A Scene-Aware Model Adaptation Scheme for Cross-Scene Online Inference on Mobile Devices

Yunzhe Li, *Student Member, IEEE*, Hongzi Zhu, *Senior Member, IEEE*, Zhuohong Deng, Yunlong Cheng, *Student Member, IEEE*, Zimu Zheng, *Member, IEEE*, Liang Zhang, *Student Member, IEEE*, Shan Chang, *Member, IEEE*, and Minyi Guo, *Fellow, IEEE*

***Abstract*—Emerging Artificial Intelligence of Things (AIoT) applications desire online prediction using deep neural network (DNN) models on mobile devices. However, due to the movement of devices, *unfamiliar* test samples constantly appear, significantly affecting the prediction accuracy of a pre-trained DNN. In addition, unstable network connection calls for local model inference. In this paper, we propose a light-weight scheme, called *Anole*, to cope with the local DNN model inference on mobile devices. The core idea of Anole is to first establish an army of compact DNN models, and then adaptively select the model fitting the current test sample best for online inference. The key is to automatically identify *model-friendly* scenes for training scene-specific DNN models. To this end, we design a weakly-supervised scene representation learning algorithm by combining both human heuristics and feature similarity in separating scenes. Moreover, we further train a model classifier to predict the best-fit scene-specific DNN model for each test sample. We implement Anole on different types of mobile devices and conduct extensive trace-driven and real-world experiments based on unmannedaerial vehicles (UAVs). The results demonstrate that Anole outwits the method of using a versatile large DNN in terms of prediction accuracy (4.5% higher), response time (33.1% faster) and power consumption (45.1% lower).**

***Index Terms*—Model inference, online algorithms, mobile devices, cross-scene, out of distribution, reliability.**

I. Introduction

MOTIVATION. Last decade has witnessed the booming development of Artificial Intelligence of Things (AIoT), an emerging computing paradigm that marries artificial intelligence (AI) and Internet of Things (IoT) technologies to enable independent decision-making at each component level of the interconnected system. In many AIoT scenarios, deep neural network (DNN) model inference (i.e., prediction) tasks are required to execute on mobile devices, referred to as the *online mobile inference* (OMI) problem, with stringent accuracy and latency requirements. For example, unmanned aerial vehicles (UAVs) need to constantly detect surrounding objects in real time [1]; a dash cam mounted on a vehicle needs to perform continuous image object detection [2]; robots in smart factories need to detect objects in production lines in real time, interact with human workers and other robots [3] or even make production decisions [4].

To address the OMI problem, however, is demanding for two reasons as follows. First, given that mobile devices constantly experience scene changes while moving (e.g., due to various lighting conditions, weather conditions, and viewing angles), the output of DNNs should remain reliable and accurate. Training a statistical learning DNN on a given dataset, as in normal deep learning paradigm, becomes difficult to guarantee the robustness, interpretability and correctness of the output of the statistical learning models when data samples are *out-of-distribution* (OOD) [5]. Second, the response time for model inference should satisfy a rigid delay budget to support real-time interactions with these devices. As mobile devices are resource-constrained in terms of computation, storage and energy, they cannot handle large DNNs. Though it would be beneficial to offload a part of or even entire inference tasks to a remote cloud, unstable communication between mobile devices and the cloud may lead to unpredictable delay.

In the literature, much effort has been made to improve DNN model inference accuracy on mobile devices but in static scenarios. One main branch aims to develop DNNs specially designed for mobile devices [6], [7], [8], [9] or to compress (e.g., via model pruning and quantization) existing DNNs to match the computing capability of a mobile device [10], [11]. Such schemes ensure real-time model inference at the expense of compromised accuracy, especially when dealing with OOD data samples. Moreover, even without considering the limited resources of mobile devices, it is also difficult to find a satisfactory neural network capable of stable inference on OOD samples for some complex tasks such as mobile decision making [3], [12]. Another branch is to divide DNNs and perform collaborative inference on both edge devices and the cloud [13], [14], [15], or to transmit compressed sensory data to the cloud for data recovery and model inference [16], [17]. These approaches need coordination with the cloud for each inference, leading to unpredictable inference delays when the communication link is unstable or disconnected. As a result, to the best of our knowledge, there is no successful solution to the OMI problem yet.

Received 2 August 2024; revised 5 March 2025; accepted 21 May 2025. Date of publication 28 May 2025; date of current version 3 September 2025. This work was supported in part by the Natural Science Foundation of China under Grant 62432008 and Grant 62472083, and in part by Natural Science Foundation of Shanghai under Grant 22ZR1400200. Recommended for acceptance by A. Taherkordi. *(Corresponding author: Hongzi Zhu.)*

Yunzhe Li, Hongzi Zhu, Zhuohong Deng, Yunlong Cheng, Liang Zhang, and Minyi Guo are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: hongzi@sjtu.edu.cn).

Zimu Zheng is with Huawei Cloud, Shenzhen 518116, China.

Shan Chang is with the School of Computer Science and Technology, Donghua University, Shanghai 201620, China.

Digital Object Identifier 10.1109/TMC.2025.3574766

Our approach: We propose *Anole*, which enables online model inference on mobile devices in dynamic scenes. We have the insight that a compressed DNN targeted for a particular scene (i.e., data distribution) can achieve comparable inference accuracy provided by a fully-fledged large DNN. The core idea of Anole is to first establish a colony of compressed scene-specific DNNs, and then adaptively select the model best suiting the current test sample for online inference. To this end, it is essential to identify scenes from the perspective of DNN models. We design a weakly-supervised scene representation learning scheme by combining both human heuristics and feature similarity in separating scenes. After that, for each identified scene, an individual compressed DNN model can be trained. Furthermore, we train a model classifier to predict the best-fit compressed DNN models for use during online inference. As a result, compelling prediction accuracy can be achieved on mobile devices by actively recruiting most capable compressed models, without any intervention with the cloud.

