A Queueing Analysis of Multi-model Multi-input Machine Learning Systems

Yuta Makino
Department of Policy and Planning Sciences
University of Tsukuba
Ibaraki, Japan
s1920505@u.tsukuba.ac.jp

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

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

***Abstract*—A multi-model multi-input machine learning system (MLS) is an architectural approach to improve the reliability of the MLS output by using multiple models and multiple sensor inputs. While the errors in MLS output can be reduced by redundancy with diversity, the performance overhead/gain caused by the employed architecture may also be concerned in safety-critical applications such as a self-driving car. In this paper, we proposed queueing models for analyzing a multi-model multi-input MLS performance in two architectures, namely a parallel MLS and a shared MLS. The parallel MLS architecture runs two different machine learning models in parallel, while the shared MLS architecture runs a single machine learning model but uses two different sensor inputs. We model the behavior of the parallel MLS by a quasi-birth-death process. On the other hand, we model dynamics of the shared MLS as a continuous-time Markov chain of GI/M/1 type. The numerical experiments on the proposed models show that the parallel MLS generally achieves better throughput performance than the shared MLS under the same parameter settings. We also show that the throughput performance of the shared MLS can be improved when the input data arrival rates are sufficiently high.**

***Index Terms*—machine learning, throughput, performance, queueing model, redundant architecture**

I. Introduction

Dependability of machine learning systems (MLSs) is becoming a fundamental challenge in safety-critical systems such as self-driving cars and autonomous robots. Prediction accuracies of machine learning models have been considerably improved recently due to advances in machine learning algorithms and computing systems. However, outputs from machine learning models are still far from perfect in use cases because the input data encountered in the operation is not the same as the samples in the training data set. The robustness of machine learning models has been extensively studied for protecting the model from adversarial examples [1], [2], [6]. Numerous testing methods for machine learning models are presented recently [7]–[9]. Validating the input data during the operation can also help reduce errors causing corner cases [10]. Nevertheless, none of the existing methods solely provide a complete solution to remove MLS output errors. Since it is not realistic to assume that MLS never outputs errors, we still need diverse efforts to improve the reliability of MLS outputs.

A multi-model multi-input MLS considered in this paper is an architecture approach to reduce error outputs by employing multi-version machine learning models and multiple inputs to determine the final output. Similar architectures have been presented in recent studies [3], [12]. The reliability gain achieved by N-version machine learning architectures are studied analytically [3], [13] as well as experimentally [11], [12], [14]. Although the existing studies on N-version architecture mainly focus on reliability improvement, none of the work addresses the performance overhead that must be concerned with real-world applications. For example, the delay of image recognition outputs in a self-driving car may cause a catastrophic consequence. If the system employs the N-version architecture, the additional performance overhead caused by multi-model or multi-input should be taken into consideration.

In this paper, we propose queueing models for a multi-model multi-input MLS. We consider two types of architectures, namely the parallel MLS and the shared MLS. In the parallel MLS, machine learning models process the input data from multiple data sources in parallel and determine the final output by voting mechanism in a comparison unit. This architecture is efficient when input data frequencies are high, while it may consume a high amount of resource. On the other hand, the shared MLS only uses a single machine learning model which receives the data from different sources and determines the final output by comparing the prediction results for different sources. The method can improve the output reliability by leveraging the input data diversity [3], while the machine learning model can be the performance bottleneck when data arrival frequencies are high. Both the architectures have their advantages and disadvantages. The objective of our study is to quantitatively analyze the performance characteristics of two different architectures for N-version machine learning systems. To this end, we focus on simple two-version architectures in which the system can use at most two different machine learning models and two data sources. We model the parallel MLS as a quasi-birth-death process that can incorporate different data arrival rates, different processing rates, and a voting process. On the other hand, we show that the performance of the shared MLS can be analyzed using a continuous-time Markov chain of GI/M/1 type whose stationary distribution can be derived algorithmically.

The rest of the paper is organized as follows. Section

Ⅱ explains the queueing models for a parallel MLS and a shared MLS. Section Ⅲ details the analysis of the proposed models. Section Ⅳ introduces the performance measures. Section Ⅴ shows the numerical examples. Section Ⅵ discusses the reliabilities of the multi-model multi-input MLSs. Finally, Section Ⅶ concludes the paper.

II. Queueing Model

In this section, we describe in detail two queueing models for the parallel MLS and the shared MLS, respectively. In the following description, we refer to the input data from the primary sensor as type 1 jobs and those from the secondary sensor as type 2 jobs. Furthermore, a module refers to a software which deploys a machine learning model.

A. Parallel MLS

In this section, we describe a queueing model for the parallel MLS. Jobs of type 1 arrive at the system according to the Poisson process with rate λ_1. Upon arrival of a type 1 job, if module 1 is free, the job is processed immediately, otherwise it waits in the queue of module 1. Jobs of type 2 also arrive at module 2 according to the Poisson process with rate λ_2 and are processed with the same manner. The processing times of module 1 and module 2 follow the exponential distributions with mean $1/\mu_1$ and $1/\mu_2$, respectively. The special feature of our model is that upon service completion, the module stops processing in cases: 1) there is not a completed job in the other module, 2) the comparison unit is not idle yet. When the processing of two jobs in the two modules are completed and the comparison unit is idle, these two jobs are forwarded to the comparison unit. The comparison completes in an exponentially distributed time with mean $1/\mu$ and after that the comparison unit becomes free. The schematic of the model is shown in Figure 1. For the sake of the analysis, the maximum allowable amount of Type 2 jobs in the parallel MLS is set to $K > 0$, but it should be large enough to represent the actual system.

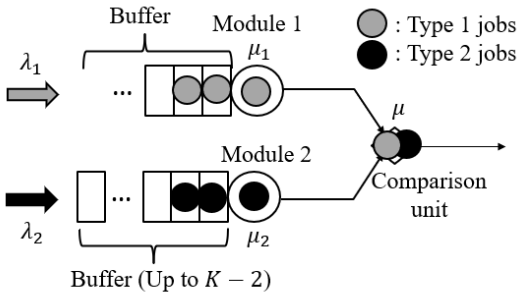

Fig. 1. The queueing model for the parallel MLS with two modules.

