Performance and Failure Cause Estimation for Machine Learning Systems in the Wild

Xiruo Liu[1], Furqan Khan[1], Yue Niu[2], Pradeep Natarajan[1], Rinat Khaziev[1], Salman Avestimehr[2], and Prateek Singhal[1]

[1] Amazon, Sunnyvale CA 94089, USA
{xiruoliu, furqankh, natarap, rinatk, prtksngh}@amazon.com
[2] University of Southern California, Los Angeles CA 90007, USA
{yueniu, avestime}@usc.edu

Abstract. Machine learning systems at the edge may fail as the real world data can be noisy and have different distribution from the training dataset which the machine learning systems were developed on. However, it is very difficult to detect the system failures and identify root cause of the failures for systems on the edge devices due to many factors such as privacy concerns, regulations, constrained computation resources and expensive error labeling. In this work, we propose a flexible and general framework, *PERF*, to estimate the performance of a machine learning system deployed at the edge device and identify the root cause of failure if it fails. *PERF*is similar yet different from the classic teacher-student paradigm. Within *PERF*, a larger performance estimation model $\mathcal{PE}$ is deployed along with the smaller target system $\mathcal{T}$ to be evaluated on the same edge device. While the device is idle, $\mathcal{PE}$ can be activated and predicts $\mathcal{T}$'s performance and the failure causes from $\mathcal{T}$'s internal and outputs features on the device without human intervention. The privacy risk can be avoided as the evaluation is done on the edge device without sending any user data to the backend cloud. We validated *PERF*on two exemplar tasks and showed promising results.

Keywords: Machine learning · Performance evaluation.

1 Introduction

Machine learning techniques have gained huge success in recent years and have been widely deployed across a large variety of domains, such as autonomous driving, conversational assistants, face recognition, natural language understanding and fraud detection [1,2,7,12]. A typical development cycle of machine learning systems includes data collection and annotation, model training, model integration, system deployment, and model upgrade or improvement. Ideally, after a system is deployed, feedbacks are collected and analyzed to understand the system performance. Especially, the causes that lead to system failures should be collected and analyzed for future improvements.

However, for a complicated machine learning system deployed at the edge (e.g., Google Home and Amazon Echo Show devices), it is very challenging to

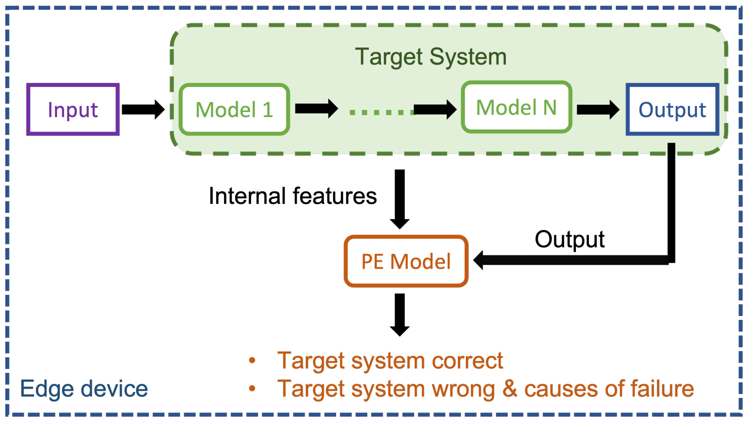

Fig. 1. *PERF* framework for performance estimation and failure cause prediction. A $\mathcal{PE}$ model is deployed aside with the target system $\mathcal{T}$ to be evaluated. While $\mathcal{T}$ performs its normal operations, it saves its output features and features extracted from selected internal nodes. When the device becomes idle and sufficient compute resources are freed, $\mathcal{PE}$ runs on the saved features from $\mathcal{T}$ and estimates if $\mathcal{T}$'s final output is correct. If $\mathcal{PE}$ predicts that $\mathcal{T}$ failed, it will also estimate the root causes of the failure.

collect feedbacks at a large scale for performance evaluation and pinpointing the cause of failures due to many reasons, such as runtime resource constraints, system complexity, privacy concerns and legal regulations. Due to the privacy concerns and regulations, we do not want to collect raw validation data and send it to the cloud for analysis as it may contain user sensitive information.

Even if it is ever possible to collect raw data on the field, deciding what data to collect for analysis is also not easy due to practical constraints. Specifically, the deployed system itself may either have few clues about its own performance or do not have enough knowledge on how the running statistics at each stage affect the final performance. In addition, considering the limited on-device resources, it is prohibitively costly to run another monitoring system in parallel with the deployed model, to continuously track the target system's performance.

