

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY  
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY  
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



## **REPORT MULTIDISCIPLINARY PROJECT**

---

**DATA STREAMING: MISSING DATA IMPUTATION  
USING GCIMPUTE AND AUTOMQ**

---

**SUPERVISOR** : Diệp Thanh Đăng, Ph.D.

—o0o—

**STUDENTS** : Lê Hữu Anh Quân - 1952942  
Nguyễn Tiến Thành - 2053437  
Nguyễn Tấn Nhật - 2152832  
Cao Võ Hoài Phúc - 2053332

HO CHI MINH CITY, DEC 2024



# Acknowledgement

Our group would like to express our sincere appreciation to Mr. Diep Thanh Dang for his unwavering support and invaluable guidance, which has greatly contributed to the success of our research. This project aims to design and implement a comprehensive IoT data streaming system to address the critical challenge of missing data. The goals include setting up and configuring a scalable data streaming platform (AutoMQ) for IoT devices, applying statistical imputation methods using gcimpute to manage incomplete data effectively, building an integrated system that combines both streaming and imputation components, and creating comprehensive documentation for system deployment, configuration, and evaluation.



# Abstract

Data streaming using the `gcimpute` package through the AutoMQ platform enables seamless real-time data processing and imputation for dynamic datasets. The `gcimpute` package, known for its efficient handling of missing data, can be integrated with AutoMQ's scalable message queuing and streaming infrastructure. This combination ensures that data streams are cleaned and prepared for analysis on-the-fly, maintaining consistency and reliability in downstream applications. By leveraging the power of AutoMQ, `gcimpute` can handle high-throughput data pipelines, making it ideal for applications in finance, healthcare, and IoT, where real-time decision-making is critical.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                                       | <b>1</b>  |
| 1.1      | Motivation . . . . .                                      | 1         |
| 1.2      | Goals . . . . .   | 3         |
| 1.3      | Thesis structure . . . . .                                | 3         |
| <b>2</b> | <b>Theoretical Background</b>                             | <b>4</b>  |
| 2.1      | Definition . . . . .                                      | 4         |
| 2.2      | Literature Review . . . . .                               | 5         |
| 2.2.1    | Missing Data Imputation . . . . .                         | 5         |
| 2.2.2    | Gaussian Copula Model for Missing Data . . . . .          | 5         |
| 2.2.3    | Integration of Streaming and Imputation Systems . . . . . | 6         |
| 2.3      | Algorithm . . . . .                                       | 6         |
| 2.3.1    | Gaussian Copula Model . . . . .                           | 6         |
| 2.3.2    | Expectation Maximization (EM) Algorithm . . . . .         | 8         |
| 2.3.3    | Scaled Mean Absolute Error (SMAE) . . . . .               | 10        |
| 2.3.4    | Mini-Batch and Online Training . . . . .                  | 11        |
| 2.3.5    | Low-Rank Gaussian Copula (LRGC) . . . . .                 | 12        |
| 2.4      | Missing data imputation process: . . . . .                | 13        |
| <b>3</b> | <b>Data Streaming Platform</b>                            | <b>15</b> |
| 3.1      | Introduction . . . . .                                    | 15        |
| 3.2      | Kafka is the Beginning . . . . .                          | 15        |



|          |   |           |
|----------|---|-----------|
| 3.3      | Technical Advantages . . . . .                        | 16        |
| 3.3.1    | Cost Efficiency . . . . .                             | 16        |
| 3.3.2    | Elasticity and Scalability . . . . .                  | 17        |
| 3.3.3    | Performance . . . . .                                 | 17        |
| 3.3.4    | Ease of Migration and Compatibility . . . . .         | 17        |
| 3.4      | Truly Cloud-Native Architecture of AutoMQ . . . . .   | 18        |
| 3.4.1    | AutoMQ Shared Storage Architecture Overview . . . . . | 18        |
| 3.4.2    | Stateless Brokers . . . . .                           | 18        |
| 3.4.3    | Automated Elasticity . . . . .                        | 20        |
| 3.4.4    | Multi-Cloud Adaptability . . . . .                    | 20        |
| 3.5      | Deployment . . . . .                                  | 21        |
| 3.5.1    | Missing Data Imputation . . . . .                     | 21        |
| 3.5.1.1  | Preparation steps . . . . .                           | 21        |
| 3.5.1.2  | Example 1 . . . . .                                   | 22        |
| 3.5.1.3  | Example 2 . . . . .                                   | 25        |
| 3.5.1.4  | Example 3 . . . . .                                   | 27        |
| 3.5.2    | Data Streaming . . . . .                              | 28        |
| <b>4</b> | <b>Evaluation</b>                                     | <b>29</b> |
| 4.1      | Missing Data Imputation . . . . .                     | 29        |
| 4.1.1    | Achievements . . . . .                                | 29        |
| 4.1.2    | Challenges . . . . .                                  | 30        |
| 4.1.3    | Recommendations for Improvement . . . . .             | 30        |
| 4.2      | Evaluation about AutoMQ . . . . .                     | 31        |
| 4.2.1    | Achievements . . . . .                                | 31        |
| 4.2.2    | Challenges . . . . .                                  | 32        |
| 4.2.3    | Recommendations for Improvement . . . . .             | 32        |
|          | <b>References</b>                                     | <b>33</b> |

# List of Figures

|            |  |    |
|------------|--|----|
| Figure 2.1 | A sample dataset with missing values . . . . .                   | 7  |
| Figure 2.2 | Rank and transform age, income, and marital status . . . . .     | 7  |
| Figure 2.3 | Fill the given table by imputed data . . . . .                   | 8  |
| Figure 2.4 | Expectation Maximization flowchart . . . . .                     | 9  |
| Figure 2.5 | Scaled Mean Absolute Error formula . . . . .                     | 10 |
| Figure 2.6 | Comparison of Mini-Batch and Standard Training: Key Differences. | 11 |
| Figure 2.7 | Mini-Batch Training process . . . . .                            | 12 |
| Figure 2.8 | Low-Rank Gaussian Copula process . . . . .                       | 12 |
| Figure 2.9 | Missing data imputation process with algorithms . . . . .        | 13 |
| Figure 3.1 | gcimpute is a Python package for missing value imputation . . .  | 21 |
| Figure 3.2 | The gcimpute package folder is opened in Visual Studio Code . .  | 22 |
| Figure 3.3 | The first example's output in a terminal . . . . .               | 24 |
| Figure 3.4 | The imputed data saved in a csv file . . . . .                   | 24 |
| Figure 3.5 | The result of Scaled Mean Absolute Error . . . . .               | 25 |
| Figure 3.6 | The second example's output in a terminal . . . . .              | 26 |
| Figure 3.7 | The third example's output in a terminal . . . . .               | 28 |

# Chapter 1

## Introduction

*In chapter 1, the overview, objectives and goals of the research's project are illustrated. The outline of the report is also presented.*

### 1.1 Motivation

The Internet of Things (IoT) has revolutionized the way we interact with and manage devices in various domains, including smart cities, healthcare, industrial automation, and home automation. By enabling devices to communicate and exchange data seamlessly, IoT systems have paved the way for more intelligent, automated, and efficient processes. However, the massive influx of data generated by IoT devices introduces significant challenges, particularly in terms of handling, streaming, and ensuring the reliability of this data. Missing data is one of the most persistent challenges in IoT systems, arising due to factors such as network disruptions, hardware malfunctions, and sensor inaccuracies. If left unaddressed, missing data can compromise the accuracy of analytics, decision-making processes, and the overall reliability of IoT systems.