In order to analyze the above model, we set necessary assumptions. We assume that the arrival intervals of jobs and the processing times of modules and the comparison times are independent of each other. The order of services is assumed to be first-come, first-served. Next, we define the necessary random variables for the analysis of this model. We define $S_{P2} := \{0, 1, ..., K\}, S_{Pstate} := \{0, 1\}, \mathbb{N}_0 := \mathbb{N} \cup \{0\}, S_P^* := S_{P2} \times S_{Pstate} \times S_{Pstate} \times S_{Pstate} \times \mathbb{N}_0$. We define the number of jobs of type 2 in the system by $N_2(t) \in S_{P2}$, the states of module processing of type 1 and type 2 and comparison processing are $N_{m1}(t), N_{m2}(t), N_{c1}(t) \in S_{Pstate}$; and the number of jobs of type 1 in the system is $N(t) \in \mathbb{N}_0$. Note that for the state of module 1, 2 and the comparison unit, 0 means the module is free and 1 means the module is processing. Finally, we define $X_P(t) := (N(t), N_2(t), N_{m1}(t), N_{m2}(t), N_{c1}(t))$. Since S_P^* includes states that $X_P(t)$ cannot reach, we define S_P as the subset of S_P^* that unreachable states are removed. Based on the above settings, it is easy to see that $\{X_P(t); t \geq 0\}$ is a Markov chain in the state space S_P whose analysis will be given in Section Ⅲ.

B. Shared MLS

In this section, we describe a queueing model for shared MLS. Jobs of type 1 and 2 arrive at the system according to Poisson processes with rate λ_1 and λ_2, respectively and we do not distinguish the types of jobs. The arrival process of arbitrary jobs (either type 1 or type 2) follow Poisson process with rate $\lambda_1 + \lambda_2$. It should be noted that the type of the arriving job is not known and the type of the job is only probabilistically determined once it enters the service. Jobs are serviced on a first-come, first-served (FCFS) discipline. Service times of jobs of type 1 are exponentially distributed with rate μ_1, while those of type 2 are exponentially distributed with rate μ_2. Upon the service completion of a job, if there exists a served job of the other type, both jobs are transferred to the comparison unit in which an exponentially distributed time with mean $1/\mu$ is further needed. In case the comparison unit is not idle, the two jobs wait at the module and the module is stopped. Otherwise, the module looks for a non-processed job of the other type from the head of the buffer one by one. In this process, jobs of the same type will be deleted until a job of the other type is possibly found. Once two jobs of the two types are transferred to the comparison unit, the module pickups one job in the head of the buffer to process. This process is repeated. The probability that a job in the buffer is a type 1 job is $\frac{\lambda_1}{\lambda_1+\lambda_2} (:= \tilde{\lambda_1})$ and the probability that it is a type 2 job is $\frac{\lambda_2}{\lambda_1+\lambda_2} (:= \tilde{\lambda_2})$. A schematic of the model is shown in Figure 2.

In order to analyze the above model, we set necessary assumptions. We assume that the arrival intervals of various jobs and the processing times of the module and comparison processes are independent of each other. The order of services is assumed to be first-come, first-served. Next, we define the necessary random variables for the analysis of this model. We define $S_{state1} := \{0, 1, 2, 3, 4, 5, 6, 7\}, S_{state2} := \{0, 1\}, \mathbb{N}_0 := \mathbb{N} \cup \{0\}, S_S^* := S_{state1} \times S_{state2} \times \mathbb{N}_0$. The state of module processing in the system is $N_m(t) \in S_{state1}$, the state of comparison processing is $N_{c2}(t) \in S_{state2}$, and the number of jobs waiting in the buffer is $L(t) \in \mathbb{N}_0$. As

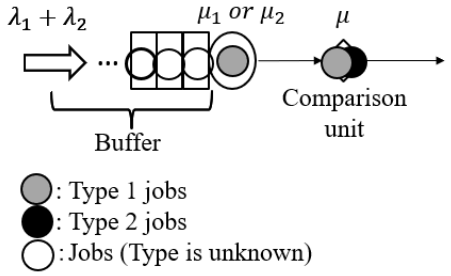

Fig. 2. The queueing model for the shared MLS with two types of jobs.

for the status of the module process, 0 means free state, 1 means that only type 1 jobs have been processed, 2 means that only type 1 job is in processing, 3 means that type 1 job has been processed and type 2 job is in processing, 4 is that only type 2 jobs have been processed, 5 means that only type 2 job is in processing, 6 means that the job of type 2 has been processed and the job of type 1 is in processing, and 7 means that both two types of jobs have been processed and waiting for comparison processing. As for the status of the comparison process, 0 means free and 1 means in process. Finally, we define $X_S(t) := (L(t), N_m(t), N_{c2}(t))$. Since S_S^* includes states that $X_S(t)$ cannot reach, we define S_S as the subset of S_S^* that unreachable states are removed. It is easy to see that $\{X_S(t); t \geq 0\}$ is a Markov chain on the state space S_S. Based on the above settings, we analyze the model in section Ⅲ.

III. Queueing Analysis

In this section, we define the infinitesimal generators for the two models described in section Ⅱ, and describe the analysis of each model.

A. Parallel MLS

When we construct the transition matrix by separating the change in the number of Type 1 jobs from the change in the other states, we can represent the infinitesimal generator Q_P (1), where O is a zero matrix of appropriate size.

$$Q_P = \begin{array}{c} \\ \mathcal{L}_0^P \\ \mathcal{L}_1^P \\ \mathcal{L}_2^P \\ \mathcal{L}_3^P \\ \mathcal{L}_4^P \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathcal{L}_0^P & \mathcal{L}_1^P & \mathcal{L}_2^P & \mathcal{L}_3^P & \mathcal{L}_4^P & \mathcal{L}_5^P & \cdots \end{array} \\ \begin{pmatrix} B_0 & C_0 & O & O & O & O & \cdots \\ A_1 & B_1 & C_1 & O & O & O & \cdots \\ O & A_2 & B_2 & C_2 & O & O & \cdots \\ O & O & A_3 & B_2 & C_2 & O & \cdots \\ O & O & O & A_3 & B_2 & C_2 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \end{array}. \quad (1)$$

$\mathcal{L}_0^P, \mathcal{L}_1^P, \mathcal{L}_i^P\ (i \geq 2)$ are the sets given as follows.

