistent with our theoretical conjecture, that is, as the batch size increases, the result accuracy will gradually decrease. The two different data sets exhibit different accuracy reduction speeds. The reason for this may be the sensitivity of different data set types to batch size change. However, the reduction in batch size comes at the expense of slower computing speed. Therefore, 25 and 18 are recommended to be used as the batch size for further experiments for the Bladder and Limb_Muscle data sets, respectively.

5.5. Memory Allocation

The memory allocation of the SGX-SczíDesk system in this project will be tested here. The selected data sets are still Quake_10x_Bladder and Quake_10x_Limb_Muscle. The parameters that affect the memory space required by the algorithm are divided into two parts: network size and batch size. A network with a larger network size means that there are more weighted matrices needed to be stored and iterated in the calculation, and a large batch size means that the memory space needs to store a larger amount of data in each training step.

5.5.1. Network Size

The size of the network size should affect the memory space occupation of the algorithm. Here we test the bladder and limb muscle data sets respectively. All training epochs are selected for five times. The testing values of the network are all numbers based on two for better data compression and decompression. Six test results of five-layer deep learning networks in different network sizes were collected, and the average memory usage was read out through the memory usage in the task manager.

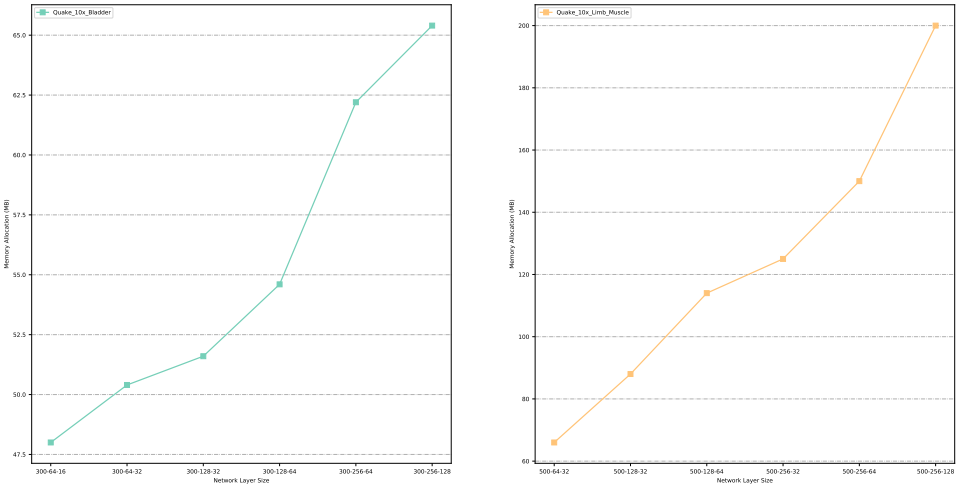


Figure 5.7: Measurement results of memory allocation (MB) for different network size.

The measurement results for the two data set are shown in Figure 5.7. The graphs clearly reflect that as the network size continues to increase, the overall memory space occupied by the algorithm is constantly increasing. But we can also find that the increase in network size does not have a significant impact on the memory space. Especially when the layer with the most units remains unchanged, the change in the number of internal layers will not cause a large burden on the overall memory usage. The increment of the memory allocation is only several MB when doubling the size of internal layer. The reason for this phenomenon is that the change in the size of the learning network will only increase the scale of partial weighted matrix value, and the number of this part will not increase exponentially as the amount of data increases.

5.5.2. Batch Size

The size of the batch size should affect the memory space occupied by the algorithm. Here we test the bladder and limb muscle data sets respectively. Data of six five-layer deep learning networks of different batch sizes are collected, and the average memory usage was read out through the memory usage of the task manager. For the network size, we use 300-128-32 and 500-128-32 for the two test data sets used in the experiment. For the training strategy, we use a value of 5-5 for both pretraining and funetraining epochs.

The measurement results for the two data sets are shown in Figure 5.8. The graphs clearly reflect that as the batch size continues to increase, the overall memory space occupied by the algorithm is

