2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Metaverse)

A Fault-Tolerant and Real-Time Framework for Efficient Resilience in Edge Inference Systems

Wenwen Liu, Yingjie Geng, Gang Wang*, Xiaoguang Liu*
College of Computer Science, TJ Key Lab of NDST, Nankai University, Tianjin, China
Email: {liuww, gengyj, wgzwp, liuxg} @nbjl.nankai.edu.cn

***Abstract*—In intelligent edge inference systems, well-trained inference models are widely deployed in resource-limited edge devices, and this is prone to slowdown/failure that inflates long-tail latency problems. In this case, efficient fault-tolerant and real-time requirements pose new challenges to the quality of service (QoS) in edge inference systems. In this paper, we present a fault-tolerant and real-time reconstruction framework and call it Flake. Motivated by exploring the long-tail latency problem, Flake builds reconstruction model to solve the fault-tolerant problem based on the coded-computation method. Due to the large size of reconstruction model, Flake uses the knowledge distillation algorithm to compress the reconstruction network to a smaller and faster one for improving efficient resilience and real-time performance. We evaluate Flake over various real-world datasets and compare it with existing fault-tolerant algorithms. The experiment results validate Flake achieves significant performances in terms of improving accuracy, reducing latency compared to other fault-tolerant algorithms. Specifically in the aspect of solving the long-tail latency problem, it reduces the gap between maximum and median latency by up to 2.5× on the premise of satisfying the accuracy requirement.**

***Index Terms*—Intelligent edge inference systems; code-computation; knowledge distillation; fault-tolerant; real-time.**

I. Introduction

Driven by model compression and acceleration technologies development, the intelligent edge system provides a promising paradigm that moves inference tasks from cloud computing centers to edge devices. It shows great potential for improving efficient resilience and QoS performance.

In generally, the training process of machine learning models is running offline in cloud servers. With the help of model compression and acceleration technologies, the well-trained models are widely deployed into edge computing devices. The inference process is running online in edge servers. AI services receive queries from user-facing terminals and return prediction results by performing inference instances. To meet service-level objectives (SLO), online interactive inference tasks have reliability and real-time requirements. For example, in autonomous driving fields, command tasks (such as path planning tasks) could not tolerate a certain amount of consecutive message losses, and monitoring tasks (such as image recognition tasks) require only tens of milliseconds end-to-end latency. However, due to the limited battery life of edge devices, network congestion and other hardware problems, it is difficult to ensure that tasks run timely and reliably. In this case, both efficient fault-tolerant and real-time requirements pose new challenge to keep QoS requirements in edge inference systems.

Therefore in this paper, we present a **F**ault-tolerant and real-time framework in **e**dge inference systems and call it **Flake**. The contributions of this paper are summarized as follows:

- First, we explore the long-tail latency problem on real-world alibaba trace, and analyze the reasons lies in the long-tail latency property.
- Second, we build the fault-tolerant reconstruction model based on coded-computation method to solve the tasks fault-tolerant problem. The model performs reconstruction inference instance over encoding queries, and decodes the output results of slowdown/failure predictions.
- Third, we take network compression algorithms to compress the fault-tolerant reconstruction network. In this part we distill intermediate-level knowledge by KL loss and get logits information outputs by KD loss from the original (teacher) network. The compressed network effectively reduces running time on the basis of guaranteed accuracy.
- Finally, we evaluate Flake over various edge inference models and compare it with three existing popular models. The experiment results show that Flake's better performance in terms of real-time, reliability.

The rest of the paper is organized as follows: Sec. II provides related work and Sec. III illustrates the existence of long tail problem in real-world and analyzes the reasons for this problem. Sec. IV provides the methodology of Flake including the overview of Flake, the description of fault-tolerant model construction based on coded-computation and network compression and training. The evaluated experiments and analysis are given in Sec. V and Sec. VI concludes this paper.

II. Related work

Fault-tolerant problem based on QoS requirements in edge inference systems has been considered as a key challenge, and mainly solved by coded-computation technology and model reconstruction approaches. In this pa-

979-8-3503-4655-8/22/$31.00 ©2022 IEEE

DOI 10.1109/SmartWorld-UIC-ATC-ScalCom-DigitalTwin-PriComp-Metaverse56740.2022.00043

per, we use coded-computation technology to build the reconstruction model, and use knowledge distillation to compress reconstruction model and reduce running time. Thus in this section we describe the related work of coded-computation technology in fault tolerance fields and knowledge distillation approach respectively.

