Reliability and Performance Evaluation of Two-input Machine Learning Systems

Kazuya Wakigami
Department of Computer Science
University of Tsukuba
Tsukuba, Japan
wakigami.kazuya@sd.cs.tsukuba.ac.jp

Fumio Machida
Department of Computer Science
University of Tsukuba
Tsukuba, Japan
machida@cs.tsukuba.ac.jp

Tuan Phung-Duc
Department of Policy and Planning Sciences
University of Tsukuba
Tsukuba, Japan
tuan@sk.tsukuba.ac.jp

***Abstract*—The multiple-input machine learning system (MLS) is a system architecture exploiting data diversity to improve the output reliability of the system by comparing prediction results on multiple input data. While the output reliability is enhanced by redundancy, the architecture imposes additional costs and non-negligible processing overheads. The performance of multiple-input MLSs has been theoretically investigated in the previous study using queueing analysis. However, it is little known how real MLSs are impacted by the multiple predictions and comparison processes needed in the architecture. In this paper, we implement two-input MLSs in two different configurations, a parallel type architecture and a shared type architecture, and evaluate the reliability, performance, and energy consumption of the system by experiments. Our empirical results unveil several advantages of the shared type architecture that can suppress the increases in response time and energy consumption by using a shared machine learning module for predictions of two inputs. We also compare the results of the performance simulation of two-input MLS with the empirical results. While we confirm the effectiveness of the simulation, we also find some gaps in the real observations. For example, we observe that the inference time distribution fits well in the log-normal distribution rather than the exponential distribution assumed in the simulation. Our findings could be useful for developing performance models for multiple-input MLSs.**

***Index Terms*—Energy consumption, Machine learning system, Performance, Reliability, Simulation**

I. Introduction

In recent years, systems and services using machine learning (ML) models have been widely used. Applications of MLs are expanding in the fields requiring safety and high reliability, such as medical image diagnosis and autonomous vehicles. In such application fields, prediction errors may cause serious problems, and hence, improving the reliability of MLSs is left as a fundamental challenge. In terms of the reliability of MLSs, existing studies focus on improving the robustness of ML models by generating adversarial samples that induce prediction errors [1], and verifying the safety of neural networks subject to adversarial examples [2]. These studies mainly consider the performance and robustness of a single ML model rather than an MLS. Meanwhile, the output reliability of MLSs can also be enhanced by introducing multiple ML models.

A redundancy architecture known as the N-version MLS can improve the system output reliability by diversifying prediction results from ML modules [3]. By using multiple models and/or inputs, the output of the system can be more reliable than relying on inference by a single ML module. Since it is necessary to execute multiple inferences and compare the results, it is concerned that response time and throughput performance can decrease in real systems. The performance of two-input MLSs has been theoretically investigated in the previous study using queueing analysis [4]. However, the existing studies have not verified the performance characteristics of two-input MLSs with real MLSs. Since queueing analysis is based on theoretical assumptions, such as exponentially distributed service time and the Poisson job arrival, it is essential to validate the model's assumptions with real data.

In this paper, we implement two-input MLSs and empirically investigate the performance characteristics of real MLSs using diversified inputs. We consider image classification as an ML task and measure the response time from data input to the final output, the amount of processing per unit time (throughput), and the mean energy consumption of the system during the experiments. We conduct the same experiments for two types of two-input MLSs, which are the parallel type architecture and the shared type architecture. For the input data distribution, the Poisson distribution and the constant interval inputs are considered. Our performance experiment results show that performance characteristics of empirical results are generally similar to the results obtained by the queueing model analysis. However, we also find some gaps between the empirical and theoretical results. For example, the inference time of the ML model fits well with the log-normal distribution instead of the exponential distribution assumed in the queueing model. In addition to the performance measurement experiment, we tuned the simulation programs developed in the other study [5] to adapt our experimental configuration to simulate the MLS performance and compare the simulation with the empirical results. As a result, we observe that the response time of the empirical results is shorter than the simulation result in the parallel type architecture, while it is longer than the simulation results in the shared type architecture. There are several reasons for the difference, but in common, the shared type architecture MLS has a shorter average response time than the parallel type.

We make the following contributions in this paper.

- The performance characteristics of two-input MLSs are empirically investigated. The parallel and shared type architectures are compared in terms of reliability, throughput, response time, and energy consumption.
- We show that the observed performance characteristics of two-input MLSs are generally similar to the theoretical results based on queueing analysis. However, our experimental results reveal that the ML inference time distribution better fits the log-normal distribution rather than the exponential one.
- By comparing the simulation results with the empirical results, we demonstrate the usefulness of the performance simulation for analyzing the performance of MLSs.

The rest of the paper is organized as follows. In Section II, we discuss the related work. In Section III, we describe the two-input MLSs and introduce the parallel type architecture and shared type architecture. Section IV explains the experimental system. Section V presents the details of the experiment procedure. Section VI describes experimental results. Section VII compares simulation results with empirical results. Finally, Section VIII gives our conclusion.

II. Related work

Regarding the performance of multiple-input MLSs, theoretical studies have been conducted using queueing models. The throughput of the parallel and the shared type architectures has been evaluated by modeling with queueing theory [4]. When the service rate is sufficiently high, it is shown that the parallel type architecture has higher throughput than the shared type architecture. A similar evaluation using a simulation program was presented in [5]. The simulation results showed that the shared type architecture had advantages in response time and energy consumption. While these theoretical studies rely solely on numerical analysis or simulation, the performance characteristics of a real MLS have not been empirically investigated. There is also a study regarding the modeling of MLS aiming to optimize services on serverless platforms using a Bayesian optimizer [8]. In this study, the model leverages Markovian Arrival Processes (MAPs), while the service time is assumed to follow the exponential distribution.