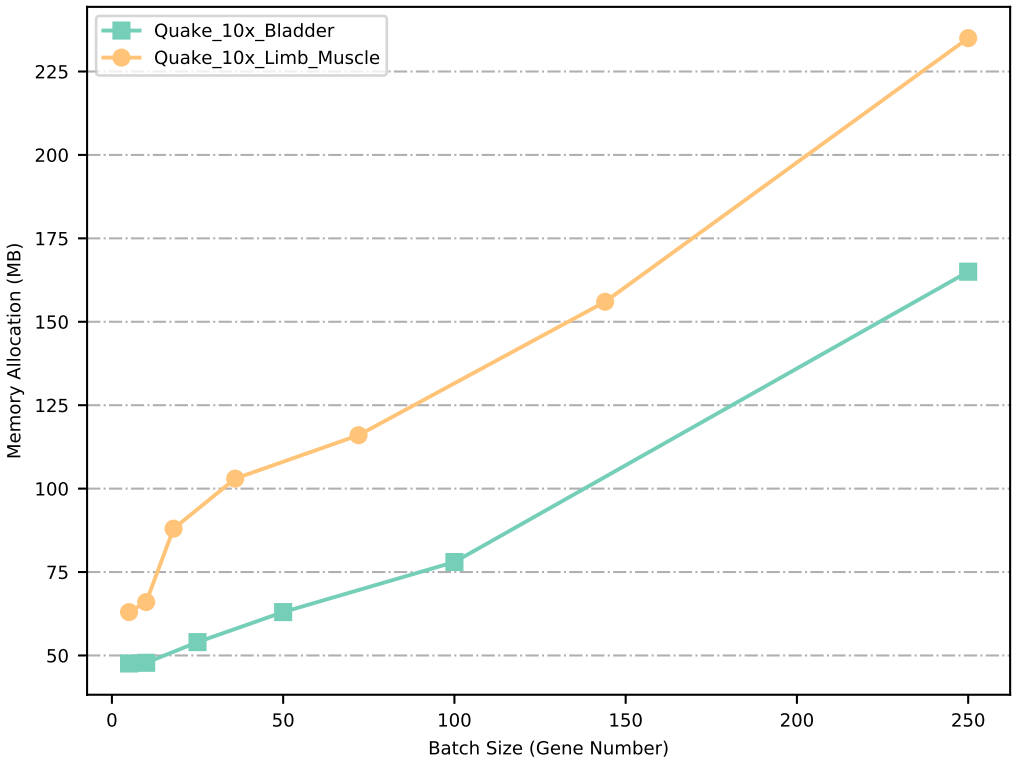


Figure 5.8: Measurement results of memory allocation (MB) for different batch size.

constantly improving. At the same time, the increase in batch size will have a significant impact on the increase in memory space usage. The reason is that batch size corresponds to the size of the input data in the network during each training. And the size of this value will directly determine the amount of the memory space required for each training epoch. Therefore, in view of the limited space of SGX, it can be said that the relatively small batch size is more in line with our needs for the project.

5.6. Running Time

The running time of the algorithm is an important parameter for the hardware test of the data set in this project. In the previous benchmark test, we can already find that when the algorithm runs in Intel SGX, it will take about 15-20 times the running time outside of SGX. This part will measure the time required for the SGX-SzciDesk system in detail. In order to avoid duplication of work, we will only use Quake_10x_Bladder for and analysis in this part. This data set contains 2500 cells, and the number of high variable genes is selected as 300.

The impact on running time mainly includes two parts: network size and batch size. In the part of network size measurement, we will separately measure the three main parts of the algorithm: pre-training, K-means and funetraining's contribution with different network sizes. At the same time, the total running times of different network sizes are recorded and the reasons are analyzed. In the batch size measurement part, we will measure the running time under different batch sizes and analyze the reasons.

5.6.1. Network Size

The size of the network size should affect the running time of the system. Date set Quake_10x_Bladder is used for the measurement with network size increasing from 300-64-16 to 300-256-64. The training epoch are all five and K-means iteration is five with batch size 25. Pretraining, K-means and funetraining running time are tested and recorded separately.