Data streaming systems play a crucial role in managing and processing IoT data in real-time. They ensure that data flows continuously and efficiently between IoT devices, servers, and end-user applications. However, the dynamic and heterogeneous nature of



IoT environments makes data streaming inherently complex. A reliable streaming platform must accommodate varying data rates, provide scalability to handle large volumes of data, and ensure data integrity, even when faced with missing or incomplete entries. Addressing these challenges requires a robust data streaming solution complemented by advanced techniques to mitigate the impact of missing data.

Traditional methods of handling missing data, such as mean imputation or simple interpolation, often fall short in complex IoT environments. These methods fail to capture the underlying relationships and dependencies among variables, leading to inaccurate or biased results. Advanced statistical techniques, such as those based on Gaussian copula models, offer a more sophisticated approach by modeling the interactions between variables and providing probabilistic imputation. The `gcimpute` library, which leverages the Gaussian copula model, is particularly suited for handling mixed data types, streaming datasets, and large-scale imputation tasks. Its ability to provide confidence intervals for imputed values further enhances the reliability of the imputed data.

The integration of a robust data streaming platform like AutoMQ with an advanced imputation library such as `gcimpute` offers a promising solution to the challenges of IoT data management. AutoMQ's ability to handle real-time data flows and `gcimpute`'s capability to address missing data synergize to create a powerful system for IoT applications. This combination not only ensures the seamless transmission of data but also improves the quality and reliability of the insights derived from it.

In this context, this project is motivated by the pressing need to develop a unified system that combines data streaming and advanced imputation techniques to address the challenges of incomplete data in IoT. By achieving this, the project aims to contribute to the field of IoT by providing a scalable, reliable, and efficient framework for real-time data processing and analytics. This system has the potential to enhance the performance and





applicability of IoT solutions across various industries, paving the way for smarter and more robust IoT ecosystems.

## 1.2 Goals

This project aims to design and implement a comprehensive IoT data streaming system capable of addressing the critical issue of missing data. The goals include:

- Setting up and configuring a scalable data streaming platform (AutoMQ) for IoT devices.
- Applying statistical imputation methods using `gcimpute` to manage incomplete data effectively.
- Building an integrated system combining streaming and imputation components.
- Creating comprehensive documentation for system deployment, configuration, and evaluation.

## 1.3 Thesis structure

There are four chapters in this project:

- Chapter 1: Provides an overview of the project, including its motivation, goals, scope, and thesis structure.
- Chapter 2: Explains the definitions, foundational concepts, reviews relevant literature, and discusses algorithms for data streaming and imputation.
- Chapter 3: Describes the design, architecture, and deployment of the data streaming platform (AutoMQ) used in the project.
- Chapter 4: Presents the evaluation of each phase in our project, and conclusions drawn from the performance analysis of the system.

## Chapter 2

# Theoretical Background

### 2.1 Definition

**Data streaming** refers to the continuous flow of data generated and processed in real time. Unlike traditional batch processing, where data is collected, stored, and processed at specific intervals, streaming involves handling data as it is produced, enabling near-instantaneous analysis and decision-making. This method is widely used in applications such as IoT devices, financial transactions, social media feeds, and real-time analytics platforms. Technologies like Apache Kafka, Apache Flink, and AutoMQ facilitate efficient data ingestion, processing, and management. Data streaming is essential for modern systems that require low-latency operations, scalability, and the ability to process vast amounts of data in motion.

**Data imputation** is the process of replacing missing or incomplete data with substituted values to ensure the dataset is complete and reliable for analysis. Missing data can arise due to various reasons, such as sensor errors, incomplete surveys, or data corruption. Imputation techniques range from simple methods like replacing missing values with the mean, median, or mode to advanced methods such as k-nearest neighbors (KNN),



multiple imputation, or machine learning algorithms. Proper imputation preserves data integrity and reduces bias, ensuring more accurate statistical modeling or machine learning predictions. It is a critical step in data preprocessing, as incomplete data can negatively impact analytical outcomes.

## **2.2 Literature Review**

### **2.2.1 Missing Data Imputation**

Handling missing data has been a long-standing challenge in data analysis. Traditional methods such as mean substitution and k-nearest neighbors (KNN) imputation have been widely applied but often fail to capture the underlying data structure. Recent advancements, such as multiple imputation and machine learning-based methods, have shown promise in improving accuracy.

The `gcimpute` package, introduced by Zhao and Udell (2024) [1], models mixed data types using a Gaussian copula framework. This approach separates the marginal distribution of variables from their joint dependency structure, enabling robust imputation across continuous, binary, ordinal, and truncated data types. The method is particularly effective for high-dimensional and incomplete datasets, offering both single and multiple imputation options.

### **2.2.2 Gaussian Copula Model for Missing Data**

The Gaussian copula model offers a semiparametric approach for modeling multivariate dependencies. It transforms observed variables into latent Gaussian variables while preserving their marginal distributions. This model has demonstrated superior perfor-



mance in imputing missing data, particularly for mixed datasets where variable types vary significantly. Zhao and Udell (2024) [1] extended this model with computational optimizations, such as mini-batch training and low-rank approximations, to handle large-scale and streaming data efficiently.

### 2.2.3 Integration of Streaming and Imputation Systems

Combining real-time data streaming with statistical imputation presents a significant advancement in IoT data processing. Previous studies have explored the integration of streaming platforms with data preprocessing pipelines, but the coupling of platforms like AutoMQ with advanced imputation methods like `gcimpute` remains an emerging field. This project builds upon existing research to develop a unified system that addresses the challenges of scalability and real-time imputation.

## 2.3 Algorithm

### 2.3.1 Gaussian Copula Model

The **Gaussian copula** is a method for modeling dependencies between variables, even when these variables have different marginal distributions (such as categorical and continuous). Copulas separate the modeling of marginal distributions from the dependencies between variables, allowing for flexible handling of various types of data.

- **Copula Basics:** A copula is a function that links multivariate distributions to their marginal distributions. The Gaussian copula, specifically, is based on the multivariate normal (Gaussian) distribution. By using it, `gcimpute` can capture dependencies across mixed data types without needing to assume that the data follows a normal distribution.
- In the below example, copula basics help explain how `gcimpute` uses the Gaussian

copula model to handle dependencies among the mixed variables (Age, Income, MaritalStatus) when some values are missing.

**Figure 2.1:** A sample dataset with missing values

| Age | Income | MaritalStatus |
|-----|--------|---------------|
| 25  | 45000  | Single        |
| 32  | NaN    | Married       |
| NaN | 38000  | Divorced      |
| 45  | 55000  | NaN           |
| NaN | NaN    | Single        |

- **Gaussian Copula Process:**

- Rank Transformation: Each variable is transformed to follow a uniform distribution between 0 and 1. This transformation maps the raw data to a copula-friendly format.

**Figure 2.2:** Rank and transform age, income, and marital status

| Age (Rank)<br>Values:<br>[25, 32, 45] | Income (Rank)<br>Values: [45000,<br>38000, 55000] | MaritalStatus (Rank)<br>Values: Single = 1,<br>Married = 2,<br>Divorced = 3 | Age (Normal) | Income (Normal) | MaritalStatus (Normal) |
|---------------------------------------|---|---|--------------|-----------------|------------------------|
| 0                                     | 0.3333  | 0   | -Inf         | -0.4308         | -Inf                   |
| 0.3333                                | 0   | 0.3333  | -0.4308      | -Inf            | -0.4308                |
| 0.6667                                | 0.6667  | 0.6667  | 0.4308       | 0.4308          | 0.4308                 |

- Gaussian Copula Modeling: A Gaussian Copula is fit to the transformed data to

model the dependencies between variables. This allows the imputation process to leverage correlations effectively.

