ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

UDC 004.8: 629.7.05: 623.67 doi: 10.32620/reks.2025.3.05

Olga SOLOVEI, Tetyana HONCHARENKO

Kyiv National University of Construction and Architecture, Kyiv, Ukraine

A BAYESIAN-DRIVEN FEEDFORWARD NEURAL NETWORK MODEL FOR KAFKA CLUSTER LATENCY FORECASTING

*The **subject matter** of this article is the process of designing the architecture of a Feedforward neural network model based on the discrete Bayesian Network and a new method for setting the initial weights that connect neurons across layers. The **goal** of this study is to develop a Neural Network model designed to forecast end-to-end latency in a Kafka cluster. The proposed model can be used as a tool to predict the end-to-end latency of Kafka clusters based on the given configuration parameters and performance metrics. This study resolved the following tasks: developed and validated a discrete Bayesian network to understand the factors influencing end-to-end latency in Kafka clusters; conducted a sensitivity analysis on the discrete Bayesian network; created a matrix with initial weights derived from the sensitivity analysis in the Bayesian network to initialize weights in FFNN model; designed FFNN architecture for predicting the Kafka cluster end-to-end latency and configured its parameters; trained and evaluated the designed FFNN model. Methods from theories were used to conduct the research: big data processing, probabilistic graphical models and Bayesian inference theory, artificial neural networks and deep learning theories, graph theory, and machine learning optimization. The following **results** were obtained: a trained FFNN model Mean Square Error showed consistent decrease across epochs, so we concluded that the model can be deployed and used as a tool to forecast Apach Kafka latency for given configuration parameters and performance metrics. The comparison of the Mean Square Error values when FFNN model is initialized with weights derived from the strength of influence in the Bayes Network and FFNN model which is set the same initial weights but scaled by Kaiming He factor demonstrated that Kaiming He scaling factor primarily improves the initial phase of training by stabilizing weight initialization. Therefore, we recommend scaling the initial weights as specified in our method to optimize FFNN training process. **Conclusions**. The scientific novelty of the results obtained is as follows: 1) a new methodology for defining the architecture of a Feedforward Neural Network (FFNN) based on the discrete Bayesian network structure is introduced; 2) the initial weights that connect neurons across layers are set.*

Keywords: *Kafka cluster latency; Bayes Network; Feedforward Neural network; strength of influence; initial weights.*

1. Introduction

IoT devices, sensors, and wearable sensing devices have become data sources for construction-based information systems [1]. Consequently, the architecture of such systems has evolved to include streaming processing engines, such as Apache Kafka, which serves as a distributed storage system. This system consumes streaming messages from Kafka producers and retains them until they are retrieved by Kafka consumers. The efficiency of a Kafka cluster is measured by its end-to-end latency, which is the time between the moment when an application that includes a Kafka producer sends event data and the moment when the consumer logic of the Kafka program receives the event [2]. Information systems for building construction projects rely heavily on Kafka clusters' low end-to-end latency, as such systems require immediate decisions based on real-time data. They also need real-time data to monitor site safety and coordinate numerous teams in a timely manner [3].

1.1. Motivation

Given the limited formal methods available to predict Kafka cluster performance under different system conditions, the Kafka cluster configuration is selected based on the system's performance tests. This may lead to configuration changes post-deployment if Kafka cluster performance is lower than expected, which may breach the unavailability threshold, which must be avoided. Therefore, when Kafka clusters are included in information systems for building construction projects, models and methods are required to diagnose and evaluate their efficiency [4].

A neural network model to predict end-to-end latency in a Kafka cluster can be an effective approach, leveraging the capabilities of machine learning to model complex non-linear relationships and interactions between various system parameters and performance metrics. However, the effectiveness of a neural network model will depend on how well the network's architecture

and parameters, such as an initial weight between neurons in different layers reflect the casual relations between various performance characteristics of Kafka cluster [5]. At the same time, a Bayes network (Bayes Net) has become a method for analyzing and diagnosing complex systems, as it provides the ability to model causal relationships and answer probabilistic queries. For example, in research [6, 7] Bayes Networks (Bayes net) were used to identify potential issues and prioritize maintenance and repair needs in building condition assessment. In this context, the Bayes Net was used to model the relationships between building components and their condition.

1.2. Motivating Use Case

This work is motivated by the challenge of real-time data processing in large-scale urban infrastructure projects, specifically the construction of a new subway tunnel beneath a densely populated area [8]. In this high-risk environment, modern civil engineering practices rely on a dense network of Internet of Things (IoT) sensors to mitigate geological risks and optimize the Tunnel Boring Machine (TBM) operational efficiency. The integrity of the project and personnel safety are directly dependent on the timely and reliable processing of this sensor data. The system architecture is a centralized, high-throughput streaming platform. The primary data sources were heterogeneous IoT sensors deployed in and around the tunnel. All data streams are ingested into a central, fault-tolerant message queue that is implemented using an Apache Kafka cluster. This cluster serves as the data backbone, decoupling the sensors (producers) from the various data analysis and storage systems (consumers). The project's real-time analytics platform is the primary consumer, which is responsible for anomaly detection, visualization for TBM operators, and long-term data archival.

The operational safety model is critically dependent on low-latency data processing. Based on real-time feedback from the geotechnical sensors, TBM operators manually and automatically adjust machine parameters. A safety requirement is defined as follows: a sudden spike in ground pressure detected by piezometers must be processed and trigger a system-wide alert or an automated partial TBM shutdown within a strict T-millisecond.

Problem Definition: a latency spike in the Kafka pipeline could delay a critical alert, causing the TBM to operate under unsafe geological pressures. This could fail in the tunnel face, damage to the machine, and extreme risk to personnel.

Proposed Solution in the current research: to address this challenge, we propose employing a Feedforward Neural Network (FFNN) trained to forecast latency spikes. The prediction of FFNN is fed into a decision process. If the forecasted latency exceeds a predefined operational threshold, the system triggers an automated remediation action before the actual latency becomes critical. The action involves increasing the number of consumer instances. This forces a partition rebalances within Kafka, distributing the data load across more processing units. The system reduces consumer lag and stabilizes the pipeline latency by increasing parallel processing capacity, ensuring the threshold safety requirement is consistently met.

The goal of the current research is to propose a Neural Network model designed to forecast end-to-end latency in a Kafka cluster, with a focus on enhancing prediction accuracy, model trustworthiness, and model convergence. The proposed model is intended to be a practical tool for predicting end-to-end latency in Kafka clusters based on given configuration parameters and performance metrics.

To achieve this goal, a new method is proposed for defining the architecture of a Feedforward Neural Network based on the discrete Bayesian Network structure and for setting the initial weights that connect neurons across layers.

1.3. State of the art

The Kafka cluster consists of servers, which are often referred to as brokers. In Kafka cluster, one server is always assigned the "controller" role and the "leader" role. The "controller" is responsible for administrative tasks, while the "leader" is the server that first receives streaming data. Kafka uses ZooKeeper as a centralized service for managing and coordinating Kafka brokers in the cluster. Each Kafka broker manages topics to which producers send streaming data and where consumers tune in. Kafka replicates data across various servers included in the Kafka cluster to ensure high availability, durability, and fault tolerance. The number of necessary copies is determined by the topic configuration using the replication factor parameter. Apach Kafka documents [9] specify the end-to-end Kafka cluster event streaming process through a sequence of steps (Fig. 1).

When a producer sends event data on a topic with a single partition, the event lands in the receive socket buffer on the broker; from there, it is picked up by the Network Thread and placed in the shared request queue.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

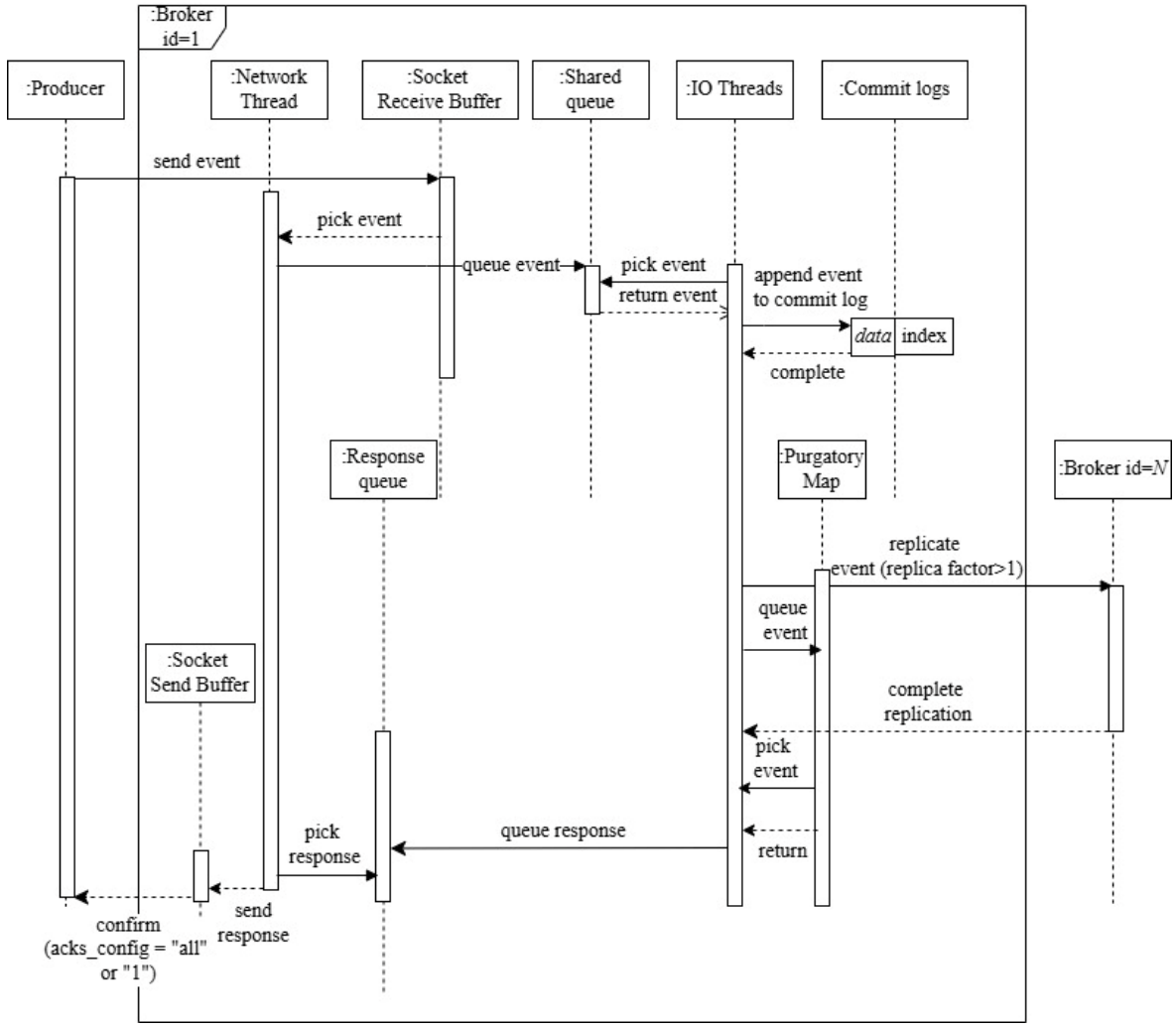

Fig. 1. Kafka cluster event streaming process

Kafka's IO thread picks up the event and registers it to a commit log organized on disk in segments, where each segment holds part of the log. The registered event data from RAM is saved on the leader server's disk and moved to the "Purgatory Map" queue, where they are held until copied to other servers in the Kafka cluster. These servers send requests to the leader server to receive copying event data. The leader server sends the event data in response once it is copied to the disk. Until brokers do not complete the copying process, they will continually send requests to the leader.