$$\begin{aligned}
\mathcal{L}_0^P :=& \{(0,0,0,0,0)\} \cup \cdots \cup \{(0,K,0,0,0)\} \cup \{(0,1,0,1,0)\} \\
& \cup \cdots \cup \{(0,K,0,1,0)\}, \\
\mathcal{L}_1^P :=& \{(1,0,0,0,0)\} \cup \{(1,0,1,0,0)\} \cup \cdots \cup \{(1,K,1,0,0)\} \cup \{(1,1,0,1,0)\} \\
& \cup \cdots \cup \{(1,K,0,1,0)\} \cup \{(1,1,0,0,1)\} \cup \cdots \cup \{(1,K,0,0,1)\} \\
& \cup \{(1,1,1,1,0)\} \cup \cdots \cup \{(1,K,1,1,0)\} \cup \{(1,2,0,1,1)\} \\
& \cup \cdots \cup \{(1,K,0,1,1)\}, \\
\mathcal{L}_i^P :=& \{(i,0,0,0,0)\} \cup \{(i,0,1,0,0)\} \cup \cdots \cup \{(i,K,1,0,0)\} \cup \{(i,1,0,1,0)\} \\
& \cup \cdots \cup \{(i,K,0,1,0)\} \cup \{(i,1,0,0,1)\} \cup \cdots \cup \{(i,K,0,0,1)\} \\
& \cup \{(i,1,1,1,0)\} \cup \cdots \cup \{(i,K,1,1,0)\} \cup \{(i,1,1,0,1)\} \cup \cdots \\
& \cup \{(i,K,1,0,1)\} \cup \{(i,2,0,1,1)\} \cup \cdots \cup \{(i,K,0,1,1)\} \cup \{(i,2,1,1,1)\} \\
& \cup \cdots \cup \{(i,K,1,1,1)\}.
\end{aligned}$$

In $\mathcal{L}_i^P$, i corresponds to the number of type 1 jobs in the system, the elements of $\mathcal{L}_i^P$ are sorted in the increasing order of $N_2(t)$ and the last three components are sorted in an appropriate order convenient for computation. Block matrix $B_i\,(i \geq 0)$ represents the state transition when the number of type 1 jobs does not change, the block matrix $C_i\,(i \geq 0)$ represents the state transition when the number of type 1 jobs increases by one, and the block matrix $A_i\,(i \geq 1)$ represents the state transition when the number of type 1 jobs decreases by one. For the elements of each matrix, please refer to the appendix.

Next, we compute the stationary distribution of (1). Because $\{X_P(t) \in S_P \,|\, t \geq 0\}$ defined in section Ⅱ is a continuous-time Markov chain, which can be seen as a quasi-birth-death process [4]. We calculate the stationary distribution by referring to the method shown in [4]. We define the stationary distribution $\pi_{i,j,k,m,l}^P$ of $X_P(t)$ for $(i,j,k,m,l) \in S_P$ as follows.

$$\begin{aligned}
\pi_{i,j,k,m,l}^P := \lim_{t\to\infty} P(&N(t) = i, N_2(t) = j, N_{m1}(t) = k \\
&N_{m2}(t) = m, N_{c1}(t) = l).
\end{aligned}$$

In addition, we define π_i^P as follows.

$$\pi_i^P := (\pi_{i,j,k,m,l}^P)_{(i,j,k,m,l)\in\mathcal{L}_i^P}.$$

π_i^P that represents the stationary distribution when the number of type 1 jobs in the system is fixed to i.

B. Shared MLS

When we construct the transition matrix by separating the change in the number of jobs in buffer from the change in the other states, we can represent the infinitesimal generator Q_S (2), where O is a zero matrix of appropriate size.

$$Q_S = \begin{array}{c} \\ \mathcal{L}_0^S \\ \mathcal{L}_1^S \\ \mathcal{L}_2^S \\ \mathcal{L}_3^S \\ \mathcal{L}_4^S \\ \mathcal{L}_5^S \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathcal{L}_0^S & \mathcal{L}_1^S & \mathcal{L}_2^S & \mathcal{L}_3^S & \mathcal{L}_4^S & \mathcal{L}_5^S & \cdots \end{array} \\ \begin{pmatrix} B_0 & C_0 & O & O & O & O & \cdots \\ B_1 & A_1 & A_0 & O & O & O & \cdots \\ B_2 & A_2 & A_1 & A_0 & O & O & \cdots \\ B_3 & A_3 & A_2 & A_1 & A_0 & O & \cdots \\ B_4 & A_4 & A_3 & A_2 & A_1 & A_0 & \cdots \\ B_5 & A_5 & A_4 & A_3 & A_2 & A_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \end{array}. \quad (2)$$

$\mathcal{L}_0^S, \mathcal{L}_i^S$ $(i \geq 1)$ are the sets given as follows.

$$\begin{aligned}\mathcal{L}_0^S :=& \{(0,0,0)\} \cup \cdots \cup \{(0,6,0)\} \cup \{(0,0,1)\} \cup \cdots \cup \{(0,7,1)\},\\ \mathcal{L}_i^S :=& \{(i,2,0)\} \cup \{(i,3,0)\} \cup \{(i,5,0)\} \cup \{(i,6,0)\} \cup \{(i,2,1)\} \cup \{(i,3,1)\}\\ & \cup \{(i,5,1)\} \cup \{(i,6,1)\} \cup \{(i,7,1)\}.\end{aligned}$$

In $\mathcal{L}_i^S$, i corresponds to the number of jobs in buffer. The block matrix B_0, A_1 represents the state transition when the number of jobs in buffer does not change, the block matrix C_0, A_0 represents the state transition when the number of jobs in buffer increases by one, the block matrix $A_i\,(i \geq 2)$ represents the state transition when the number of jobs in buffer decreases by $i-1$, and the block matrix $B_i\,(i \geq 1)$ represents the state transition when the number of jobs in buffer decreases by i. For the elements of each matrix, please refer to the appendix.

Next, we compute the stationary distribution of (2). Because $\{X_S(t) \in S_S \,|\, t \geq 0\}$ defined in section Ⅱ is a continuous-time Markov chain of GI/M/1 type, we calculate the stationary distribution by referring to the method shown in [5]. We define the stationary distribution $\pi_{i,j,k}^S$ of $X_S(t)$ for $(i,j,k) \in S_S$ as follows.

$$\pi_{i,j,k}^S = \lim_{t\to\infty} P(L(t) = i, N_m(t) = j, N_{c2}(t) = k).$$

In addition, we define π_i^S as follows.

$$\pi_i^S := (\pi_{i,j,k}^S)_{(i,j,k)\in\mathcal{L}_i^S}.$$

π_i^S that represents the stationary distribution when the number of jobs waiting in the buffer is fixed to i.