Challenges and contributions: The Anole design faces three main challenges. First, how to obtain model-friendly scenes and train scene-specific DNNs from public datasets is unclear, as the distribution that a DNN model can characterize is implicit. One naive way is to use semantic attributes (e.g., time, location, weather and light conditions) of data to define scenes of similar data samples. However, as shown in our empirical study, DNNs trained on such scenes cannot reach satisfactory prediction accuracy even on their respective training scenes. To tackle this challenge, we design a scene representation learning algorithm that combines semantic similarity and feature similarity of data to filter out scenes. Specifically, human heuristic is first used to define scenes of similar semantic attribute values, referred to as *semantic scenes*. Then, a scene representation model, denoted as $\mathcal{M}_{scene}$, is trained using the indices of semantic scenes as labels. After that, we can obtain embeddings of all data samples extracted with $\mathcal{M}_{scene}$ and believe such embeddings can well characterize semantic information. Therefore, by conducting multi-granularity clustering on these embeddings, we can obtain clusters of data samples with similar semantic information in feature space, referred to as *model-friendly scenes*. Finally, a compressed DNN can be trained on each model-friendly scene, constituting a model repository for use.

Second, given a test sample, how to determine the best-fit models or whether such models even exist in the model repository is hard to tell. To deal with this challenge, we train a model classifier, denoted as $\mathcal{M}_{decision}$, to predict the best model for use. Specifically, for each model-friendly scene, we select those data samples in the scene that can be accurately predicted by the corresponding DNN and use the index of the DNN as the label to train $\mathcal{M}_{decision}$. Instead of testing all data samples, we use Thompson sampling to establish balanced training sets at a low computational cost. With a well-trained $\mathcal{M}_{decision}$, the most suitable compressed models can be predicted and the prediction confidence can be used to indicate whether such models exist.

Last, how to deploy those pre-trained compressed DNNs on mobile devices with constrained memory is non-trivial. We have the observation that the utility of models follows a power-law distribution over all test videos. This implies that it is feasible to cache a small number of most frequently used compressed models and take a least frequently used (LFU) model replacement strategy.

We implement Anole on three typical mobile devices, i.e., Jetson Nano, Jetson TX2 NX and a laptop, with each equipped with a CPU/MCU and an entry-level GPU, to conduct the image object detection task on moving vehicles. Specifically, we train the $\mathcal{M}_{scene}$ based on Resnet18 [18], a pack of 19 compressed DNNs based on YOLOv3-tiny [19], and the $\mathcal{M}_{decision}$ based on Resnet18 accordingly, using three driving video datasets collected from multiple cities in different counties. We conduct extensive trace-driven and real-world experiments using UAVs. Results demonstrate that Anole is lightweight and agile to switch best models with low latencies of 61.0 ms, 13.9 ms, and 52.0 ms on Jetson Nano, Jetson TX2 NX, and the laptop, respectively. In cross-scene (i.e., seen but fast-changing scenes) setting, Anole can achieve a high F1 prediction accuracy of 56.4% whereas the F1 score of a general large DNN model and a general compact DNN are 50.7% and 45.9%, respectively. In hard new-scene (i.e., unseen scenes) setting, Anole can maintain a high F1 score of 48.7% whereas the F1 score of the general large DNN and the general compact DNN drops to 46.6% and 41.1%, respectively.

We highlight the main contributions made in this paper as follows:

- A new solution to the OMI problem by recruiting a pack of compact but specialized models on resource-constrained mobile devices, without any intervention with the cloud during online model inference;
- A scene partition method that effectively facilitates the training of specialized models by leveraging both semantic and feature similarity of the data;
- We have implemented Anole on typical mobile devices and conducted extensive trace-driven and real-world experiments on 3 typical tasks, the results of which demonstrate the efficacy of Anole.

II. Problem Definition

A. System Model

We consider three types of entities in the system:

- *Mobile devices:* Mobile devices have constrained computational power and a limited amount of memory but are affordable for running and storing compressed DNNs. Such devices may be moving while performing online inference tasks at the same time. They are battery-powered, desiring lightweight operations. In addition, they can communicate with a cloud server via an unstable wireless network connection for offline model training and downloading.
- *Cloud server:* A cloud server has sufficient computational power and storage for offline model training. During online inference, the cloud server is not involved.
- *Complex environment*: We consider practical environments where background objects and light conditions have distinct spatial and temporal distributions. When mobile devices move in such a complex environment, they constantly experience fast scene changes.

TABLE I
NOTATIONS

Acronym	Description
C_j^k	j-th Cluster over H_i with clustering number k
D	All available labeled data
$\mathcal{M}_0$	Compressed model trained using all available data
$\mathcal{M}_i$	Compressed DNN model with index i
$\mathcal{M}_{big}$	Big model
$\mathcal{M}_{scene}$	Model for scene representation
$\mathcal{M}_{decision}$	End-to-end decision model for model selection
$\mathcal{M}_{test}$	Model selected for inference on x_{test}
M	A set of trained models
M^*	An optimal set of M
H_i	Embedding of i-th sample using $\mathcal{M}_{scene}$
U	Universal set of all possible data
x	Data sample in dataset D
x_{test}	Test sample during online inference
Γ_i	Subset of D with index i
Γ_i^{sem}	i-th scene partitioned based on semantic attributes
Ψ_i	Data distribution that $\mathcal{M}_i$ can characterize
Ψ_i^{sub}	Balanced subset of Ψ_i

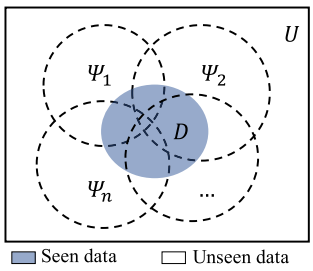


Fig. 1. Illustration of the online mobile inference problem, where data distributions characterized by statistical models (depicted as dashed disks) are implicit and not easy to understand.