After a broker completes replication, the pending event data are removed from the Purgatory Map queue and placed in the response queue. The Network Thread then takes a response and sends it into Send socket buffer. The Kafka producer receives confirmation from the "leader server" that events have been securely stored after being copied on all servers (when the "acks_config" parameter is set to "all"). The Kafka producer can also receive confirmation as soon as the data is added to the logging journal on the server (when the "acks_config" parameter is set to "1") or may not expect any confirmation at all if the "acks_config" parameter is set to "0". Using the poll() function, Kafka consumers continuously send requests to the leader server to retrieve data from the topic. In response, they receive messages immediately after the Kafka producer has received confirmation from the server (in cases where the "acks_config" parameter is "1" or "all") or as soon as the event is added to the logging journal on the leader server (when "acks_config" is "0"). From Fig. 1 can be concluded that the end-to-end latency of the Kafka event streaming process can be measured by time T_L, which equals the sum of the following time periods:

1. The time required to collect events into a batch before sending them to the server is referred to as producer time ($T_{producer}$).
2. The time from when the event data are received in the receive socket buffer to when they are saved on the leader-server disk, referred to as the leader-server commit time ($T_{leadercommit}$).
3. The time from when the message is saved on the leader-server's disk to the completion of its copying on the servers included in the Kafka cluster, referred to as replica time ($T_{replica}$).

4. The time from when the Kafka producer receives confirmation from the leader-server ("acks_config"=1 or "acks_config"="all"), or when a certain amount of data is saved on the disk of the leader-server ("acks_config"= 0), to when the Kafka consumer receives the event data, referred to as consumer fetch time (T_{fetch}).

Therefore, the formal definition of the end-to-end latency time (T_L) of Kafka cluster is provided in equation (1):

$$T_L = T_{producer} + T_{leadercommit} + T_{replica} + T_{fetch}. \quad (1)$$

A previous study [10] confirmed that the log-based architecture of Kafka is optimal for scalable, durable, and high-throughput data ingestion and predictive maintenance in Condition Monitoring (CM). The study concluded that Kafka outperforms RabbitMQ in producer throughput, whereas RabbitMQ achieves higher consumer throughput, demonstrating its superiority in rapid message consumption scenarios.

Study [11] proposed a model for Kafka cluster configuration comprising three subsystems in series: a producer group, an Apache Kafka cluster, and a consumer group, each containing three parallel units operating under a 1-out-of-3 strategy. The created model has been proven to improve system robustness and efficiency in handling failures in streaming data.

In a previous study [12], Apache Kafka was used as the backbone of the data ingestion layer to manage high-throughput data streams in real time for an Internet banking system. The Apache Kafka cluster was configured with multiple producers and consumers. This configuration ensured scalability by dynamically adjusting the data ingestion rate based on the number of active producers, making it suitable for high-velocity Internet banking data.

A latency-aware and resource-efficient approach to dynamic event consumer provisioning in distributed event queues for real-time cloud applications was explored in [13]. The proposed solution models consumer provisioning as a two-dimensional bin packing problem and addresses the challenge of blocking synchronization, which affects high-percentile latency. An extension to the bin-pack autoscaler is introduced to mitigate tail latency. The experimental results provide insights into optimizing the model for workloads with high variance in processing time.

From our perspective, the Kafka cluster configuration models proposed in [10-13] could further improve performance predictability by incorporating a machine learning model capable of forecasting Kafka cluster performance based on configuration parameters and performance metrics.

Feedforward Neural Networks (FFNNs), combined with gradient descent optimization techniques such as backpropagation and algorithms such as Adaptive Moment Estimation (Adam), are widely regarded as foundational machine learning models for regression and classification tasks. FFNNs architecture includes one input layer, one or more hidden layer(s), and a single output layer. The number of neurons in the input and hidden layers and the number of hidden layers in a FFNN are critical architectural choices that significantly affect the network's performance [14]. If these parameters are incorrectly chosen, it can negatively impact the model's ability to learn from the data and achieve high accuracy or generalization. Currently, there is no universally recommended procedure to determine the optimal number of units or layers in an FFNN architecture for forecasting Kafka cluster performance. Therefore, in this study, we propose a method that specifies how to define FFNN architecture based on the Bayesian Network structure.

FFNN training begins by setting initial values for the following parameters: weights and bias. A learning rate that determines the step size for weight updates during optimization must be specified as a hyperparameter [15].

Numerous studies have discussed various weight initialization methods for neural networks and emphasized their importance as weight start significantly influences neuron activation [16]. Orthogonal initialization is a recent method that initializes weights as orthogonal matrices, which helps preserve the norms of activations and gradients, contributing to stable training dynamics. Sparse initialization initializes weights with a sparse structure, promoting sparsity in network activations and facilitating efficient computation [17].

Kaiming Uniform initialization is designed to work with Rectified Linear Unit (ReLU) activation function. The weights of each layer are initialized from a uniform distribution with zero mean and a variance calculated based on the number of neurons *n* in the layer [18]:

$$\sigma^2 = 2/\mathrm{n}, \text{ or } \sigma = \sqrt{2/\mathrm{n}}. \quad (2)$$

Once the variance σ^2 is calculated according to (2), the weights are sampled from a uniform distribution in the range $[-\sqrt{3}\sigma, \sqrt{3}\sigma]$.

Given that weights in a neural network dictate the strength of the inputs in determining the output of a neuron, we propose setting their initial values based on the strength of influence obtained from a sensitivity analysis on a Bayesian network. However, to keep the variance of activations and gradients relatively constant across different layers of the network, we scale the values of the strength of influence using the Kaiming He scale factor (2), which is specifically designed for networks with ReLU activation function.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

1.4. Objectives and tasks

This study aims to design the architecture of a Feedforward neural network model based on the discrete Bayesian Network and a new method for setting the initial weights that connect neurons across layers. The goal of the neural network is to predict future latency issues based on present and historical cluster configurations and performance metrics. To achieve these objectives, the following tasks must be completed:

1. A discrete Bayesian network was developed and validated to diagnose and understand the factors influencing end-to-end latency in Kafka clusters.
2. A sensitivity analysis was conducted on the discrete Bayesian network to identify the strength of the factors influencing the Kafka cluster latency.
3. To initialize weights in FFNN, create a matrix of initial weights derived from the sensitivity analysis in the Bayesian network.
4. Design the FFNN architecture and configure its parameters.
5. Train and evaluate FFNN model.

Paper Structure. In Section 1 after the "Introduction" the "Motivation**"** formulates the challenges with Kafka cluster performance predictability and specifies how this study's goal proposes to address those challenges. The "State of the art" provides the summaries of the related research. The main tasks to be resolved in achieving the current research goal are listed in the "Objectives and tasks" section.

Section2. "Materials and Methods" includes the formal specifications of the methods and models employed in this study.

Section 3. "Development of Feedforward Neural Network Model" details the steps taken to construct a discrete Bayes network model to diagnose Kafka cluster end-to-end latency. This section also describes the process of deriving Feedforward Neural Network architecture from to forecast Kafka cluster end-to-end latency.

Section 4. "Experimental Study" describes the steps followed to gather the data required for learning the parameters of the Bayesian network and parameters to be used to train a Feedforward neural network. Metrics for evaluating the quality of a developed Bayesian Network. Metrics for evaluating the proposed Feedforward Neural Network model's quality.

Section 5. "Results and Discussion" includes the practical outcomes of these tests.

The Conclusion section outlines the recommendations drawn from this research.

2. Materials and research methods

The defined tasks (1-6) will be addressed through sequential processes, where the results of the process "Design Bayes Network Model" will serve as input data for the subsequent process "Design Neural Network Model" (Fig. 2). Each process is described in detail below.

1.1. Define Bayes Network structure. The structure of the discrete Bayesian network that models the relationship between X and Y is an acyclic graph where Y is a set of hidden nodes $Y_{j=\overline{1,n}} = \{y_j\}$ that dependent on the set of observed nodes $X_{i=\overline{1,k}} = \{x_i\}$, which serve as independent parent nodes. The influence of the parent nodes x_i on y_i is expressed by the joint probability according to Bayes' rule: $P(y_1, y_2, \dots y_n) = \prod_{i=1}^{n} y_i|X$.

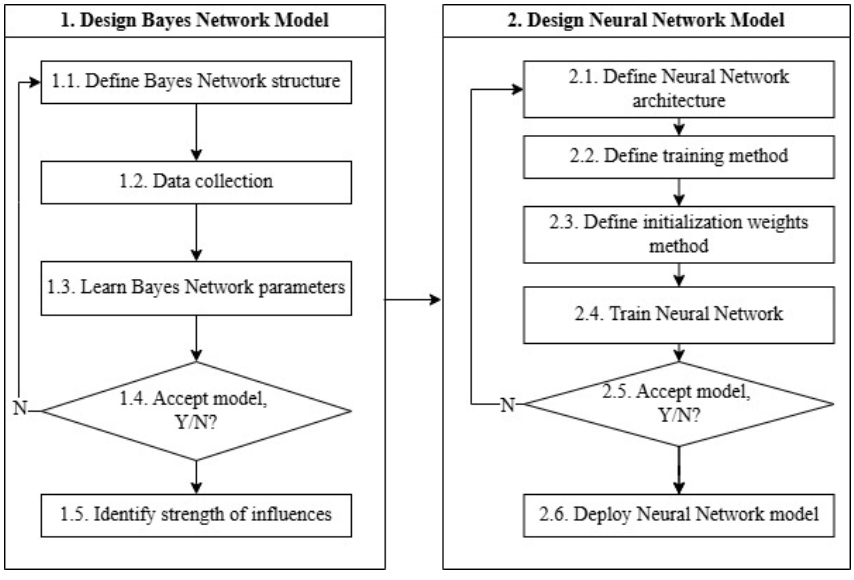

Fig. 2. Method to define FFNN architecture based on Bayes Network

The prior probabilities for $x_i \in X$ are defined by parameter θ_X (3) and the conditional probabilities of $y_j \in Y$ are defined by θ_y (4).

$$\theta_X = \left\{\theta_{x_{11}}, \dots, \theta_{x_{1N[s_{x_1}]}}, \dots, \theta_{x_{k1}}, \dots, \theta_{x_{kN[s_{x_k}]}}\right\}, \quad (3)$$

$$\theta_y = \left\{\theta_{y_1|x_{11}}, \dots, \theta_{y_1|x_{1N[s_{x_1}]}}, \dots, \theta_{y_m|x_{11}}, \dots, \theta_{y_m|x_{1N[s_{x_1}]}}\right\}, \quad (4)$$

where $N[s_{x_1}], N[s_{x_2}], N[s_{x_k}]$ – the number of states S that are defined for $x_i \in X$.