Therefore, we would like to answer the following questions: *Can we design a generic solution that accurately estimates the performance of a deployed target machine learning system without revealing any private user data? Furthermore, can we identify the failure causes without human intervention on spot?*

To answer these questions, we propose a general performance estimation framework *PERF*, as shown in Figure 1. *PERF* has two major components: a target system $\mathcal{T}$, which is the system to be evaluated and may consist of several cascaded models, and a performance estimation model $\mathcal{PE}$, which is deployed together with $\mathcal{T}$ on the same edge device. When the device is idle, $\mathcal{PE}$ accesses both the internal and output features cached by $\mathcal{T}$, and uses it to estimate whether $\mathcal{T}$ worked correctly. More importantly, if $\mathcal{T}$ failed, $\mathcal{PE}$ analyzes how anomalies in the input space affect $\mathcal{T}$'s result. It is important to realize the difference between

PERF and the teacher-student paradigm of knowledge distillation, which will be discussed further in Section 2. In *PERF*, the larger model $\mathcal{PE}$ leverages running statistics from the smaller $\mathcal{T}$ to produce its prediction, which is opposite to the teacher-student paradigm where a smaller student model learns from a larger teacher model.

The target system $\mathcal{T}$ often performs real time tasks and hence has tight throughput and latency requirements. Also, an edge device typically has limited compute and storage constraints, which put on resource consumption constraints on $\mathcal{T}$. As a result, individual models in $\mathcal{T}$ are often small and quantized, which limits their representation power. In order to keep the best experience for users, we do not modify $\mathcal{T}$, which is optimized for its designated task. *On the other hand, $\mathcal{PE}$ may run offline on the data saved from $\mathcal{T}$ and does not need to satisfy the same throughput and latency constraints as $\mathcal{T}$.* Therefore, $\mathcal{PE}$ may have larger representation power to perform more complex tasks with better accuracy.

Another important advantage of *PERF* is privacy protection as it neither requires raw user data nor sending any user information to the cloud. Also, compared to conventional methods of analyzing anomalies in inputs [5], our formulation takes a step further and predicts whether certain anomalies in inputs cause failure of the deployed system. To the best of our knowledge, this work comes as the first effort to estimate the deployed machine learning system's performance and *identify the root causes of failures*.

In this paper, we discuss the related work in Section 2. Then in Section 3, we first introduce the general framework of *PERF* and then elaborate an exemplar design for face directness detection task. Dataset creation and evaluation for the face directness detection task will be presented in Section 4 and Section 5. Then an additional image classification task will be discussed in Section 6 to demonstrate the generalizability and flexibility of *PERF*.

2 Related work

The problem of interest is related but different from estimating a system's generalization capability. Instead, we aim at obtaining an unbiased empirical performance estimation because commonly used metrics, e.g., precision and recall, are affected by the data distribution in the test set of the development dataset, which may differ from the real deployment environment. Therefore, even if there is no new concept introduced, the empirical performance estimate may differ.

Performance estimation – Performance estimation is an active research area in analyzing machine learning systems, where direct measurements for a system's performance are usually difficult. For instance, in the image captioning task [10,18], it is difficult to provide a quantitative metric evaluating the model's captioning quality. Therefore, [14] proposes an estimation model that probes into the captioning model and collects internal representations. In addition to image captioning, work such as [11] also designed a performance estimation model to examine a conversational assistant system with cascaded neural networks. In [3], a framework was proposed to analyze errors and their impacts on the model performance in object detection and instance segmentation tasks. However, this work does not analyze the root causes of errors and can not apply to other domains. To the best of our knowledge, we have not seen a general framework for automated performance estimation and failure causes analysis.

Anomaly detection – Comparing to identifying the root cause of failures for machine learning models, anomaly detection [5] has been studied for a long time and it aims at identifying outliers that deviate from the majority of the samples. Given an input to a system, anomaly detection does not analyze the causal relationship between the anomalies and the system output, i.e., whether anomalies in the input affect the system's performance.

Knowledge distillation – Our problem formulation is similar yet different from the teacher-student paradigm of knowledge distillation (KD) [9], where a larger teacher model is used to distill knowledge into a smaller student model. The teacher model is expected to perform better than the student model due to its larger representation capacity. The key difference between KD and *PERF*is that in KD the parameters of the smaller student model are learned from the outputs of the larger teacher model, whereas in *PERF*the parameters of the larger $\mathcal{PE}$ model are learned from the smaller $\mathcal{T}$ model and annotations. Note that $\mathcal{PE}$ is only a critic of $\mathcal{T}$ and does not provide any direct feedback to $\mathcal{T}$.