B. Problem Formulation

Given the set of all available labeled data, denoted as D, a compressed DNN model, denoted as $\mathcal{M}_i$, can be trained on a particular dataset, denoted as Γ_i, which is a subset of D, i.e., $\Gamma_i \subseteq D$ for $i \in \mathbb{N}$. For instance, Γ_i can be established based on some semantic attributes of data. Table I lists all notations defined in this work.

Assume that a set of n models $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$ have been pre-trained on respective training datasets $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$ and the *implicit* data distributions that those models can characterize are $\{\Psi_1, \Psi_2, \ldots, \Psi_n\}$, respectively, which means that if a data sample $x \in \Psi_i$ for $i \in [1, n]$, model $\mathcal{M}_i$ guarantees to output accurate prediction for x. We have the following proposition:

Proposition 1. Though $\mathcal{M}_i$ is trained on Γ_i, not all data samples in Γ_i necessarily belong to Ψ_i, i.e., $\Gamma_i \not\subset \Psi_i$.

As illustrated in Fig. 1, given D, we can train such a set of n models $M = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$ so that $D \subset \bigcup_{i=1}^{n} \Psi_i$. As in mobile settings, any data sample $x \in U$ can be encountered where U is the universal set of all possible data, the online mobile inference problem is to identify an optimal subset of M, denoted as M^*, that maximize the prediction accuracy for x. The problem can be discussed in the following three cases of different difficulties: 1) $x \in D$: in this case, M^* is known since x is seen before, i.e., $M^* = \{\mathcal{M}_i : x \in \Psi_i, i \in [1, n]\}$; 2) $x \notin D$ and $x \in \bigcup_{i=1}^{n} \Psi_i$: in this case, x is not seen before and $M^* = \{\mathcal{M}_i : x \in \Psi_i, i \in [1, n]\}$ exists but how to find the M^* is hard; 3) $x \in U - \bigcup_{i=1}^{n} \Psi_i$: in this case, as x is not seen before and M^* does not exist regarding existing M, how to make best-effort online prediction for x is challenging. A remedy for this case is to train new models to deal with x and the like in the future.

The main difficulty of the online mobile inference problem lies in *how to determine whether an unseen x belongs to Ψ_i for $i \in [1, n]$*. According to Proposition 1, simply comparing the similarity of semantic attributes between x and Γ_i for $i \in [1, n]$ would not work. Another concern is *how to achieve the best-effort inference accuracy within a specific latency budget even if M^* does not exist*.

III. Empirical Study

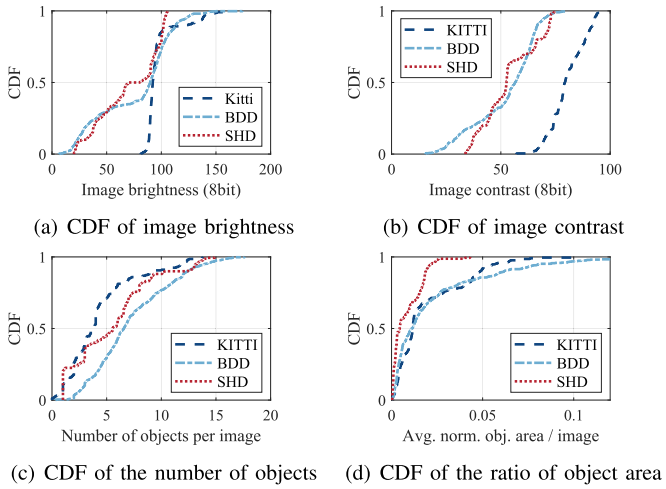
We first investigate how DNNs behave in mobile settings, taking the typical online object detection as the example task.

A. Driving Video Datasets

We select three representative driving datasets for evaluation. Video clips in these datasets are recorded from multiple vehicles moving at various speeds in different cities, experiencing different light and weather conditions, which makes them ideal for systematic evaluation. Specifically, these datasets are as follows:

- KITTI [20]: comprises 389 stereo and optical flow image pairs, stereo visual odometry sequences of 39.2 km length, and more than 200 k 3D object annotations captured in cluttered scenarios (up to 15 cars and 30 pedestrians are visible per image). For online object detection, KITTI consists of 21 training sequences and 29 test sequences.
- BDD100k [21]: contains over 100 k video clips regarding ten autonomous driving tasks. Clips of 720 p and 30 fps were collected from more than 50 thousand rides in New York city and San Francisco Bay Area, USA. Each clip lasts for 40 seconds and is associated with semantic attributes such as the scene type (e.g., city, streets, residential areas, and highways), weather condition and the time of the day.
- SHD: contains 100 driving video clips of one minute recorded in March 2022 with a 1080 p dashcam in Shanghai city, China. Clips were collected from ten typical scenarios, including highway, typical surface roads, and tunnels, at different time in the day. LabelImg [22] is employed to label objects in all images.

We random select 10 video clips from KITTI, 44 clips from BDD100 k, and 10 clips from SHD, forming a dataset of 64 video clips containing 16,145 image samples in various scenarios. Fig. 2 shows the cumulative distribution functions (CDFs) about foreground objects and illumination condition over all frames in the dataset, demonstrating diverse driving scenarios. We partition these 64 clips into seen (i.e., involved in model training) and unseen (i.e., not used in model training) categories with a ratio of 9:1. For each seen clip, we further divide frames into training, validation, and testing image sets with a ratio of 6:2:2.



(a) CDF of image brightness (b) CDF of image contrast

(c) CDF of the number of objects (d) CDF of the ratio of object area

Fig. 2. The dataset of 64 randomly selected driving video clips demonstrates a large diversity in terms of image light conditions and foreground object distributions.

B. Mobile Inference Accuracy Analysis

We first train a big object detection model based on YOLOv3, denoted as $\mathcal{M}_{big}$, and a compressed object detection model based on YOLOv3-tiny, denoted as $\mathcal{M}_0$, using all available training data of 8,714 images in seen clips. Grid search is used to choose the hyper-parameters during model training. Particularly, $\mathcal{M}_{big}$ incurs $10\times$ computational cost for inference compared to $\mathcal{M}_0$.