When the observed data to train Bayes net exists then θX, θy can be computed using the likelihood function $L(\theta|D)$ (5) which represents the joint distribution of the probabilities of the observed data D [18, 19].

$$L(\theta|D) = \prod_{t=1}^{d} P(X(t), Y(t)|\theta) = \prod_{t=1}^{d} P(X(t)|\theta)P(Y(t)|X(t), \theta) = \theta_{x_{11}}^{N[x_{11}]} \cdot \dots \cdot \theta_{x_{1N[s_{x_1}]}}^{N\left[x_{1N[s_{x_1}]}\right]} \cdot \dots \cdot \theta_{x_{k1}}^{N[x_{k1}]} \cdot \dots \cdot \theta_{x_{kN[s_{x_k}]}}^{N\left[x_{kN[s_{x_k}]}\right]} \cdot \dots \cdot \theta_{y_1|x_{11}}^{N[y_1|x_{11}]} \cdot \dots \cdot \theta_{y_1|x_{1N[s_{x_1}]}}^{N\left[y_1|x_{1N[s_{x_1}]}\right]} \cdot \dots \cdot \theta_{y_m|x_{11}}^{N[y_m|x_{11}]} \cdot \dots \cdot \theta_{y_m|x_{1N[s_{x_1}]}}^{N\left[y_m|x_{1N[s_{x_1}]}\right]}, \quad (5)$$

where $N\left[y_1|x_{1N[s_{x_1}]}\right], N\left[y_m|x_{1N[s_{x_1}]}\right]$ – the number of times $\theta_{y_1|x_{1N[s_{x_1}]}}$ is included in (5).

1.2. Data collection. Since the prior probabilities of the observed variables *X* are unknown it is necessary to create a data set *D* in order to calculate θ_X, θ_y. The data will be collected by executing different scenarios created to test Kafka cluster performance under different system loads. The collected continuous values of performance measures from Kafka Producers, Servers, and Consumers must be discretized to be utilized in the discrete Bayesian network model. Hierarchical clustering methods groups data based on two criteria: distance metric and linkage method, beginning when each data point is a separate cluster and merging them until a single cluster is formed. When the linkage method is "Ward" and the distance metric is either "Euclidean" or "Manhattan," the resulting clusters tend to be relatively compact, equally sized, and more robust to outliers [20]. Therefore, hierarchical clustering will be employed in this research to discretize the continuous values.

1.3. Learn Bayes Network parameters. The task of determining the network parameters is to find the solution of equations (5) in partial derivatives.

1.4. Accept model. The data collected in step 1.2 includes the logged values of the end-to-end latency of Kafka cluster for the given configuration parameters and the observed performance metrics. These data will be used to compare the evidence from the Bayes network. The developed Bayes Network model will be accepted if the expected test results match the actual results.

1.5. Identify the strength of influence. The posterior probability $P(Y|X)(p)$ of the child variable Y due to a change in the parameters of the parent variable X is expressed as the ratio of two linear functions of the parameter (p) (6)

$$P(Y|X)(p) = \frac{a \cdot p + b}{c \cdot p + 1}, \quad (6)$$

where a, c – the angular coefficients in the linear equation;

b – the shift along the OY- axis;

p – is the probability that the network parameters will take certain values.

The partial derivative of $P(Y|X)(p)$ with respect to p measures the sensitivity of child node $y \in Y$ to changes in the parent node. The derivative is given by:

$$Dr = \frac{\partial(P(Y|X)(p))}{\partial p} = \frac{a - bc}{(c \cdot p + 1)^2}. \quad (7)$$

The strength of the influence I_{ij} from changes in the values of the parent node (i) on the posterior probability (6) of the child node (j) is determined by the product of parameter range's interval width W_i and the absolute value of the derivative (7). It is calculated using the following expression [20]:

$$I_{ij} = W_i \cdot Dr. \quad (8)$$

2.1. Define Neural Network architecture. Given that the "Design Bayes Network" process is completed with the validated model and the calculated strength of the influence I_{ij} as per (8). The architecture of Feedforward fully connected Neural Network is defined by the following design principles: $X_{i=\overline{1,k}} = \{x_i\}$ neurons will form the input layer; the neurons in the hidden layers correspond to the hidden variable Y of Bayes net and the number of hidden layers is derived from the structure of a Bayes Network. A single neuron in the output layer will be "end-to-end latency" as described by (1). Neurons between layers are fully connected; however, the initial weights for connections that do not exist in the Bayes Network model are set to 0.

2.2. Define the training method. For each hidden layer (l) with weights W^l and biases b^l, the output h^l and preactivation z^l are calculated as follows:

$$h^l = ReLu(W^l h^{l-1} + b^l), \quad (9)$$

where ReLU is f(x)=max(0,x).

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

For the output layer L with weights W^L and bias b^L predicted value $\hat{y}$ with linear activation function is as follows:

$$\hat{Y} = W^L h^{L-1} + b^L \quad (10)$$

2.3 Define the initialization weights method. Given an adjacency matrix of the Bayes Network $A_{m\times m}$, has $a_{ii}=0$; $a_{ij}=1$ when node i is connected to node j and $a_{ij}=0$ otherwise; m= |X|+|Y| is a total number of nodes in the Bays Network. Then the corresponding initial weights matrix W for FFNN is set as follows:

$$w_{ij} = \begin{cases} 0, \text{where } a_{ij} = 0 \\ I_{ij}, \text{ where } a_{ij} <> 0' \end{cases} \quad (11)$$

where I_{ij} is the strength of influence (8).

2.4. Train Neural Network. The training of the FFNN will aim to minimize a loss function, such as Mean Squared Error (MSE) for regression tasks.

2.5. Accept model. The proposed FFNN model will be accepted when the number of epochs increases and the value of MSE consistently decreases.

2.6. The accepted FFNN to be deployed and ready to use with new datasets to forecast Kafka cluster end-to-end latency.

3. Development of Feedforward Neural Network Model

3.1. Design a discrete Bayes Network model structure to diagnose Kafka cluster end-to-end latency

A discrete Bayesian network to diagnose the growth of end-to-end latency in an Apache Kafka cluster defines the target node to be "end-to-end latency time (T_L)", and the Kafka cluster configuration parameters are considered as the observed variables, and the Kafka performance metrics are treated as the hidden variables for each term in equation (1). To describe causal dependencies using an acyclic graph that represents the Bayes Net structure, we employ a notation in which a set with a hidden node Y is identified by the function f_m^n This function includes parameters X, which are the observed nodes. Here, 'n' denotes the level of the node in the graph, and 'm' is the index of the term in equation (1). The definitions for sets X and Y are provided below.

The elements of set X_1={ $x_{11},\ldots,x_{15}$} include:

x_{11}="acks_config" configures the level of acknowledgment required from the leader-server for producers, determining when a message write is considered successful. It directly influences the trade-offs between message delivery durability and availability, thereby affecting the reliability of data transmission within a Kafka cluster;

x_{12}="buffer.memory" specifies the total amount of memory that the producer can use for buffering. If the buffer is completely filled, additional messages will be blocked or discarded depending on the blocking policy;

x_{13} = "max.inflight.requests.per.connection" - defines the maximum number of unacknowledged requests that can be sent to the server on one connection. If this maximum is reached, the producer's batches will be blocked until confirmation from the server is received;

x_{14}="socket.receive.buffer.bytes" specifies the network socket buffer size for receiving data. The chosen buffer size can affect the server's message processing time;

x_{15} = "max.poll.records" defines the maximum number of records that a Kafka consumer can handle in one call to the poll() method.

A higher value can reduce queue times if the consumer handles larger batches efficiently, thereby decreasing response delays.

The elements of set Y_1 = {y_{11}, y_{12}} are performance measures as follows:

y_{11}=RequestQueueTimeMs (Request Queue Time Milliseconds) measures the time a request spends waiting in the request queue before being processed by the broker. High values indicate that the broker is overloaded;

y_{12}=ResponseQueueTimeMs (Response Queue Time Milliseconds) measures the time a response spends in the response queue after being processed and before being sent back to the client. A high value indicates that the broker struggles to promptly dispatch responses.

The elements of set X_2={x_{21}, x_{22}, x_{23}} are the performance measures:

x_{21}="log.flush.interval.ms" specifies the maximum time, in milliseconds, that a message can remain in the log buffer before it is flushed to disk. Setting a lower value for this parameter means that logs will be flushed to disk more frequently, which could result in an increase in the log flush rate but potentially decrease the flushing time (log flush time) because the amount of data to be written at each flush could be smaller. Conversely, a higher value for this parameter could decrease the log flush rate while increasing the log flush time as more data accumulates before each flush;

x_{22}="replica factor" determines the number of data copies (replicas) that will be maintained across different brokers. A higher replica factor not only increases redundancy and fault tolerance but also impacts log flush dynamics. More replicas mean that each message needs to be flushed in multiple places, potentially increasing the overall time taken for flush operations (as well as the system's I/O overhead). This might lead to a decrease in the

log flush rate because the system is managing more flush operations across replicas;

x_{23}=replica.fetch.min.bytes indicates the minimum amount of data that a follower replica must collect before sending a fetch request to the leader replica. Increasing the replica.fetch.min.bytes results in larger batch sizes being fetched by each follower. Larger batch sizes can improve throughput but might result in fewer fetch requests. If the followers fetch data less frequently but in larger batches, the leader might accumulate more unflushed data, potentially increasing the amount of data to flush when the log flush occurs. The impact on resource utilization might decrease the log flush rate.

The elements of set $Y_2 = \{y_{21}\}$ is a performance measure: y_{21}="LogFlushRateAndTimeMs" (Log Flush Rate Milliseconds) represents the rate and time taken to flush log data from memory to disk. This metric is crucial for understanding the performance and efficiency of data durability and storage in your Kafka cluster.

The elements of set $X_3=\{x_{31},...,x_{34}\}$ are Kafka configuration parameters:

x_{31}= "linger.ms" - how long the producer will collect event data to form a batch. If this parameter is set to a value greater than zero, the producer will accumulate messages in the buffer for a specified time;

x_{32} = "batch.size" the maximum batch size in bytes. Once the batch reaches this size, it will be sent regardless of whether the time specified by "linger.ms" has elapsed;

x_{33} = "compression.type" determines the message compression type, which requires additional processing time before being sent;

x_{34} = "fetch.min.bytes" defines the minimum number of bytes that must be copied to the disk of the leader-server before becoming available for Kafka consumers to fetch. If the threshold set by fetch.min.bytes is not reached, the server leader will wait until a sufficient amount of data accumulates before sending a response to the consumer. The use of fetch.min.bytes can balance the number of requests and the scale of transmitted data.