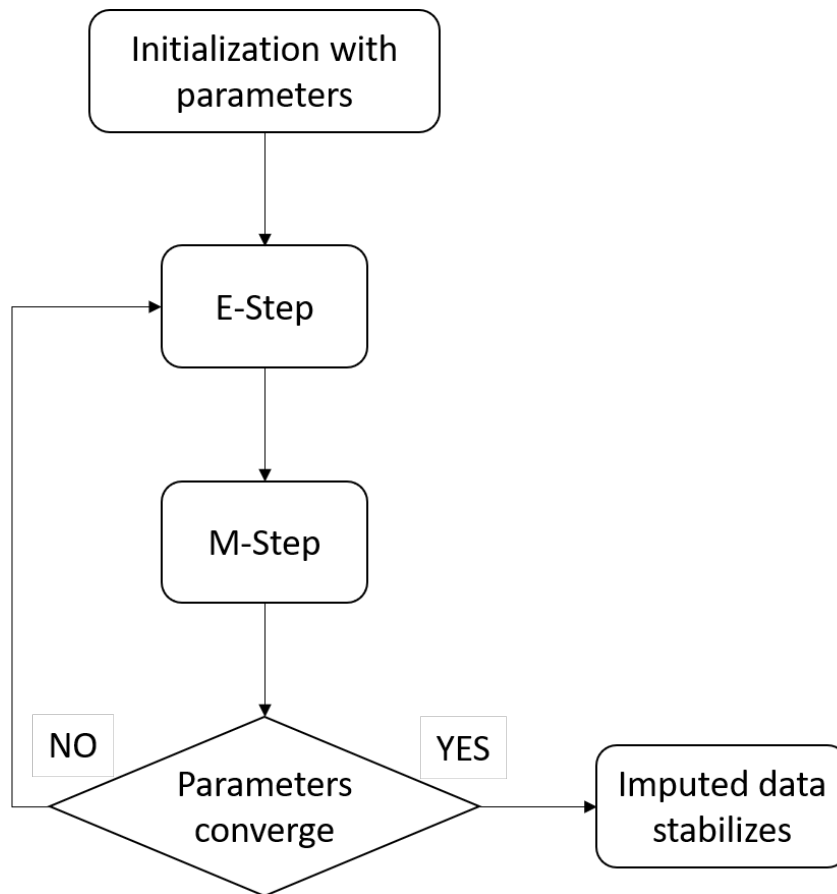
- Imputation of Missing Values: Missing values are imputed using the learned copula. This involves sampling from the joint distribution conditioned on the observed values.
- Inverse Rank Transformation: After imputation, the rank transformation is inverted to map the imputed values back to the original scale of the data.

| Age | Income | MaritalStatus |
|-----|--------|---------------|
| 25  | 45000  | Single        |
| 32  | 42000  | Married       |
| 29  | 38000  | Divorced      |
| 45  | 55000  | Married       |
| 30  | 41000  | Single        |

**Figure 2.3:** Fill the given table by imputed data

### 2.3.2 Expectation Maximization (EM) Algorithm

To estimate the copula correlation matrix and perform imputation, **gcimpute** uses the EM algorithm. In each iteration, the algorithm updates the expected covariance matrix of latent variables, then uses **maximum likelihood estimation (MLE)** to update the correlation matrix.



**Figure 2.4:** Expectation Maximization flowchart

Illustrating the above diagram:

- **Parameters Initialization:** Start with initial guesses for the parameters of the copula model, such as the copula correlation matrix or other distributional parameters.
- **E-step (Expectation Step):** Estimate missing data or latent variables using current parameters, leveraging conditional distributions from the copula structure.
- **M-step (Maximization Step):** Update copula parameters (e.g., correlation matrix) by maximizing the likelihood function with Maximum Likelihood Estimation (MLE).
- **Convergence:** Repeat E and M steps until:

- Parameter changes are below a threshold ( $\Delta < \varepsilon$ ).
- Log-likelihood stabilizes.
- The imputed data reaches a stable distribution.

### 2.3.3 Scaled Mean Absolute Error (SMAE)

The imputation evaluation is done using a metric such as the Scaled Mean Absolute Error (SMAE). This metric helps to assess how accurately the missing data was imputed by comparing the imputed values with the true (or "ground truth") dataset. Here's how it works:

$$\text{SMAE} = \frac{\text{MAE}}{\text{Range of Observed Data}} = \frac{\frac{1}{n} \sum_{i \in \text{missing}} |x_{\text{true},i} - x_{\text{imputed},i}|}{\max(x_{\text{obs}}) - \min(x_{\text{obs}})}$$

#### Explanation of Variables:

- $x_{\text{true},i}$ : True value of the  $i$ -th data point.
- $x_{\text{imputed},i}$ : Imputed value of the  $i$ -th data point.
- $x_{\text{obs}}$ : Observed (non-missing) values in the feature.
- $n$ : Number of missing values.

**Figure 2.5:** Scaled Mean Absolute Error formula

After computing SMAE for each imputation method:

- Smaller SMAE values indicate better performance, meaning the imputed values are closer to the true values relative to the range of observed data.
- Larger SMAE values suggest that the imputation method is less effective at approximating the true values.
- To assess the effectiveness of the different imputation methods, rank them based on their SMAE scores.



### 2.3.4 Mini-Batch and Online Training

To handle large and streaming datasets, gcimpute supports mini-batch processing, where smaller batches of data are processed iteratively. In streaming contexts, gcimpute can update its model incrementally as new data arrives, making it suitable for time-sensitive or evolving data environments.

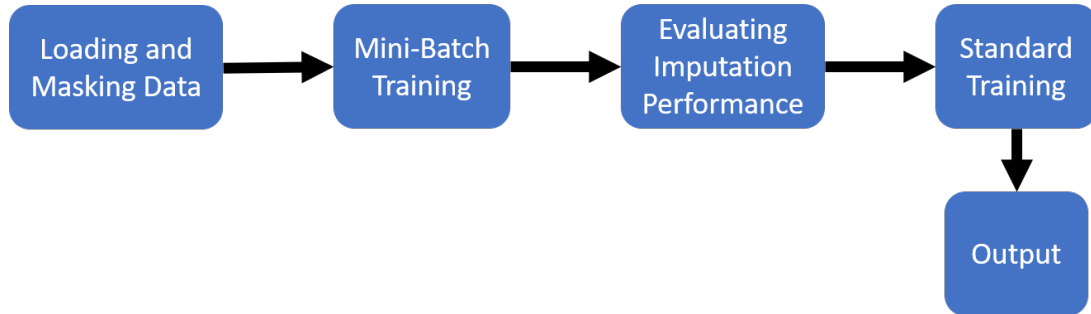
| Aspect                     | Mini-Batch Training                                | Standard Training                             |
|----------------------------|--|---|
| <b>Data Processing</b>     | Processes data in smaller chunks (mini-batches).   | Processes the entire dataset at once.         |
| <b>Memory Efficiency</b>   | More memory-efficient, suitable for large data.    | Requires all data to fit into memory.         |
| <b>Computational Speed</b> | Generally faster, especially for large datasets.   | Slower due to processing the entire dataset.  |
| <b>Use Case</b>            | Large datasets, real-time learning, deep learning. | Smaller datasets, stable gradient estimation. |

**Figure 2.6:** Comparison of Mini-Batch and Standard Training: Key Differences.

The process of comparing the runtime and performance of a Mini-Batch Training method versus a Standard Training approach using a Gaussian Copula model:

- **Data Loading and Masking:** The dataset is loaded, and some percentages of the data are randomly masked (missing).
- **Mini-Batch Training:**
  - The Gaussian Copula model is trained in mini-batch offline mode, processing smaller data chunks.
  - The imputation is performed, and the runtime is measured.
  - The imputation error is calculated using SMAE (Scaled Mean Absolute Error).
- **Standard Training:** The model is trained with the entire dataset at once (standard training mode), and the runtime is measured.