Then, we further define ten distinct scenes, denoted as Γ_i for $i \in [1, 10]$, based on the associated semantic attributes of all available training samples, such as {*good weather, daytime, urban*}, and for each scene, we train a compressed object detection models, denoted as $\mathcal{M}_i$ for $i \in [1, 10]$.

Fig. 3(a) plots the boxplot of the F1 score obtained with different models on all the 58 testing sets of seen clips. It can be seen that the compressed model trained on all available data samples, denoted as $\mathcal{M}_0$, cannot compete with a full-fledged model $\mathcal{M}_{big}$. However, for each test sample, there exists a compressed model trained on one of the divided scenes, denoted as $\mathcal{M}_{best}$, achieving the best accuary among all compressed models. Moreover, the performance of $\mathcal{M}_{best}$ is even better than that of $\mathcal{M}_{big}$. As a result, we have the following observation:

Observation: Though any single compressed model generally has a lower prediction accuracy than the big model, there exists a compressed model that can achieve comparable accuracy as the big model for each specific scene.

From the above observation, it is possible to achieve appealing inference accuracy of the big model at a low cost of a compressed model, if we can identify an appropriate compressed model that best fits the current test sample. An intuitive model selection scheme is to choose the compressed model $\mathcal{M}_k$ trained on the dataset Γ_k with similar semantic attribute values to the current test sample. This idea assumes that $\Gamma_k \subset \Psi_k$. Fig. 3(b) plots the normalized F1 score for each $\mathcal{M}_k$ tested on each Γ_k. It can be seen that the highest F1 scores do not consistently appear along the diagonal, which makes the intuitive model selection scheme hard to work. Another scheme is to define scenes based on feature similarity, where data samples with similar features are clustered. Given identified scenes, compressed DNNs can be trained and the best DNN can be selected according to the feature similarity between the test samples and existing scenes. Similar model selection results can be seen in Fig. 3(c).

IV. Overview of Anole

As illustrated in Fig. 4, Anole consists of two parts, i.e., offline scene profiling and online model inference.

Offline Scene Profiling (OSP): OSP is deployed on cloud servers for offline scene partitioning and scene-specific model training, which integrates three components as follows:

1) Training Compressed Models (TCM): Given the available labelled dataset D, TCM first divides D into appropriate training datasets and train a scene representation model $\mathcal{M}_{scene}$ and a pack of n compressed models $M = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$;

2) Adaptive Scene Sampling (ASS): As $\{\Psi_1, \Psi_2, \ldots, \Psi_n\}$ are implicit, ASS is to adaptively sample $\{\Psi_1, \Psi_2, \ldots, \Psi_n\}$ based on Thompson sampling from all available dataset D to obtain balanced subsets of $\{\Psi_1, \Psi_2, \ldots, \Psi_n\}$ in D, denoted as $\{\Psi_1^{sub}, \Psi_2^{sub}, \ldots, \Psi_n^{sub}\}$, which can be used as labels for decision model training;

3) Training Decision Model (TDM): An end-to-end decision model $\mathcal{M}_{decision}$ is trained using $\{\Psi_1^{sub}, \Psi_2^{sub}, \ldots, \Psi_n^{sub}\}$, which can be used to select suitable compressed models for testing samples.

Online Model Inference (OMI): OMI is deployed on mobile devices for online model inference. Before online inference, pre-trained $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$ and $\mathcal{M}_{decision}$ need to be downloaded. The core idea of OMI is to compare testing data samples with $\{\Psi_1, \Psi_2, \ldots, \Psi_n\}$ in feature space and select the most suitable compressed models for model inference. To this end, OMI integrates two components:

1) Model Selection Strategy (MSS): During online inference, test sample, denoted as x_{test}, will be fed to the $\mathcal{M}_{decision}$, which predicts the suitability probability of $\mathcal{M}_i$ for all $i \in [1, n]$ with respect to x_{test}. These probabilities are used for ranking models.

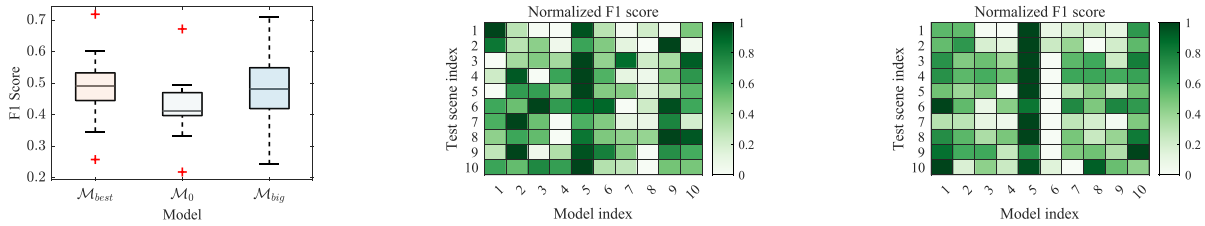
2) Cache-based Model Deployment (CMD): Given the model ranking, CMD identifies the model with the highest suitability probability in the model cache, denoted as $\mathcal{M}_{test}$, for online inference. If the model with the highest suitability probability is missed, CMD takes the LFU strategy to update models in the cache.

3) Model Inference (MI): $\mathcal{M}_{test}$ is applied to x_{test} for conducting local prediction.

V. Offline Scene Profiling

A. Training Compressed Models

1) Training Dataset Segmentation: We first define semantic scenes based on semantic attributes of data. It is non-trivial, however, to manually define appropriate scenes as semantic attributes have different dimensions and different granularities. For example, for driving images, "urban" and "daytime" are



(a) Boxplot of F1 score, where $\mathcal{M}_{best}$ denotes the best compressed models of each scene

(b) Heatmap of normalized F1 score tested on ten selected scenes defined with semantic attributes

(c) Heatmap of normalized F1 score tested on ten selected scenes defined based on feature similarity

Fig. 3. A best-fit compressed model can achieve comparable prediction accuracy provided by a large DNN, whereas compressed DNNs trained using existing scene partitioning methods fail to perform well on their corresponding training dataset.