A. Coded-computation Technology

Coded-computation technology uses erasure coding to recover the model outputs. Compared with traditional erasure-data, the coded-computation technology performs computation over encoded data and uses an erasure code to reconstruct the results of computation rather than the data blocks themselves [1].

In neural networks training process, coded-computation is used in gradient computation. Dutta et al. [2] performs coded-computation over the linear operations of neural networks. During the decoding process, coded-computation requires distributed computing and runs many decoding steps. These may increase latency and computation cost. Aoudia et al. [3] explored that using coded-computation to learn error correcting codes for the fault-tolerant communication. It recovers data units transmitted over a noisy channel. Jack Kosaian et al. [1] proposed the parity model that enables erasure-coded resilience in inference systems. It used coded-computation learning-based approach to reconstruct a well-trained neural network model that recoveries slow or failed predictions. However, the parity model need to take much long training time, and it could not well deal with multiple-fault-tolerant problem.

B. Knowledge Distillation

Knowledge Distillation (KD) is one of the promising methods that compress the machine learning model for terminal mobile device tasks. It distills knowledge from a larger deep neural network (known as Teacher) to guide training of a smaller and faster network (known as Student).

To solve the problem of the cumbersome models for deployment, Bucilua et al. [4] proposed the model compression that transfers the knowledge from a large model into a small model without an information drop in accuracy. It mainly mimics the teacher model to training the student model. Furthermore, in [5] Hinton et al. first proposed KD that used for model compression. They introduced a new type of ensemble leveraged a much simpler model with fewer parameters to match the performance of a larger model via aligning the softened softmax output. It shows great performance of rapid training. Based on this research, other methods explored different to improve the performance of a student. For example, in [6] authors transferred the intermediate features from a teacher model into a small student model. In [7] Park et al. proposed structure knowledge to take network structure knowledge from teacher model. And in [8], Yang et al. adopt graph-based information to learn knowledge from teacher model.

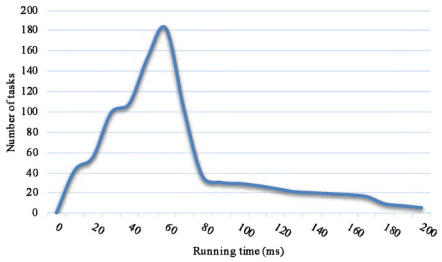

Fig. 1. The long tail latency phenomenon in Alibaba cluster trace.

However, these previous methods are trapped in an original "teacher-student" manner. They use an one-way KD loss function to learning knowledge from a high-capacity teacher model to a compact student model. These may lead the valuable knowledge be ignored to distillate.

In this paper, our goal is to recover the outputs from slowdown/failure requests in fault-tolerant inference systems. We perform coded-computation to reconstruct a fault-tolerant model that solve the fault-tolerant problem under QoS requirements. In order to reduce the latency of fault-tolerant model in inference process, we adopt KD method to match the real-time performance, in which we use KL loss to distill intermediate layer knowledge, and use KD loss to get logits information outputs from the teacher network.

III. Long Tail Latency Phenomenon

In large-scale online inference services, low response times is required consistently to meet users' QoS requirement. Most edge inference tasks are deployed in distributed parallel computing system, services typically dispatch requests and aggregate responses from multiple servers. However, the slowdown/failure problem brings great challenges to low latency requirement. It takes on the long-tail property.

To prove the long-tail property existing in real world, we made a verification experiment. There we adopt tasks from Alibaba cluster trace [9] that used by academics studying edge computing. The job data set includes four subsets, each job has various tasks, each task contains various instances. A task is considered to running completed after all instances belongs it completing running. In this paper, we select 1021 tasks with inter-dependencies to explore their response time. The results are shown in Fig. 1.

From Fig. 1 we can observe a long-tail property of response time: merely 12.72% of tasks have a high contribution of over 80% to the final operation decision performance. We can see that tasks' response time fluctuates markedly over operations with task inference in terms of average and variance.

The reasons for this long-tail latency phenomenon are analyzed as follows:

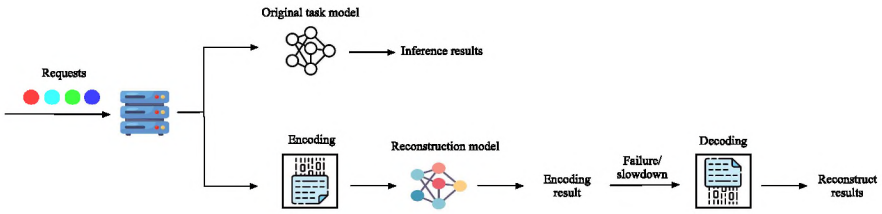

Fig. 2. The overview of Flake.