3 PERF framework

*PERF*is a general framework that may work with both simple and complex systems. As shown in Figure 1, there is a target system $\mathcal{T}$ on the edge device that we want to estimate its performance and predict its failure cause if it fails. $\mathcal{T}$ may cache its output and selected internal features during runtime, which will be used as the inputs to a performance estimation model $\mathcal{PE}$. When the device is idle, $\mathcal{PE}$ takes the cached features and predict $\mathcal{T}$'s performance and failure causes.

3.1 Problem formulation

As the first example of *PERF*, we examine a computer vision task of face directness detection, i.e., determining if a person's face is facing the camera as illustrated in Fig. 2. With a $\mathcal{T}$ model deployed on a device predicting the face directness, our objective is to design a $\mathcal{PE}$ model hat is deployed alongside $\mathcal{T}$ to estimate if its output is correct and error causes if $\mathcal{T}$ fails. To be specific, given intermediate features X and output features Y from $\mathcal{T}$, $\mathcal{PE}$ produces an estimation est as $est = \mathcal{PE}(X, Y)$. Note that X may be an ensemble of multiple feature vectors extracted from different internal layers of $\mathcal{T}$. est may be either *Correct* (i.e., $\mathcal{T}$ is correct) or at least one of the following errors. Note that errors we consider here are the most commonly observed failure causes in real deployment environments and are not mutually exclusive.

1. *Localization error*: $\mathcal{T}$ fails to localize the person's face correctly at the face detection stage, and hence causes a wrong face directness prediction. For example, this error can be a shifted bounding box detection or a mis-detection.

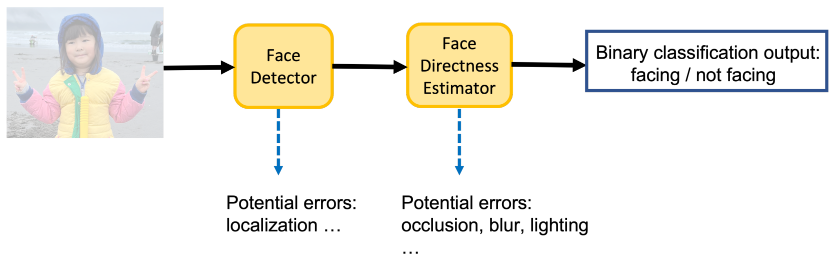
Face Detector
Face Directness Estimator
Binary classification output: facing / not facing
Potential errors: localization …
Potential errors: occlusion, blur, lighting …

Fig. 2. An example of the target system for the face directness prediction task that includes two cascaded models. An input image (not a customer image, generated for illustration purpose) first passes through a face detection model to generate a face crop, where potential errors such as localization errors may occur. Then the face crop is fed into a face directness estimator to predict whether the face is facing the camera, where causes such as blur, occlusion or poor lighting may cause the estimator to fail.

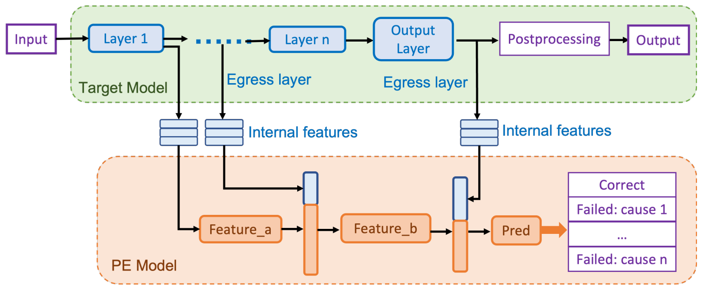

Fig. 3. An example design of *PERF*. Given an input, $\mathcal{PE}$ leverages internal features from $\mathcal{T}$ to predict $\mathcal{T}$'s performance. This exemplar $\mathcal{PE}$ includes three major components: *feature_ a* extracts internal features from an early layer of $\mathcal{T}$; *feature_ b* combines the output of *feature_ a* and features from a middle layer of $\mathcal{T}$ as its input; *pred* combines *feature_ b*'s output and features from $\mathcal{T}$'s final layer as its input to produce the performance estimation. $\mathcal{PE}$ outputs the correctness of $\mathcal{T}$ and predicts failure causes.

2. *Occlusion error*: the face is partially occluded by an object (e.g., cup, glasses, mask) that leads to $\mathcal{T}$'s failure.
3. *Blur error*: the image is too blurry, which may happen on a moving subject.
4. *Lighting error*: the lighting condition of the environment is poor (e.g., too dark or too bright) that causes $\mathcal{T}$ failure.