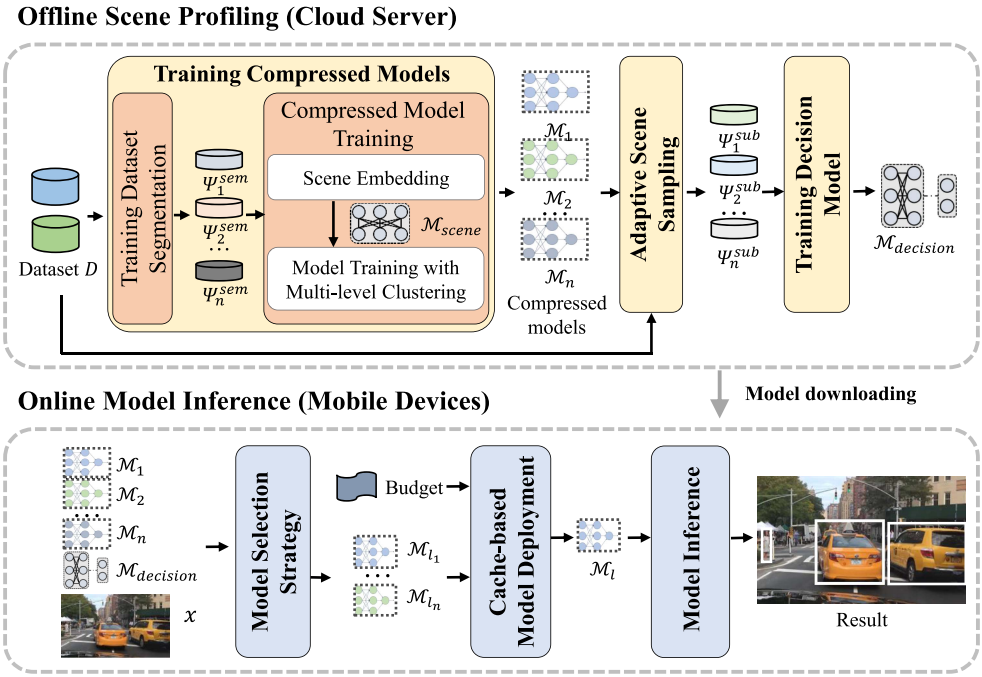


Fig. 4. System architecture of Anole, which consists of the offline scene profiling on cloud servers and the online model inference on mobile devices. Communication between both parts is carried out offline.

spatial and temporal attributes, respectively, in different dimensions; "urban" and "street" are spatial attributes but in different granularities. Scenes defined with fine-grained attributes would have insufficient number of samples to train a model whereas scenes defined with coarse-grained attributes would lose the diversity of models. Specifically, we heuristically select fine-grained attributes in each orthogonal dimension to separate data samples into m scenes, denoted as $\{\Gamma_1^{sem}, \Gamma_2^{sem}, \ldots, \Gamma_m^{sem}\}$. For instance, as for driving images, we define semantic scenes according to 120 combinations of attributes in three dimensions, i.e., {*clear*, *overcast*, *rainy*, *snowy*, *foggy*} in weather, {*highway*, *urban*, *residential*, *parking lot*, *tunnel*, *gas station*, *bridge*, *toll booth*} in location and {*daytime*, *dawn/dusk*, *night*} in time.[1] Note that annotations given in some public datasets may be insufficient for defining fine-grained scenes. For these datasets, we manually label each video clip in the datasets with required sementic attributes as it is easy to identify conditions such as weather, location type and time in a day.

2) Compressed Model Training: We employ a training strategy, integrating both semantic similarity and feature similarity of data samples to train diverse compressed models, which consists of the following two steps, as described in Algorithm 1.

Scene Embedding: Given semantic scenes $\{\Gamma_1^{sem}, \Gamma_2^{sem}, \ldots, \Gamma_m^{sem}\}$, we train a scene classifier, denoted as $\mathcal{M}_{scene}$, using samples in each Γ_i and the index of the scene as label. For each scene dataset Γ_i for $i \in [1, m]$, the hidden features on the last layer of $\mathcal{M}_{scene}$, denoted as H_i, are used as the embeddings of Γ_i.

Model Training with Multi-level Clustering: Instead of training compressed models directly from Γ_i for $i \in [1, m]$, we further consider the feature similarity of data samples by clustering embeddings in all H_i and train compressed models on obtained

[1]Note that these scenes are defined at a very fine-grained level, to the extent that they may not have enough samples to train a satisfactory model. They will be clustered further to a moderate granularity for model training.

Algorithm 1: CMT-DM Algorithm.

Input: Semantic-defined scenes Γ_i^{sem} for $i \in [1, m]$, preset number n of compressed models to be trained, threshold δ at which the model performance meets the required criteria, embeddings of all scenes H_i for $i \in [1, m]$.

Output: Compressed models specific for scenes $M^{rep} = \{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n\}$.

// Model training with multi-level clustering.

1 Compressed model repository $M^{rep} \leftarrow \{\}$;
2 clustering number $k \leftarrow 2$;
3 **while** $|M^{rep}| < n$ **do**
4 Cluster on $\{H_1, H_2, \cdots, H_m\}$ with clustering number k;
5 Train k compressed models $\mathcal{M}_j^k$ for $j \in [1, k]$;
6 **for** *each* $\mathcal{M}_j^k$ **do**
7 $p_j \leftarrow$ evaluation performance of $\mathcal{M}_j^k$ on its vaildation set;
8 **if** $p_j > \delta$ **then**
9 $\mathcal{M}_{|M^{rep}|+1} \leftarrow \mathcal{M}_j^k$;
10 M^{rep}.append($\mathcal{M}_{|M^{rep}|+1}$);
11 $k \leftarrow k + 1$;
12 **return** $M^{rep} = \{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n\}$

Algorithm 2: CMT-SM Algorithm.