- Due to the limited battery life of devices, network congestion and other hardware failures, tasks suspend or terminate execution.
- Due to the co-located tasks may compete for server resources (such as CPU core, processor cache, memory bandwidth and network bandwidth), the tasks was waiting in queue to increase response latency.
- Due to the dynamic and heterogeneous resource servers when common fault-tolerant mechanisms tend to introduce additional latency, the system resource is over consume.

Regarding the above points, slowdown/failure tasks may affect the QoS of the whole service because of the dependency relationship among tasks. Therefore in this paper, we present Flake that solves the long tail latency problem. The following section is the description of Flake's methodology.

IV. Methodology

A. Overview

Flake aims to enable real-time and fault-tolerant inference with lower resource-overhead. Fig.2 plots the overview of Flake: The system receives requests and the reconstruction model encodes them to get input into the reconstruction model. Then all request input data is transmitted into models respectively, and get the corresponding output results. When some tasks do not return the corresponding inference result by slowdown/failure problems in specified time threshold, we can get the reconstruct result by decoding the Flake's result and other original inference results.

Thus the Flake consists two main parts:

- **Fault-tolerant model construction based on coded-computation.** This part is to restructure a fault-tolerant network model by the code-computation method. First, we take original model requests input into the addition encoder. Then all request input data is transmitted into models respectively, and we will get the corresponding output results. If come across slowdown or failure tasks, the reconstruction model will take subtraction decode action to get the reconstruct response result.
- **Network compression and training.** This part is to compress the fault-tolerant model for reducing inference time and energy consumption by knowledge distillation method. It includes four parts: network compression, distilling the middle layer knowledge, minimizing the accuracy loss and training the network.

The detailed description of each part is as follows.

B. Fault-tolerant model construction based on coded-computation

An intelligent edge inference system deploys machine learning inference models in edge servers to serve inference services with low latency and high reliability. It accepts task queries from terminal users, allocates them to edge server with corresponding inference instances and performs, then we will get the prediction results.

In this scenario, a new task query starts coming into task queuing by First-In First-Out (FIFO) turning. As for the allocation of task slots, this paper adopts a simple First Come First Serve (FCFS) rule.

In [10], authors proposed timing bounds for each task that trades off between real-time and fault-tolerant requirements. In order to identify under what conditions a task may be lost and reconstruction model start, we use this method to determine the running time bound of the tasks. Once the task does not response the prediction result beyond the timing bound, we start the code-computation reconstruction model. The description of code-computation reconstruction model is as follows.

Using the code-computation method [11] to reconstruct the inference prediction of slowdown/failure tasks is different from the traditional erasure codes. It encodes the original models input, then combines them as input to the reconstruction model, thus getting the reconstruction result by running a decode inference model. In this process, we get reconstruction through DNN model for encoders and decoders, not performing encode/decode computation over data. Inspired by [1], to reduce latency and computational cost, we adopt the simple addition/subtraction erasure code to encode/decode the unavailable predictions. The reasons we choose this simple encoder/decoder are: it (1) enables to accurately reconstruct unavailable predictions; (2) runs fast and contributes to solving long-tail latency problem; (3) applicable to a broad range of inference tasks.

To obtain input with a consistent format, we allocate the same type of tasks with the same network model to running in the same edge server. When a task comes across

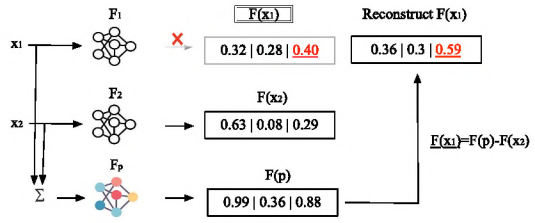

Fig. 3. The code-computation method to reconstruct the inference prediction of slowdown/failure tasks.