Recently, some empirical studies of multi-input MLSs use real datasets to evaluate the system output reliability. The reliability of a 3-version image classification model architecture was investigated in [6]. The impact of data diversification using image transformations was studied in [7]. The study presented the Neuron Coverage Improvement Rate (NCIR) to explore efficient combinations of diversified data that improve the system's reliability. However, these empirical studies did not consider the performance overhead imposed by multiple predictions and their comparisons.

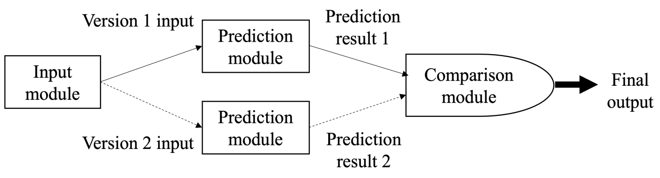
Energy efficiency is another important performance aspect of MLSs. In the previous study [9], the latency and energy consumption of the object detection model are evaluated. In addition, a method has been proposed to reduce energy consumption by enhancing the post-processing of data during object detection [10]. Another study focused on image classification tasks and comprehensively analyzed and considered accuracy, inference time, and energy consumption [11]. However, these works did not consider the reliability of MLS outputs, which is associated with MLS architectures. In this paper, we implement two-input MLSs with the parallel type and shared type architectures using small computers and compare their performance empirically by throughput, response time, and energy consumption to argue their trade-offs.

III. Two-input MLSs

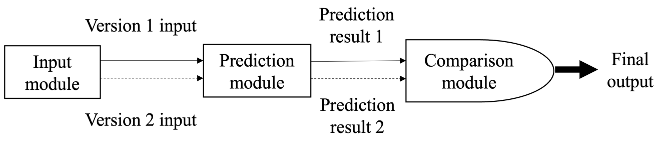
In this section, we explain two-input MLSs with parallel and shared type architectures for performance evaluation. Both types of MLSs use two input data and compare different inference results from the two data. For example, we perform image classification on images taken by two different cameras and compare the results. These two input images are not identical but similar. Therefore, the inference results are expected to be the same. If the inference results are not matched, the system can find that at least one of the results is wrong, and hence, an incorrect system output can be suppressed. In a real MLS, an input data stream is generated by a camera or a sensor. In our experiments, a module that generates the data for inference is called an Input module. The Input module generates two versions of input data to introduce the data diversity, which are called version 1 input and version 2 inputs, respectively. Although these two versions of input data are not identical, both data represent the same target. The version 1 and version 2 input are generated independently and sent to modules for ML predictions. A module that deploys an ML model for inference on the input data is called a Prediction module. The Parallel type architecture MLS uses two Prediction modules in parallel, while the shared type architecture MLS uses one Prediction module.

As shown in Fig. 1, in the parallel type architecture in Fig. 1(a), version 1 and version 2 input are sent to different Prediction modules, whereas in the shared type architecture in Fig. 1(b), two input data are sent to the same Prediction module. All the inference results are sent to the Comparison module that decides the final output of the system. The Comparison module compares the corresponding inference results for version 1 and version 2 input. If the results are matched (i.e., predicted labels are identical), the Comparison module outputs the results as the final output. Otherwise, the module discards the results. In the experiment, we use an image classification task in which an inference result is considered correct if the result matches the label of the original image.

The performance of these architectures can be characterized by some system parameters. Table. I shows the parameters of two-input MLSs. Assume the job arrival rate of version 1 and version 2 input is λ_1 and λ_2 respectively, and the service rate of the Prediction module for version 1 and version 2 input is μ_1 and μ_2, respectively. Let μ be the service rate of the Comparison module. Let K be the maximum buffer size of the Prediction modules. These parameters are also



(a) Parallel type architecture



(b) Shared type architecture

Fig. 1. Architectures of two-input MLS

used in the queueing models [4]. The detailed behavior of job processing in the two architectures is explained in the following subsections.

TABLE I
PARAMETERS OF TWO-INPUT MLSS

Parameter	Description
λ_1	Arrival rate of version 1 input.
λ_2	Arrival rate of version 2 input.
μ_1	Service rate of the Prediction module for version 1 input.
μ_2	Service rate of the Prediction module for version 2 input.
μ	Service rate of the Comparison module.
K	Maximum size of the buffer in the Prediction modules.

A. Parallel type architecture

The parallel type architecture is shown in Fig. 1(a). When the input data arrives from the Input module to the Prediction module, if the corresponding Prediction module is idle, the module starts processing the data. If the Prediction module is processing the previous arrival data, the input data is stored in the buffer of the Prediction module. When the inference result is obtained after the completion of the processing of the Prediction modules, the module goes into a waiting state, and the state of the other Prediction module is checked. If the other Prediction module is also idle, each one sends the inference result to the Comparison module and checks the buffer. Otherwise, the Prediction module waits for the completion of the processing of the other Prediction module. If there is still input data in the buffer, the Prediction module starts processing the data at the head of the buffer. Otherwise, the module becomes idle until the next data arrives.