IV. Performance Measures

The throughput of parallel MLS T_P can be defined as follows.

$$T_P = \pi_1^P e_{P1}^{**} + \sum_{i=2}^{\infty} \pi_i^P e_{P2}^{**},$$

where e_{P1}^{**} is a column vector of size $5K+1$ with all elements in $2K+3 \sim 3K+2$ rows and $4K+3 \sim 5K+1$ rows being 1 and the other elements being 0, and e_{P2}^{**} is a column vector of size $7K$ and with all elements in $2K+3 \sim 3K+2$ and $4K+3 \sim 7K$ rows being 1 and the other elements being 0. For elements in e_{P1}^{**}, e_{P2}^{**}, 1 corresponds to the state that the throughput is calculated.

The throughput of shared MLS T_S can be defined as follows.

$$T_S = \pi_0^S e_{S0}^{*} + \sum_{i=1}^{\infty} \pi_i^S e_{S1}^{*},$$

where e_{S0}^{*}, e_{S1}^{*} are column vectors given by

$$\begin{aligned} e_{S0}^{*} =& \quad (0,0,0,0,0,0,0,1,1,1,1,1,1,1)^T,\\ e_{S1}^{*} =& \quad (0,0,0,0,1,1,1,1,1)^T. \end{aligned}$$

For elements in e_{S0}^{*}, e_{S1}^{*}, 1 corresponds to the state that the throughput is calculated.

V. Numerical Results

In this section, we present numerical results based on the analysis presented in Sections Ⅱ and Ⅲ.

A. Parallel MLS

In the numerical results, we perform Monte Carlo simulations in addition to the numerical calculations based on the analysis results in section Ⅲ to have a double check. First, we calculate T_P by varying the value of μ with $\mu_1 = \mu_2 = 5.0, K = 50$ while λ_1 is fixed to 0.9, 1.4, 1.9, 2.4 and λ_2 is fixed to 1.0, 1.5, 2.0, 2.5. The results are shown in Fig. 3.

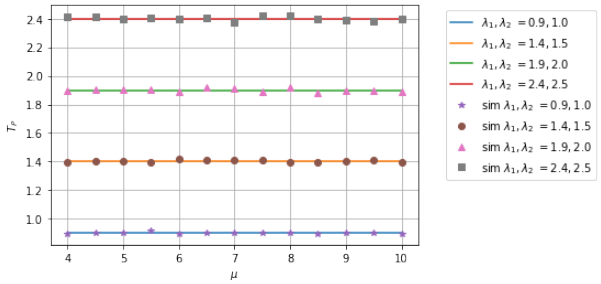

Fig. 3. Throughputs of the parallel MLS by varying the value of μ.

- The value of T_P increases as the arrival rates $\lambda_1 = \lambda_2$ increase simultaneously.
- The value of T_P is almost insensitive to the value of μ.
- The value of T_P is the same as the value of arrival rate for type 1 jobs λ_1.

Next, we calculate T_P by varying the value of $\mu_1 = \mu_2$ with $\mu = 7.0, K = 50$ while λ_1 is fixed to 0.9, 1.4, 1.9, 2.4 and λ_2 is fixed to 1.0, 1.5, 2.0, 2.5. The results are shown in Fig. 4.

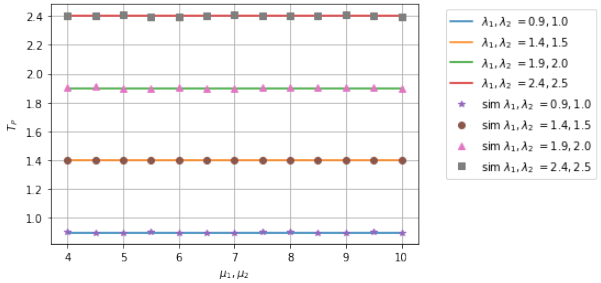

Fig. 4. Throughputs of the parallel MLS by varying the value of $\mu_1 = \mu_2$.

- The value of T_P increases as the arrival rates $\lambda_1 = \lambda_2$ increase simultaneously.
- The value of T_P is almost insensitive to the value of $\mu_1 = \mu_2$.
- The value of T_P is the same as the value of arrival rate for type 1 jobs λ_1.

In addition, we calculate T_P by varying the value of λ_1 with $\lambda_2 = 2.5, \mu_1 = \mu_2 = 5.0, \mu = 7.0, K = 50$. The results are shown in Fig. 5.

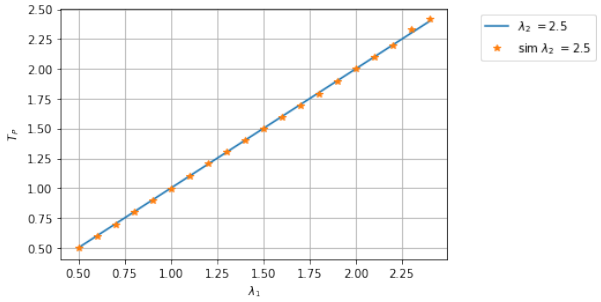

Fig. 5. Throughputs of the parallel MLS by varying the value of λ_1.

In Fig. 5, the value of T_P increases linearly when the arrival rate λ_1 increases. This observation suggests that the value of throughput is the same as the value of arrival rate for type 1 jobs λ_1. This is because type 1 jobs are not lost.

Furthermore, we calculate T_P by varying the value of λ_2 with $\lambda_1 = 0.9, \mu_1 = \mu_2 = 5.0, \mu = 7.0, K = 50$. The results are shown in Fig. 6.

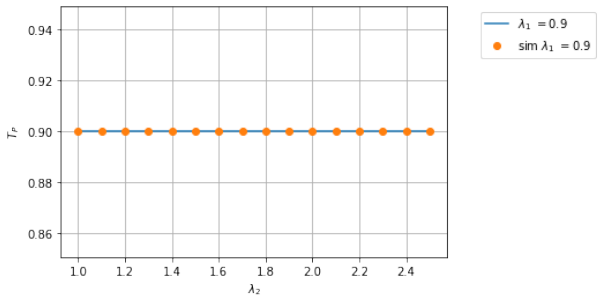

Fig. 6. Throughputs of the parallel MLS by varying the value of λ_2.

In Fig. 6, the value of T_P is almost insensitive to the value of λ_2.

These observations from Fig. 3 to Fig. 6 suggest that the value of T_P increases as the value of λ_1 increases.