Input: Semantic-defined scenes Γ_i^{sem} for $i \in [1, m]$, preset number n of compressed models to be trained, threshold δ at which the model performance meets the required criteria, embeddings of all scenes H_i for $i \in [1, m]$, initial clustering number K.

Output: Compressed models specific for scenes $M^{rep} = \{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n\}$.

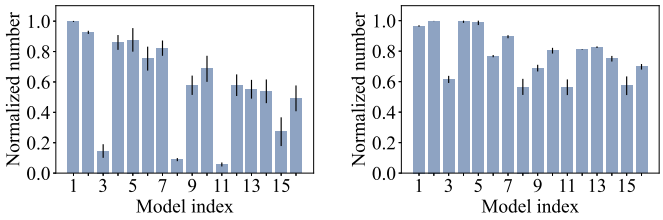
// Model training with multi-level clustering.

1 Compressed model repository $M^{rep} \leftarrow \{\}$;
2 $\boldsymbol{C} = \{C_1, C_2, \cdots, C_K\} \leftarrow$ clusters on $\{H_1, H_2, \cdots, H_m\}$ with clustering number K;
3 **while** $|M^{rep}| < n$ **do**
4 Train K compressed models $\mathcal{M}_j$ for $j \in [1, K]$ on clusters $\{C_1, C_2, \cdots, C_K\}$;
 // performance evaluation for each candidate compressed models.
5 **for** *each* $\mathcal{M}_j^k$ **do**
6 $p_j \leftarrow$ evaluation performance of $\mathcal{M}_j^k$ on its vaildation set;
7 **if** $p_j > \delta$ **then**
8 $\mathcal{M}_{|M^{rep}|+1} \leftarrow \mathcal{M}_j^k$;
9 M^{rep}.append($\mathcal{M}_{|M^{rep}|+1}$);
10 $\boldsymbol{C}^s = \{C_1^s, C_2^s, \cdots, C_K^s\} \leftarrow$ ascending sorted clusters according to their performance;
 // clusters combination.
11 **for** *each* $C_{k_1}^s$ *for* $k_1 \in [1, K]$ **do**
12 **for** *each* $C_{k_2}^s$ *for* $k_2 \in [2, K]$ **do**
13 $p_{k_1,k_2} \leftarrow$ performance of the model $\mathcal{M}_{k_1,k_2}$ trained on $C_{k_1,k_2}^s = C_{k_1}^s \cup C_{k_2}^s$;
14 **if** $p_{k_1,k_2} > p_{k_1}$ *and* $p_{k_1,k_2} > p_{k_2}$ **then**
15 $\boldsymbol{C} \leftarrow \boldsymbol{C} \cup \{C_{k_1,k_2}^s\} \setminus \{C_{k_1}^s, C_{k_2}^s\}$;
16 **else if** $p_{k_1,k_2} > p_{k_1}$ *and* $p_{k_1,k_2} \leq p_{k_2}$ **then**
17 $\boldsymbol{C} \leftarrow \boldsymbol{C} \cup \{C_{k_1,k_2}^s\} \setminus \{C_{k_1}^s\}$;
18 **else if** $p_{k_1,k_2} \leq p_{k_1}$ *and* $p_{k_1,k_2} > p_{k_2}$ **then**
19 $\boldsymbol{C} \leftarrow \boldsymbol{C} \cup \{C_{k_1,k_2}^s\} \setminus \{C_{k_2}^s\}$;
20 **return** $M^{rep} = \{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n\}$

clusters. Specifically, two versions of multi-level clustering algorithms are proposed in Anole, i.e., Compressed Model Training with Distinct Samples among Models (CMT-DM, described in Algorithm 1), and Compressed Model Training with Shared Samples among Models (CMT-SM, described in Algorithm 2).

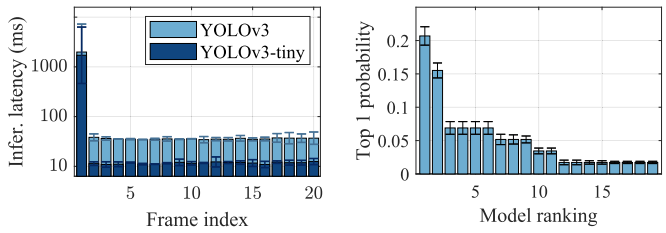
The core idea of the CMT-DM algorithm is to obtain the clusters with different levels of similarity. To this end, we conduct multiple k-means [23] clustering with k varying from 2 over embeddings in all H_i for $i \in [1, m]$. For each k, all embeddings can be divided into k clusters, denoted as C_j^k for $j \in [1, k]$ (Line 4). We train a compressed model, denoted as $\mathcal{M}_j^k$, on each clustered scene corresponding to C_j^k for $j \in [1, k]$ (Line 5) and validate its performance. If the performance of $\mathcal{M}_j^k$ exceeds a threshold δ, $\mathcal{M}_j^k$ is added to the compressed model repository (Line 6 - Line 10). This procedure repeats until a set of n compressed models $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$ are derived, where n denotes a preset number for compressed models to be trained.

The core idea of the CMT-SM algorithm is to first cluster scenes at a fine-grained level and then to try to combine the clusters with bad model training performance by pairwise combination trail. To this end, we first conduct k-means clustering with a preset clustering number K over embeddings in all H_i for $i \in [1, m]$. All embeddings can therefore be divided into K clusters, denoted as $\boldsymbol{C} = \{C_1, C_2, \ldots, C_K\}$, to get a fine-grained partitioning (Line 2). We train a model for each clusters on $\boldsymbol{C}$ and the models whose performance exceeds the threshold δ will be added to the compressed model repository (Line 5 - Line 9). Then, we sort $\boldsymbol{C} = \{C_1, C_2, \ldots, C_K\}$ in ascending validated performance order to obtain $\boldsymbol{C}^s = \{C_1^s, C_2^s, \ldots, C_K^s\}$ for further combination of clusters (Line 10). Next, we start from the two clusters with the worst training performance, i.e., C_1^s and C_2^s, attempting to combine them pairwise, and verify whether the training performance, denoted as p_{k_1,k_2} of combined cluster, denoted as $C_{k_1,k_2}^s = C_{k_1}^s \cup C_{k_2}^s$, is higher than that of C_1^s and C_2^s, denoted as p_{k_1} and p_{k_2}, respectively (Line 11 - Line 19). If p_{k_1,k_2} is larger than both p_{k_1} and p_{k_2}, C_1^s and C_2^s will be removed from the clusters set $\boldsymbol{C}$ and C_{k_1,k_2}^s will be added; If only p_{k_1,k_2} is larger than only one of p_{k_1} or p_{k_2}, only the less one (p_{k_1} or p_{k_2} which is less) will be removed from $\boldsymbol{C}$ and C_{k_1,k_2}^s will be added. This procedure repeats until a set of n compressed models are derived.