B. Shared type architecture

The shared type architecture is shown in Fig. 1(b). When the input data arrives from the Input module to the Prediction module, if the Prediction module is idle, the module starts processing the data. When the Prediction module completes the processing and results for both version 1 and version 2 input are obtained, the Prediction module sends the results to the Comparison module and checks the buffer to find the next data. If each result has not been obtained, the module enters the waiting state and checks the buffer to obtain the other version of input data. If the Prediction module is processing the previous arrival data, the input data is stored in the buffer of the Prediction module like the parallel type architecture. If the Prediction module is in a waiting state, the module checks its own buffer. If the version of data at the head of the buffer is different from the one just finished, the module starts processing the data. If the input data version is the same as the finished data, the checked data is dropped from the buffer, and the module checks the next data. When the buffer becomes empty, the module stays idle and waits for the next data arrival.

IV. IMPLEMENTATION

This section describes the experimental system developed to evaluate the performance of the two-input MLSs.

A. System configuration

We built the experiment system composed of four PINEs, which are widely used single-board computers. Each machine is configured as an Input module, a Prediction module, or a Comparison module. PINE is a single-board computer developed by PINE 64 with Quad-core ARM Cortex-A53. The specifications of the machine used in the experiment are as follows.

- CPU: Quad-core ARM Cortex-A53 Processor@1152Mhz
- RAM Memory: 2GB
- OS: Armbian 22.05.3 Focal

The four PINEs are connected to a switching hub via Gigabit Ethernet and used for deploying an Input module, two Prediction modules, and a Comparison module. PINEs communicate with each other using the TCP protocol. Fig. 2 shows our experimental system. The left-most machine is assigned as the Input module, the second one and the third one are the Prediction modules, and the right-most machine is worked as the Comparison module.

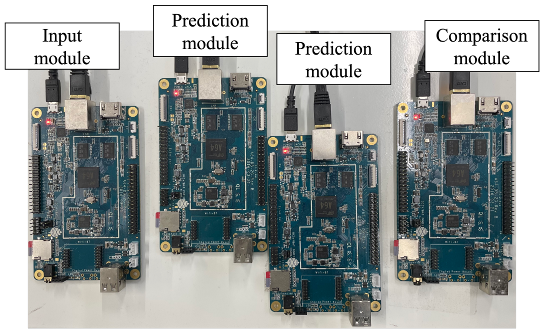


Fig. 2. The experimental system composed of four PINEs

B. ML model

We consider an image classification task for the MNIST dataset as an ML model. MNIST is a dataset of handwritten digit images [12]. A cloud-based development environment

Google colaboratory and the ML framework PyTorch are used to train an ML model. We used a convolutional neural network (CNN) trained with 60000 MNIST training data. For the training, we used ReLU (Rectified Linear Unit) as the activation function, Cross-entropy Loss as the loss function, and Adam as the optimization function. The accuracy of the trained model for 10000 MNIST test data is 95.9%.

C. Modules of the MLS

We implemented the Python programs that run on each module.

Input module. The Input module generates input data from the MNIST test data and sends the data to the Prediction module. The test data is divided into ten batch data by each correct label, and the module generates an input data stream for each batch data randomly. The same test data is reused if the size of the test data for each class is less than 10000. The two versions of data to be sent are the original MNIST data (version 1 input) and shifted data by one pixel to the upside (version 2 input). Any other kinds of small image perturbations can be used to generate version 2 input as studied in [7], but one-pixel shifted data is used in our study. We assume that the correct label for version 2 input is the same as the correct label for the corresponding version 1 input. The input data are sent in two interval patterns, following the Poisson distribution and sending the constant time intervals.

Prediction module. In the parallel type architecture system, the same ML model is deployed to two PINEs and used for the Prediction modules. On the other hand, in the shared type architecture system, one PINE is used for a Prediction module. We use PyTorch to run the ML model in the module. When the input data arrives at the module, if the corresponding module is in a processing or waiting state, the input data is stored in the buffer without processing the next inference.

Comparison module. We also assigned one PINE for the Comparison module and implemented a process to decide the final output of the system by comparing two inference results for version 1 and version 2 input in the order of arrival. When two results are the same, the Comparison module outputs the result. Otherwise, the module does not make any output.

V. Experiment procedure

This section describes the performance evaluation experiments using real two-input MLSs introduced in the previous section. First, we describe performance and reliability metrics of interest and then explain the performance measurement procedure.

A. Performance and reliability metrics

We use the following metrics to evaluate the reliability and performance of the output of the MLSs.

Comparison ratio. The number of comparison processes is counted by the number of data pairs compared in the Comparison module. We define the comparison ratio as the ratio of the number of comparison processes over the total number of data pairs sent from the Input module. The comparison ratio can deteriorate due to data loss during the transmission or drops at the buffer.

Correct output ratio. The number of outputs represents the number that the Comparison module produces output (i.e., the output is made when two inference results are the same). We define the number of correct outputs as the number of outputs that match the correct labels. The correct output ratio is computed by dividing the number of correct outputs by the number of outputs.

Response time for the strict condition and the loose condition. In two-input MLSs, there is a time lag between the times when each version of data in the pair is sent from the Input module. When we calculate the response time based on the data sent earlier, the response time includes the waiting time in the buffer. Thus, this kind of response time becomes a large value. The response time based on the data sent earlier is called the response time for the strict condition. On the other hand, the response time based on the data sent later is called the response time for the loose condition. The response time for the strict condition is always longer than the response time for the loose condition.