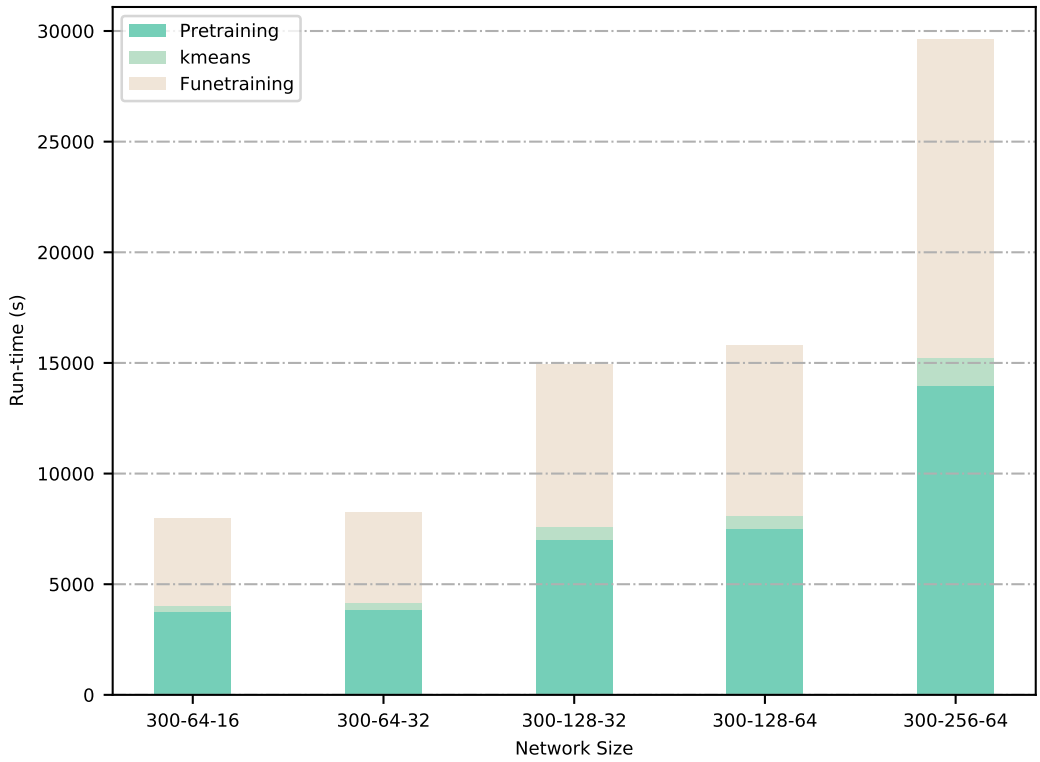


Figure 5.9: Running time of the system with different network size for data set Quake_10x_Bladder.

The measurement results for the data set are shown in Figure 5.9. It can be observed that the training time occupies most of the total running time. At the same time, the time occupied by the two training steps are almost the same for all measurements. This is because that the training parameters of the two training are similar. It can also be observed that as the size of the second layer network increases, the total running time increases in proportion to it. This is because the second-layer network

has an important influence on the size of the overall training parameter, and the increase in its size directly corresponds to the increase of the weight matrix processed in the training network. The third layer network size does not contribute much to the overall computing time, because this size will only affect the weight matrix size of the encoder and decoder from the second to the third layer. Therefore, we can say that the increase in the number of network layers closer to the input/output layer has a more negative effect on the total running time especially for large data sets in Intel SGX.

5.6.2. Batch Size

Batch size also has an impact on running time. A large batch size means a smaller slice number in each training process. In this experiment, we choose a five-layer neural network with a network size of 300-64-16 for data testing. The training epoch are all five and K-means iteration is five with batch size 25. The batch size increase from 5 to 100.

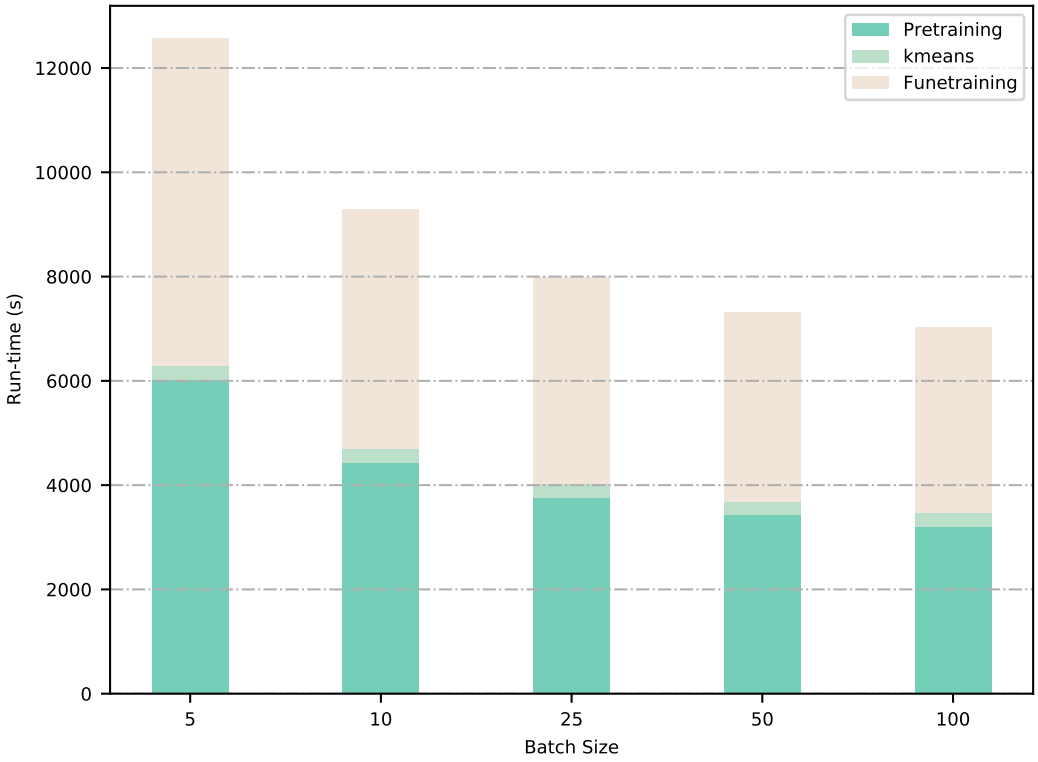


Figure 5.10: Running time of the system with different batch size for data set Quake_10x_Bladder.