3.2 Model design

Target system $\mathcal{T}$: As shown in Figure 2, we chose a face directness prediction system with two stages: face detection and directness prediction. In the first stage, a face detector model finds the bounding box of a face in the image, which is used to crop out the face. In the second stage, a directness prediction model takes the face crop from the previous stage as its input and predicts whether the person is facing the camera. The failure causes considered here extend over both stages. For example, when $\mathcal{T}$ fails, a localization error implies that the face detector failed to produce a good bounding box, and hence affected the final result. If the bounding box has good quality, then the system failure might be due to an error in the second stage. As a result, the capability of identifying failure causes also helps to pinpoint which component fails in a complex $\mathcal{T}$.

Performance estimation model $\mathcal{PE}$: Here we elaborate an example instantiation of *PERF*framework as shown in Figure 3. At a high level, $\mathcal{PE}$ leverages data saved from different stages of $\mathcal{T}$, and learns how different factors in the input space affect $\mathcal{T}$'s outputs. Note that there is no raw or processed user data sent to the cloud and thus the privacy risk is minimized. Also, $\mathcal{PE}$ works on processed data cached by $\mathcal{T}$ and can be activated when device is idle (e.g., during midnight). Therefore $\mathcal{PE}$ may benefit from less compute resource constraints.

Given an input image and $\mathcal{T}$, $\mathcal{PE}$ includes three major components (*feature_ a*, *feature_ b* and *pred*), which learn how well $\mathcal{T}$ performs from internal representations extracted from different places in $\mathcal{T}$. *feature_ a* learns semantic context from low dimension features extracted from an early layer of $\mathcal{T}$; *feature_ b* concatenates the output features of *feature_ a* and internal features from a middle layer of $\mathcal{T}$ as its input; *pred* concatenates the output features of *feature_ b* and internal features from the final layer of $\mathcal{T}$ as its input to produce the performance estimation. Specifically, via the feature extraction and concatenation, $\mathcal{PE}$ learns (1)key relevant information in features learned from the early stage of $\mathcal{T}$, which has not been over biased by $\mathcal{T}$ and at the time reduces privacy risks; (2)what features $\mathcal{T}$ extracted and learned throughout its processing pipeline that lead to its final performance.

Note that $\mathcal{PE}$ does not simply detect specific input characteristics, such as the magnitude of blur or illumination. Instead, it aims at analyzing the causal relationship between patterns in the input data and the performance of the target system. Therefore, $\mathcal{PE}$ includes a set of binary classifiers to generate its output: the first classifier predicts if $\mathcal{T}$ works correctly, while the rest classifiers are error classifiers and estimate the error causes if the first classifier predicts $\mathcal{T}$'s failure. The outputs from the first classifier and the error classifiers are mutually exclusive as $\mathcal{T}$ can not be correct and wrong at the same time. On the other hand, multiple error classifiers may produce positive outputs simultaneously as multiple factors might interweave together and cause $\mathcal{T}$'s failure in the real world scenarios.

4 Augmented dataset

4.1 Iterative approach for continuous learning

For evaluating the performance of the deployed target model and determining the causes of failures, it would be ideal to have annotated data samples that faithfully reflect $\mathcal{T}$'s failures. However, it is generally very difficult, if possible at all, to collect such type of data from devices deployed on the field due to limitations such as availability, privacy concerns and legal regulations. Not to mention that the real world environment can be dynamic and failure causes may change from time to time. Also there may be a lot of factors, known and unknown, that cause the deployed target system to fail. It is infeasible to exclusively and directly collect data samples of all potential failure causes.

Therefore, one practical approach would be leveraging *PERF* framework to detect $\mathcal{T}$'s failures offline, and extract the embeddings that learn the characteristics of the data causing failures if $\mathcal{T}$ failed. Then the device can send the evaluation results and embeddings to the backend cloud. At the cloud, those collected data can be aggregated and used for further improvements for both $\mathcal{T}$ and $\mathcal{PE}$. Especially for $\mathcal{PE}$'s development, as it is very challenging to collect data from the field at a large scale due to many constraints discussed in Section 1, one viable approach is to create a synthetic dataset for the training purpose by using these aggregated embeddings collected from the devices that capture the characteristics of failure causes. This approach has been partially explored by the work in [17], where synthetic images can be generated from a customized GAN [6] to match patterns extracted from provided sketches. This datatset creation method may become an iterative approach for updating $\mathcal{T}$ and $\mathcal{PE}$ whenever needed.

To the best of our knowledge, currently we do not have devices deployed on the field that capture embeddings relevant to the failure causes. Hence, to start with, we created an augmented dataset for developing the $\mathcal{PE}$ model. We envision that after validating the *PERF* approach with the starting synthetic dataset, $\mathcal{PE}$ can be shipped together with $\mathcal{T}$ on devices and then continuously provide feedback and data on the $\mathcal{T}$'s performance to the backend. As a result, an updated synthetic dataset as aforementioned can be created for future improvements and system updates.