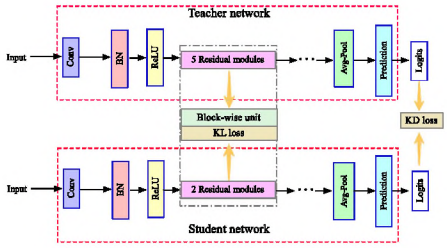

Fig. 4. An example of network compression.

failure/slowdown problems(it does not return the response result after the timing bound), the reconstruction model will decode its inference results to reconstruct the response results of failure/slowdown request.

Herein the simple addition/subtraction encoder/decoder was adopt to reconstruct slowdown/failure task prediction. Shown in Fig. 3, when a prediction from task F_1 is failure/slowdown, first, we normalize the requests of original tasks to ensure the consistency of the format. And the request of reconstructing model is the summation of the pairwise tasks, noted as $p = x_1 + x_2$. The input dataset of reconstruct model is the same as p, the labels are generated as the corresponding element-wise summation of $F(x_1)$ and $F(x_2)$ vectors. Then we input the dataset and labels into the reconstruct model F_p to produce reconstruct instance $F(p)$. When the system monitors $F(x_1)$ is slowdown/failure, the subtraction decoder reconstructs prediction result of $F(x_1)$ as $F(p) - F(x_2)$.

In this process, Flake leverages code-computation to implement encode/decode mechanism throughout the whole inference process, this greatly increases the security of data transmission. On the other hand, reconstruction model enable reconstruct the inference prediction results for every slowdown/failure task by utilizing one extra instance to perform a reconstruction model, and for each reconstructed task, reconstruction model only need 1× less additional resources that the case no slowdown/failure occurred. Similarly, by using the multiple reconstruction models, it achieves the purpose of solving multiple tasks failure problem. However, the size of the reconstruction model is larger than the original model, which will increase the reconstruction inference time and computational cost. In order to compress reconstruction while keep its accuracy, we propose network compression method, that is described as next part.

C. Network compression and training

DNN is becoming the footstone for ubiquitous AI. It performs by combining the lower and mid-level features to form abstract high level features. Existing KD method effectively learns a small student model from a large DNN teacher model to get model compression and acceleration.

In this paper, based on the observation of edge inference DNN model, we aim at the design objectives as follows:

- to reduce the aggregate complexity of the reconstruction model for reducing latency and computational cost;
- to distill the middle layer knowledge;
- to minimize the accuracy loss of the overall model's output.

Network compression. To reduce the aggregate complexity of the reconstruct (student) model, our approach distills limited knowledge and ignores rich features encoded in the middle layers of teacher networks. In practice, we reuse the architecture of the original (teacher) model and adopt a subset of the layers in the original model to design the student model, some of little-importance layers are skipped (for example, the Resnet-50 model is used as the student network and the Resnet-152 model is used as the teacher network in experiment on accuracy). In order to ensure that the performance of student network is close to teacher network as much as possible, we extract features in mid-level layers of teacher networks to guide the student's learning process, and optimize the loss function of student networks to boost the performance and the output prediction accuracy.

For considering the reader to understand our mean easier, we present an example. Fig. 4 depicts a case that shows the procedure of network compression and student model building. In the example, the original (teacher) model's high-level layer consists of 5 low-level layers: Convolution, BN, ReLU, Average Pooling layers, and Prediction. In the student model, we reuse the whole architecture. To compact student model we take only 2 Residual modules and skip 3 layers from teacher's 5 Residual modules, and a Block-wise unit is defined. We take extract features in the Block-wise unit using KL loss. Then we use KD loss to enforce the student network to learn from the softened output of the teacher network. The process of knowledge distillation will be described in detail below.

Distill the middle layer knowledge. Knowledge transfer techniques are employed to train student model that mimic the outputs of the original (teacher) model. A well-trained original (teacher) network is beneficial for promoting the performance of the student via providing a various of knowledge. To distill the middle layer knowledge, we use KL loss to reduce the compress block's differences at

the feature-map level.

For well-trained original (teacher) network T and compressed student S pair, we design a block-wise unit to enable the student network get intermediate-level feature knowledge from teacher. First we determine the feature transfer block of the teacher and student network. That's the corresponding compressed block from the teacher and student network, which changes according to network type. The intermediate-level feature transfer takes place in every block-wise unit. Then the total intermediate-level feature distills from the teacher network is defined as the sum of each feature block-wise pair.