The measurement results for the data set are shown in Figure 5.10. It can be observed that with the increment of batch size, the running time decreases. The training time is reduced as the batch size increases, while the part of the K-means time remains unchanged. There are generally two reasons for this result. The first aspect is the improvement of memory utilization. Large batch size can improve the parallelization efficiency in the process of large matrix multiplication. The second aspect is that the number of iterations required to run an epoch (full data set) is reduced, which can further speed up the processing of the same amount of data. But at the same time, as the batch size increases, the size of the array passed in the network in each training epoch also increases. Therefore, the speed of overall running time decrements also decreases.

5.7. Full system parameter selection

Based on the previous measurement results, we can make a summary. When the system of this project is actually running in the SGX environment. There are three main parameters to consider: accuracy, memory allocation and running time. These three parameters will change with the size of the training network, the number of training epochs and the batch size. The requirement for accuracy is that ARI is greater than 0.5 under normal circumstances. The limitation on memory mainly comes from the maximum heap size of SGX, where this size is about 90MB. There is generally no special limitation on training time, but for a data set composed of several thousands cells, the training time should generally not exceed 24 hours.

For accuracy. The network size has local optimal values for different data set sizes. As the training epoch increases, the accuracy will increase significantly. A smaller batch size can ensure higher accuracy. For memory allocation. The increase in network size and batch size will significantly increase memory. For running time, the increase in network size and training epoch will significantly increase the burden of running time, and a smaller batch size will also increase it.

Therefore, in general, increasing the network size may improve the accuracy of the results, but it will bring higher memory allocation and larger running time. Increasing training epoch will greatly improve the accuracy, but the running time will grow proportionally. A small batch size will bring higher precision and smaller memory allocation, but it will have a negative impact on running time.

We choose a balanced solution here to try to achieve the trade-off between accuracy, memory allocation and running time. For the Quake_10x_Bladder data set, we chose 300 high-variable genes, a five-layer learning network with layer size 300-128-32, batch size 5 and training epoch 25 for each training step. For the Quake_10x_Limb_Muscle data set, we chose 500 high-variable genes, a five-layer learning network with layer size 500-128-32, batch size 18 and training epoch 5 for each training step.

Comparison between the test results of this parameter selection with the original SczïDesk algorithm, CIDR, SIMLR and RaceID can show its advantages and disadvantages. The measurement results for the data set are shown in Figure 5.11. ARI is around 0.7 for the two data sets. Time consumption is around 4 hours and 6 hours respectively with memory allocation less than 90MB.

In benchmark measurement part we have concluded that there is an overhead ratio of around 20 between same operation code inside and outside SGX. Therefore, we can take the division of this fixed overhead and make comparison between the system with several existing methods. It can be observed that SGX-SczïDesk system achieves a same running time level with other existing methods. In order to achieve the accuracy and safety of the overall system, the running time of the system has made a big sacrifice. The required memory space is very small compared to other common solutions. The result is shown in Figure 5.12. The overall system result meets our application requirements.