(a) Random sampling (b) Adaptive sampling

Fig. 5. (a) An example of compressed models being unevenly sampled with random sampling; (b) our adaptive sampling algorithm can mitigate the unbalanced sampling problem.



(a) Average inference latency (b) Utility of compressed models

Fig. 6. (a) Average latency of model inference on consecutive frames over all test clips; (b) the probability of being the top one model, following a long-tailed distribution.

B. Adaptive Scene Sampling

To obtain $\{\Psi_1^{sub}, \Psi_2^{sub}, \ldots, \Psi_n^{sub}\}$, a straightforward idea is to randomly pick a number of samples X from D and test $\mathcal{M}_i$ for $i \in [1, n]$. If a $\mathcal{M}_i$ can achieve satisfactory prediction accuracy on sample $x \in X$, x belongs to Ψ_i^{sub}. As $\{\Psi_1, \Psi_2, \ldots, \Psi_n\}$ may be biased in D, such random sampling algorithm also generates unbalanced $\{\Psi_1^{sub}, \Psi_2^{sub}, \ldots, \Psi_n^{sub}\}$. To solve the unbalanced sampling problem, however, is not intuitive, because of Proposition 1. Proposition 1 holds that we can not know a sample belongs to which distribution from all the distributions those models can characterize (i.e., $\{\Psi_1, \Psi_2, \ldots, \Psi_n\}$) without high computational cost experiments. In order to obtain a balanced $\{\Psi_1^{sub}, \Psi_2^{sub}, \ldots, \Psi_n^{sub}\}$ at a low computation cost, we design an adaptive sampling algorithm based on Thompson sampling [24].

Specifically, in the k-th sampling round for $k \in \mathbb{N}$, we first examine if the training set Γ_i of $\mathcal{M}_i$ for $i \in [1, n]$ has been well sampled by checking $|S_i| > \frac{\log(1-\theta^{\frac{1}{|\Gamma_i|}})}{\log(1-\frac{1}{|\Gamma_i|})}$, where S_i is the set of samples sampled from Γ_i; θ is the confidence of being well sampled; and $|\cdot|$ is the number of elements in a set.

Then, for each training set Γ_i that has not been well sampled, we estimate a sampling probability p_i^k based on a Beta distribution $Beta(\alpha_i^{k-1}, \beta_i^{k-1})$, where α_i^{k-1} and β_i^{k-1} are the two parameters of the Beta distribution of Γ_i, updated in the previous round. As a result, the training set Γ_k with the highest sampling probability will be sampled.

Finally, all $Beta(\alpha_i^k, \beta_i^k)$ will be updated as follow:

$$Beta\left(\alpha_i^k, \beta_i^k\right) = \begin{cases} Beta(\alpha_i^{k-1}+1, \beta_i^{k-1}), & \text{if } \Gamma_i \text{ is sampled;} \\ Beta(\alpha_i^{k-1}, \beta_i^{k-1}+1), & \text{otherwise.} \end{cases}$$

This procedure repeats until a specific number of κ samples are collected. Fig. 5 shows the normalized $|S_i|$ for all the $\mathcal{M}_i$ for $i \in [1, n]$ where $n = 16$, using the random sampling algorithm and our adaptive sampling algorithm, respectively. It can be seen that our adaptive sampling algorithm can effectively mitigate the unbalanced sampling problem.

C. Training Decision Model

Given the sampling results $\{\Psi_1^{sub}, \Psi_2^{sub}, \ldots, \Psi_n^{sub}\}$, we train an end-to-end decision model $\mathcal{M}_{decision}$ to effectively represent and distinguish $\{\Psi_1^{sub}, \Psi_2^{sub}, \ldots, \Psi_n^{sub}\}$ by employing a parameter-frozen scene representation network $\mathcal{M}_{scene}$ and neural-network-based classifier.

Specifically, we use $\mathcal{M}_{scene}$ as a backbone neural network to obtain scene representation, denoted as h_i^s, for every data sample $x_i \in \Psi_i^{sub}, i \in [1, n]$. In this way, h_i^s will retain the scene-related information. The model decision here can be formulated as a multi-class classification problem. The label of x for decision model training is a vector, referred to as a *model allocation vector* $v^x = \{v_i^x, i \in [1, n]\}$, where the i-th element v_i^x, denotes whether $x \in \Psi_i^{sub}$. The cross entropy loss function [25] is used for training the decision model. Note that during the training of decision model $\mathcal{M}_{decision}$, the parameter of $\mathcal{M}_{scene}$ is frozen to improve training efficiency and enhance the generalization of $\mathcal{M}_{decision}$ [26].

VI. Online Model Inference

A. Model Selection Strategy

Given the set of pretrained models $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$ and decision model $\mathcal{M}_{decision}$ downloaded from a cloud server, a mobile device needs to select most suitable compressed models for online inference. Specifically, it utilizes $\mathcal{M}_{decision}$ to output the model allocation vector v^x for a testing sample x, i.e., $v^x = \mathcal{M}_{decision}(x)$, where the i-th element v_i indicates the suitability probability that model $\mathcal{M}_i$ is suitable for x. Therefore, we can rank all compressed models according to their suitability probablities for x using v^x. It should be noted that for the uncertainty of scenario duration, model selection should be conducted on every testing sample, taking into account the fast-changing data distributions in the perspective of compressed models.