We need to distill M intermediate feature distributions from teacher T, that gets $\{f_i^S\}_{i=1}^M$ from $\{f_i^T\}_{i=1}^M$. We adopt the KL divergence to make intermediate feature distribution of student's compress block approximate the teacher's. The loss of the $i-th$ block pair among T and S can be expressed as follows:

$$\mathcal{L}_{KL}^{block}(f_i^T, f_i^S). \tag{1}$$

The sum of each feature block-wise pair is defined as:

$$\mathcal{L}^{Inter} = \sum_{i=1}^{M} \mathcal{L}_{KL}^{block}(f_i^T, f_i^S), \tag{2}$$

Thus the KL divergence is included in the loss function to improve the feature extracting ability of the student network if we define the compressed block-wise unit.

Minimize the accuracy loss. However more valuable dark knowledge in logits information and high layer has been not distilled to instruct the student by KL methods. Some important differences between T and S still remain.

To minimize the accuracy loss and get more logits knowledge from teacher's network, we use the KD loss function. For well-trained original (teacher) network T and compressed student S pair, T acts as the teacher network to provide logits information to student S. The logits information will take the dark knowledge to the student network that can not be explicitly provided by the training data samples. We use it to understand how a teacher generalize given tasks in a more efficient form.

The logits values of original (teacher) network T is denoted as z_T, and we get logits values of compressed student S network is denoted as z_S. Given logits z as the output of the last fully connected layer and z_i the logit for the i-th class. A temperature parameter produces a softer probability distribution over classes, we denote it as τ. Therefore, a softmax function with τ is formulated as follows:

$$\tau(o_i^S) = softmax(\frac{z_S}{\tau}) = \frac{\exp(z_i^S/\tau)}{\sum_j \exp(z_j^S/\tau)}, \tag{3}$$

$$\tau(o_i^T) = softmax(\frac{z_T}{\tau}) = \frac{\exp(z_i^T/\tau)}{\sum_j \exp(z_j^T/\tau)}, \tag{4}$$

Algorithm 1 Training the of algorithm

Require: The well-trained original (teacher) model T, Training set x, label set y.

Ensure: The student model.

1: **repeat**
2: Input images set x_i and its label y to the well-trained original (teacher) model T .
3: Compute the loss $\mathcal{L}^{Inter}$ with equation (2).
4: Compute cross entropy loss $\mathcal{L}^{KD}$ with equation (5).
5: Update the parameters of student S by the *SGD* algorithm.
6: **until** Maximum iterations or convergence are reached.

where, if τ gets a higher temperature, it will produce a softer probability distribution over classes to transfer important information to student network. In particular, $\tau \to 1$, all classes divide the same probability; $\tau \to 0$, the soft targets become one-hot labels, i.e., the hard targets. Hence, τ controls the importance of each soft target.

Both the soft targets $\tau(o_i^S)$ and $\tau(o_i^T)$ are important for improving the performance of the student network [12]. Based on results of them, we adopt KD loss to improve the convergence rate and the accuracy under the guidance of the original (teacher) [13]. It is formulated to match the logits between the teacher model and the student model as follows,

$$\mathcal{L}^{KD} = \gamma\tau^2\mathcal{H}(\tau(o_i^T), \tau(o_i^S)), \tag{5}$$

where γ is the balanced weight coefficient of $\mathcal{L}^{KD}$. And we use $\gamma = 0.7$ and $\tau = 4$ to carry out KD training [14]. It acts as a carrier to transfer to logits knowledge from deeper and wider neural networks to shallower and thinner neural networks instead of the true labels.

Training the network. Based on above elements, the student network could be generated offline. Algorithm 1 illustrates the training process. We require the well-trained original (teacher) model, and its training set x and label set y as input. In Algorithm 1, the student with transferring knowledge are trained by the loss functions $\mathcal{L}^{Inter}$ and $\mathcal{L}^{KD}$. For the $\mathcal{L}^{Inter}$, we utilize the in-network compression method to preserve overall accuracy. The student network get intermediate-level feature knowledge from teacher. This provides strong supervision to the student. Moreover, for $\mathcal{L}^{KD}$, it enforces the student network to learn from the softened output of the teacher network. Thus, using mixed loss function, we constantly update the parameters of student by the SGD algorithm, until the maximum iterations or convergence are reached. The student gradually obtains valuable information from the original (teacher) network, and effectively integrate them to achieve enhanced performance.

V. Experiments

In this section, we conduct both offline and online experiments to verify the effectiveness of Flake. All the experiments are implemented by PyTorch based on real-world datasets. We first describe the experimental setup, that including inference tasks and datasets, hardware configurations, inference tasks, datasets and baseline methods. To illustrate Flake's fault tolerant and real-time performance, we carry out a series of experiments on latency and accuracy. All above contents are described in detail below.