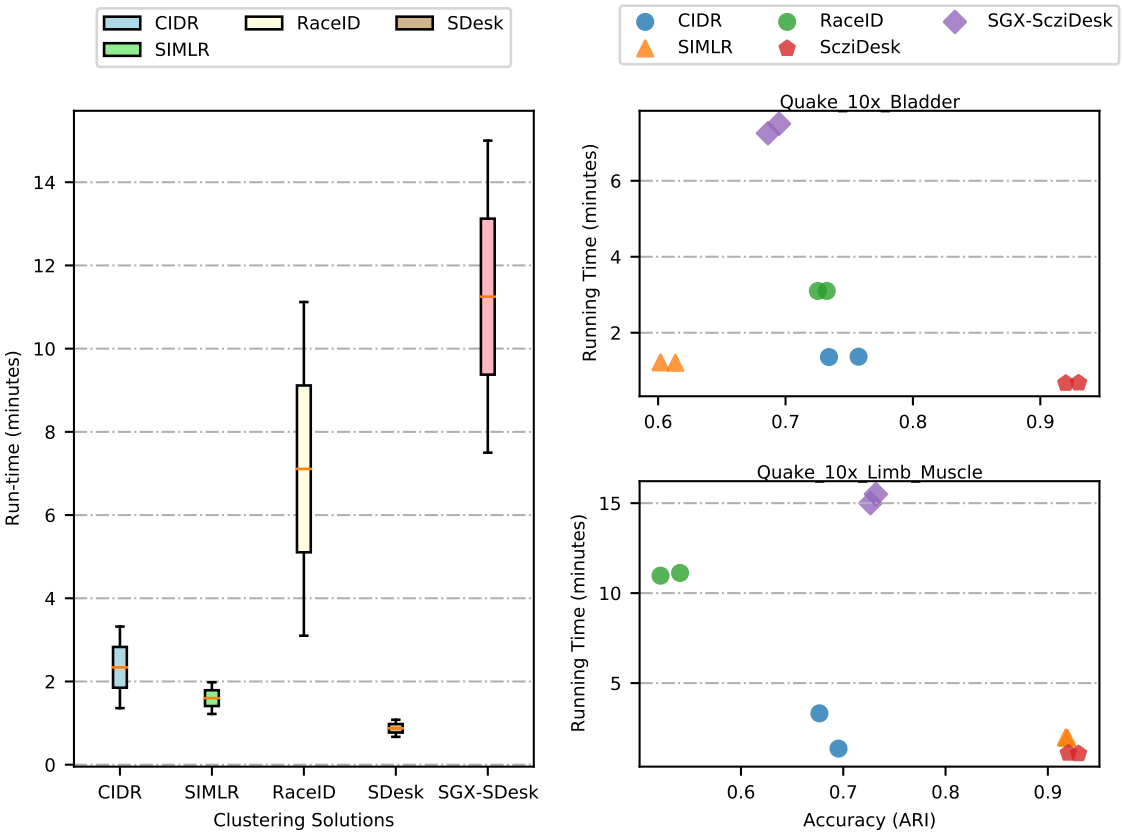


Figure 5.11: Measurement results of running time and ARI for different clustering solutions on data set Quake_10x_Bladder and Quake_10x_Limb_Muscle. Running time for SGX-SczíDesk is 1/20 of the original time.

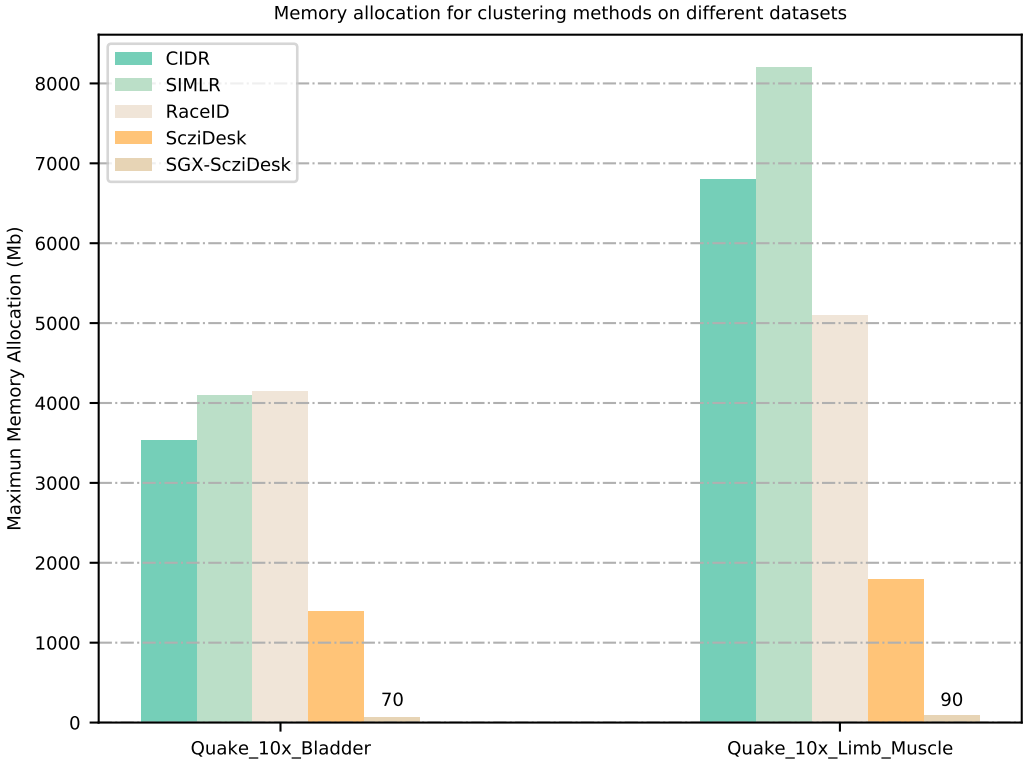


Figure 5.12: Memory allocation for clustering solutions on different data sets

6

Conclusions and recommendations

6.1. Conclusion

In this thesis, we have analyzed the implementation of RNA sequence data using Intel SGX secure hardware. We are now able to answer the research questions in the first chapter:

1. *How to enable Intel SGX hardware in a specific trusted system?*

   The application of SGX mainly involves the setting of related hardware and the help of third-party platforms. No matter in the operating environment of Windows or Linux, after starting the SGX setting in the bios interface, we need to download the SGX SDK, SGX PSW and SGX driver to ensure the normal call of the SGX hardware space by the CPU. This project also uses Fortanix-EDP as the development platform. This platform will provide a direct interface between the Rust language project and the enclave memory space.

2. *How to develop deep learning algorithms in SGX?*

   The development of deep learning algorithms is mainly designed in three aspects. Development of auxiliary functions, definition of loss function and construction of deep learning networks. In the development of auxiliary functions, the core lies in the definition of tensor-related operations and the writing of gradient functions. The development of the loss function is mainly divided into two parts, the loss function for the network parameters of the neural network and the overall loss function. The development of the deep learning network includes three aspects, the construction of the autoencoder network, the application of K-means clustering algorithm and the realization of batch training.