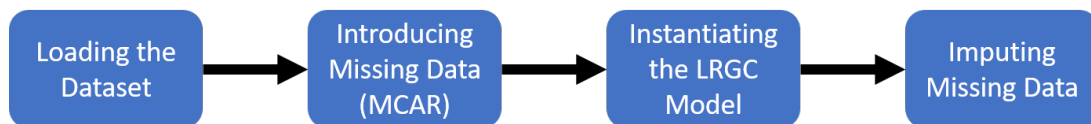
- **Output:** The runtimes and imputation error for both methods are printed.



**Figure 2.7:** Mini-Batch Training process

### 2.3.5 Low-Rank Gaussian Copula (LRGC)

The Low-Rank Gaussian Copula (LRGC) is a method used to model multivariate distributions and handle missing data by approximating the covariance matrix with a low-rank structure. This method helps to accelerate the processing of datasets with a large number of variables by reducing the complexity of the covariance matrix, making it computationally more efficient, especially for large datasets with many variables.



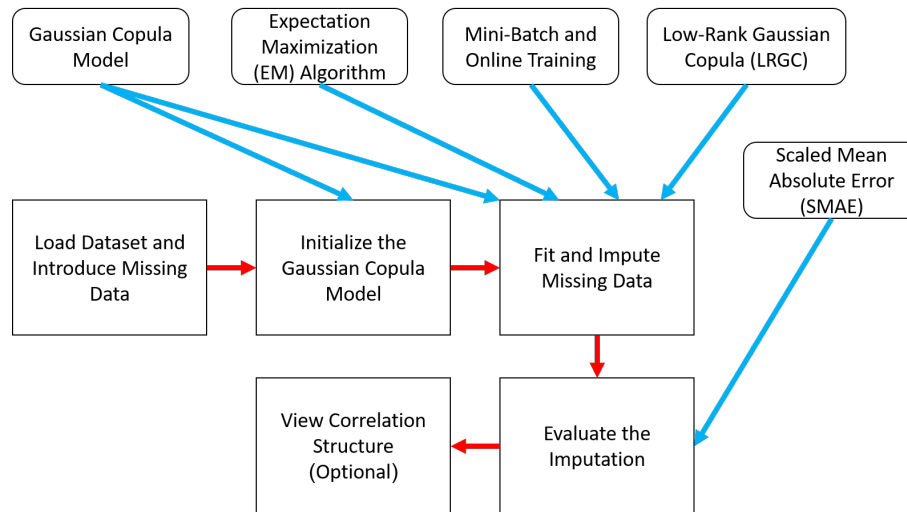
**Figure 2.8:** Low-Rank Gaussian Copula process

Illustrating the above process:

- **Loading the Dataset:** The dataset is loaded using a function that retrieves a specified number of variables (e.g., movies) and ensures a minimum number of observations per variable.
- **Introducing Missing Data:** A function randomly introduces a specified percentage of missing data (e.g., 10%) into the dataset, simulating the real-world scenario of missing values.

- **Instantiating the LRGC Model:** The Low-Rank Gaussian Copula (LRGC) model is created with a low rank value (e.g., 10), reducing the dimensionality of the covariance matrix for more efficient computation.
- **Imputing Missing Data:** The model is trained on the dataset using the `fit_transform()` method, which learns the data structure and imputes the missing values.

## 2.4 Missing data imputation process:



**Figure 2.9:** Missing data imputation process with algorithms

The missing data imputation process begins with loading a dataset containing mixed-type variables such as continuous (e.g., age), binary (e.g., yes/no diagnoses), and ordinal (e.g., severity levels). To simulate real-world data scenarios, missing values are artificially introduced at a predefined rate (e.g., 20%) across the dataset. Next, the Gaussian Copula Model is initialized, which transforms each observed variable into a uniform distribution and maps it into a latent Gaussian space using the inverse Gaussian CDF. This allows the copula to capture the dependency structure across variables in the dataset, regardless of their types. The Expectation Maximization (EM) Algorithm is then applied to iteratively estimate the missing values. In the E-step, conditional expectations



are computed to predict the missing entries given the observed data, while in the M-step, the latent Gaussian parameters (mean and covariance) are updated to maximize the likelihood of the complete data.

For large datasets, where memory and computational resources are constrained, the Low-Rank Gaussian Copula (LRGC) is employed to approximate the dependency structure with a low-rank representation, reducing the model's complexity while maintaining accuracy. Additionally, Mini-Batch and Online Training techniques ensure the process is scalable by splitting the dataset into smaller batches, where model parameters and imputed values are updated incrementally. This also allows real-time data processing when new observations arrive. Once the imputation is complete, the results are evaluated using the Scaled Mean Absolute Error (SMAE), which measures the mean absolute error (MAE) of the imputed values and scales it by the standard deviation of the observed data to provide a normalized performance metric. The SMAE helps assess imputation quality across datasets with varying scales and variable types, offering a robust evaluation framework for the Gaussian Copula-based imputation process.

## **Chapter 3**

# **Data Streaming Platform**

### **3.1 Introduction**

AutoMQ is a Kafka alternative designed with a cloud-first philosophy. It innovatively re-designs the storage layer of Apache Kafka to leverage cloud infrastructure effectively, delivering a 10x cost reduction and a 100x increase in elasticity. Moreover, AutoMQ maintains 100% compatibility with Kafka, allowing seamless migration for existing users. By separating persistence between EBS (Elastic Block Store) and S3, AutoMQ achieves unprecedented levels of efficiency. The platform delivers low latency, high throughput, and low cost, making it not only performant but also easy to use. These features combine to provide a unified solution for modern data streaming needs.

### **3.2 Kafka is the Beginning**

Apache Kafka has been the cornerstone of modern distributed messaging systems since its inception. Designed to handle real-time data feeds, Kafka revolutionized the way organizations process and analyze data streams. By offering high-throughput, fault-tolerant messaging capabilities, Kafka set the standard for data streaming platforms. However, as the demand for scalability and cost-efficiency grew, Kafka began to show limitations in



a cloud-centric world.

Kafka's architecture relies heavily on a tightly coupled storage and compute design, which necessitates substantial resource provisioning. While this approach ensures reliability, it incurs high operational costs and introduces complexity in scaling. For example, Kafka requires extensive replication for fault tolerance, leading to significant overhead in terms of storage and compute resources. Additionally, the necessity of managing both the broker and storage clusters adds administrative complexity.

These limitations highlighted the need for an evolved architecture that could harness the elasticity and cost advantages of cloud infrastructure. AutoMQ emerges as a solution to these challenges, retaining Kafka's core functionality while reimagining its architecture for the cloud era.

## 3.3 Technical Advantages

AutoMQ offers a range of technical advantages that address the limitations of traditional messaging systems like Kafka:

### 3.3.1 Cost Efficiency

One of the standout features of AutoMQ is its ability to deliver a 10x cost reduction compared to Kafka. This is achieved through:

- **Decoupling Storage and Compute:** AutoMQ leverages cloud storage systems like S3, which offer high durability at a fraction of the cost of traditional replication-based approaches.
- **Spot Instances:** By adopting a stateless broker design, AutoMQ can run efficiently on cost-effective Spot instances, further reducing computational expenses.
- **Elastic Scaling:** AutoMQ dynamically adjusts its resources based on workload demands, ensuring optimal utilization and cost savings.