Energy consumption. We used a Bluetooth watt checker to measure energy consumption. We start measuring and recording the energy consumption when the experiment is started. We calculate the average energy consumption during the experiment from the recorded file.

B. Performance measurements

The data input interval is set to 0.1 seconds. In the case of data input interval following the Poisson distribution, data is sent with arrival rate $\lambda_1 = \lambda_2 = 10$. The maximum size of the buffer of the Prediction module is set to $K = 80$. When the new data arrives at the Prediction module, if the size of the buffer is larger than or equal to K, the data is discarded. We also measure the performance of a single version of the MLS for the purpose of comparison. In the single version system, the Comparison module simply outputs the inference results from the Prediction module. Each performance measurement is conducted five times, and we use the average values for the evaluation.

C. Service time measurements

We also measure the service time of the Prediction module in the MLSs. Service time indicates the time required for module processing (i.e., ML model inference). A single version MLS is used for measuring the service time of the Prediction module. We send data 500 times for each batch data corresponding to numbers 0 to 9, and measure the service times for individual samples. We also measure the service time of the Comparison module of the two-input MLSs. In the experiment, we sent data 10000 times to each of the parallel type architecture and the shared type architecture, and measured the time taken to obtain comparison results.

VI. Results

A. Reliability of output

Table. II shows the mean correct output ratio of the performance measurements mentioned in Section V-B in the parallel type architecture and the shared type architecture with the Poisson distribution data input and the constant interval input. The first row shows the case that the data input interval follows the Poisson distribution. The correct output ratio is 99.58 % for the parallel type architecture MLS, and 99.88 % for the shared type. Both values are higher than 96.78 %, which is the accuracy of the single version MLS. The results indicate that two-input MLS can improve the reliability of the output by exploiting data diversity, as expected from the theoretical results [3]. Comparing the two architectures, the shared type architecture has a slightly higher correct output ratio. The second row shows the case that the data input interval is constant. The correct output ratio is 99.73 % for the parallel type architecture, and 99.53 % for the shared type. Both values are higher than 96.76 %, which is the accuracy of the single version MLS. In the case of constant input data arrival, both two-version MLS architectures have higher values than the case of the single version system. These results indicate that the input data interval does not affect the correct output ratio.

TABLE II
Mean correct output ratio of the performance measurements

	Parallel type	**Shared type**	**1-ver.**
Poisson distribution	0.9958	0.9988	0.9678
Constant interval	0.9973	0.9953	0.9676

Next, Table. III shows the comparison ratio in the Comparison module for both input data intervals, following the Poisson distribution and arrival interval. In the case of the data input interval following the Poisson distribution, the comparison ratio of the shared type architecture is 66.39 %, and the remaining 33.61 % is discarded in module processing or lost during the communication. This result certainly agrees with the theoretical observation that the throughput decreases to 2/3 in the shared type architecture [4]. On the other hand, for the data input with constant intervals, the throughput of the shared type does not decrease extremely like in the case of the Poisson arrival. This is because two input data, version 1 and version 2 input, are alternately sent by constant interval, which reduces the possibility of burst arrival of the same input data. The comparison ratio of the parallel type is close to one but slightly lower than the single version.

TABLE III
Mean comparison ratio of the performance measurements

	Parallel type	**Shared type**	**1-ver.**
Poisson distribution	0.9943	0.6639	0.9997
Constant interval	0.9982	0.9996	0.9993

B. Response time

Table. IV shows the average response times for the two different input data arrival patterns. The response times are divided into two exclusive conditions: the strict condition and the loose condition.

TABLE IV
Mean response time of the performance measurements

(a) Both of the strict and the loose conditions [s]

	Parallel type	**Shared type**	**1-ver.**
Poisson distribution	1.7722	0.1176	0.0585
Constant interval	0.3101	0.0925	0.0437

(b) Strict condition [s]

	Parallel type	**Shared type**
Poisson distribution	3.4753	0.1679
Constant interval	0.5725	0.1328

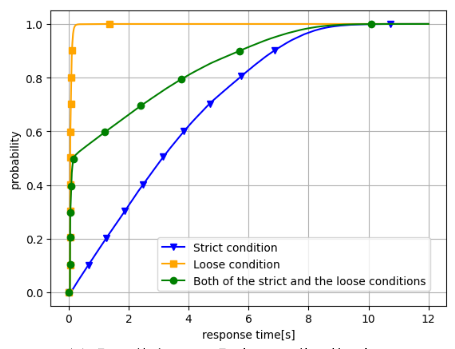
(c) Loose condition [s]

	Parallel type	**Shared type**
Poisson distribution	0.0690	0.0674
Constant interval	0.0478	0.0522

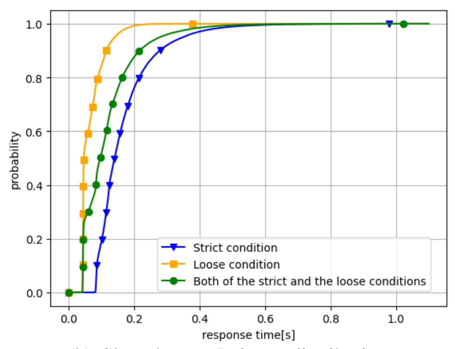
Table. IV(a) shows the mean response times in both strict and loose conditions. In the Poisson distribution case, the response times in both the parallel type and the shared type architecture MLSs are longer than that of the single version MLS. The response times of the parallel type architecture are significantly affected by the waiting time in the buffer due to the randomness of the data arrival. The response times in the constant interval case for both architectures of two-input MLSs are also longer than that of the single version MLS. We observe the response times in constant interval inputs are shorter than the response times in the Poisson distribution case. Similarly, the response time of the parallel type architecture is greatly affected by the waiting time in the buffer due to random arrival times. For the Poisson distribution case, the parallel type MLS takes 15.1 times longer than the shared type. In the case of constant intervals, the parallel type takes 3.35 times longer than the shared type.