4.2 Augmented dataset creation

For the face directness prediction task, we created augmented datasets from a real world dataset *ETH-XGaze* [19] by applying augmentation operations to approximate common causes of failures. *ETH-XGaze* dataset includes face crop images of size 224×224 with a large range of head poses. We selected images from 11 random participants in *ETH-XGaze* training set as the base to generate our augmented datasets. Our augmented training set was generated from 7 participants with 69714 images, while both the validation set and the test set used 2 participants' images with 15426 and 17226 images.

The *ETH-XGaze* dataset provides head pose annotations in the form of (*pitch, yaw*) in radian. Therefore, face directness labels can be generated as **facing** if $\cos(\text{yaw}) \cdot \cos(\text{pitch}) > \theta \cdot \pi/180$; otherwise **not-facing**. Here, θ is an angular threshold for determining face directness, i.e., facing or not facing. We set it to 30 degree when creating the augmented datasets.

We selected four representative failure causes (i.e., *localization error*, *occlusion error*, *blur error* and *light error*) and applied data augmentation operations to approximate their effects on images from *ETH-XGaze*. The operations we used to stimulate these commonly observed errors are described below:

1. *Localization error*: to stimulate localization error from a face detector, we randomly changed the size and position of the bounding box of a face in the image within a predefined range.
2. *Occlusion error*: we collected 15 images of occlusion objects, which include mugs, glasses, masks and hands. We randomly picked one occlusion object, scaled it to a proper size and then composite onto the original image to cover the corresponding part of the face. For example, mugs and hands cover the mouth area; glasses cover eyes area; and masks cover the nose and eyes area. The exact position of the occlusion object is determined by the face landmarks from a face mesh estimator.
3. *Blur error*: we used Albumentations library [4] to apply a motion blur operation with a blur convolution kernel to the image.
4. *Lighting error*: we randomly changed the brightness and contrast of an image to simulate different lighting conditions.

The annotations of our augmented dataset include the face directness label and error causes applied on the images. To create a dataset that effectively simulates errors occurring in real environments, we saved 20% original images untouched and applied two operations randomly selected from four aforementioned augmentation operations on each of the rest images. We name this dataset *2-cause dataset*. The purpose of applying multiple augmentations to one image is to approximate the complicated real world environment where multiple factors may coexist and collectively cause the target system to fail. For ablation study purposes, we also created another simpler *1-cause dataset*, where only one randomly chosen augmentation operation was applied to 80% of the images.

5 Evaluations

5.1 Target system evaluation

Following the system architecture described in Figure 2, we constructed a face directness estimator by training a head pose estimation model on the original *ETH-XGaze* training dataset, and then applying a postprocessing step to determine the face directness from the predicted head pose vector as described in Section 4.2. The head pose estimator uses a MobileNetV2 (with width multiplier 1.0) [16] as its backbone and replaces the original top layer with a fully connected layer for the head pose estimation purpose.

Note that Figure 2 shows a cascaded target system, which includes a face detector and a face directness estimator. In our experiments, since the images from *ETH-XGaze* are cropped faces already, we skipped the face detector. And the localization error introduced by augmentation operations can still represent one of the potential causes of the face detector failure in the wild.

We compared $\mathcal{T}$'s performance on the original unmodified images as well as on our augmented datasets in Table 1. It shows that $\mathcal{T}$ performs very well on the non-augmented dataset with 98.84% average accuracy. On the *1-cause* dataset, the average accuracy drops to 85.56%. The accuracy drops even further to 80.73% on the *2-cause* dataset as it introduces more complicated perturbations.

Dataset	Non-aug	1-cause	2-cause
Ave accuracy	98.84%	85.56%	80.73%

Table 1. The average accuracy of $\mathcal{T}$ on the non-augmented dataset and two augmented datasets for the face directness detection task. The accuracy on non-augmented is very high (98.84%). It drops to 85.56% on the *1-cause* dataset, and drops even further to 80.73% on the *2-cause* dataset as it introduces more complicated perturbations.