The elements of set $Y_3 = \{y_{31}, y_{32}\}$ are performance measures as follows:

y_{31}=BytesInPerSec measures the total number of bytes being received per second by a Kafka server from all producers. High values may indicate the need to add processing resources or adjust producer configurations;

y_{32}=BytesOutPerSec – measures the rate at which data are sent from the Kafka brokers to the consumers. While high values can indicate good consumer throughput, they can also signal that consumers are demanding data at a rate that might strain the server, especially if combined with high values in BytesInPerSec.

The elements of set $X_4=\{x_{15},x_{31}, x_{32}, x_{33}, x_{41}\}$ are the performance measures:

x_{15}, x_{31}, x_{32}, x_{33} are from sets X_1,X_3 respectively;

x_{41}= "the number of producers " - the relationship between the number of producers and the producer average batch size (batch.size.avg) can be influenced by the configuration of the producers. As the number of producers increases, the contention for network and broker resources can also increase, potentially leading to backpressure and longer wait times for batch accumulation. In this case, each producer might reach its batch.size limit more frequently due to data sending delays, potentially reducing the batch.size.avg if not all producers are consistently filling their batches to the maximum configured size.

The elements of set $y_4 = \{y_{41}\}$ is a performance measure: y_{41}= "batch_size_avg" a producer average batch size measures the average size of message batches sent by the producer to a Kafka broker. A larger average batch size means that more records are sent per request, which can improve throughput but may also result in higher latencies.

The metric to evaluate a node "end-to-end Kafka cluster latency time T_L" is equal to the sum of

$$y_1^1 = f_1^1(y_{11}, y_{21}, y_{31}, y_{41}); \; y_2^1 = f_2^1(y_{12}, y_{21}, y_{32});$$
$$y_3^1 = f_3^1(y_{12}, y_{21}, y_{32}, y_{41}); \; y_4^1 = f_4^1(y_{11}, y_{12}, y_{21}),$$

where y_1^1 is TotalTimeMs, request=Produce which equates to the total time taken to handle a Produce request. It is directly influenced by:

y_{11}, where a long queue could increase the total handling time;

y_{21}, as frequent or slow log flushes can affect the processing speed of a produce request;

y_{31} since heavier incoming data rates might slow down processing;

y_{41}, as larger batch sizes might take longer to process times.

y_2^1 presents TotalTimeMs for the request=Fetch which corresponds to the total time needed to complete a fetch request. It is impacted by:

y_{12}, where delays in response handling can extend the total time;

y_{21}, as, if data needs to be fetched from disk, flush rate/times can play a significant role;

y_{32}, where high output rates can indicate faster retrieval but also depend on network and broker load;

y_3^1 is TotalTimeMs, request=FetchConsumer - related to fetch requests initiated by consumers and can be affected by the same factors as TotalTimeMs, request=Fetch but additionally by y_{41} depending on how quickly batches are gathered and sent to consumers, impacting total fetch time for consumers;

y_4^1 represent TotalTimeMs, request=OffsetCommit - time taken to commit offset details, is influenced by $y_{11;}$

y_{12} as any queuing delays directly add to the total commit time; y_{21}, where committing an offset might require log interactions, thereby affected by log operations.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

With the above knowledge the structure of Bayes Net is specified as follows:

$$Y_1 = f_1^0(X_1); Y_2 = f_2^0(X_2); Y_3 = f_3^0(X_3); Y_4 = f_4^0(X_4);$$
$$Z = y_1^1 + y_2^1 + y_3^1 + y_4^1$$

and it's schema is illustrated on Fig. 3.

To describe Bayes Network (Fig. 3) with variables in the study, the following notation is used:

<N, S, G(S)>, (12)

where N – name of variable;

S – a set of state $S_{i=\overline{1,N}} = \{s_i\}$ expressed as linguistic terms;

G(S) –a set of values for each state $s_i \in S$.

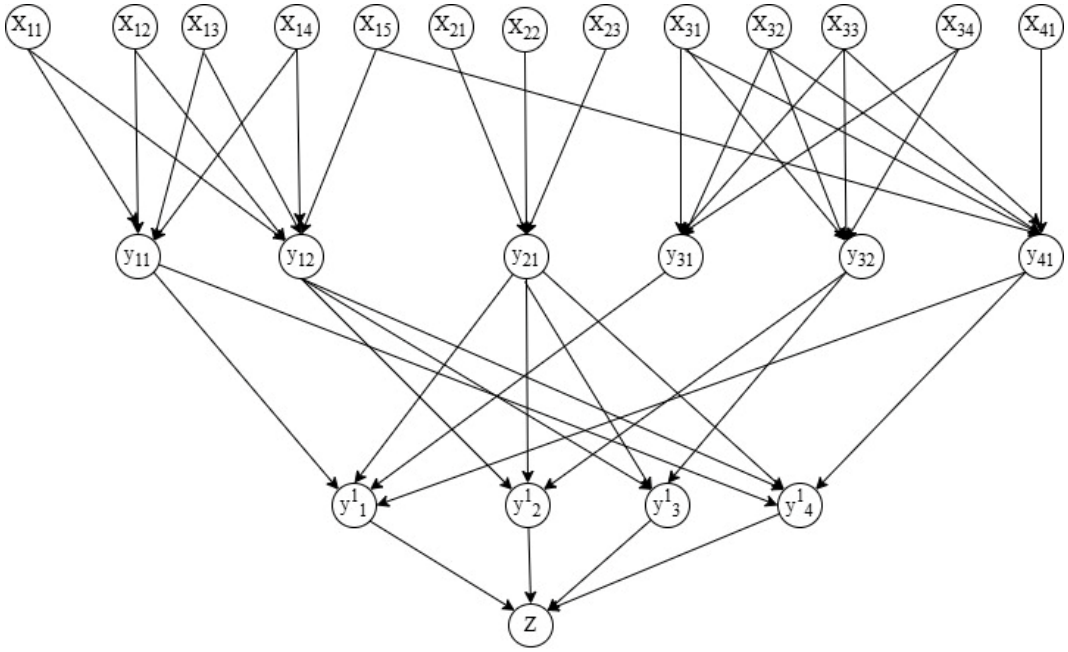

Fig. 3. Structure of a discrete Bayes Network for diagnosing Kafka cluster end-to-end latency

3.2 Design an architecture of Feedforward Neural Network discrete Bayes Network model structure to diagnose Kafka cluster end-to-end latency

Based on the structure of Bayes Network (Fig. 3) an architecture of fully connected FFNN is recorded in Table 1 and illustrated on Fig. 4.

Equation (13) specifies the predicted value of the Kafka cluster's end-to-end latency corresponding to the architecture outlined in Table 1.

$$\hat{y} = f\left(W^3 \mathrm{ReLU}\left(W^2 \mathrm{ReLU}\left(W^1 x_{k\times 1} + \vec{b}_n^1\right) + \vec{b}_p^2\right) + \vec{b}_z^3\right), \quad (13)$$

where f(z)=z; W^1, W^2, W^3 – weights initialization matrix (14) - (16).

Solid lines on Fig. 4 correspond to connections with initial weights not equal to zero in matrices (15) – (17). Dashed lines indicate connections for which no influence was identified based on sensitivity analysis.

Table 1

An architecture of fully connected FFNN

Layer	Layer neurons	Initial weights scaled by Kaiming He scale factor	Bias
Input, k=13	$x_{k\times 1} \epsilon X$	$W^1 = \left(\sqrt{2/k}\right) W^1_{n\times k}$	$\vec{b}_n^1 \leftarrow 0$
First hidden, n=6	$\{y_{11}, y_{12}, y_{21}, y_{13}, y_{23}, y_{41}\}$	$W^2 = \left(\sqrt{2/n}\right) W^2_{p\times n}$	$\vec{b}_p^2 \leftarrow 0$
Second hidden, p=4	$y_1^1; y_2^1; y_3^1; y_4^1$	$W^3 = \left(\sqrt{2/p}\right) W^3_{z\times p}$	$\vec{b}_z^3 \leftarrow 0$
Output	Z		

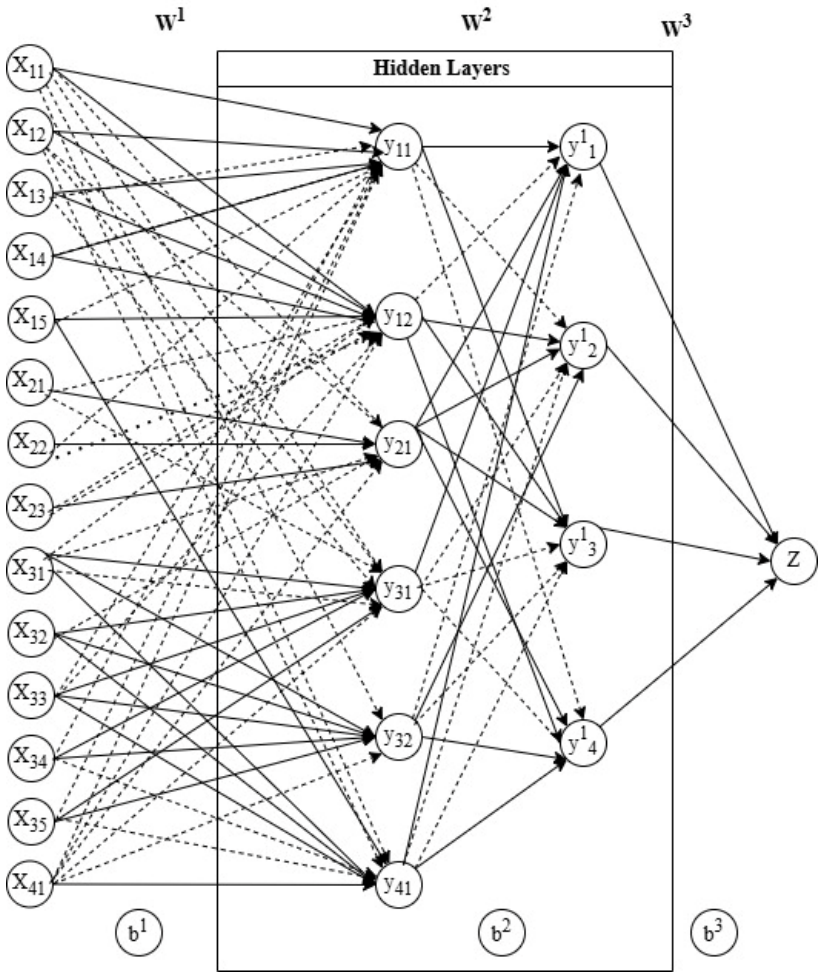

Fig. 4. Fully connected FFNN to forecast Kafka cluster end-to-end latency

$$W^1_{6\times 13} = \begin{pmatrix} w^1_1 & w^1_3 & w^1_5 & w^1_7 & & & & & & & & & & \\ w^1_2 & w^1_4 & w^1_6 & w^1_8 & w^1_9 & & & & & & & & & \\ & & & & & w^1_{11} & w^1_{12} & w^1_{13} & & & & & & \\ & & & & & & & & w^1_{14} & w^1_{17} & w^1_{20} & w^1_{23} & & \\ & & & & & & & & w^1_{14} & w^1_{18} & w^1_{21} & w^1_{24} & & \\ & & & & w^1_{10} & & & & w^1_{16} & w^1_{19} & w^1_{22} & & w^1_{25} & \end{pmatrix}, \quad (14)$$

$$W^2_{4\times 6} = \begin{pmatrix} w^2_1 & & w^2_6 & w^2_{10} & & w^2_{13} \\ & w^2_3 & w^2_7 & & w^2_{11} & \\ w^2_2 & w^2_4 & w^2_8 & & & \\ & w^2_5 & w^2_9 & & w^2_{12} & w^2_{14} \end{pmatrix}, \quad (15)$$

$$W^3_{1\times 4} = (w^3_1 \quad w^3_2 \quad w^3_3 \quad w^3_4). \quad (16)$$

4. Experimental Study

4.1. Data Collection

The observed variables for Bayes Network with the structure on Fig. 3 are defined according to (13) in Table 2 and hidden variables are described by states {"LowTime", "MeanTime", "HighTime"} and the set of values for each state is identified programmatically by executing hierarchical clustering.