### 3.3.2 Elasticity and Scalability

AutoMQ's architecture is designed for seamless scalability. Its stateless brokers and shared storage architecture allow for:

- **Instant Scaling:** Brokers can be added or removed within seconds, responding to traffic spikes or dips without manual intervention.
- **Auto-Balancing:** Dynamic partition reassignment eliminates bottlenecks and ensures an even distribution of workload across the cluster.

### 3.3.3 Performance

Despite its focus on cost and elasticity, AutoMQ does not compromise on performance. Key metrics include:

- **Low Latency:** AutoMQ achieves single-digit millisecond latency, ensuring real-time processing for critical applications.
- **High Throughput:** The platform can handle large-scale data streams, matching or exceeding Kafka's performance levels.
- **Enhanced Catch-Up Reads:** Consumers can recover quickly from lag without impacting overall system performance.

### 3.3.4 Ease of Migration and Compatibility

AutoMQ is fully compatible with Apache Kafka's protocol, ensuring that existing Kafka users can migrate seamlessly without modifying their applications. This compatibility extends to features like compact topics, idempotent producers, and transactional messaging, making AutoMQ a drop-in replacement for Kafka.



## 3.4 Truly Cloud-Native Architecture of AutoMQ

AutoMQ's cloud-native design is a cornerstone of its success, reflecting a deep understanding of the unique advantages offered by modern cloud environments.

### 3.4.1 AutoMQ Shared Storage Architecture Overview

The Shared Storage architecture introduced by AutoMQ follows the principle of separating storage and compute. By offloading storage to cloud services, it makes the computation layer completely stateless, rendering the entire architecture elastic and fully exploiting cloud-native advantages.

- **Shared Storage Layer:** AutoMQ selects the largest object storage and EBS storage from cloud providers as the storage medium for the shared storage layer. EBS serves as shared WAL storage, while object storage functions as the primary data store.
- **Stateless Compute Layer:** AutoMQ replaces Apache Kafka's native Log storage with its self-developed stream storage library—S3Stream, offloading Broker storage to cloud storage. This leverages EBS's high performance, object storage's low cost, and high throughput characteristics, making the compute layer stateless.
- **Control Layer:** After fully offloading the storage state, AutoMQ makes it incredibly easy to achieve second-level partition reassignment, automatic scaling, and continuous self-balancing of traffic. Consequently, AutoMQ has built-in Controller components within its core, such as the Auto Scaling and Auto Balancing components, which are responsible for cluster scaling and traffic self-balancing, respectively.

### 3.4.2 Stateless Brokers

How AutoMQ Achieves Complete Statelessness. S3Stream comprises two storage components:





- **WAL Storage:** The storage medium for WAL Storage is diverse; it can use EBS for WAL or S3 for WAL.
- **S3 Storage:** Object storage is used as the primary storage for data.

S3Stream accesses object storage via the HTTP protocol, which is entirely stateless. Therefore, if S3 is chosen as the storage medium for WAL when deploying AutoMQ, the entire architecture is completely stateless. However, if EBS is chosen for WAL and accessed through the file API, achieving a stateless architecture becomes challenging. The core of AutoMQ is using EBS's multi-attach capability. By turning EBS into shared storage, complete statelessness can be achieved. The core process is straightforward:

1. Upon detecting the failure of Broker A, the Controller will attach its EBS WAL to Broker B in a multi-attach manner.
2. Broker B will take over and complete the recovery upload of the small amount of data in the WAL that is not stored in S3.
3. At this point, the state of Broker A is offloaded, and the Controller will subsequently reassign the partitions originally belonging to Broker A evenly to other Brokers.

The above process design can be applied to:

- Fault recovery scenarios
- Scaling down
- Decommissioning processes

Traditional messaging systems rely on a tightly coupled storage layer, which increases complexity and limits flexibility. AutoMQ's use of S3Stream enables a stateless broker architecture, transforming the system from a Shared-Nothing to a Shared-Storage model. This stateless design offers several benefits:



- **Simplified Operations:** For AutoMQ, daily operations become sufficiently simple. After a Broker node shuts down, its state is completely transferred, with clients entirely unaffected. Operations personnel have ample time to decide whether the shut-down Broker needs to be brought back online or decommissioned permanently. Cluster upgrades can also be completed at low cost and risk through rolling updates.
- **Automatic Scaling:** Stateless AutoMQ can scale up or down freely, similar to a microservice application or a Kubernetes Deployment, achieving true auto-scaling and saving significant costs.
- **Use of Spot Instances:** Cloud providers offer Spot instances that can be up to 90% cheaper than regular virtual machines. However, due to the nature of Spot instances being subject to termination at any time, only stateless applications can take advantage of them.

### 3.4.3 Automated Elasticity

Automation is a key feature of AutoMQ's architecture. The system includes controllers that continuously monitor metrics such as load and partition distribution. These controllers enable:

- **Auto-Scaling:** Resources are dynamically adjusted based on real-time demand, ensuring efficient operation at all times.
- **Auto-Balancing:** Hotspots are identified and resolved automatically, optimizing performance across the cluster.

### 3.4.4 Multi-Cloud Adaptability

AutoMQ is designed to operate efficiently across different cloud providers, adapting its architecture to leverage the unique features of each platform. For example:

- On AWS, AutoMQ uses a combination of S3 One Zone and EBS to achieve low-latency writes and zone failure tolerance.
- On Azure and Google Cloud, regional EBS solutions are used for WAL, ensuring high availability and reliability.

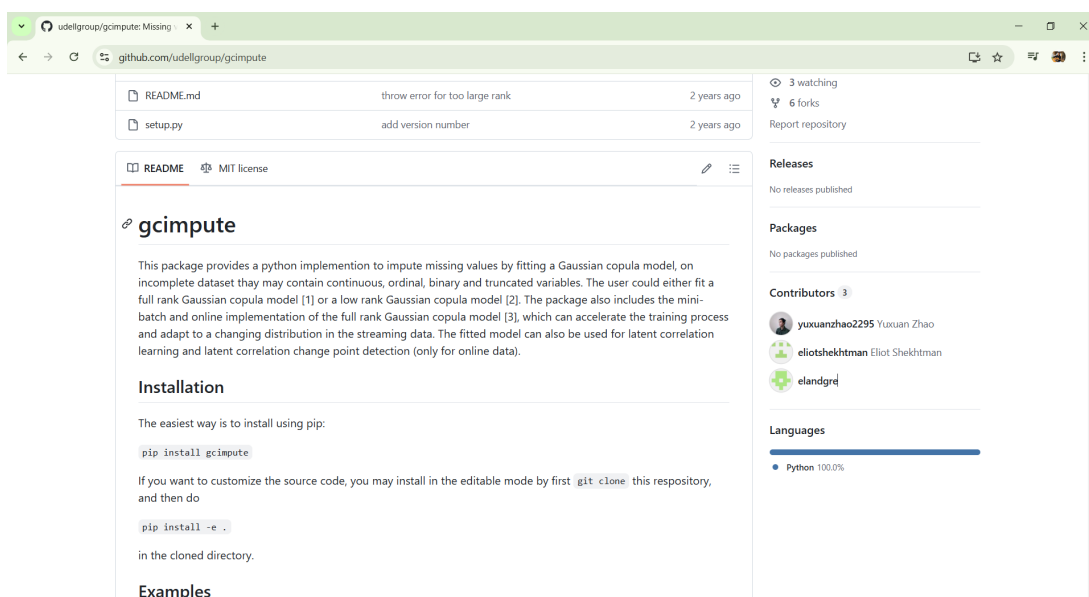
## 3.5 Deployment

Everyone can view the source code in our team's [GitHub Repository](#).