$\mathcal{PE}$	**$\mathcal{T}$'s Egress Layer**	**Layer**	**Layer params**	**Input**	**Output**
Feature_a	17th	Conv2D	7×7, 64, stride 2	112×112	56×56
		Residual Block	$[3 \times 3, 64]$x2	56×56	26×28
Feature_b	57th	Conv2D	3×3, 256, stride 1	28×28	28×28
		Residual Block	$[3 \times 3, 256] \times 2$	28×28	4×4
			$[3 \times 3, 512] \times 2$		
			$[3 \times 3, 1024] \times 2$		
Pred	Final	Conv2D	$[3 \times 3, 128]$	3	131
		Ave pool	pool size 2		

Table 2. An example $\mathcal{PE}$ model architecture used for the face directness detection task. The *feature_ a* module includes 2 residual blocks and extracts features from the an early layer (17th) of $\mathcal{T}$. The *feature_ b* module includes 6 residual blocks and combines the features learn by *feature_ a* and the features extracted from a node of in the middle of $\mathcal{T}$ as its input. The *pred* module learns from both *feature_ b*'s output and $\mathcal{T}$'s output to predict the performance of $\mathcal{T}$ and its potential failure causes.

5.2 Performance estimation model evaluation

We adopted a ResNet-18 [8] based architecture, where both *feature_ a* and *feature_ b* modules use several residual blocks depending on where to extract features from $\mathcal{T}$. Similar to ResNet-18, after every 2 residual blocks, we downsized features by 2. In the classification layer, we used 6 binary classifiers: one classifier is to predict $\mathcal{T}$'s correctness; four classifiers are for predictions of aforementioned four error causes; and the last classifier is for other failure causes which are not introduced by augmentation operations. Overall, the computation and memory overhead of $\mathcal{PE}$ is similar to ResNet-18.

As shown in Figure 3, *feature_ a* and *feature_ b* modules extract features from $\mathcal{T}$. Theoretically, they may extract features from any layers in $\mathcal{T}$. The choice of the extracted internal node may lead to different $\mathcal{PE}$ model architecture and bring impacts on the $\mathcal{PE}$ model performance. Table 2 shows the architecture details of an example $\mathcal{PE}$ model, which extracts features from the 17th and 57th layer of the target model as the inputs for *feature_ a* and *feature_ b* components. In this model, *feature_ a* consists of 2 residual blocks, and *feature_ b* consists of 6 residual blocks. Worth to note that unlike $\mathcal{T}$, which may have limited design choices due to the strict runtime constraints on the latency and memory consumption, the architecture design of the $\mathcal{PE}$ model has more freedom and hence possibly better performance as it typically runs when the device is idle and hence does not have the same constraints as $\mathcal{T}$.

To train the PE model in Table 2, we used a SGD optimizer with momentum 0.9 and weight_decay 0.0004. We adopted a multi-step learning rate decay strategy with initial learning rate 0.1 and decaying it by 0.1 after every 10 epochs. We trained 30 epochs with batch size 64.

We first evaluated the $\mathcal{PE}$ model described in Table 2 and trained on *2-cause* augmented dataset. Table 3 shows the performance of this $\mathcal{PE}$ model. Overall, it performs well on the test set of *2-cause* dataset, and it can predicate if the target model fails with an accuracy of 95.01%. The True Negative Rate (TNR) is 98.18%, which shows that the $\mathcal{PE}$ model predicts correctly most of the time when the target model $\mathcal{T}$ predicts correctly. When $\mathcal{T}$ fails, $\mathcal{PE}$ predicts the failure correctly with a True Positive Rate (TPR) 82.41%. Note that when $\mathcal{T}$ fails, a correct prediction of the $\mathcal{PE}$ model requires that the $\mathcal{PE}$ model predicts both the failure and the causes of the failure correctly. We further analyzed how the $\mathcal{PE}$ model performs on identifying the causes of the failures. The breakdown analysis on the failure causes in Table 3 shows that for the known causes which were introduced to the augmented dataset, the $\mathcal{PE}$ model is able to identify the failure causes with a high accuracy (varying from 89.71% to 92.82%). For unknown causes (i.e., *Other causes*), the accuracy drops to 80%.

Accuracy	**TNR**	**TPR**
95.01%	98.18%	82.41%
Breakdown accuracy for cause of failure		
Localization		89.71%
Occlusion		91.25%
Blur		92.82%
Lighting		91.71%
Other causes		80%

Table 3. Performance analysis of a $\mathcal{PE}$ modeltrained and tested on the *ETH-XGaze 2-cause* dataset for the face directness detection task. The overall accuracy of predicting if the target model predicts correctly is high (95.01%) on the test set. The breakdown analysis shows that the $\mathcal{PE}$ model can predict the failure causes with a high accuracy (varying from 89.71% to 92.82%) for the known causes introduced by augmentation operations. For unknown causes (i.e., *Other causes*), the accuracy drops to 80%.