To collect a data to learn Bayes Network parameters (5), (6) we designed scenarios to test different Kafka cluster characteristics:

1. A scenario to measure a Kafka cluster latency with high throughput emphasis with durability.
2. A scenario to measure a Kafka cluster latency with low latency and average throughput.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Table 2

Specification of variable for Bayes network

Name of variable	Set of states	Set of values
acks	“none”, “leader”, ”all”	0,1,”all”
batch.size	“small”, “moderate”, ”large”	[1000..8192], (8192..16384],(16384..32768]
replica factor	“no fault tolerance”, “fault tolerance”	“1”,”2”
linger.ms	“none”, “default”, “high”	0,[1..50),[50..100]
compression type	“enable”, “disable”	“none”,“gzip”,”snappy”
buffer.memory	“small”, ”default”, “large”	[1MB..8MB],(8MB..50MB],(50MB..96MB]
max.inflight.requests.per.connect	“single”, “moderate”, “high”	1,(2..5],(5..15]
socket.receive.buffer.bytes	“small”, ”default”, “large”	[100KB..500KB],(500KB..1MB),[1MB..2MB]
log.flush.interval.ms	“frequent”, “less frequent”, “not frequent”	[0..100),[100..500),[500..1000]
replica.fetch.min.bytes	“frequent”, “less frequent”, “not frequent”	[1KB..100KB],(100KB..50KB],(50KB..1MB]
max.poll.records	small, moderate, large	[100..300],(300..500],(500..2000]
fetch.min.bytes	very frequent, frequent, not frequent	[1KB..100KB],(100KB..50KB],(50KB..1MB]
number of producers	small, moderate, large	[0..5),[5..10),[10..15]

3. A scenario to measure a Kafka cluster latency with a balanced throughput and latency with fault tolerance.

4. A scenario to measure a Kafka cluster latency with a stress test.

5. A scenario to measure a Kafka cluster latency with a high fetch size for Bulk Processing.

Table 3 lists the values of the observed variables for each scenario. For each scenario Kafka producer will be sent 400 messages making the final dataset to include 2000 observations.

The experiments will be conducted on a system with processor 11th Gen Intel(R) Core(TM) i7-1185G7@ 3.00GHz 3.00 GHz and 32 GB RAM. Apache Kafka version: 3.8.1. Bayes Network construction, learning paraments and sensitivity analysis will be performed in program GeNIe Academic. Development of FFNN will be performed in Python.

Table 3

The values of the observed variables for scenario 1-5

Scenario № / Name of variable	1	2	3	4	5
acks	all	leader	all	none	all
batch.size	large	small	moderate	large	large
replica factor	fault tolerance	no fault tolerance	Fault tolerance	no fault tolerance	Fault tolerance
linger.ms	high	none	default	none	high
compression type	enable	disable	enable	disable	enable
buffer.memory	large	default	default	small	large
max.inflight.requests.per.connect	moderate	single	moderate	high	moderate
socket.receive.buffer.bytes	large	small	small	small	large
log.flush.interval.ms	less frequent	frequent	less frequent	frequent	not frequent
replica.fetch.min.bytes	not frequent	Frequent	not frequent	not frequent	not frequent
max.poll.records	large	small	small	large	large
fetch.min.bytes	frequent	very frequent	frequent	very frequent	not frequent
number of producers	large	small	large	large	small

4.2. Metrics for evaluating the quality of a developed Bayesian Network

The normalized log-likelihood is used as a metric to evaluate how effectively the Bayesian Network has learned patterns in a given dataset D:

$$\frac{\ln(L)}{n} = \frac{1}{n}\sum_{i=1}^{n}\ln(P(x_i)), \quad (17)$$

where $P(x_i)$ represents the joint probability of the i-th observation, calculated using the conditional probability tables (CPTs) learned from the dataset D, and n denotes the total number of observations in D.

This metric, $\frac{\ln(L)}{n}$ provides a normalized measure of model fit by indicating the average log-probability assigned to each observation. A widely used set of practical guidelines for interpreting the normalized log-likelihood is provided within the documentation for the influential Bayesian network software, Netica (Norsys, 2023). These heuristics suggest:

$\frac{\ln(L)}{n} > -0.5$ the Bayesian network fits data with high accuracy, and well-captured relationships. No further revision is required.

$-1.0 < \frac{\ln(L)}{n} \leq -0.5$ the Bayesian network effectively captures key patterns and dependencies. The model is deemed acceptable, and immediate structural improvement is not required.

$-1.5 < \frac{\ln(L)}{n} \leq -1.0$ the Bayesian network has an acceptable fit, however further refinement of the network structure is recommended to improve its ability to model the dataset.

$\frac{\ln(L)}{n} \leq -1.5$ – the Bayesian Network fails to adequately explain the data, indicating a poor fit. The model cannot be accepted and requires significant structural revision or redevelopment.

4.3. Metrics for evaluating the quality of a proposed FFNN model

The following evaluation metrics will be employed to compare the quality of the feedforward neural network (FFNN) with the proposed weight initialization method and architecture (Fig. 4) against the quality of an FFNN with the same architecture but initial weights set using the Xavier/Glorot uniform initialization method and to align with the evaluation framework in [21]:

Mean Absolute Error (MAE) measures how closely the model's predicted Kafka cluster latency ($\hat{L}_i$) aligns with the actual Kafka cluster latency (L) in the dataset:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|L_i - \hat{L}_i\right|. \quad (18)$$

Root Mean Squared Error (RMSE) penalizes deviations more heavily than MAE, as outliers have a squared impact. It is useful for evaluating the accuracy of the model in cases where large latency prediction errors are significant:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(L_i - \hat{L}_i\right)^2} \quad (19)$$

Coefficient of Determination R^2 to measure how well the model learned patterns in the dataset. Measures the proportion of variance in latency captured by the model:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(L_i - \hat{L}_i)^2}{\sum_{i=1}^{n}(L_i - \bar{L})^2}, \quad (20)$$

where $\bar{L}$ is the mean of the actual Kafka latencies.

Convergence speed measures how efficiently the model reaches an acceptable loss threshold (T) during training in terms of time:

$$S_t = \min\{t_i | Loss_i \leq T\}, \quad (21)$$

Where t_i is the training time at epoch i.
Alternatively, the convergence speed can be expressed in terms of epoch as follows:

$$S_i = \min\{i | Loss_i \leq T\}, \quad (22)$$

where i is the epoch index, the Loss is calculated based on Mean Squared Error (MSE).

Key Performance indicators to quantify the percentage improvement or degradation in key metrics such as R^2 and MSE when comparing the proposed Bayesian-guided weight initialization to the Xavier initialization, the following indicators will be calculated:
Percentage change in R2:

$$\Delta_{R^2}\,\% = \left(\frac{R_1^2 - R_2^2}{R_1^2}\right) \times 100, \quad (23)$$

where R_1^2, R_2^2 are the coefficients of determination of the models to be compared.

Percentage change in MSE:

$$\Delta_{MSE}\,\% = \left(\frac{MSE_1 - MSE_2}{MSE_1}\right) \times 100, \quad (24)$$

MSE_1, MSE_2 are the Mean Squared Errors of the models for comparison.

Statistical significance test to quantify whether the performance improvements are meaningful will be based on paired t-test:

$$t = \frac{\bar{a}}{s_d/\sqrt{n}}, \quad (25)$$

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

where $\bar{a}$ – mean of differences; s_d – Standard Deviation of differences.

t- test's the Null Hypothesis (H_0): no significant difference in performance between two FFNN models.

5. Results and Discussion

Because of executed scenarios 1-5, the values for hidden variables $y_1^1; y_2^1; y_3^1; y_4^1$ of Bayes network from Fig. 3 are recorded, and their values per scenario are illustrated on Fig. 5.

The lowest end-to-end latency, equal to 9.44 ms, was achieved in scenario 2 where only the leader replica must acknowledge the messages. This speeds up the process because acknowledgments from all replicas are not required; the absence of multiple replicas removes the overhead related to replicating data. Furthermore, disabling compression type helped avoid the time and computational power needed for data compression and decompression; the absence of multiple numbers for concurrent requests helped optimize the usage of resources. As a result, the TotalTime metric for all types is low, so their cumulative influence on end-to-end latency is minimal.

In scenario 4, the end-to-end latency was 13 ms, which was a 37% increase compared to scenario 2. This increase was due to allowing many numbers of unacknowledged requests to be sent to the server on one connection, which impacted the Total Time when the requests were "Produce" and "OffsetCommit". However, as no acknowledgments are required, the TotalTime for types "Fetch" and "FetchConsume" remained low, so there was not a drastic effect on end-to-end latency.

In the scenario 1 and 3, the end-to-end latencies were 40 ms and 43.38 ms, showing increases of 331% and 359.5% compared to scenario 2, respectively. The reasons for these increases include the requirement for acknowledgments from all servers, which negatively impacted the time it takes for each message to be considered successfully sent. Along with infrequent flushes, this impacted the TotalTime for requests labeled as "Produce" and "OffsetCommit." These last two factors contributed to end-to-end latency growth.

In scenario 5, the end-to-end latency was 122.42 ms, showing an increase in 1196% due to the requirement for acknowledgments from all replicas, which significantly increased the waiting times. The overhead of managing multiple replicas slowed down the data processing. "not frequent" settings in log.flush.interval.ms and replica.fetch.min.bytes delayed data flushing and fetching, contributing to increased response times, i.e., TotalTime, for requests of the "Fetch" and "FetchConsumer" types.

Figure 6 shows the developed Bayesian Network, which was constructed and trained using the parameters learned from the collected dataset. The calculated log-likelihood for the network is Ln(L)=-1785.6, and the corresponding normalized log-likelihood is $\frac{\mathrm{Ln(L)}}{n} = \frac{-1785.6}{2000} \sim -0.89$, where n=2000 represents the total number of observations in the dataset. A normalized log-likelihood of −0.89 falls within the "Good" fit quality category, indicating that the Bayesian Network is well-suited to model the dataset, and can be used to perform sensitivity analysis to identify the strength of influences between nodes in the network.