Table. IV(b) shows the average response time for the strict condition (i.e., the response time includes the waiting time in the buffer of the Prediction module). For the Poisson distribution case, the response time for the parallel type architecture is 20.7 times longer than that for the shared type. Comparing the response times between the parallel and shared type architectures, the parallel type takes 4.31 times longer than the shared type. Since all the response times for the strict condition include the waiting time in the buffer of the Prediction module, the difference in the response times between the parallel type and the shared type becomes large compared with the response times for both conditions in Table. IV(a).

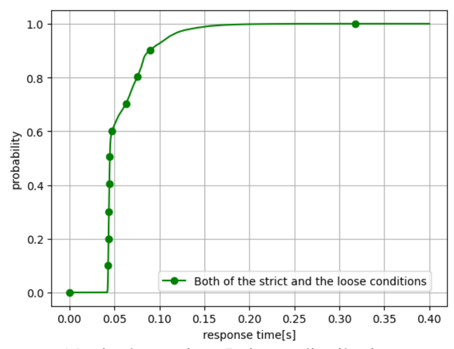
Table. IV(c) shows the average response time for the loose condition (i.e., the response time does not include the waiting time in the buffer). For the Poisson distribution case, the



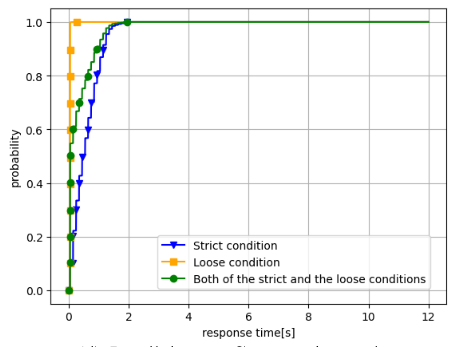
(a) Parallel type, Poisson distribution



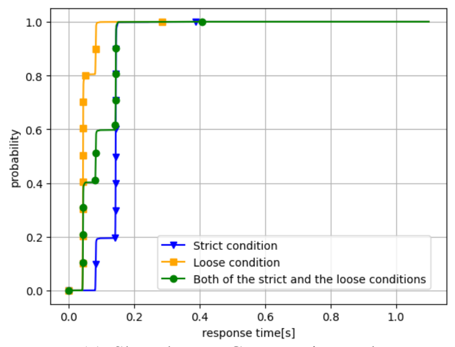
(b) Shared type, Poisson distribution



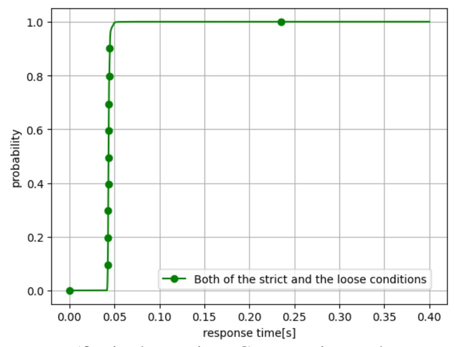
(c) single-version, Poisson distribution



(d) Parallel type, Constant interval



(e) Shared type, Constant interval



(f) single-version, Constant interval

Fig. 3. Response time distributions of the strict and the loose condition

response time of the parallel type architecture takes 1.02 times longer than that of the shared type. The response time for the parallel type architecture is 0.92 of that for the comparing shared type. In the response time for the loose conditions, the waiting time in the buffer is not included. Therefore, there is no large difference in the response time between the two architectures.

Fig. 3 shows the distributions of the response time for the strict, loose, and both conditions in different architectures. The horizontal axis indicates response time in seconds, while the vertical axis indicates the probability. Referring to Fig. 3(a) and Fig. 3(b), in the case of the Poisson distribution, the response time of both architectures is influenced by the response time of the loose condition. The probability is sharply increased at first and then slowly increased to 100 %. Also, referring to Fig. 3(d), Fig. 3(e), and Fig. 3(f), the response time increases step by step when data is sent at constant intervals.

C. Energy consumption

We measured the energy consumption every second. The average energy consumption during the experiment is 12.03 W for the parallel type architecture and 8.96 W for the shared type architecture. The energy consumption of the shared type architecture was 25.52 % smaller than the parallel type one. This is due to the difference in the number of machines used for each architecture. In the parallel type architecture case, the system is configured with four machines, while in the shared type case, the system uses only three machines.