B. Shared MLS

First, we calculate T_S by varying the value of μ with $\mu_1 = \mu_2 = 5.0$ while λ_1, λ_2 are fixed to 1.0, 1.5, 2.0, 2.5. The results are shown in Fig. 7.

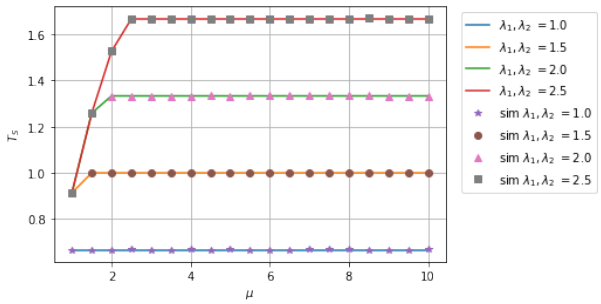

Fig. 7. Throughputs of the shared MLS by varying the value of μ.

- The value of T_S increases as the arrival rates $\lambda_1 = \lambda_2$ increase simultaneously.
- The value of T_S is almost insensitive to the value of μ with $\lambda_1 = \lambda_2 = 1.0$.
- The value of T_S increases as the value of μ increases to some extent when λ_1, λ_2 are fixed to 1.5, 2.0, 2.5.
- The value of T_S is almost insensitive to the value of μ from a certain value as λ_1, λ_2 are fixed to 1.5, 2.0, 2.5.
- The value of T_S is at most $\frac{2}{3}$ of the arrival rate $\lambda_1 = \lambda_2$ as the value of μ is increased.

These observations suggest that the value of T_S is sensitive to the value of μ and λ_1, λ_2 and is at most $\frac{2}{3}$ of the arrival rate $\lambda_1 = \lambda_2$.

Next, we calculate T_S by varying the value of $\mu_1 = \mu_2$ with $\mu = 7.0$ while λ_1, λ_2 are fixed to 1.0, 1.5, 2.0, 2.5. The results are shown in Fig. 8.

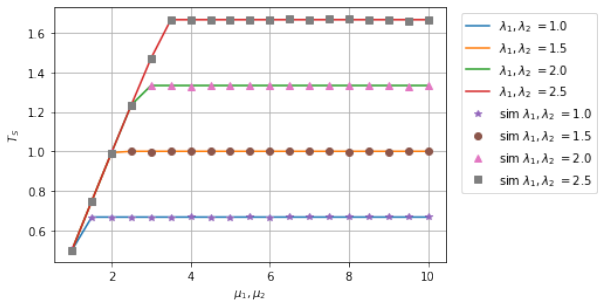

Fig. 8. Throughputs of the shared MLS by varying the value of $\mu_1 = \mu_2$.

- The value of T_S increases as the arrival rates $\lambda_1 = \lambda_2$ increase simultaneously.
- The value of T_S increases as the value of $\mu_1 = \mu_2$ increases to some extent.
- The value of T_S is almost insensitive to the value of μ from a certain value.
- The value of T_S is at most $\frac{2}{3}$ of the arrival rate $\lambda_1 = \lambda_2$ as the value of μ increases.

These observations suggest that the value of T_S is sensitive to the value of μ_1, μ_2 and λ_1, λ_2 and is at most $\frac{2}{3}$ of the arrival rate $\lambda_1 = \lambda_2$.

In addition, we calculate T_S by varying the value of $\lambda_1 = \lambda_2$ while μ_1, μ_2 are fixed to 5.0, 7.0, 9.0, 11.0 and μ is fixed to 7.0, 9.0, 11.0, 13.0. The results are shown in Fig. 9.

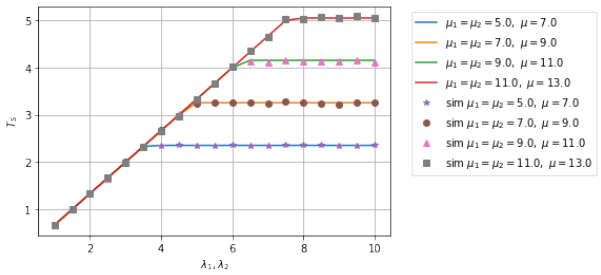

Fig. 9. Throughputs of the shared MLS by varying the value of $\lambda_1 = \lambda_2$.

- The value of T_S increases as the arrival rates $\lambda_1 = \lambda_2$ simultaneously increase to some extent.
- The value of T_S is almost insensitive to the value of λ_1, λ_2 from a certain value. Furthermore, before the saturation, T_S is equal to $2/3$ of $\lambda_1 = \lambda_2$. The saturated throughput depends on $\lambda_1, \lambda_2, \mu_1, \mu_2, mu$ in a complicated manner and increases as these parameters increase.

Furthermore, we calculate T_S by varying the value of λ_1 with $\mu_1 = \mu_2 = 9.0, \mu = 11.0$ while λ_2 are fixed to 1.0, 1.5, 2.0, 2.5. The results are shown in Fig. 10.

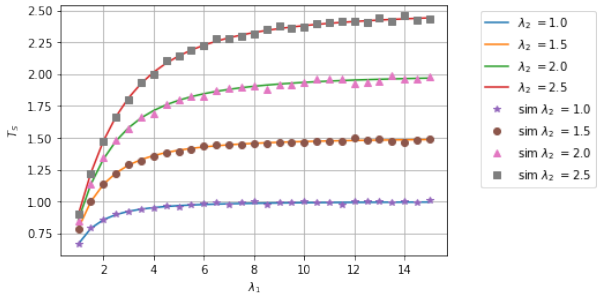

Fig. 10. Throughputs of the shared MLS by varying the value of λ_1.

- The value of T_S increases as the value of λ_1 increases and asymptotes to the value of λ_2.
- The value of T_S increases as the value of λ_2 increases.

C. Comparison of two types of processing methods

By comparing the numerical results of the parallel MLS's throughputs and the shared MLS's throughputs, we obtain the following observations.

- When each processing rate is sufficiently large, the parallel MLS achieves higher performance than that of shared MLS while the arrival rates are the same.
- When each processing rate is sufficiently large, the shared MLS may achieve a better performance than a parallel MLS if the arrival rate of one type of job can be increased.