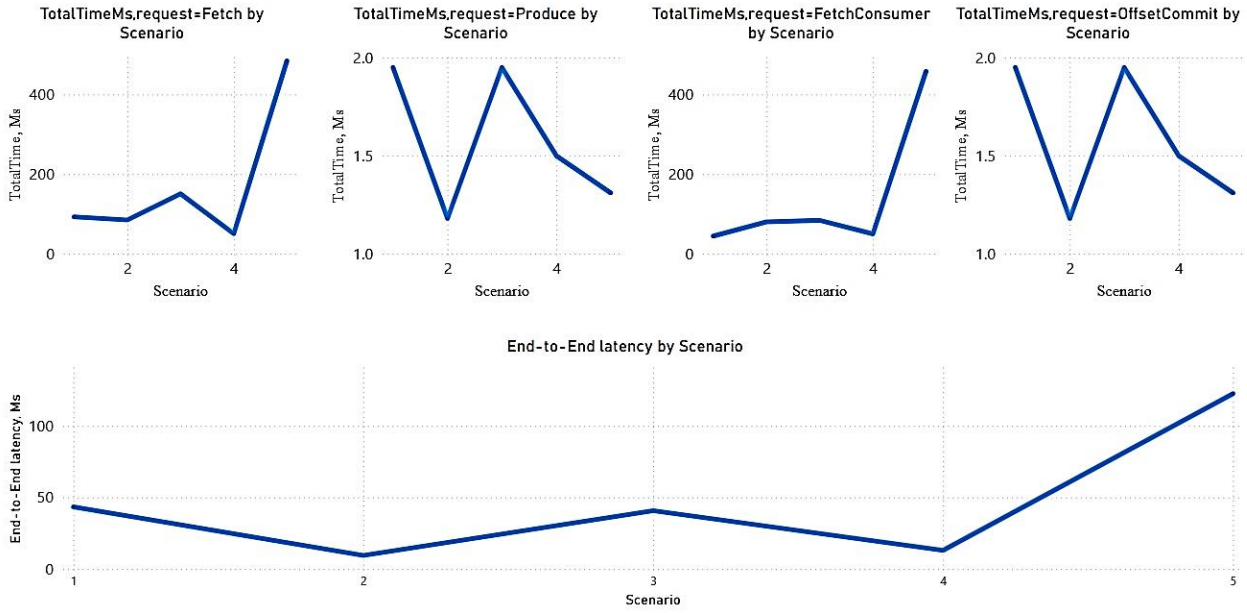

Fig. 5. Kafka Cluster performance metric values for the scenarios 1-5

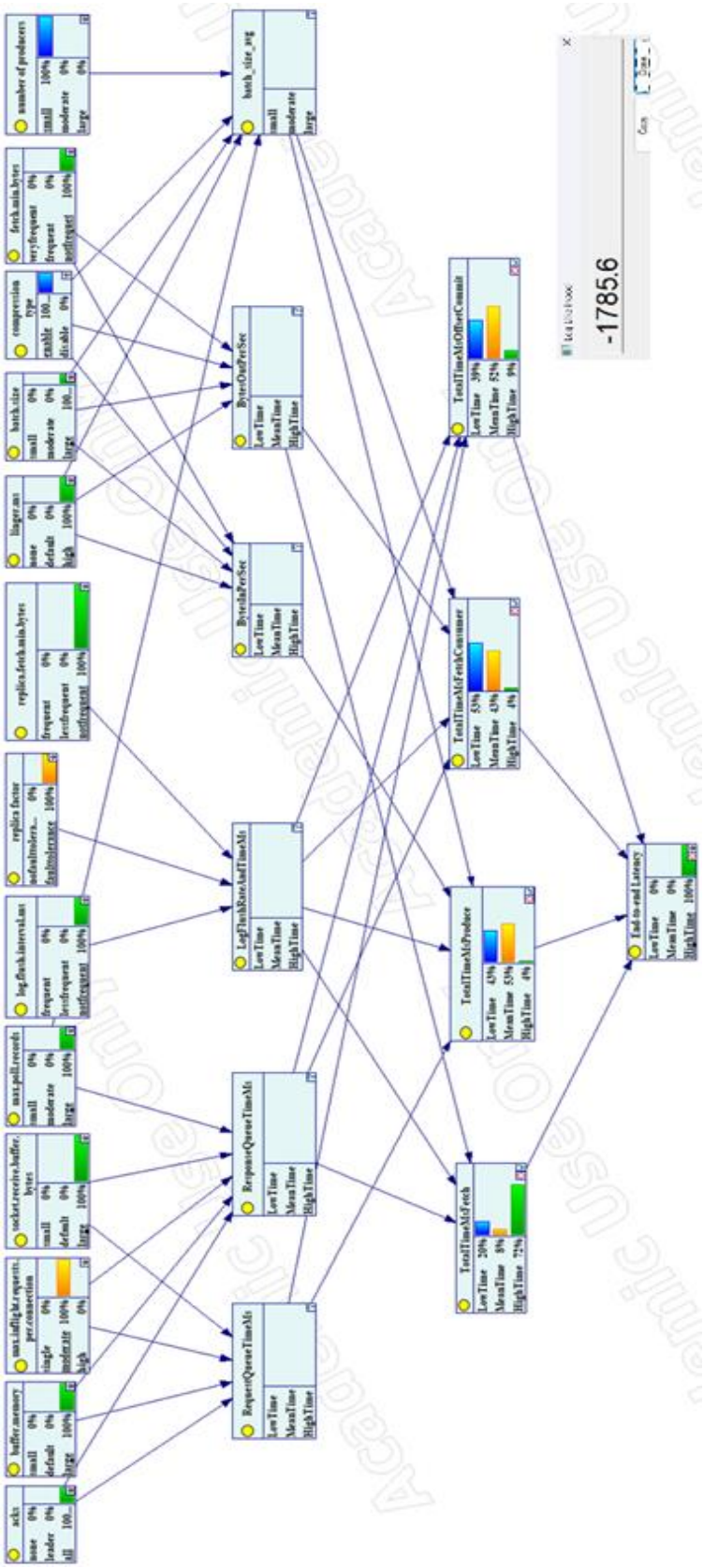

Fig. 6. Bayes Network to diagnose end-to-end latency of Kafka cluster with calculated evidence

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

The states of the observed variables are set for testing purposes according to scenario 5. The evidence obtained from the Bayesian network is as follows: the "end-to-end latency" of the Kafka cluster will be high when the TotalTime for the "Fetch" request is high, with a probability of 72%; additionally, "BytesInPerSec" and "BytesOutPerSec" are likely to be in a MeanTime state with probabilities of 52% and 56% respectively.

These results correspond to the test results obtained with scenario 5 and allow us to conclude on the capability of the developed discrete Bayesian network to diagnose growth in end-to-end latency in an Apache Kafka cluster by simulating the effects of changes in cluster configuration parameters.

The strength of the influences in the Bayes Network was identified as a result of the completed sensitivity analysis (Fig. 7). The different thicknesses of the network arcs represent the magnitude of the influence strength, which further indicates that the initial weights for FFNN model sampled from, for example, uniform or random distributions will not represent the connection influences correctly. The nodes "Fetch" and "FetchConsumer" are highlighted in red to emphasize their greater influence on the target node, "End-to-end latency", which aligns with the test scenario 5 result.

On Fig. 8 (a) is visualized a convergence training speed in terms of time and on Fig. 8 (b) – a convergence training speed in terms of the epoch for FFNN model with architecture according to Fig. 4 and weights initialize based on (14)-(16) (denoted as "Bayesian-Guided FFNN") and FFNN model with architecture according to Fig. 4 and weights initialized from a uniform distribution and scaled using the Xavier/Glorot Uniform start method (denoted as Standard FFNN). Bayesian-Guided FFNN model converged within 7.78 s and 744 epochs compared to 8.19 s and 943 epochs it took by Standard FFNN.

On Fig.9. are collected key measures to compare models' performance: Bayesian-Guided FFNN achieved an MSE of 186.12, a reduction of 18.14% relative to the Standard FFNN (227.24). A lower RMSE value of 13.64 for Bayesian-Guided FFNN versus 15.07 showed RMSE reduction by 9.48%. With a MAE of 10.89, the Bayesian-Guided FFNN outperforms FFNN (12.1), showing 9.99% lower error in absolute prediction terms. The Bayesian-Guided FFNN achieved a higher R^2 value of 0.96, demonstrating better predictive capability and accuracy in explaining the variance in the dataset. Improvement in R^2 0.914% relative to the Standard FFNN.

MSE distributions on Fig.10 show consistent differences but R^2 distributions are less differentiable, indicating minimal variability between the models in terms of variance explanation. The results of the statistical significance testing confirm that the Bayesian-Guided FFNN significantly outperforms the Standard FFNN in reducing the overall error ($p=0.000003<0.05$, $t=-4.68$). However, no statistically significant difference was observed in the ability of the models to explain variance ($p=0.491729>0.05$, $t=-0.69$).

6. Conclusions

A feedforward neural network (FFNN) model is proposed to forecast end-to-end latency in a Kafka cluster. To address the problem where the effectiveness of a neural network model depends on how well its architecture and parameters are selected, we defined a new methodology for designing the architecture of a FFNN model based on the discrete Bayesian Network and a new method for setting the initial weights that connect neurons across layers.

The proposed method involves two sequential processes. The results from the "Design Bayes Network" process – a verified discrete Bayesian Model to diagnose Kafka cluster latency and determine the strength of influence between nodes in the Bayes Network were used as input parameters for "Design Neural Network" process. These inputs are used to design an architecture for FFNN model and set the initial weights matrices based on the strength of the influences received on Bayes Network.

The constructed Bayesian Network achieved a normalized log-likelihood of − 0.89, which falls within the "Good" fit quality range. This score confirms that the network effectively models the collected dataset and captures the underlying probabilistic dependencies between the Kafka configuration parameters and latency metrics.

The created FFNN model was tested with a dataset collected from repeatedly executed scenarios in which Kafka cluster works under different system loads.

The developed FFNN model achieves significant improvements in prediction accuracy (reduction in MSE by 18.14%, showing the ability to reduce overall prediction errors more effectively than the Standard FFNN) and training efficiency (faster training convergence with reduced training time by 5% (7.78 s vs. 8.19 s for Standard FFNN); reduced the number of epochs to converge by 21% (744 epochs vs. 943 epochs), fulfilling the research goal. However, its ability to better explain variance (as reflected in R2) compared with the Standard FFNN requires further investigation. Overall, the developed FFNN model is a reliable and practical model for Kafka cluster latency forecasting, leveraging domain-specific knowledge for enhanced performance.