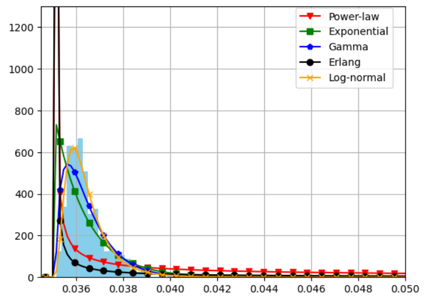
D. Service time of the Prediction module

Table. V shows the mean service time, standard deviation, and coefficient of variation (CV) obtained from the performance measurement experiments. For the input data interval following the Poisson distributions, the mean service time is 36.29 ms, the standard deviation is 1.46 ms, and the CV value is 0.0403. For the constant interval, the mean service time is 35.62 ms, the standard deviation is 1.26 ms, and the CV value is 0.0354. The service rate of the Prediction module is 27.56 for the Poisson distribution and 28.07 for the constant interval. Although the arrival process is assumed to be independent of the service time distribution in the queueing analysis, the service time in the real system is slightly affected by the arrival process (i.e., the processing rate of the computer likely slows down by the randomness of the arrival process).

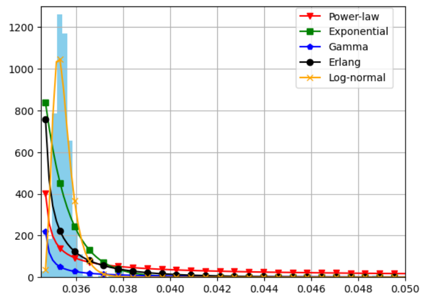
TABLE V
SERVICE TIME STATISTICS OF THE PREDICTION MODULE

	Poisson distribution	**Constant interval**
Mean [ms]	36.29	35.62
Standard deviation [ms]	1.46	1.26
CV	0.0403	0.0354
Service rate	27.56	28.07

Then, we use the Python Fitter library, which is provided for fitting probability distributions to observed data. The service time measurement results are fitted in five types of distributions: the log-normal distribution (lognorm), the



(a) Poisson distribution



(b) Constant interval

Fig. 4. Fitting results of service time of the Prediction module

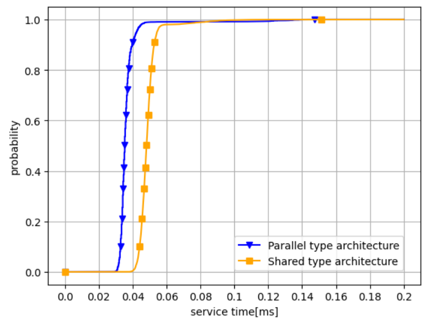


Fig. 5. Service time of the Comparison module

exponential distribution (expon), the power-law distribution (powerlaw), the Erlang distribution (erlang), and the gamma distribution (gamma). Table. VI shows the fitting parameters for the five distributions. Location parameters μ are almost the same, while the shape parameters and scale values σ differ in individual distributions. Table. VII shows the results of calculating the Sum of Squared Error (SSE). When the input data follows the Poisson distribution, the log-normal distribution achieves the best fit, followed by the gamma distribution and the exponential distribution. In the case of constant intervals, the order of the best-fitting results is the log-normal distribution, the exponential distribution, and the power-law distribution. Fig. 4 shows fitting results of the service time distribution of the Prediction module. The horizontal axis represents service time, and the vertical axis represents the number of data. From the results shown in Table. VI and Fig. 4, the log-normal distribution is considered the best-fitted distribution regardless of the type of input data arrivals. The results imply that the exponentially distributed service time of an ML module assumed in the queueing analysis [4], [5], [9] needs to be adjusted when considering a more accurate performance prediction.

TABLE VI
FITTING PARAMETERS OF SERVICE TIME OF THE PREDICTION MODULE

(a) Poisson distribution

	μ[s]	σ[s]	**Shape parameter**
Log-normal	0.0349	0.0013	$s = 0.578$
Exponential	0.0352	0.0013	-
Power-law	0.0352	0.0225	$a = 0.319$
Erlang	0.0352	0.8910	$k = 0.140$
Gamma	0.0351	0.0008	$k = 1.609$

(b) Constant interval

	μ[s]	σ[s]	**Shape parameter**
Log-normal	0.0344	0.0010	$s = 0.418$
Exponential	0.0345	0.0010	-
Power-law	0.0345	0.0220	$a = 0.304$
Erlang	0.0345	0.0033	$k = 0.313$
Gamma	0.0345	0.0188	$k = 0.047$

TABLE VII
SSE [10^5] OF FITTING RESULTS OF SERVICE TIME

	Poisson distribution	**Constant interval**
Log-normal	0.341	2.30
Exponential	5.96	25.4
Power-law	12.5	33.9
Erlang	15.0	37.1
Gamma	1.87	37.9

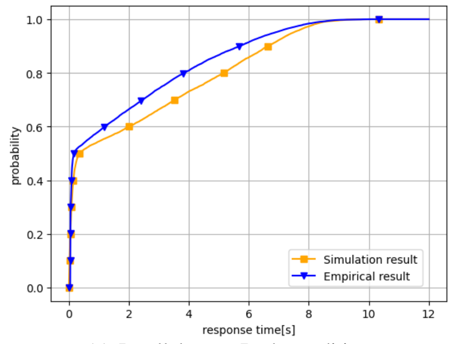
E. Service rate of the Comparison module

Out of 10000 input data, 9943 comparison results were obtained for the parallel type architecture MLSs, and 6612 comparison results were obtained for the shared type. We obtained the mean service time, the standard deviation, the minimum value, and the maximum value for each architecture. The service time is calculated from the mean processing time. Table. VIII shows the results of service time statistics spent in the Comparison module. Both service rates of the parallel type and the shared type architecture MLSs are sufficiently large, and both standard deviations are sufficiently small. The service rate of the parallel type is larger than that of the shared type. In the case of the parallel type, the Comparison module receives two inference results at the same time, while in the case of the shared type, the module receives an inference result each time the Prediction module obtains the result. Thus, the number of processing steps is increased, and the service rate becomes slightly decreased in the shared type case.