Regarding the first observation, for example, when comparing the case where the arrival rates are almost the same, the value of T_S is 1.0 for $\lambda_1 = \lambda_2 = 1.5$ shown in Fig. 7, while the value of T_P is 1.4 for $\lambda_1 = 1.4, \lambda_2 = 1.5$ shown in Fig. 3. This means that using twice the number of modules results in approximately 1.4 times the performance compared to that of shared MLS. For example, if a multi-model multi-input MLS is applied to image diagnosis in the medical field, it may be difficult to change the arrival rate of the data, such as MRI images. Therefore, when the arrival rate is the same, a parallel MLS can achieve higher performance.

Regarding the second observation, for example, the value of T_S is 1.0 for $\lambda_1 = \lambda_2 = 1.5$ shown in Fig. 7, while the value of T_P is 1.4 for $\lambda_1 = 1.4, \lambda_2 = 1.5$ shown in Fig. 3. However, by increasing the arrival rate of the shared MLS to $\lambda_1 = \lambda_2 = 2.5$, the value of T_S becomes approximately 1.6, which is higher than the value of $T_P = 1.4$ for $\lambda_1 = 1.4, \lambda_2 = 1.5$. In another example, the value of T_S is approximately 0.67 for $\lambda_1 = \lambda_2 = 1.0$ shown in Fig. 10, but by setting only λ_1 to a value of 6.0 or higher in the shared MLS, the value of T_S becomes approximately 1.0, which is higher than the value of $T_P = 0.9$ for $\lambda_1 = 0.9, \lambda_2 = 1.0$. This means that, for example, if a multi-model multi-input MLS is applied to image recognition of traffic signals and signs in automated driving, the arrival rate of image data can be increased by increasing the number and frequency of images to be recognized by the camera installed in the automated vehicle. This allows us to achieve higher performance even in the case of a shared MLS that uses only one module.

VI. Implications to MLS reliability

Regardless of the different architectures discussed above, outputs from MLS are screened by comparing two prediction results in the comparison unit. If the prediction results are not consistent, either one of the predictions is wrong, and hence the comparison unit does not produce the final output. Therefore, the probability that MLS outputs error can be decreased by adopting either the parallel MLS or the shared MLS. While the parallel MLS can exploit the both diversities of machine learning models and input data (which is referred to as DMDI architecture in [3]), the shared MLS only benefits from the input data diversity. Following the reliability model for N-version machine learning system [3], the parallel MLS is expected to be higher reliability than the shared MLS. In contrast to the existing studies, we show the throughput impacts of a multi-module multi-input MLS. Since the MLS does not output any results as long as the prediction results are agreed in the comparison unit, the higher throughput must be encouraged for applications using the prediction results.

VII. Conclusion

In this paper, we proposed queueing models for a multi-model multi-input MLS into two types of processing schemes; the parallel MLS and the shared MLS. We defined the throughput measure on the proposed models. The findings from our numerical analysis and simulation as follows; i) When the processing rate μ_1, μ_2 and μ are sufficiently large, the parallel MLS achieves a higher throughput than the shared MLS if the arrival rate λ_1, λ_2 cannot be changed. ii) The throughput of the shared MLS improves by increasing the arrival rate λ_1, λ_2.

As a future work, we would like to model a multi-model multi-input MLS with more modules and more types of data as a queueing model and evaluate the performance of the system.

VIII. Acknowledgement

The second and the third authors were supported in part by JSPS KAKENHI Grant Numbers 18K18006 and 19K24337, respectively.

Appendix

We describe each of the block matrices used in the infinitesimal generators Q_P and Q_S defined in section Ⅲ. In the following, the element in the ith row from the top and jth column from the left of a block matrix is called the (i, j)

element of that matrix. And for instance, the (i,j) element of a block matrix A is denoted as $(A)_{i,j}$.

A. Parallel MLS

In this section, we describe each of the block matrices used in the infinitesimal generator Q_P. First, B_0 is a $(2K+1)$-order square matrix that represents the transition of states such as the number of type 2 jobs or states of module and comparison processing when there are no type 1 jobs in the system. Each element $(B_0)_{i,j}$ is defined as follows.

$$(B_0)_{i,j} = \begin{cases} \lambda_2 & i = \{2, ..., K, K+2, ..., 2K\}, j = i+1, \\ \lambda_2 & i = 1, j = K+2, \\ \mu_2 & i = \{K+2, ..., 2K+1\}, j = i-K, \\ \Phi^0_{i,j} & i = j, \\ 0 & (otherwise), \end{cases}$$

where $\Phi^0_{i,j} = -\left(\sum_{j \neq i} B_{0,i,j} + \lambda_1\right)$.

B_1 is a $(5K+1)$-order square matrix that represents the transition of states when there is one job of type 1 in the system. Each element $(B_1)_{i,j}$ is defined as follows.

$$(B_1)_{i,j} = \begin{cases} \lambda_2 & i \in I_1, j = i+1, \\ \lambda_2 & (i,j) = (1, K+3), (2, 3K+3), (2K+3, 4K+3), \\ \mu_2 & i = \{K+4, ..., 2K+2\}, j = i+3K-1, \\ \mu_2 & i = \{3K+3, ..., 4K+2\}, j = i-3K, \\ \mu_2 & i = \{4K+3, ..., 5K+1\}, j = i-2K+1, \\ \mu_2 & i = K+3, j = 2K+3, \\ \mu_1 & i = \{4, ..., K+2\}, j = i+4K-1, \\ \mu_1 & i = \{3K+3, ...4K+2\}, j = i-2K, \\ \mu_1 & i = 3, j = 2K+3, \\ \Phi^1_{i,j} & i = j, \\ 0 & (otherwise), \end{cases}$$

where $\Phi^1_{i,j} = -\left(\sum_{j=1}^{2K+1} A_{1,i,j} \sum_{j \neq i} B_{1,i,j} + \lambda_1\right)$, and I_1 is the set defined as follows.

$$\begin{aligned} I_1 := \{&3, ..., K+1, K+3, ..., 2K+1, 2K+4, ..., 3K+1, \\ &3K+3, ..., 4K+1, 4K+3, ..., 5K\}. \end{aligned}$$

B_2 is a $7K$-order square matrix that represents the transition of states when there are two or more jobs of type 1 in the system. Each element $(B_2)_{i,j}$ is defined as follows.