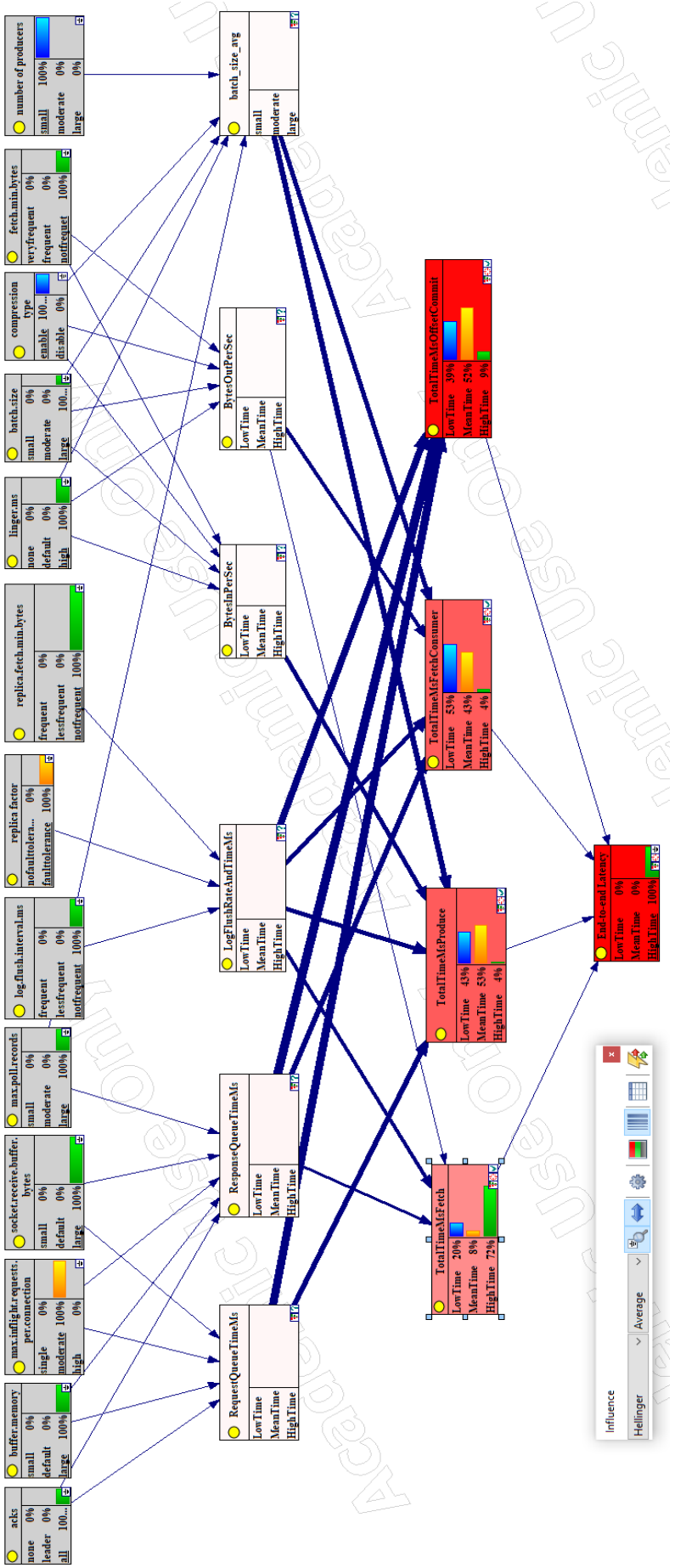

Fig. 7. Strength of influence from parameters changes on end-to-end latency of Kafka cluster

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

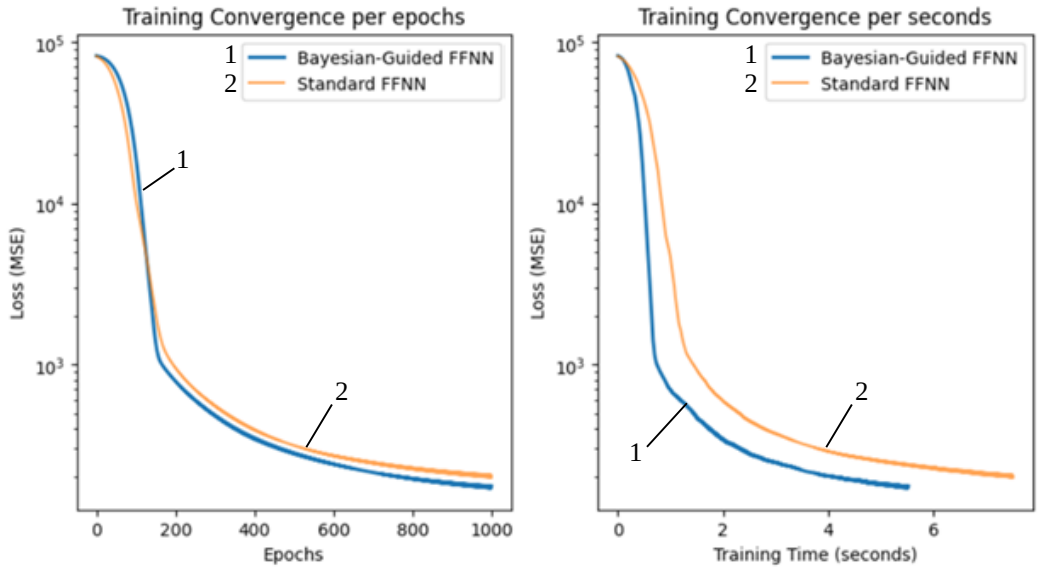

Fig. 8. (a) Convergence training speed in term of epochs; (b) convergence training speed in term of time

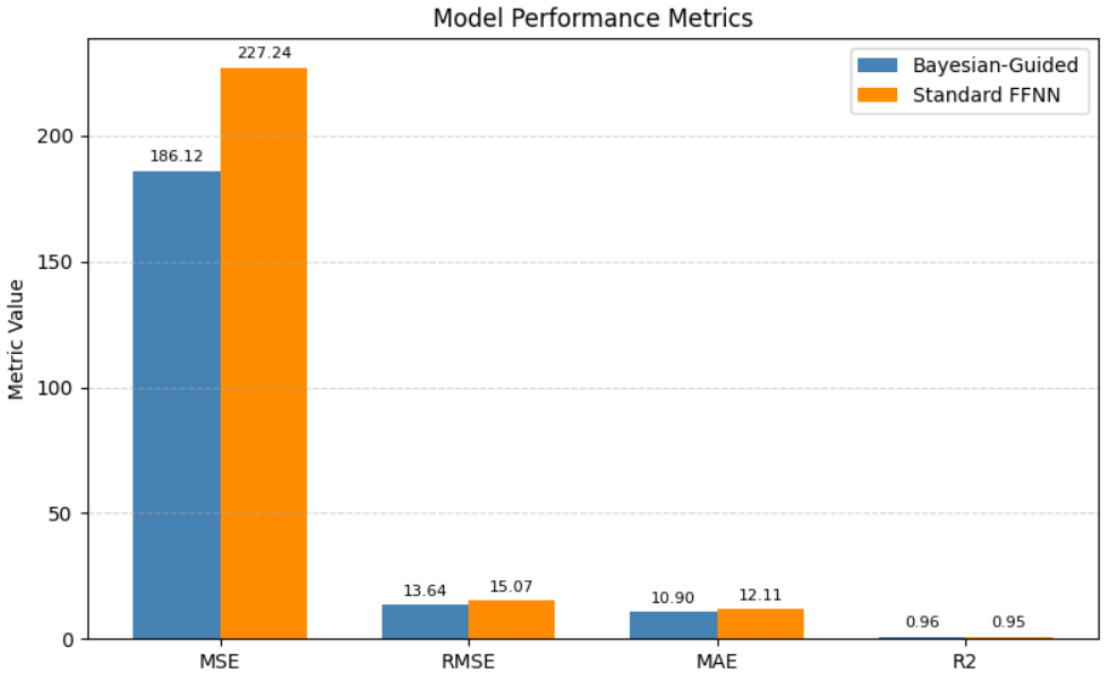

Fig. 9. Key measures to compare models' performance MSE, RMSE, MAE, R^2

Future work will focus on the refinement of the Model Architecture, particularly by exploring the architecture that combines insights from Bayesian Networks with Mutual Information (MI) scores to improve the overall predictive performance in Kafka cluster latency forecasting.

The current Bayesian-Guided FFNN leverages the influence of the Bayesian Network to establish initial weights, encoding domain-specific dependencies within the model. While effective, further improvements can be achieved by integrating Mutual Information (MI) scores to quantify pairwise feature dependencies, providing an additional signal for optimizing feature relationships. This hybrid approach combines the probabilistic reasoning of Bayesian Networks with the statistical dependency analysis of MI, resulting in richer representations of the underlying structure of Kafka workload and configuration data.

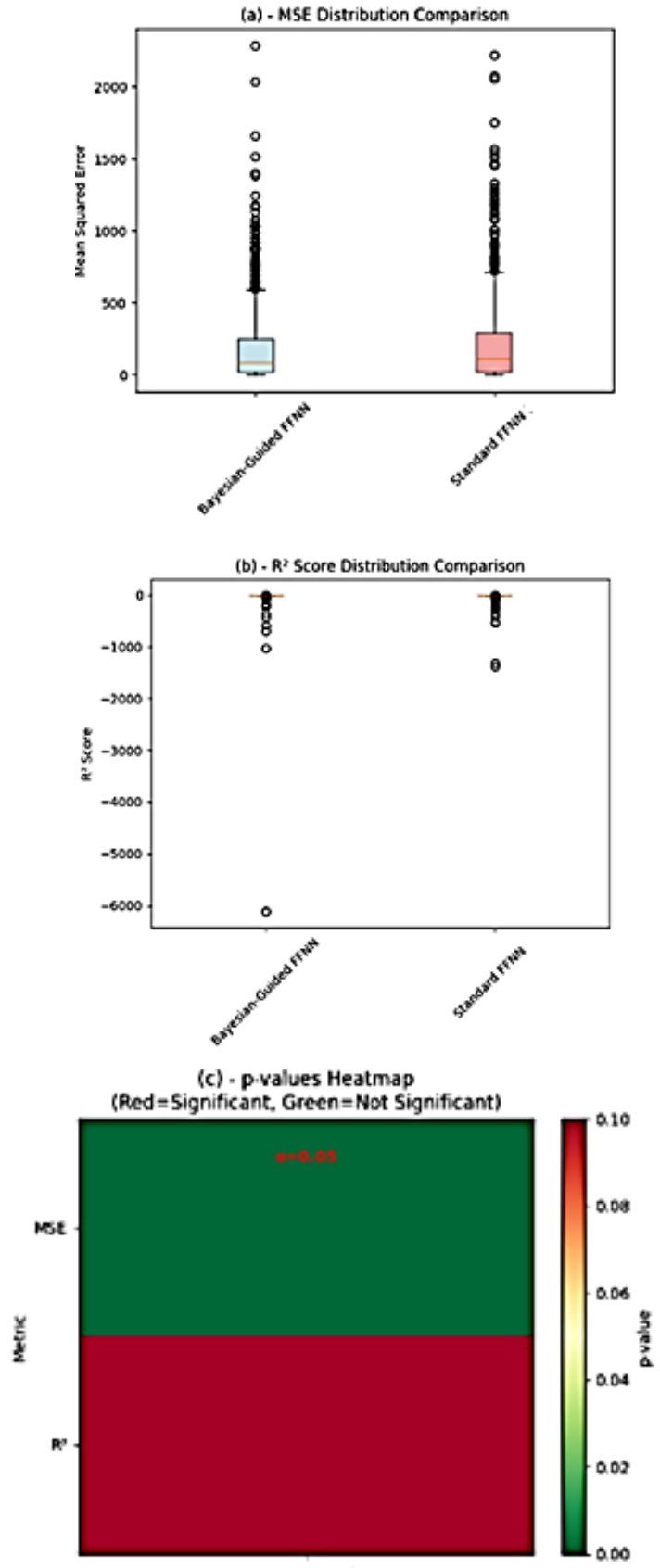

Fig. 10. Statistical Comparison Bayesian-Guided vs Standard FNN:
(a) – MSE distributions comparison; (b) – R^2 distributions comparison;
(c) – p-values of statistical significance testing for differences in MSE and R^2

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Contributions of authors: Olga Solovei - conceptualization, methodology**;** formulation of tasks, analysis**;** development of model, software, verification**;** analysis of results, visualization**;** writing original draft preparation. **Tetiana Honcharenko -** supervision, conceptualization, review and editing**.**

Conflict of Interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, author ship or otherwise, that could affect the research and its results presented in this paper.