5.3 Ablation study

We created two augmented datasets with different complexities. A cross-dataset comparison was conducted to understand how models generalize to different scenarios. As shown in Table 4, we trained two models with the same architecture as described in Table 2 on both *1-cause* and *2-cause* datasets. Both models perform well on the test set that corresponds to their training set as shown in the second row and the last row of Table 4 for known causes (i.e., localization, occlusion, blur and lighting). However, the model trained on the *1-cause* dataset

Training set	Test set	Localization	Occlusion	Blur	Lighting	Others
1-cause	1-cause	99.87%	99.73%	93.65%	98.39%	61.54%
	2-cause	68.73%	59.33%	55.02%	50.59%	40%
2-cause	1-cause	96.6%	96.22%	98.99%	96.78%	69.23%
	2-cause	89.71%	91.25%	92.82%	91.71%	80%

Table 4. Performance comparison of two $\mathcal{PE}$ models trained on *1-cause* and *2-cause* augmented *ETH-XGaze* dataset. Both models perform well on the test set that corresponds to their training set for known causes. However, the model trained on the *1-cause* dataset does not perform well on the *2-cause* test set. On the other hand, the model trained on the *2-cause* has consistent performance on both test sets. This observation indicates that models trained on a dataset with more complex perturbations mixed together generalize well on a simpler dataset, where only a single type of perturbation is applied to the images.

Feature_a input	Feature_b input	Overall Accuracy	TNR	TPR
Image	57th layer	95.41%	98.55%	82.9%
17th layer	57th layer	95.01%	98.18%	82.41%
57th layer	119th layer	94.56%	97.86%	81.46%

Table 5. Performance comparison of different feature extraction choices on *2-cause* dataset in face directness detection task. We compared three choices for the inputs of *feature_ a* and *feature_ b* modules. In the first choice, *feature_ a* takes the original image in the dataset as its input, while *feature_ b* extracts the features generated by the 57th layer of $\mathcal{T}$. In the second choice, the inputs of *feature_ a* and *feature_ b* are extracted from $\mathcal{T}$'s 17th layer and 57th layer respectively. In the last choice, the inputs of *feature_ a* and *feature_ b* are taken from the later part of $\mathcal{T}$, i.e., 57th and 119th layer. The results show that these three design choices have similar performance, though the first choice is the best and the last choice is the worst. This indicts that without sacrificing privacy, our design may still achieve good performance.

does not perform well on the *2-cause* test set as shown by the third row in Table 4. On the other hand, the model trained on the *2-cause* has consistent performance on both test sets, where the accuracy for known causes are all above 89%. Other models trained on these two datasets with different $\mathcal{PE}$ model architectures also show a similar pattern. This observation indicates that models trained on a dataset with more complex perturbations mixed together generalize well on a simpler dataset, where only a single type of perturbation is applied to one image. This implies that by aggregating features collected from deployed edge devices and using them to create a complex augmented dataset, models trained with this type of datasets may generalize reasonably well on both complex and simple scenarios.

With the flexible *PERF* framework, there may be different design choices, e.g., choices for the positions to extract features from the target system $\mathcal{T}$. Here we explored three choices for the inputs of *feature_ a* and *feature_ b* modules. In the first experiment, *feature_ a* takes the original image in the dataset as its input, while *feature_ b* extracts the embeddings generated by the 57th layer of

$\mathcal{T}$. This choice may violate privacy considerations. But it is useful to understand if using the raw user data would bring benefits. In the second choice, the inputs of *feature_a* and *feature_b* are extracted from $\mathcal{T}$'s 17th layer and 57th layer respectively. In the last choice, the inputs of *feature_a* and *feature_b* are taken from the later part of $\mathcal{T}$, i.e., 57th and 119th layer. The results in Table 5 show that these three design choices have similar performance, with the first choice being slightly better and the last choice being the worst. This indicates that without using the raw user data and sacrificing user privacy, our design may still achieve good performance.

6 PERF on an image classification task

Accuracy	**TNR**	**TPR**
75.18%	62.84%	79.25%
Breakdown accuracy for cause of failure		
Localization		88.65%
Blur		99.23%
Lighting		94.99%
Other causes		98.37%

Table 6. Performance of a $\mathcal{PE}$ model trained and tested on the augmented *CIFAR-10 2-cause* dataset for an image classification task, where $\mathcal{T}$ is a 17 layers baseline model. The overall accuracy for predicting $\mathcal{T}$'s performance is reasonably well. $\mathcal{PE}$ is able to estimate the error causes with high accuracy from 88.95% to 99.23% for all three introduced causes as well as for unknown causes. This shows that *PERF*framework is a general approach that may be applied to various scenarios.