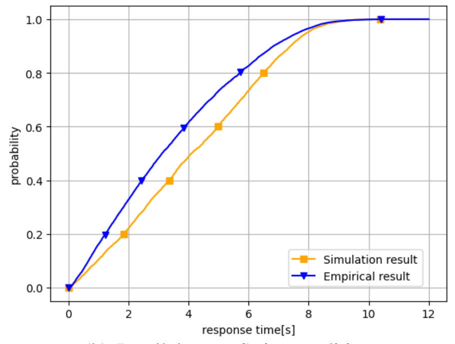
In addition, 5000 data are sampled from comparison results for each architecture, and the distribution of the service time is investigated. Fig. 5 shows the service time distribution of the Comparison module. The horizontal axis represents service time, and the vertical axis represents the probability. As observed in the standard deviation in Table. VIII and Fig. 5, the service time of the Comparison module is most likely constant.

VII. COMPARISON WITH SIMULATION RESULTS

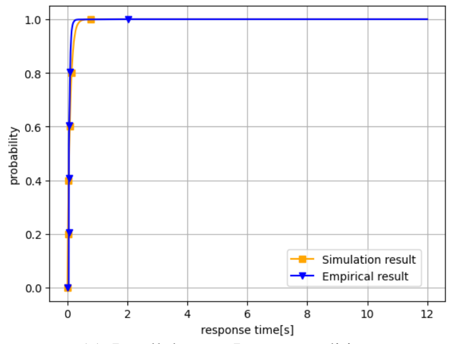
In this section, we compare the response time measured in the real MLSs and the response time obtained by a simulation program. The simulation program is developed to evaluate the parallel and the shared type architecture MLSs using the queueing model [5]. The performance model used in the simulation is based on the assumption that the data



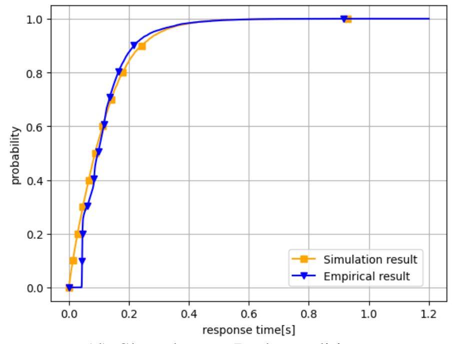
(a) Parallel type, Both condition



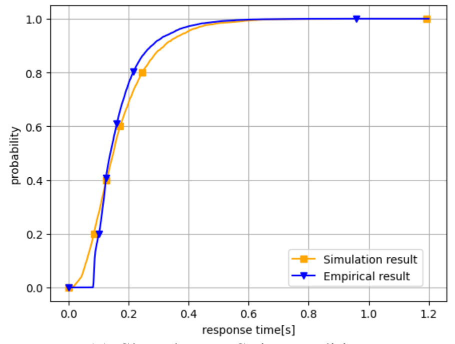
(b) Parallel type, Strict condition



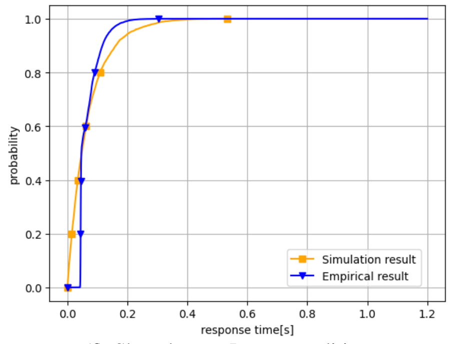
(c) Parallel type, Loose condition



(d) Shared type, Both conditions



(e) Shared type, Strict condition



(f) Shared type, Loose condition

Fig. 6. Response time distributions of the simulation and the empirical result

TABLE VIII
SERVICE TIME STATISTICS OF THE COMPARISON MODULE

	Parallel type	**Shared type**
Mean [ms]	0.0366	0.0489
Standard deviation [ms]	0.00871	0.00639
Minimum [ms]	0.0284	0.0358
Maximum [ms]	0.170	0.151
Service rate	2.73×10^4	2.04×10^4

TABLE IX
COMPARISON OF THE SIMULATION AND THE EMPIRICAL RESULT

(a) Parallel type architecture [s]

	Simulation	**Empirical**
Mean	2.061	1.772
Standard deviation	6.748	5.537
Minimum	0.0000468	0.0422
Maximum	10.500	11.160

(b) Shared type architecture [s]

	Simulation	**Empirical**
Mean	0.111	0.118
Standard deviation	0.0099	0.0072
Minimum	0.0000257	0.0411
Maximum	0.857	1.040

arrival follows the Poisson distribution and the service time is exponentially distributed.

A. *Configuration of simulation programs*

We configure the parameters of the simulation program to be consistent with the experiment system. For both the parallel type architecture and shared type architecture, the maximum queue size of each job is set to $K = 80$, which is the same as the maximum size of the buffer of the implemented MLSs. The arrival rates of jobs for version 1 and version 2 input are set as $\lambda_1 = \lambda_2 = 10$. The service rates of the Prediction module are set as $\mu_1 = \mu_2 = 27.56$ for the parallel type architecture and $\mu_1 = \mu_2 = 28.07$ for the shared type architecture. The service rate of the Comparison module is set as $\mu = 2.73 \times 10^4$ for the parallel type architecture and $\mu = 2.04 \times 10^4$ for the shared type architecture. We executed five simulations each for the parallel type architecture and the shared type architecture, and collected 10000 data samples from the simulation results, excluding the first 1000 data.