$$(B_2)_{i,j} = \begin{cases} \lambda_2 & i \in I_2, j = i+1, \\ \lambda_2 & (i,j) = (1, K+3), (2, 3K+3), \\ \lambda_2 & (i,j) = (2K+3, 5K+3), (4K+3, 6K+2), \\ \mu_2 & i = \{K+4, ..., 2K+2\}, j = i+5K-2, \\ \mu_2 & i = \{3K+3, ..., 4K+2\}, j = i-3K, \\ \mu_2 & i = \{5K+3, ..., 6K+1\}, j = i-3K+1, \\ \mu_2 & i = \{6K+2, ..., 7K\}, j = i-2K+2, \\ \mu_2 & i = K+3, j = 4K+3, \\ \mu_1 & i = \{4, ..., K+2\}, j = i+6K-2, \\ \mu_1 & i = \{3K+3, ...5K+2\}, j = i-2K, \\ \mu_1 & i = 3, j = 4K+3, \\ \Phi^2_{i,j} & i = j, \\ 0 & (otherwise), \end{cases}$$

where $\Phi^2_{i,j} = -\left(\sum_{j=1}^{5K+1} A_{2,i,j} \sum_{j \neq i} B_{2,i,j} + \lambda_1\right)$, and I_2 is the set defined as follows.

$$\begin{aligned} I_2 := \{&3, ..., K+1, K+3, ..., 2K+1, 2K+4, ..., 3K+1, \\ &3K+3, ..., 4K+1, 4K+4, ..., 5K+1, 5K+3, ..., 6K, \\ &6K+2, ..., 7K-1\}. \end{aligned}$$

C_0 is a matrix of size $(2K+1) \times (5K+1)$ that represents the transition of the number of jobs of type 1 from 0 to 1. Each element $(C_0)_{i,j}$ is defined as follows.

$$(C_0)_{i,j} = \begin{cases} \lambda_1 & i = \{1, ..., K+1\}, j = i+1, \\ \lambda_1 & i = \{K+2, ..., 2K+1\}, j = i+2K+1, \\ 0 & (otherwise). \end{cases}$$

C_1 is a matrix of size $(5K+1) \times (7K)$ that represents the transition of the number of jobs of type 1 from 1 to 2. Each element $(C_1)_{i,j}$ is defined as follows.

$$(C_1)_{i,j} = \begin{cases} \lambda_1 & i = \{1, ..., 2K+2, 3K+3, ..., 4K+2\}, j = i, \\ \lambda_1 & i = \{2K+3, ..., 3K+2\}, j = i+2K, \\ \lambda_1 & i = \{4K+2, ..., 5K+1\}, j = i+2K-1, \\ 0 & (otherwise). \end{cases}$$

C_2 is a $7K$-order square matrix that represents the transition of the number of in-system jobs of type 1 from i to $i+1$ ($i \geq 2$), defined as

$$C_2 = diag(\lambda_1, ..., \lambda_1).$$

A_1 is a matrix of size $(5K+1) \times (2K+1)$ that represents the transition of the number of jobs of type 1 from 1 to 0. Each element $(A_1)_{i,j}$ is defined as follows.

$$(A_1)_{i,j} = \begin{cases} \mu & i = \{2K+3, ..., 3K+2\}, j = i-2K-2, \\ \mu & i = \{4K+3, ..., 5K+1\}, j = i-3K-1, \\ 0 & (otherwise). \end{cases}$$

A_2 is a matrix of size $(7K) \times (5K+1)$ that represents the transition of the number of jobs of type 1 from 2 to 1. Each element $(A_2)_{i,j}$ is defined as follows.

$$(A_2)_{i,j} = \begin{cases} \mu & i = \{2K+5, ..., 3K+2\}, j = i+2K-2, \\ \mu & i = \{4K+3, ..., 5K+2\}, j = i-4K-1, \\ \mu & i = \{5K+3, ..., 6K+1\}, j = i-4K, \\ \mu & i = \{6K+2, ..., 7K\}, j = i-3K+1, \\ \mu & (i,j) = (2K+3,1), (2K+4, 2K+3), \\ 0 & (otherwise). \end{cases}$$

A_3 is a $7K$-order square matrix that represents the transition of the number of Type 1 jobs from i to $i-1\,(i \geq 3)$. Each element $(A_3)_{i,j}$ is defined as follows.

$$(A_3)_{i,j} = \begin{cases} \mu & i = \{2K+5, ..., 3K+2\}, j = i+4K-3, \\ \mu & i = \{4K+3, ..., 5K+2\}, j = i-4K-1, \\ \mu & i = \{5K+3, ..., 6K+1\}, j = i-4K, \\ \mu & i = \{6K+2, ..., 7K\}, j = i-3K+1, \\ \mu & (i,j) = (2K+3,1), (2K+4, 4K+3), \\ 0 & (otherwise). \end{cases}$$

B. Shared MLS

In this section we describe each of the block matrices used in the infinitesimal generator Q_S. First, A_0 is a 9th-order square matrix that represents the transition of the number of jobs in the buffer from i to $i+1\,(i \geq 1)$, defined as follows

$$A_0 = diag(\lambda_1 + \lambda_2, ..., \lambda_1 + \lambda_2).$$

B_0 is a 15th-order square matrix that represents the transition of states when the number of jobs in the buffer is zero. Each element $(B_0)_{i,j}$ is defined as follows.

$$\begin{aligned} &(B_0)_{1,3} = (B_0)_{5,7} = (B_0)_{8,10} = (B_0)_{12,14} = \lambda_1, \\ &(B_0)_{1,6} = (B_0)_{2,4} = (B_0)_{8,13} = (B_0)_{9,11} = \lambda_2, \\ &(B_0)_{3,2} = (B_0)_{7,8} = (B_0)_{10,9} = (B_0)_{14,15} = \mu_1, \\ &(B_0)_{4,8} = (B_0)_{6,5} = (B_0)_{11,15} = (B_0)_{13,12} = \mu_2, \\ &(B_0)_{8,1} = (B_0)_{9,2} = \cdots = (B_0)_{15,8} = \mu. \end{aligned}$$

Denoting $I := \{1, 2, ..., 15\}$, $J := \{1, 2, ..., 9\}, i \in I$, we define the diagonal components of B_0 as follows.

$$(B_0)_{i,i} = -\left(\sum_{j \in I \setminus \{i\}} (B_0)_{i,j} + \sum_{j \in J} (C_0)_{i,j} \right).$$

The other elements of $(B_0)_{i,j}$ are all 0.