### 3.5.1 Missing Data Imputation

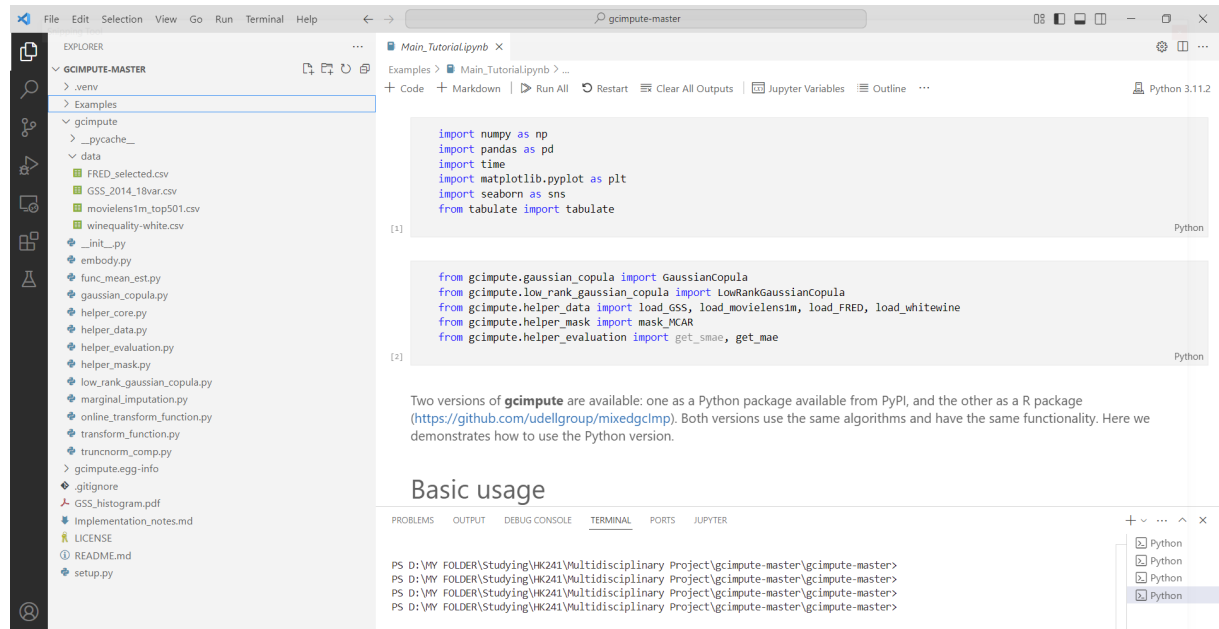
#### 3.5.1.1 Preparation steps

First, our team follows instructions in [The gcimpute package](#) to use its Python implementation for imputing missing values by fitting a Gaussian copula model on incomplete datasets. This approach supports datasets with a mix of continuous, ordinal, binary, and truncated variables.



**Figure 3.1:** gcimpute is a Python package for missing value imputation

Then, once you have downloaded this directory, you can open and use it in Visual Studio Code as follows:



**Figure 3.2:** The gcimpute package folder is opened in Visual Studio Code

Finally, after successfully running three examples in this package, you can create new Python programs and run them within this directory.

### 3.5.1.2 Example 1

To demonstrate a basic usage example of Gaussian Copula imputation and evaluate the imputation performance using SMAE, our team made the following program:

```
import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from gcimpute.gaussian_copula import GaussianCopula
from gcimpute.low_rank_gaussian_copula import LowRankGaussianCopula
from gcimpute.helper_data import load_GSS, load_movielens1m
```

```
from gcimpute.helper_mask import mask_MCAR
from gcimpute.helper_evaluation import get_smae, get_mae
print("Example 1: Basic usage")
# File paths
imputed_data_path =
    r"\\wsl.localhost\Ubuntu\home\anhquan2682001\imputed_data.csv"
smae_results_path =
    r"\\wsl.localhost\Ubuntu\home\anhquan2682001\smae_results.txt"

# Load GSS dataset (simulated here for demonstration)
data = load_GSS() # e.g., includes age, income, health status, etc.
# Introduce 10% missing data randomly for testing
data_with_missing = mask_MCAR(X=data, mask_fraction=.1, seed=101)

# Impute missing data using Gaussian Copula
model = GaussianCopula(verbose=1)
imputed_data = model.fit_transform(X=data_with_missing)
# Assuming data is the original dataset without missing values
# Calculate Scaled Mean Absolute Error (SMAE)
smae = get_smae(imputed_data, x_true=data, x_obs=data_with_missing)
print(f"Scaled Mean Absolute Error (SMAE): {smae.mean():.3f}")

# Save imputed data to a CSV file
imputed_data_df = pd.DataFrame(imputed_data, columns=data.columns)
imputed_data_df.to_csv(imputed_data_path, index=False)

# Save SMAE results to a text file
with open(smae_results_path, 'w') as f:
    f.write(f"Scaled Mean Absolute Error (SMAE): {smae.mean():.3f}\n")
    f.write("SMAE values per feature:\n")
```

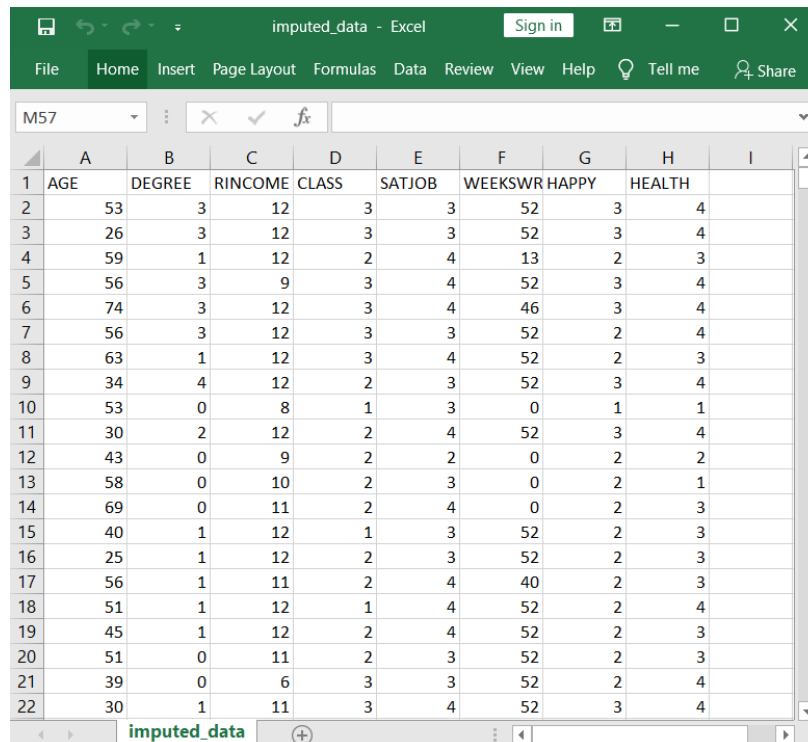
```
for i, col in enumerate(data.columns):  
    f.write(f"{col}: {smae[i]:.3f}\n")
```