B. *Results*

The simulation results are summarized in Table. IX. The mean response time of empirical results of the parallel type architecture is 14.02% shorter than that of the simulation results. The standard deviation is 17.95% smaller than that of the simulation results. In the shared type architecture, the mean response time is 6.31% longer than that of the simulated results. The standard deviation is 27.27% smaller than that of the simulation results. The empirical minimum response times are longer than the simulated minimum response time for both architectures. This is due to the communication overhead in the deployed system, e.g., the communication delay has a lower bound that cannot be an extremely small value.

Fig. 6 shows the response time distributions of simulation and empirical results. For the parallel type architecture, in Fig. 6(a), Fig. 6(b), and Fig. 6(c), the empirical results are mostly higher probability than that of the simulation results. For the shared type architecture, in Fig. 6(d), Fig. 6(e), and Fig. 6(f), comparing the two response time distributions, the simulation result has a higher probability in the range of < 0.04 because the minimum value is limited due to the communication overhead when the input interval is very small in the real MLSs. However, the empirical values become a higher probability in other ranges, similar to the parallel type case.

VIII. Conclusion

In this paper, we conducted experiments to evaluate the reliability, performance, and energy consumption of the MLS in the parallel type and the shared type architectures. We confirmed that the reliability of MLS is enhanced by the two-input MLS in both architectures exploiting data diversity. For the throughput performance, we observed almost the same results as the theoretical results [5]. In the case of data input interval following the Poisson distribution, the throughput of the shared type architecture decreases to 2/3, consistent with the theoretical study [4]. In terms of response time, the parallel type architecture took 20.7 times longer than the shared type architecture. The difference in the response times between the shared type architecture MLS and the single version in the strict condition is 0.11 seconds, which is caused by the influence of the input data interval (i.e., the interval is set to 0.1 seconds in the experiments). Regarding energy consumption, the parallel type architecture consumes 1.34 times more energy than the shared type architecture. This is due to the difference in the number of machines used for the architectures.

Comparing the results of the simulation and the empirical results, the response time of the empirical result is shorter than that of the simulation results in the parallel type architecture. This is due to the difference in the distribution of the service time of the Prediction module. In the simulation program, the module service times of the Prediction module and the Comparison module are assumed to follow the exponential distributions. However, we find that the service time distribution fits better with the log-normal distribution. For more accurate simulation, it is encouraged to use the log-normal distribution in the simulation analysis as well. In the shared type architecture case, the mean response time of the simulation result is shorter than that of the empirical result.

In summary, we confirmed that the shared type architecture MLS has a lower energy consumption and a shorter response time than the parallel type architecture MLS. The shared type architecture is effective in improving reliability while suppressing increases in response time and energy consumption. However, since the shared type architecture reduces the throughput, the parallel type architecture is preferable in terms of reliability.

Acknowledgment

We would like to thank S. Nishio for his assistance with the simulation program. This work is supported in part by JSPS KAKENHI Grant Numbers 18K18006 and 22K17871.

References

[1] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," https://arxiv.org/abs/1412.6572, 2015.

[2] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," Proceedings of International Conference on Computer Aided Verification, pp.3-29, 2017.

[3] F. Machida, "N-version machine learning models for safety critical systems," Proceedings of DSN Workshop on Dependable and Secure Machine Learning, pp. 48-51, 2019.

[4] Y. Makino, T. Phung-Duc, and F. Machida, "A queueing analysis of malti-model multi-input machine learning systems," Proceedings of The 4th DSN Workshop on Dependable and Secure Machine Learning, 2021.

[5] S. Nishio, Y. Makino, T. Phung-Duc, and F. Machida, "Performance Analysis of Energy-Efficient Reliable Machine Learning System Architectures," http://dx.doi.org/10.2139/ssrn.4431918, 2023.

[6] F. Machida, "On the diversity of machine learning models for system reliability," Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 276-285, 2019.

[7] M. Takahashi, F. Machida, and Qiang Wen, "How data diversification benefits the reliability of three-version image classification systems," Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 34-42, 2022.

[8] A. Ali, R. Pinciroli, F. Yan, E. Smirni, "Optimizing inference serving on serverless latforms," Proceedings of the VLDB Endowment, Volume 15, Issue 10, pp. 2071–2084, 2022.

[9] J. Lee, P. Wang, R, Xu, V. Dasari, N. Weston, Y. Li, S. Bagchi, and S. Chaterji, "Virtuoso: Video-based intelligence for real-time tuning on SOCs," ACM Transactions on Design Automation of Electronic Systems, Association for Computing Machinery New York, NY, United States, 2022.

[10] J. Deng, Y. Pan, T. Yao, W. Zhou, H. Li, T. Mei, "Relation distillation networks for video object detection," Proceedings of IEEE/CVF International Conference on Computer Vision (ICCV), pp. 7022-7031, 2019.

[11] A. Canziani, A. Paszke, E. Culurciello, "An analysis of deep neural network models for practical applications," arXiv preprint arXiv: 1605.07678, 2016.

[12] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, Volume 86, Issue 11, pp. 2278-2324, 1998.