C_0 is a matrix of size 15×9 that represents the transition of the number of jobs in the buffer from 0 to 1. Each element $(C_0)_{i,j}$ is defined as follows.

$$\begin{aligned} &(C_0)_{3,1} = (C_0)_{4,2} = (C_0)_{6,3} = (C_0)_{7,4} = (C_0)_{10,5} = (C_0)_{11,6} \\ &= (C_0)_{13,7} = (C_0)_{14,8} = (C_0)_{15,9} = \lambda_1 + \lambda_2. \end{aligned}$$

The other elements of $(C_0)_{i,j}$ are all 0.

$B_k\,(k \geq 1)$ is a matrix of size 9×15 that represents the transition of the number of jobs in the buffer from k to 0. Each element $(B_k)_{i,j}$ is defined as follows.

$$\begin{aligned} &(B_k)_{1,2} = (B_k)_{5,9} = \mu_1(\tilde{\lambda_1})^k, \\ &(B_k)_{1,4} = (B_k)_{5,11} = \mu_1(\tilde{\lambda_1})^{k-1}\tilde{\lambda_2}, \\ &(B_k)_{3,7} = (B_k)_{7,14} = \mu_2(\tilde{\lambda_2})^{k-1}\tilde{\lambda_1}, \\ &(B_k)_{3,5} = (B_k)_{7,12} = \mu_2(\tilde{\lambda_2})^k. \end{aligned}$$

Also, we define the following elements $(B_1)_{i,j}$.

$$\begin{aligned} &(B_1)_{4,10} = \mu_1\tilde{\lambda_1},\ (B_1)_{4,13} = \mu_1\tilde{\lambda_2}, \\ &(B_1)_{2,10} = \mu_2\tilde{\lambda_1},\ (B_1)_{2,13} = \mu_2\tilde{\lambda_2}, \\ &(B_1)_{9,3} = \mu\tilde{\lambda_1},\ (B_1)_{9,6} = \mu\tilde{\lambda_2}. \end{aligned}$$

The other elements of $(B_k)_{i,j}\,(k \geq 1)$ are all 0.

A_1 is a 9th-order square matrix that represents the transition of other states when the number of jobs in the buffer is greater than or equal to 1. Each element $(A_1)_{i,j}$ is defined as follows.

$$\begin{aligned} &(A_1)_{8,9} = \mu_1,\ (A_1)_{6,9} = \mu_2, \\ &(A_1)_{5,1} = (A_1)_{6,2} = (A_1)_{7,3} = (A_1)_{8,4} = \mu. \end{aligned}$$

Denoting $I := \{1, 2, ..., 9\}$, $J := \{1, 2, ..., 15\}, i \in I$, we define the diagonal components of A_1 as follows.

$$(A_1)_{i,i} = -\left(\sum_{j \in J} (B_1)_{i,j} + \sum_{j \in I \setminus \{i\}} (A_1)_{i,j} + \sum_{j \in I} (A_0)_{i,j} \right).$$

The other elements of $(A_1)_{i,j}$ are all 0.

$A_k\,(k \geq 2)$ is a 9th-order square matrix that represents the transition of the number of jobs in the buffer decreasing by $k-1$. Each element $(A_k)_{i,j}$ is defined as follows.

$$\begin{aligned} &(A_k)_{1,2} = (A_k)_{5,6} = \mu_1(\tilde{\lambda_1})^{k-2}\tilde{\lambda_2}, \\ &(A_k)_{3,4} = (A_k)_{7,8} = \mu_2(\tilde{\lambda_2})^{k-2}\tilde{\lambda_1}. \end{aligned}$$

Also, we define the following elements $(A_2)_{i,j}$.

$$\begin{aligned} &(A_2)_{4,5} = \mu_1\tilde{\lambda_1},\ (A_2)_{4,7} = \mu_1\tilde{\lambda_2},\ (A_2)_{2,5} = \mu_2\tilde{\lambda_1}, \\ &(A_2)_{2,7} = \mu_2\tilde{\lambda_2},\ (A_2)_{9,1} = \mu\tilde{\lambda_1},\ (A_2)_{9,3} = \mu\tilde{\lambda_2}. \end{aligned}$$

The other elements of $(A_k)_{i,j}\,(k \geq 2)$ are all 0.

REFERENCES

[1] I. Goodfellow, J. Shlens, and C. Szegedy, Explaining and Harnessing Adversarial Examples, https://arxiv.org/abs/1412.6572, 2014.

[2] X. Huang, M. Kwiatkowska, S. Wang, M. Wu, Safety Verification of Deep Neural Networks, In Proc. of International Conference on Computer Aided Verification, pp. 3-29, 2017.

[3] F. Machida, N-version machine learning models for safety critical systems, 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, 2019.

[4] Latouche, G., Ramaswami, V., Introduction to matrix analytic methods in stochastic modeling, Society for Industrial and Applied Mathematics, 1999.

[5] Adan I., Leeuwaarden V.J., Selen J., "Analysis of structured Markov processes," arXiv:1709.09060v1, 2017.

[6] E. Wong and Z. Kolter, Provable defenses against adversarial examples via the convex outer adversarial polytope, in International Conference on Machine Learning, pp. 5286-5295, 2018.

[7] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, Machine learning testing: Survey, landscapes and horizons, IEEE Transactions on Software Engineering, 2020.

[8] Y. Tian, K. Pei, S. Jana, and B. Ray, Deeptest: Automated testing of deep-neural-network-driven autonomous cars, in Proceedings of the 40th international conference on software engineering, pp. 303-314, 2018.

[9] C. Murphy, G. E. Kaiser, and M. Arias, An approach to software testing of machine learning applications. In SEKE, vol. 167, 2007.

[10] W. Wu, H. Xu, S. Zhong, M. Lyu, and I. King. Deep validation: Toward detecting real-world corner cases for deep neural networks. In Proc. of the 49th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 125-137, 2019.

[11] F. Machida, On the diversity of machine learning models for system reliability, IEEE Pacific Rim Int'l Symp. on Dependable Computing (PRDC), pp. 276-285, 2019.

[12] H. Xu, Z. Chen, W. Wu, Z. Jin, S. Kuo, M. R. Lyu, NV-DNN: towards fault-tolerant DNN systems with N-version programming, In Proc. of the DSN Workshop on Dependable and Secure Machine Learning, pp. 44-47, 2019.

[13] A. Gujarati, S. Gopalakrishnan, and K. Pattabiraman, New wine in an old bottle: N-version programming for machine learning components, in 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 283-286, 2020.

[14] S. Latifi, B. Zamirai, and S. Mahlke, PolygraphMR: Enhancing the reliability and dependability of CNNs, in 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 99-112, 2020.