A. Experimental Setup

Inference tasks and datasets. Learning a reconstruct model is bound up with the original inference tasks. Therefore, we evaluate Flake on popular image classification tasks, that including Resnet-50 [15], Resnet-152 [15], MobilenetV3 [16]. **Resnet-50** and **Resnet-152** belong to Resnet (residual nets) with a depth of 50 layers and 152 layers. They are widely used neural networks on image classification tasks. **Mobilenet-V3** is a dedicated mobile computer vision architecture that takes a combination of hardware aware network architecture search (NAS) and the NetAdapt algorithm. Moreover, we choose three popular image classification datasets, that including **CIFAR-10** [17], **CIFAR-100** [17], and **Tiny-ImageNet-200 datasets** (the tiny version of ImageNet ILSVRC 2012 [18]).

Hardware configurations. Referred to the widely-used cloudlet-discovery project [19] architecture, we establish our edge computing simulation environment. The hierarchy of simulation environment is 2-tier, which includes an edge server for running inference tasks and one cloud server for training network model:

- A laptop computer with Intel NCS 2 Myriad X as edge server for inference.
- A desktop computer with NVIDIA GeForce RTX 2080 Ti as cloud server for training.

Baseline methods. To demonstrate the effectiveness of our proposed method, we employ the following fault-tolerant methods for comparison.

- **Single-Replica Placement (SRP)**: For every inference instance, we deploy a extra copy in edge server. It performs the single-replica inference model to reconstruct the prediction result when occurs slowdown or failure problem.
- **ParM(PM)** [1]: ParM is a neural reconstruction network trained by the original network architecture (without model compression and acceleration). It encodes (addition) multiple queries into a reconstruction result and decodes (subtraction) the slow or failed predictions to get the approximations prediction result of unavailable predictions. Notice that it is not applicable when multiple unavailable queries occur.
- **Naive KD (NKD) [5]:** Naive KD takes standard training of student model by learning from the teacher's main performance. To get the model compression and acceleration, it mimics the feature extraction and classification of teacher network to get fewer parameters and computations.

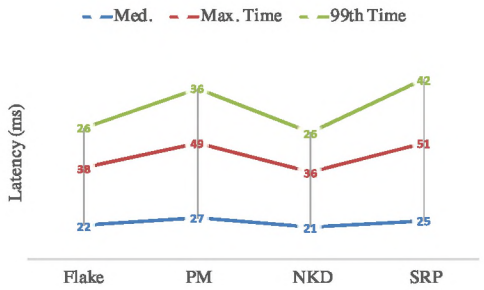

Fig. 5. The latency metrics {the maximum latency, the median latency, the 99th percentile queries latency} of Flake, PM, SRP and NKD.

B. Experiment on Latency

In this subsection, we experiment on Flake's real-time performance based on the MobileNet-V3 model instance. We perform the same training protocol as [16]. All original model instances are trained from scratch on tiny-ImageNet-200 dataset. We construct reconstruction models based on the original model by proposed network compression method. Specifically, for NKD and Flake, we construct models using MobilenetV3 pairs as the teacher-student networks. The MobilenetV3 small model is used as the student network and the MobilenetV3 large model is used as the teacher network. Then carrying about the network compression method by standard training, we could get the compressed reconstruction inference model. We drawn test queries from tiny-ImageNet-200 dataset. To avoid selection bias, the queries are got from 200 distinct categories rather than the usual 2 categories in each prediction inference. The queries are pulled into inference instances with different rates to evaluate the real-time performance. **The latency** is measured by the time between when the inference instances receives a query and when the corresponding prediction result is returned. Unless special noted, all experiments take this latency scale.

Long-tail latency reduction. Herein, we take the 200qps query rate and adopt {the maximum latency, the median latency, the 99th percentile queries latency} experiment metrics. From the gaps of between the maximum latency and the median latency, and between the maximum latency and the 99% queries latency, we can capture long-tail latency reduction performance.