3. *How to ensure the security of data and the normal transmission of data?*

   In order to protect the security of data transmission, the AES128 encryption scheme is applied to the transmission of data between the enclave and the client side. The data transmission between these two ends is based on the TCP protocol.

4. *How do such SGX compute pipelines compare with non-SGX state-of-the-art work?*

   Two standard data sets are tested during the measurement process. Different system parameters are also analyzed and discussed for accuracy, time requirements and memory space occupation. Results show that this system can have comparable results with other traditional clustering schemes in terms of accuracy without consuming too long running time. At the same time, this system shows very strong security and low memory allocation.

6.2. Recommendations

We have identified several possible directions for continuing our work.

Implementation Approach

In our project application, we adopted a plan to place both training and prediction in a safe space. This solution brings high data and algorithm security but at the same time it also brings a great burden on the running time. We have noticed that there are different research topics focusing on how to split the training and prediction parts and only put the prediction part that requires less resources into the SGX security hardware for operation. This may become an important possible topic for optimizing the SGX based neural network development process.

Memory Allocation

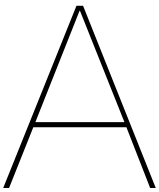
In our project application, we notice that an important factor limiting the application of the algorithm is the total memory size of the SGX hardware. When the input data and network size become larger, it becomes very difficult to run the project in the SGX hardware. Therefore, how to effectively reduce the memory space required by the algorithm under the premise of ensuring accuracy and running time can become a potential issue.

Clustering Algorithm

A core part of this project is to perform K-means algorithm clustering on latent space. We also notice that there are many clustering algorithms that have been proven to be superior to traditional K-means, so the possible optimization of this part will greatly improve the performance of the overall system. The choice in the clustering algorithm part can become a potential research direction.

Security Instruction

In this project, the method to ensure the security of data and algorithms is the security features of the AES128 encryption system and enclave itself. We also found that for the SGX system, Intel also provides other security enhancement solutions for non-local host users and processors for security enhancement. For these possible SGX unique safety instructions, we can conduct potential research and measurement for the impact of these safety instructions on performance and accuracy.



Appendix: Contribution to Rust-autograd

Reported issues

Rust-autograd-46: Bug for g.argmax
Link: `https://github.com/raskr/rust-autograd/issues/46`

Rust-autograd-45: Alternatives for tf.where()
Link: `https://github.com/raskr/rust-autograd/issues/45`

Rust-autograd-44: Gradient error for tensor of different dimensions
Link: `https://github.com/raskr/rust-autograd/issues/44`

Rust-autograd-43: Support for lgamma function
Link: `https://github.com/raskr/rust-autograd/issues/43`

B

Appendix: SGX Set-up Process Under Linux Environment

The basic hardware enablement process in the Linux environment is more complicated. And the official instruction may have many problems in actual use, here is a start-up process used in this project for reference.

Build and Install the Intel® SGX Driver. There are two ways to install the SGX driver, one is to run the executable file to install directly by default, and the other is to install the source code. Source code installation is more complicated and prone to problems. It is recommended to run the executable file to install directly by default. Proceed as follows:

1. Go to Linux-sgx-repo and download the three files (download to the /home/<username>/Downloads folder by default).
2. Go to the download folder and enter

   ```
   sudo chmod 777 sgx_linux_x64_driver_2.11.0_0373e2e.bin
   ```

   to give this .bin file execute permissions
3. Run this .bin file with:

   ```
   sudo ./sgx_linux_x64_driver_2.11.0_0373e2e.bin
   ```

Build and install the Intel® SGX SDK and Intel® SGX PSW Package. The process is shown as follows:

1. Install the tools needed to compile the SGX SDK with command:

   ```
   sudo apt-get install libssl-dev libcurl4-openssl-dev protobuf-compiler
    libprotobuf-dev debhelper cmake reprepro unzip
   ```

2. Get the source code from the git with:

   ```
   git clone https://github.com/intel/linux-sgx.git
   cd linux-sgx && make preparation
   ```

If the following error is reported: No such file or directory exists. That comes from the lack of dcap module in your system. Dcap is actually a security basic module of RA in ECDSA mode. Intel recommends installing it when installing the driver (that is, there may be a statement directly to write the response package Under this ./external/dcap_source folder). But the prerequisite for this service to work is that the CPU supports FLC. However, 6th Intel processor do not support FLC. The current solution is to down the entire project on sgx-driver-new, and then copy the two folders ./QuoteVerification and ./QuoteGeneration directly to ./ Under external, then run make preparation again.