After running this code, our program prints results on the terminal:

```
Example 1: Basic usage  
Iter 1: copula parameter change 0.0867, likelihood -9.6268  
Iter 2: copula parameter change 0.0495, likelihood -9.5544  
Iter 3: copula parameter change 0.0288, likelihood -9.5091  
Iter 4: copula parameter change 0.0176, likelihood -9.4828  
Iter 5: copula parameter change 0.0114, likelihood -9.4672  
Iter 6: copula parameter change 0.0077, likelihood -9.4576  
Convergence achieved at iteration 6  
Scaled Mean Absolute Error (SMAE): 0.889
```


**Figure 3.3:** The first example's output in a terminal

Moreover, in a Ubuntu folder, there are two output files containing imputed data and SMAE results.



|    | A   | B      | C       | D     | E      | F       | G     | H      | I |
|----|-----|--------|---------|-------|--------|---------|-------|--------|---|
| 1  | AGE | DEGREE | RINCOME | CLASS | SATJOB | WEEKSWR | HAPPY | HEALTH |   |
| 2  | 53  | 3      | 12      | 3     | 3      | 52      | 3     | 4      |   |
| 3  | 26  | 3      | 12      | 3     | 3      | 52      | 3     | 4      |   |
| 4  | 59  | 1      | 12      | 2     | 4      | 13      | 2     | 3      |   |
| 5  | 56  | 3      | 9       | 3     | 4      | 52      | 3     | 4      |   |
| 6  | 74  | 3      | 12      | 3     | 4      | 46      | 3     | 4      |   |
| 7  | 56  | 3      | 12      | 3     | 3      | 52      | 2     | 4      |   |
| 8  | 63  | 1      | 12      | 3     | 4      | 52      | 2     | 3      |   |
| 9  | 34  | 4      | 12      | 2     | 3      | 52      | 3     | 4      |   |
| 10 | 53  | 0      | 8       | 1     | 3      | 0       | 1     | 1      |   |
| 11 | 30  | 2      | 12      | 2     | 4      | 52      | 3     | 4      |   |
| 12 | 43  | 0      | 9       | 2     | 2      | 0       | 2     | 2      |   |
| 13 | 58  | 0      | 10      | 2     | 3      | 0       | 2     | 1      |   |
| 14 | 69  | 0      | 11      | 2     | 4      | 0       | 2     | 3      |   |
| 15 | 40  | 1      | 12      | 1     | 3      | 52      | 2     | 3      |   |
| 16 | 25  | 1      | 12      | 2     | 3      | 52      | 2     | 3      |   |
| 17 | 56  | 1      | 11      | 2     | 4      | 40      | 2     | 3      |   |
| 18 | 51  | 1      | 12      | 1     | 4      | 52      | 2     | 4      |   |
| 19 | 45  | 1      | 12      | 2     | 4      | 52      | 2     | 3      |   |
| 20 | 51  | 0      | 11      | 2     | 3      | 52      | 2     | 3      |   |
| 21 | 39  | 0      | 6       | 3     | 3      | 52      | 2     | 4      |   |
| 22 | 30  | 1      | 11      | 3     | 4      | 52      | 3     | 4      |   |

**Figure 3.4:** The imputed data saved in a csv file

 **smae\_results - Notepad**  
File Edit Format View Help  
Scaled Mean Absolute Error (SMAE): 0.889  
SMAE values per feature:  
AGE: 0.932  
DEGREE: 0.868  
RINCOME: 0.944  
CLASS: 0.872  
SATJOB: 0.927  
WEEKSWRK: 0.700  
HAPPY: 0.963  
HEALTH: 0.906

**Figure 3.5:** The result of Scaled Mean Absolute Error

#### Key Insights:

- WEEKSWRK has the lowest SMAE, suggesting it is imputed most accurately.
- HAPPY and RINCOME have the highest SMAE, indicating these features are more challenging to impute accurately.
- The overall SMAE of 0.889 indicates moderate imputation performance.

#### 3.5.1.3 Example 2

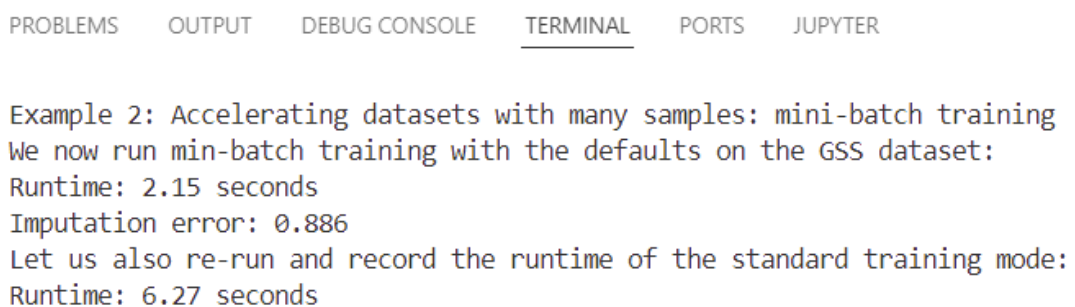
To demonstrate how to accelerate the imputation process using mini-batch training compared to the standard training mode on the GSS dataset, we use the following code:

```
print("\nExample 2: Accelerating datasets with many samples: mini-batch  
training")  
print("We now run min-batch training with the defaults on the GSS  
dataset:")  
data_gss = load_GSS()  
gss_masked = mask_MCAR(X=data_gss, mask_fraction=.1, seed=101)
```

```
t1=time.time()
model_minibatch = GaussianCopula(training_mode='minibatch-offline')
Ximp_batch = model_minibatch.fit_transform(X=gss_masked)
t2=time.time()
print(f'Runtime: {t2-t1:.2f} seconds')
smae_batch = get_smae(x_imp=Ximp_batch, x_true=data_gss, x_obs=gss_masked)
print(f'Imputation error: {smae_batch.mean():.3f}')

print("Let us also re-run and record the runtime of the standard training
      mode:")
t1=time.time()
GaussianCopula().fit_transform(X=gss_masked)
t2=time.time()
print(f'Runtime: {t2-t1:.2f} seconds')
```

After running this code, our program prints results on the terminal:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  JUPYTER

Example 2: Accelerating datasets with many samples: mini-batch training
We now run min-batch training with the defaults on the GSS dataset:
Runtime: 2.15 seconds
Imputation error: 0.886
Let us also re-run and record the runtime of the standard training mode:
Runtime: 6.27 seconds
```

**Figure 3.6:** The second example's output in a terminal

Key Observations:

- **Runtime Efficiency:** Mini-batch training significantly reduces runtime (by 65%) compared to standard training. This is especially beneficial for datasets with a large number of samples.





- **Imputation Accuracy:** Mini-batch training achieves an imputation error (SMAE) of 0.886, showing a slight improvement over standard training's 0.889, demonstrating its capability to handle missing data efficiently.

#### 3.5.1.4 Example 3

To demonstrate the use of Low-Rank Gaussian Copula (LRGC) for accelerating the imputation process on a large dataset with many variables (features) compared to the standard Gaussian Copula (GC) method, we built the following program:

```
print("\nExample 3: Accelerating datasets with many variables: low rank  
structure")  
  
data_movie = load_movielens1m(num=400, min_obs=150, verbose=True)  
movie_masked = mask_MCAR(X=data_movie, mask_fraction=0.1, seed=101)  
  
a = time.time()  
model_movie_lrgc = LowRankGaussianCopula(rank=10, verbose=1)  
m_imp_lrgc = model_movie_lrgc.fit_transform(X=movie_masked)  
print(f'LRGC runtime {(time.time()-a)/60:.2f} mins.')a = time.time()  
model_movie_gc = GaussianCopula(verbose=1)  
m_imp_gc = model_movie_gc.fit_transform(X=movie_masked)  
print(f'GC runtime {(time.time()-a)/60:.2f} mins.')  
mae_gc = get_mae(x_imp=m_imp_gc, x_true=data_movie, x_obs=movie_masked)  
mae_lrgc = get_mae(x_imp=m_imp_lrgc, x_true=data_movie,  
                  x_obs=movie_masked)  
print(f'LRGC imputation MAE: {mae_lrgc:.3f}')print(f'GC imputation MAE: {mae_gc:.3f}')
```

After running this code, our program prints results on the terminal:

```
Example 3: Accelerating datasets with many variables: low rank structure
The loaded dataset consists of 914 users and 400 movies with 53.3% ratings observed
Iteration 1: noise ratio 0.6464, copula parameter change 0.0967, likelihood -411.3549
Iteration 2: noise ratio 0.6368, copula parameter change 0.0428, likelihood -410.6035
Iteration 3: noise ratio 0.6299, copula parameter change 0.0249, likelihood -410.0521
Iteration 4: noise ratio 0.6247, copula parameter change 0.0167, likelihood -409.5867
Iteration 5: noise ratio 0.6208, copula parameter change 0.0121, likelihood -409.2098
Iteration 6: noise ratio 0.6178, copula parameter change 0.0093, likelihood -408.9108
Convergence achieved at iteration 6
LRGC runtime 0.09 mins.
Iter 1: copula parameter change 0.2972, likelihood -395.2120
Iter 2: copula parameter change 0.1265, likelihood -388.8526
Iter 3: copula parameter change 0.0688, likelihood -383.2187
Iter 4: copula parameter change 0.0428, likelihood -378.8466
Iter 5: copula parameter change 0.0299, likelihood -375.2380
Iter 6: copula parameter change 0.0222, likelihood -372.2628
Iter 7: copula parameter change 0.0174, likelihood -369.7298
Iter 8: copula parameter change 0.0140, likelihood -367.5678
Iter 9: copula parameter change 0.0116, likelihood -365.6854
Iter 10: copula parameter change 0.0098, likelihood -364.0453
Convergence achieved at iteration 10
GC runtime 2.50 mins.
LRGC imputation MAE: 0.581
GC imputation MAE: 0.615
PS D:\MY FOLDER\Studying\HK241\Multidisciplinary Project\gcimpute-master\gcimpute-master>
```

**Figure 3.7:** The third example's output in a terminal

Some analytics:

- **Runtime Efficiency:** 96% faster runtime than the standard Gaussian copula for  $p=400$  variables, with greater speedups at higher dimensions.
- **Imputation Accuracy:** Improves imputation error from 0.615 to 0.581, with small errors ( 0.5 stars) on a 1–5 rating scale.

### 3.5.2 Data Streaming

Everyone can view our [User Manual](#) for data streaming deployment.

# Chapter 4

## Evaluation

### 4.1 Missing Data Imputation

#### 4.1.1 Achievements

- **Accurate Imputation Results:** GCImpute achieves high accuracy in imputing missing values, especially in datasets with complex structures, outperforming traditional methods like mean, median, and KNN imputation. It excels in capturing non-linear relationships and interdependencies among data points.
- **Scalability for Large Datasets:** GCImpute can handle large-scale datasets efficiently by utilizing the parallelization capabilities of deep learning frameworks. This allows it to process high-dimensional data where traditional imputation methods struggle.
- **Versatility Across Domains:** GCImpute has been successfully applied in various domains, including bioinformatics, IoT systems, and recommendation systems, showcasing its adaptability to diverse types of data.



### 4.1.2 Challenges

- **Graph Construction Dependency:** The performance of GCImpute is heavily dependent on the quality of the graph representation. Poorly constructed graphs can lead to suboptimal or inaccurate imputations.
- **Data-Specific Tuning:** GCImpute often requires careful tuning of hyperparameters, such as the graph structure, learning rate, and model architecture, which can be time-consuming and require deep expertise.
- **Training Time:** Compared to simpler methods, GCImpute has a longer training time, particularly for very large or sparse datasets. This makes it less suitable for real-time or time-sensitive tasks.

### 4.1.3 Recommendations for Improvement

- **Automated Graph Construction:** Implement automated and optimized graph construction techniques to reduce the dependency on manually designed or domain-specific graphs. Leveraging graph similarity measures or embedding techniques can improve robustness.
- **Model Simplification:** Develop lightweight versions of GCImpute that reduce computational complexity and training time while retaining performance, making the model more accessible for smaller datasets and systems.
- **Real-Time Adaptation:** Enhance GCImpute's ability to perform imputation in real-time environments, such as streaming data systems, by optimizing graph updates and model inference speed.



## 4.2 Evaluation about AutoMQ

The project undertaken by the AutoMQ data streaming platform team has shown significant progress in understanding and implementing a suitable innovative data streaming solution. Here is a review of their efforts, achievements and areas for improvement:

### 4.2.1 Achievements

- **Foundational Knowledge of AutoMQ:** The team was successful in gaining a solid understanding of the core concepts of AutoMQ, in addition to the platform's breakthroughs compared to Kafka. This foundational knowledge was essential to explore and leverage AutoMQ's capabilities for the team's project and guide further development.
- **Feature Comparison with Kafka:** By conducting a comprehensive comparison between AutoMQ and Kafka, the team highlighted the advantages of AutoMQ, especially in the context of cloud computing used to compute and process massive data streams. This analysis provides valuable insights for potential adopters and contributors to projects that require platforms with AutoMQ's capabilities and tasks.
- **Basic Implementation of AutoMQ Functionality:** The team has implemented the basic functionality of AutoMQ on local machines. Although there is a limitation that the system cannot be deployed on the cloud due to some objective reasons, this achievement emphasizes our practical skills in software development and the ability to convert theoretical knowledge into functional results that can be applied to the project in the best way.



### 4.2.2 Challenges

- **Limited Deployment Scope:** Due to objective constraints, the team was only able to run AutoMQ locally, missing out on the opportunity to demonstrate its full potential in a cloud environment. This limitation resulted in the inability to realize the full functionality of AutoMQ in practice.

### 4.2.3 Recommendations for Improvement

- **Focus on Cloud Deployment:** Overcoming the objective barriers to deploying AutoMQ in the cloud should be a top priority for the team. This will allow us to benchmark AutoMQ's performance in the intended environment and demonstrate its full advantages over Kafka and other traditional streaming platforms in a more holistic manner.
- **Demonstrating Advanced Features:** Although the team has successfully developed AutoMQ locally and running it, it is necessary to link to cloud platforms such as S3 to take advantage of all the potential and resources that AutoMQ can bring to the project.

# References

- [1] Y. Zhao and M. Udell, “Gcimpute: A package for missing data imputation”, *Journal of Statistical Software*, vol. 108, no. 4, Feb. 2024.
- [2] “Automq website”. (), [Online]. Available: <https://www.automq.com/>.
- [3] “Automq repository”. (), [Online]. Available: <https://github.com/AutoMQ/automq>.
- [4] “Automq architecture overview”. (), [Online]. Available: <https://docs.automq.com/automq/architecture/overview>.
- [5] “Automq s3stream overview”. (), [Online]. Available: <https://docs.automq.com/automq/architecture/s3stream-shared-streaming-storage/overview>.
- [6] “Automq technical advantages”. (), [Online]. Available: <https://docs.automq.com/automq/architecture/technical-advantage/overview>.