As displayed in Fig. 5, NKD gets the minimal gap between the maximum latency and median latency, followed by Flake, PM, SRP. Similarly, NKD gets the minimal gap between the 99th percentile queries latency and median latency, followed by Flake, PM, SRP. To more cleanly observation, TABLE I describes the gap between the maximum latency and the median latency (GMM), and

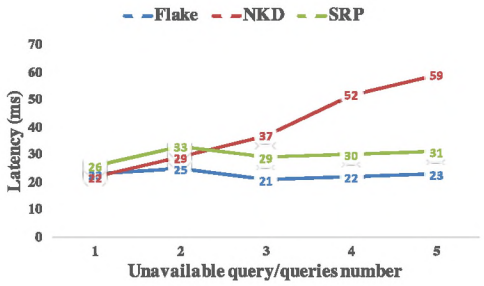

Fig. 6. Test latency by varying unavailable query/queries number.

the gap between the 99.9th percentile queries latency and the median latency (GPM) of Flake, PM, NKD and SRP. The reason for this is that NKD's network model is the simplest compared to other reconstruction models. It does not contain a code-computation module and compresses the original model by KD. Thus, it reduces the computational latency of failure/slowdown queries. The gap between the maximum latency and median latency of Flake is only 1 millisecond short of NKD, this result is acceptable for reconstruction system. Flake contains code-computation module, which enable to recovery the prediction results and also effectively improve the security of failure/slowdown queries. Moreover, due to utilizing the better performing compression network methods, Flake can recovery the prediction results more accurately than Naive KD (next experiment proves this). Since PM employs the original network structure, which is more complex than NKD and Flake, it takes a longer reconstruction time. At last for SRP, it recoveries prediction results by repeating transmission of queries, this will reduce the network transmission speed and occupy a large storage space. Therefore, Flakes reduces the gap between tail and median latency by 2× compared to PM, and by 2.5× compared to SRP. Though it is slightly longer than NKD, the ability of long-tail latency reduction is remarkable.

TABLE I
THE COMPARISON OF LATENCY METRICS.

	Flake	PM	NKD	SRP
GMM (ms)	16	22	15	26
GPM (ms)	4	9	5	17

Test latency by varying unavailable query/queries number. To investigate the effect of the unavailable query/queries number on latency, we pull different numbers of unavailable query/queries into reconstruction system. The unavailable query/queries number is set to {1, 2, 3, 4, 5} respectively. All the experimental results are averaged latency over 3 running times to prevent the effect of accidental samples. Due to PM only supports single query reconstruction, this experiment does not involve it.

As shown in Fig. 6, the Flake's latency is a slight change (less than 5 ms, which is negligible) facing the unavailable query/queries numbers increase. The reason for this is that Flake recalls a previous query instance that has run successfully from the cache queue as slowdown/failure code-computation partners to reconstruct prediction results. No matter how it changes, Flake could get the code-computation partners (original models) that recovery slowdown/failure prediction results, it has little influence by unavailable query/queries numbers. For NKD, we only put one reconstruction model in inference system. The unavailable query/queries are reconstructed by executing NKD model one after the other, leading to its latency increases as the unavailable query/queries numbers increase. For SPR, it also has a slight change. The reason for this is that we put one reconstruction model for every inference system. Even this result did not trade off a large amount of its storage consumption. As we expected, Flake enables us to guarantee no increased latency and storage load with multiple unavailable queries.

C. Experiment on Accuracy

We next evaluate Flake's accuracy performance compared to baselines. The accuracy experiments are conducted on Resnet with different depths. The original inference model instances are trained from Resnet-152, and for NKD and Flake, the Resnet-50 model is used as the student network and the Resnet-152 model is used as the teacher network. To investigate the performance of accuracy on different scale datasets, we conduct the teacher-student pairs (ResNet152/ResNet50) on CIFAR-10 and CIFAR-100 datasets. We perform the training setup refer to [15] based on CIFAR datasets. Specifically, for NKD and Flake, the student model should be trained offline. We train the student network through the Resnet-50 model. The Stochastic Gradient Descent (SGD) method is applied to training for 400 iterations with the batch size of 64. The learning rate starts from 0.1 and is divided by 10 at epoch {100, 200, 300 and 400}. We set the weight decay to be 5e-4 and T to be 4 for generating soften predictions. Then carrying about the KD by training, we could get the compressed reconstruction inference model. We pull queries that get from all distinct categories in CIFAR into inference instances to evaluate the accuracy performance. The unavailable query/queries number is set to 1. The results are summarized in TABLE II.