Financing

This study was conducted without financial support.

Data Availability

Data will be made available upon reasonable request.

Use of Artificial Intelligence

The authors confirm that they did not use artificial intelligence methods while creating the presented work.

All the authors have read and agreed to the published version of this manuscript.

References

1. Honcharenko, T., Khrolenko, V., Gorbatyuk, I., Liashchenko, M., Bodnar, N., & Sherif, N. H. Smart Integration of Information Technologies for City Digital Twins. *In 2024 35th Conference of Open Innovations Association (FRUCT)*, IEEE, 2024, pp. 253-258. DOI: 10.23919/FRUCT61870.2024.10516358.

2. Raptis, T. P., & Passarella, A. A survey on networked data streaming with apache kafka. *IEEE Access*, 2023, vol. 11, pp. 85333-85350. DOI: 10.1109/ACCESS.2023.3303810.

3. Solovei, O., Honcharenko, T., & Fesan, A. Tekhnolohiyi upravlinnya velykymy danymy proyektiv mis′koho budivnytstva [Technologies to manager big data of urban building projects]. *Upravlinnya rozvytkom skladnykh system – Management of Development of Complex Systems*, 2024, no. 60, pp. 121–128, DOI: 10.32347/2412-9933.2024.60.121-128. (In Ukrainian).

4. Vogel, A., Henning, S., Ertl, O., & Rabiser, R. A systematic mapping of performance in distributed stream processing systems. *In 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 2023, pp. 293-300. DOI: 10.1109/SEAA60479.2023.00052.

5. Metta, C., Fantozzi, M., Papini, A., Amato, G., Bergamaschi, M., Galfrè, S. G., Marchetti, A., Veglio, M., Parton, M., & Morandin, F. Increasing biases can be more efficient than increasing weights. *In Proceedings of the 2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024, pp. 2810-2819. DOI: 10.1109/WACV57701.2024.00279.

6. Hosamo, H. H., Nielsen, H. K., Kraniotis, D., Svennevig, P. R., & Svidt, K. Improving building occupant comfort through a digital twin approach: A Bayesian network model and predictive maintenance method. *Energy and Buildings*, 2023, vol. 288, article no. 112992. DOI: 10.1016/j.enbuild.2023.112992.

7. Bortolini, R., & Forcada, N. A probabilistic performance evaluation for buildings and constructed assets. *Building Research & Information*, 2020, vol. 48, iss. 8, pp. 838-855. DOI: 10.1080/09613218.2019.1704208.

8. Mousavi, M., Shen, X., Zhang, Z., Barati, K., & Li, B. IoT-Bayes fusion: Advancing real-time environmental safety risk monitoring in under-ground mining and construction. *Reliability Engineering & System Safety*, 2025, vol. 256, article no. 110760. DOI: 10.1016/j.ress.2024.110760.

9. *Kafka Producer Configuration Reference for Confluent Platform*. Available at: https://docs.confluent.io/platform/current/installation/configuration/producer-configs.html. (accessed 12.01.2025).

10. Pacella, M., Papa, A., Papadia, G., & Fedeli, E. A Scalable Framework for Sensor Data Ingestion and Real-Time Processing in Cloud Manufacturing. *Algorithms*, 2025, vol. 18, iss. 1, article no. 22. DOI: 10.3390/a18010022.

11. Elshoubary, E. E., & Radwan, T. Studying the Efficiency of the Apache Kafka System Using the Reduction Method, and Its Effectiveness in Terms of Reliability Metrics Subject to a Copula Approach. *Applied Sciences*, 2024, vol. 14, iss. 15, article no. 6758. DOI: 10.3390/app14156758.

12. Sathupadi, K., Achar, S., Bhaskaran, S. V., Faruqui, N., & Uddin, J. BankNet: Real-Time Big Data Analytics for Secure Internet Banking. *Big Data and Cognitive Computing*, 2025, vol. 9, iss. 2, article no. 24. DOI: 10.3390/bdcc9020024.

13. Ezzeddine, M., Baude, F., Huet, F., & Laaziz, F. Latency Aware and Resource-Efficient Bin Pack Autoscaling for Distributed Event Queues: Parameters Impact and Setting. *SN Computer Science*, 2025, vol. 6, article no. 219. DOI: 10.1007/s42979-025-03740-9.

14. Harle, S. M. Advancements and challenges in the application of artificial intelligence in civil engineering: a comprehensive review. *Asian Journal of Civil Engineering*, 2024, vol. 25, iss. 1, pp.1061-1078. DOI: 10.1007/s42107-023-00760-9.

15. Moller, M. Efficient training of feed-forward neural networks. *DAIMI Report Series*, 1993, no. 464, article no. PB-464. pp. 136-173. DOI: 10.7146/dpb.v22i464.6937.

16. Narkhede, M. V., Bartakke, P. P., & Sutaone, M. S. A review on weight initialization strategies for neural networks. *Artificial intelligence review*, 2022, vol. 55, pp. 291-322. DOI: 10.1007/s10462-021-10033-z.

17. Ebid, S. E., El-Tantawy, S., Shawky, D., & Abdel-Malek, H. L. Correlation-based pruning algorithm

with weight compensation for feedforward neural networks. *Neural Computing and Applications*, 2025, vol. 37, pp. 6351-6367. DOI: 10.1007/s00521-024-10932-6.

18. Kitson, N. K., Constantinou, A. C., Guo, Z., Liu, Y., & Chobtham, K. A survey of Bayesian Network structure learning. *Artificial Intelligence Review*, 2023, vol. 56, pp. 8721-8814. DOI: 10.1007/s10462-022-10351-w.

19. Lu, N. Y., Zhang, K., & Yuan, C. Improving causal discovery by optimal bayesian network learning. *Proceedings of the AAAI Conference on artificial intelligence*, 2021, vol. 35, iss. 10, pp. 8741-8748. DOI: 10.1609/aaai.v35i10.17059.

20. Tawakuli, A., & Engel, T. Make your data fair: A survey of data preprocessing techniques that address biases in data towards fair AI. *Journal of Engineering Research*, 2024. DOI: 10.1016/j.jer.2024.06.016.

21. Kharchenko, V., Fesenko, H., & Illiashenko, O. Quality models for artificial intelligence systems: characteristic-based approach, development and application. *Sensors*, 2022, vol. 22, iss. 13, article no. 4865. DOI: 10.3390/s22134865.

Received 15.01.2025, Accepted 25.08.2025

МОДЕЛЬ НЕЙРОННОЇ МЕРЕЖІ ПРЯМОГО ПОШИРЕННЯ ДЛЯ ПРОГНОЗУВАННЯ ЗАТРИМКИ KAFKA КЛАСТЕРА

О. Л. Соловей, Т. А. Гончаренко

Предметом вивчення в статті є процес проектування архітектури моделі нейронної мережі прямого поширення (FFNN) на основі дискретної байєсівської мережі та методи визначення початкових ваг, які з'єднують нейрони між шарами нейронної мережі. **Метою** є розробка моделі нейронної мережі типу FFNN, призначеної для прогнозування наскрізної затримки в Kafka кластері. Запропонована модель буде використовуватися як інструмент для прогнозування затримки кластера Kafka на основі заданих параметрів конфігурації та показників продуктивності. Для досягнення мети в дослідженні вирішені **завдання**: розроблено та перевірено дискретну байєсовську мережу для розуміння факторів, що впливають на затримку в кластері Kafka; проведено аналіз чутливості дискретної байєсівської мережі на основі чого створив матрицю з початковими вагами, для початкової ініціалізації вагових коефіцієнтів FFNN моделі; розроблено архітектуру FFNN моделі для прогнозування затримки Kafka кластера та визначені її параметри; виконано навчання розробленої FFNN моделі і проведена оцінка здатності моделі прогнозувати потенційну затримку Kafka кластера. Для проведення дослідження були використані **методи з теорій**: обробки великих даних; імовірнісні графічні моделі та байєсовська теорія логічного висновку; штучні нейронні мережі та теорії глибокого навчання; теорія графів; оптимізація машинного навчання. Отримані такі **результати.** Модель FFNN була протестована, і значення середньої квадратичної помилки показали послідовне зниження протягом епох. Також були отримані результати які демонструють, що коефіцієнт масштабування Kaiming He покращує початкову фазу тренування, стабілізуючи ініціалізацію ваг. **Висновки**. Наукова новизна отриманих результатів полягає в наступному: 1) запропоновано нову методологію визначення архітектури нейронної мережі типу FFNN на основі дискретної структури байєсівської мережі; 2) розроблено новий метод встановлення початкових ваг, які з'єднують нейрони між шарами. Оскільки отримані значення середньої квадратичної помилки показали послідовне зниження протягом епох, ми дійшли висновку, що розроблена модель FFNN може бути розгорнута та використана як інструмент для прогнозування затримки Apach Kafka кластера. Запропонований в даній роботі метод визначення початкових ваг для FFNN є корисним для оптимізації процесу тренування моделі типу FFNN.

Ключові слова: затримка Kafka кластера; мережа Байєса; нейронна мережа прямого поширення; сила впливу; початкові ваги.

Соловей Ольга Леонідівна – докторантка каф. інформаційних технологій, канд. техн. наук, доц. каф. інформаційних технологій проектування та прикладної математики, Київський національний університет будівництва та архітектури, Київ, Україна.

Гончаренко Тетяна Андріївна – д-р техн. наук, доц., зав. каф. інформаційних технологій, Київський національний університет будівництва та архітектури, Київ, Україна.

Olha Solovei – Doctoral Student of the Department of Information Technologies, Associate Professor at the Department of Information Technologies of Design and Applied Mathematics, Kyiv National University of Construction and Architecture, Kyiv, Ukraine,
e-mail: solovey.ol@knuba.edu.ua, ORCID: 0000-0001-8774-7243, Scopus Author ID: 58173727100.

Tetiana Honcharenko – Doctor of Technical Science, Associated Professor, Head of the Department of Information Technologies, Kyiv National University of Construction and Architecture, Kyiv, Ukraine,
e-mail: goncharenko.ta@knuba.edu.ua, ORCID: 0000-0003-2577-6916, Scopus Author ID: 57204204504.