3. Enter the ./linux-sgx folder (in fact, it should have been in this folder all the time), and execute make sdk and make sdk_install_pkg from the command line

4. Install SGX SDK with:

```
sudo apt-get install build-essential python
./sgx_linux_x64_sdk_${version}.bin
source ${sgx-sdk-install-path}/environment
```

5. Install SGX PSW with:

```
 echo'deb [arch=amd64] https://download.01.org/intel-sgx/sgx_repo
/ubuntu bionic main'| sudo tee /etc/apt/sources.list.d/intel-sgx.list

wegt -q0 - https://download.01.org/intel-sgx/sgx_repo/ubuntu/intel-sgx-deb.key
 | sudo apt-key add -
```

If you see [ok] after a while, the operation is successful. SGX PSW provides three services: Launch, EPID-based attestation and Algorithm agnostic attestation. Starting from version 2.8, PSW has been split into these three small services, which can be installed separately. After installation, you can find a sgx-asem-services directory in the /opt/intel/ directory.

Bibliography

[1] F. A. Wolf, P. Angerer, and F. J. Theis, "SCANPY: large-scale single-cell gene expression data analysis," *Genome Biology*, vol. 19, p. 15, Dec. 2018.

[2] "The Magic of Intel's SGX. A Tutorial on Programming a Secure… | by Daniel Ehnes | C³AI | Medium."

[3] "Architecture | Rust EDP."

[4] "Advanced Encryption Standard (AES)."

[5] K. Shekhar, S. W. Lapan, I. E. Whitney, N. M. Tran, E. Z. Macosko, M. Kowalczyk, X. Adiconis, J. Z. Levin, J. Nemesh, M. Goldman, S. A. McCarroll, C. L. Cepko, A. Regev, and J. R. Sanes, "Comprehensive Classification of Retinal Bipolar Neurons by Single-Cell Transcriptomics," *Cell*, vol. 166, pp. 1308–1323.e30, Aug. 2016.

[6] M. Hong, S. Tao, L. Zhang, L.-T. Diao, X. Huang, S. Huang, S.-J. Xie, Z.-D. Xiao, and H. Zhang, "RNA sequencing: new technologies and applications in cancer research," *Journal of Hematology & Oncology*, vol. 13, p. 166, Dec. 2020.

[7] S. Aibar, C. B. González-Blas, T. Moerman, V. A. Huynh-Thu, H. Imrichova, G. Hulselmans, F. Rambow, J.-C. Marine, P. Geurts, J. Aerts, J. van den Oord, Z. K. Atak, J. Wouters, and S. Aerts, "SCENIC: single-cell regulatory network inference and clustering," *Nature Methods*, vol. 14, pp. 1083–1086, Nov. 2017.

[8] D. A. Jaitin, E. Kenigsberg, H. Keren-Shaul, N. Elefant, F. Paul, I. Zaretsky, A. Mildner, N. Cohen, S. Jung, A. Tanay, and I. Amit, "Massively Parallel Single-Cell RNA-Seq for Marker-Free Decomposition of Tissues into Cell Types," *Science*, vol. 343, pp. 776–779, Feb. 2014.

[9] L. Chen, W. Wang, Y. Zhai, and M. Deng, "Deep soft K-means clustering with self-training for single-cell RNA sequence data," *NAR Genomics and Bioinformatics*, vol. 2, p. lqaa039, June 2020.

[10] D. Greene, P. Cunningham, and R. Mayer, "Unsupervised Learning and Clustering," in *Machine Learning Techniques for Multimedia* (M. Cord and P. Cunningham, eds.), pp. 51–90, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Series Title: Cognitive Technologies.

[11] I. Jamail and A. Moussa, "Current State-of-the-Art of Clustering Methods for Gene Expression Data with RNA-Seq," in *Applications of Pattern Recognition* (C. M. Travieso-Gonzalez, ed.), IntechOpen, July 2021.

[12] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A.-R. Sadeghi, "HardIDX: Practical and Secure Index with SGX," in *Data and Applications Security and Privacy XXXI* (G. Livraga and S. Zhu, eds.), vol. 10359, pp. 386–408, Cham: Springer International Publishing, 2017. Series Title: Lecture Notes in Computer Science.

[13] Y. Wang, J. Wang, and X. Chen, "Secure searchable encryption: a survey," *Journal of Communications and Information Networks*, vol. 1, pp. 52–65, Dec. 2016.

[14] "iDASH Genome Privacy and Security Workshop (secure genome analysis competition) - Xiaoqian Jiang."

[15] D. Wang and S. Bodovitz, "Single cell analysis: the new frontier in 'omics'," *Trends in Biotechnology*, vol. 28, pp. 281–290, June 2010.

[16] J. Eberwine, J.-Y. Sul, T. Bartfai, and J. Kim, “The promise of single-cell sequencing,” *Nature Methods*, vol. 11, pp. 25–27, Jan. 2014.

[17] E. Pennisi, “Chronicling embryos, cell by cell, gene by gene,” *Science*, vol. 360, pp. 367–367, Apr. 2018.

[18] T. Ilicic, J. K. Kim, A. A. Kolodziejczyk, F. O. Bagger, D. J. McCarthy, J. C. Marioni, and S. A. Teichmann, “Classification of low quality cells from single-cell RNA-seq data,” *Genome Biology*, vol. 17, p. 29, Dec. 2016.