To demonstrate the flexibility and generalizability of *PERF*framework, we also validated it on an image classification task using *CIFAR-10* dataset [13]. We used the same approach as discussed in Section 4.2 to create an augmented *2-cause* dataset from *CIFAR-10*. The only difference is that we did not introduce occlusion due to the small image size in *CIFAR-10*, which is only 32×32. We trained a simple 17 layers baseline model $\mathcal{T}$ (adapted from [15]) on the original *CIFAR-10* dataset to classify 10 image classes. This baseline $\mathcal{T}$ model has an accuracy of 78.78% on the test set of the clean dataset and a poor accuracy of 22.09% on the test set of the augmented dataset. We trained a simplified $\mathcal{PE}$ model which learns from the output of $\mathcal{T}$'s first layer and $\mathcal{T}$'s output. As shown in Table 6, this $\mathcal{PE}$'s overall accuracy for predicting $\mathcal{T}$'s performance is reasonably well (75.18%), with better TPR than TNR. $\mathcal{PE}$ model is able to estimate the error causes with high accuracy from 88.95% to 99.23% for all three introduced failure causes as well as for unknown causes. This shows that *PERF*framework is a general approach that may be applied to machine learning systems of different complexity in various scenarios.

7 Conclusions

In this work, we propose a generic performance estimation framework *PERF* to estimate the performance of a machine learning system deployed on the edge and identify the root cause if the system fails. Within *PERF* framework, a performance estimation model leverages both the intermediate features and the output from the target system, and analyzes hidden relations between them. By learning these relations, *PERF* can identify major factors in the input space that affect the target model's performance. We evaluated *PERF* on a face directness detection task and an image classification using *ETH-XGaze* and *CIFAR-10* datasets. The results are promising that *PERF* may achieve good performance on both predicting the target model's performance and identifying failure causes. One limitation of the current work is the predefined causes used to generate the augmented datasets. Therefore, in the future we would like to explore a combination of generative models and unsupervised approaches to augment the training dataset so as to cover more general scenarios.

References

1. Abdallah, A., Maarof, M.A., Zainal, A.: Fraud detection system: A survey. Journal of Network and Computer Applications **68**, 90–113 (2016) 1
2. Amazon: Build with amazon's newest devices and services. https://developer.amazon.com/en-US/alexa/devices/connected-devices, accessed: 2023-03-08 1
3. Bolya, D., Foley, S., Hays, J., Hoffman, J.: TIDE: a general toolbox for identifying object detection errors. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16. pp. 558–573. Springer (2020) 3
4. Buslaev, A., Iglovikov, V.I., Khvedchenya, E., Parinov, A., Druzhinin, M., Kalinin, A.A.: Albumentations: Fast and flexible image augmentations. Information **11**(2) (2020) 8
5. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) **41**(3), 1–58 (2009) 3, 4
6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Communications of the ACM **63**(11), 139–144 (2020) 7
7. Google: Introducing the new google home. https://home.google.com/the-latest/, accessed: 2023-03-08 1
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016) 9
9. Hinton, G., Vinyals, O., Dean, J., et al.: Distilling the knowledge in a neural network. stat **1050**, 9 (2015) 4
10. Hossain, M.Z., Sohel, F., Shiratuddin, M.F., Laga, H.: A comprehensive survey of deep learning for image captioning. ACM Computing Surveys (CsUR) **51**(6), 1–36 (2019) 3
11. Khaziev, R., Shahid, U., Röding, T., Chada, R., Kapanci, E., Natarajan, P.: FPI: Failure point isolation in large-scale conversational assistants. In: Proceedings of

the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track. pp. 141–148 (2022) 3

12. Kiran, B.R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A.A., Yogamani, S., Pérez, P.: Deep reinforcement learning for autonomous driving: A survey. IEEE Transactions on Intelligent Transportation Systems **23**(6), 4909–4926 (2021) 1
13. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009) 12
14. Levinboim, T., Thapliyal, A.V., Sharma, P., Soricut, R.: Quality estimation for image captions based on large-scale human evaluations. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies pp. 3157–3166 (2021) 3
15. Rahman, F.: CIFAR-10 object classification model. https://www.kaggle.com/code/faizanurrahmann/cifar-10-object-classification-best-model, accessed: 2023-03-08 12
16. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: MobileNetV2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4510–4520 (2018) 8
17. Wang, S.Y., Bau, D., Zhu, J.Y.: Sketch your own gan. In: Proceedings of the IEEE International Conference on Computer Vision (2021) 7
18. You, Q., Jin, H., Wang, Z., Fang, C., Luo, J.: Image captioning with semantic attention. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4651–4659 (2016) 3
19. Zhang, X., Park, S., Beeler, T., Bradley, D., Tang, S., Hilliges, O.: ETH-XGaze: A large scale dataset for gaze estimation under extreme head pose and gaze variation. In: Computer Vision–ECCV 2020: 16th European Conference. pp. 365–381. Springer (2020) 7