In TABLE II, we measured top-1 and top-5 test accuracy based on CIFAR-10 and CIFAR-100 dataset to compare the image classification performance. It depicts that SPR gets the maximum top-1 and top-5 accuracy, and followed by Flake, PM, NKD. For SPR, the original ResNet-152 model is assigned as the reconstruction instance network. Thus its accuracy is comparable to the original model, but its latency results and resource cost are not as favorable as Flake and NKD. For Flake, networks training uses the proposed method that shows

TABLE II
THE COMPARISON OF ACCURACY METRICS.

	Model	params	Top-1(CIFAR-10)	Top-5(CIFAR-10)	Top-1(CIFAR-100)	Top-5(CIFAR-100)
Flake	ResNet-50	**25.32M**	75.46%	96.32%	**73.79%**	92.92%
PM	ResNet-152	62.27M	75.29%	90.34%	65.93%	89.32%
NKD	ResNet-50	25.56M	69.38%	95.32%	70.84%	90.32%
SRP	ResNet-152	60.19M	**76.83%**	**97.04%**	73.68%	**95.74%**

higher accuracy than PM and NKD. Note that the Flake is 1.37% lower than SPR in terms of top-1 test accuracy, and is 0.72% lower than SPR in terms of top-5 test accuracy. This accuracy result is acceptable compared SPR's high latency and storage load. TABLE II also shows the Resnet instances are adopted on the CIFAR-100 dataset. We do this for observing the accuracy performance of Flake on large-scale datasets. We observe that the accuracy results are roughly the same as the CIFAR-10 dataset. To summarize Flake's training with offline KD can provide more comprehensive knowledge to the student network, which consequently improves the student accuracy and stability performance on different size datasets.

VI. CONCLUSIONS

.

In this paper, we propose Flake framework, which focuses on long tail latency problem in edge inference system. Flake builds reconstruction inference model by introducing code-computation and network compression methods. It leverages code-computation to implement encode/decode mechanism, that allows encoded queries to be decoded to a form that enables reconstructing the slow or failed predictions and increasing the data security in the inference process. Then Flake introduces a mixed KD method that distills intermediate-level knowledge by KL and gets high-quality outputs by KD loss function. At last, we evaluate Flake performance using popular image classification tasks in edge computing scenarios based on the CIFAR-10, CIFAR-100 and Tiny-ImageNet-200 datasets. The experimental results show that Flake achieves efficient reconstruction capability to improve QoS.

ACKNOWLEDGMENT

This work is partially supported by National Science Foundation of China (62141412, 62272252, 62272253); Science and Technology Development Plan of Tianjin (20JCZDJC00610); Key Research and Development Program of Guang Dong (No. 2021B0101310002); and the Fundamental Research Funds for the Central Universities and TKL- NDST.

REFERENCES

[1] J. Kosaian, K. Rashmi, and S. Venkataraman, "Parity models: erasure-coded resilience for prediction serving systems," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 30–46, 2019.

[2] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized polydot codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1585–1589, IEEE, 2018.

[3] F. A. Aoudia and J. Hoydis, "End-to-end learning of communications systems without a channel model," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pp. 298–303, IEEE, 2018.

[4] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

[5] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[6] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *Int. Conf. Learn. Represent. (ICLR)*, 2015.

[7] W. Park, D. Kim, Y. Lu, and M. Cho, "Relational knowledge distillation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3967–3976, 2019.

[8] Y. Yang, J. Qiu, M. Song, D. Tao, and X. Wang, "Distilling knowledge from graph convolutional networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7074–7083, 2020.

[9] "Alibaba trace." https://github.com/alibaba/clusterdata/blob/v2018/cluster-trace-v2018/trace_2018, 2018. [Online; accessed 19-July-2021].

[10] C. Wang, C. Gill, and C. Lu, "Frame: Fault tolerant and real-time messaging for edge computing," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 976–985, IEEE, 2019.

[11] J. Kosaian, K. Rashmi, and S. Venkataraman, "Learning-based coded computation," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 227–236, 2020.

[12] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[13] M. Phuong and C. Lampert, "Towards understanding knowledge distillation," in *International Conference on Machine Learning*, pp. 5142–5151, PMLR, 2019.

[14] S. Fu, Z. Li, K. Liu, S. Din, M. Imran, and X. Yang, "Model compression for iot applications in industry 4.0 via multiscale knowledge transfer," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6013–6022, 2019.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[16] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324, 2019.

[17] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Handbook of Systemic Autoimmune Diseases*, vol. 1, no. 4, 2009.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[19] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.