[19] P. Brennecke, S. Anders, J. K. Kim, A. A. Kołodziejczyk, X. Zhang, V. Proserpio, B. Baying, V. Benes, S. A. Teichmann, J. C. Marioni, and M. G. Heisler, “Accounting for technical noise in single-cell RNA-seq experiments,” *Nature Methods*, vol. 10, pp. 1093–1095, Nov. 2013.

[20] “Intel® SGX for Dummies (Intel® SGX Design Objectives).”

[21] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, “Innovative instructions and software model for isolated execution,” in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy - HASP ’13*, (Tel-Aviv, Israel), pp. 1–1, ACM Press, 2013.

[22] “Impressions of Intel® SGX performance | by Danny Harnik | Medium.”

[23] E. Macosko, A. Basu, R. Satija, J. Nemesh, K. Shekhar, M. Goldman, I. Tirosh, A. Bialas, N. Kamitaki, E. Martersteck, J. Trombetta, D. Weitz, J. Sanes, A. Shalek, A. Regev, and S. McCarroll, “Highly Parallel Genome-wide Expression Profiling of Individual Cells Using Nanoliter Droplets,” *Cell*, vol. 161, pp. 1202–1214, May 2015.

[24] P. V. Kharchenko, L. Silberstein, and D. T. Scadden, “Bayesian approach to single-cell differential expression analysis,” *Nature Methods*, vol. 11, pp. 740–742, July 2014.

[25] Z. Ji and H. Ji, “TSCAN: Pseudo-time reconstruction and evaluation in single-cell RNA-seq analysis,” *Nucleic Acids Research*, vol. 44, pp. e117–e117, July 2016.

[26] M. P. Kumar, J. Du, G. Lagoudas, Y. Jiao, A. Sawyer, D. C. Drummond, D. A. Lauffenburger, and A. Raue, “Analysis of Single-Cell RNA-Seq Identifies Cell-Cell Communication Associated with Tumor Characteristics,” *Cell Reports*, vol. 25, pp. 1458–1468.e4, Nov. 2018.

[27] P. Lin, M. Troup, and J. W. K. Ho, “CIDR: Ultrafast and accurate clustering through imputation for single-cell RNA-seq data,” *Genome Biology*, vol. 18, p. 59, Dec. 2017.

[28] B. Wang, J. Zhu, E. Pierson, D. Ramazzotti, and S. Batzoglou, “Visualization and analysis of single-cell RNA-seq data by kernel-based similarity learning,” *Nature Methods*, vol. 14, pp. 414–416, Apr. 2017.

[29] D. Grün, M. Muraro, J.-C. Boisset, K. Wiebrands, A. Lyubimova, G. Dharmadhikari, M. van den Born, J. van Es, E. Jansen, H. Clevers, E. de Koning, and A. van Oudenaarden, “De Novo Prediction of Stem Cell Identity using Single-Cell Transcriptome Data,” *Cell Stem Cell*, vol. 19, pp. 266–277, Aug. 2016.

[30] G. Eraslan, L. M. Simon, M. Mircea, N. S. Mueller, and F. J. Theis, “Single-cell RNA-seq denoising using a deep count autoencoder,” *Nature Communications*, vol. 10, p. 390, Dec. 2019.

[31] C. H. Grønbech, M. F. Vording, P. N. Timshel, C. K. Sønderby, T. H. Pers, and O. Winther, “scVAE: variational auto-encoders for single-cell gene expression data,” *Bioinformatics*, vol. 36, pp. 4415–4422, Aug. 2020.

[32] D. Grün, A. Lyubimova, L. Kester, K. Wiebrands, O. Basak, N. Sasaki, H. Clevers, and A. van Oudenaarden, “Single-cell messenger RNA sequencing reveals rare intestinal cell types,” *Nature*, vol. 525, pp. 251–255, Sept. 2015.

[33] S. E. El-Khamy, R. A. Sadek, and M. A. El-Khoreby, “An efficient brain mass detection with adaptive clustered based fuzzy C-mean and thresholding,” in *2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, (Kuala Lumpur, Malaysia), pp. 429–433, IEEE, Oct. 2015.

[34] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Jan. 2017. arXiv: 1412.6980.

[35] L. v. d. Maaten and G. Hinton, “Visualizing Data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[36] J. Xie, R. Girshick, and A. Farhadi, “Unsupervised Deep Embedding for Clustering Analysis,” *arXiv:1511.06335 [cs]*, May 2016. arXiv: 1511.06335.

[37] “Apache TVM.”

[38] “Home Page - 10x Genomics.”

[39] J. M. Santos and M. Embrechts, “On the Use of the Adjusted Rand Index as a Metric for Evaluating Supervised Classification,” in *Artificial Neural Networks – ICANN 2009* (C. Alippi, M. Polycarpou, C. Panayiotou, and G. Ellinas, eds.), vol. 5769, pp. 175–184, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.