B. Cache-Based Model Deployment

With the model allocation vector $v^x = \mathcal{M}_{decision}(x)$, compressed models can be dynamically ranked. Due to the restricted amount of memory on a mobile device, not all models may be pre-loaded into memory. To deal with this issue, we investigate the best-effort model deployment strategy.

We examine the inference latency of detecting objects on five driving video clips, using two DNN models of different size, i.e., YOLOv3 (237 MB) and YOLOv3-tiny (33.8 MB), on a Nvidia Jetson TX2 NX (ARM A57 CPU, Nvidia Pascal GPU with 4 GB memory, 32 GB flash). Fig. 6(a) plots the average inference latency of the first twenty frames over all clips. For both models,

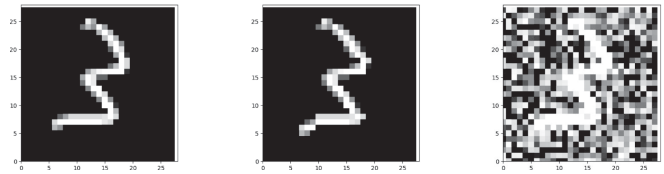


Fig. 7. Example images in the original and two constructed MNIST datasets.

TABLE II
ANOLE IS IMPLEMENTED ON THREE DIFFERENT TYPES OF MOBILE DEVICES WITH DISTINCT HARDWARE CONFIGURATIONS

Platform	CPU	GPU	GPU Mem.	Flash/Disk
Jetson Nano	ARM A57	Maxwell	2GB	32GB
Jetson TX2	ARM A57	Pascal	4GB	32GB
Laptop	i7-10750H	RTX 2070	8GB	1T

a huge delay occurs when processing the first frame. This is mainly attributed to the I/O operation for model loading and other initialization required by the deep learning framework such as Pytorch. Therefore, it is preferred to preload as many models as possible.

Given a limited video memory budget, it is tricky to pre-load best models in memory. We examine the utility of 19 YOLOv3-tiny compressed models obtained according to the algorithm stated in Section V-A (see Section VII-A2 for more details) when conducting object detection on the five driving video clips. Fig. 6(b) depicts the ratio of being the top one model over all clips for all compressed models. It can be seen that the probability of being the best model follows a power-law distribution. This observation suggests that high-level inference performance can be sustained by deploying only a small number of supreme models. Inspired by this observation, we adopt a Least Frequently Used (LFU) strategy [27] to update models in GPU memory. In the occasion of a model miss, the model with the highest suitability probability in GPU memroy will be used for inference.

VII. EVALUATION

A. Methodology

1) Tasks and Datasets: We evaluate Anole on two mobile inference tasks, i.e., handwritten digit recognition (HDR), coke grades prediction on mobile agents (CGP) and vehicle detection on driving videos (VD), based on the following datasets and real-world experiments.

- *Handwritten Digit Datasets*: The MNIST dataset [28] contains 70,000 black and white images of handwritten digits. The images are normalized to a 28×28 pixel bounding box and anti-aliased. As illustrated in Fig. 7, we also construct two datasets, i.e., rotated MNIST and noisy MNIST, by applying random rotation ranging from -45° to 45° and random gaussian noise with zero mean and a variance of 1 to the original MNIST images, respectively. We mix all three datasets and partition all images into seen and unseen categories with a ratio of 3:2. We further divide seen images into training, validation, and testing sets with a ratio of 2:1:1.
- *Coking Production Decision Making Dataset*: The Coke Production Dataset (CPD) is collected from a leading international cloud service provider. It is to predict coke grades for coke production operations where decisions with the highest grades will be taken. A total of 200 coke production experiments collected from July, 2020 to March, 2021 are collected in the CPD dataset. Each sample includes structured features such as the ratio of different coal raw materials and the quality of the refined coke is labeled for prediction. Such a prediction can be used to optimize the decision of industrial production. The samples in the CPD dataset vary from multiple factors such as raw material loactions and production specifications owing to the mobility among production in different batches. We divide the samples into training, validation and testing sets with a ratio of 3:1:1.
- *Driving Video Datasets*: We use the established driving video dataset comprising 64 clips randomly selected from KITTI, BDD100 k and SHD, with seen and unseen data divided as introduced in Section III-A.

2) Implementation: We implement the offline scene profiling on a server equipped with 128 GB RAM and 4 Nvidia 2080 Ti GPUs, running a Linux distribution. We implement online model inference on three typical mobile devices, i.e., a Nvidia Jetson Nano, a Nvidia Jetson TX2 NX and a Windows laptop. Pytorch is employed as the inference engine and TensorRT [29] is used for the run-time acceleration on both Jetson devices, running a Linux distribution. OpenCV is compiled on CPU for balancing the usage of CPU and GPU. The hardware configurations are shown in Table II. For the HDR task, because of small image size, we direct conduct multi-level clustering on image pixels without training the $\mathcal{M}_{scene}$ and use a multi-layer perceptron (MLP) [30] with two layers to train the $\mathcal{M}_{decision}$ for its light-weight characteristic. Compressed models for image classification are trained based on LeNet [28]. For the CGP task, we directly use the structured features without training the $\mathcal{M}_{scene}$ and use a decision tree (DT) model [31] based on Scikit-Learn [32] as $\mathcal{M}_{decision}$ for its superior performance and interpretability. Compressed models for coke grade prediction are trained based on Linear Regression (LinearR) [33]. For the VD task, ResNet18 [18] and a MLP of two layers are used to train the $\mathcal{M}_{scene}$ and the $\mathcal{M}_{decision}$, respectively. Compressed models for object detection are fine-tuned on YOLOv3-tiny [19] pre-trained on the COCO [34] dataset. Details of all deployed models are listed in Table III. Compressed models of HDR and VD tasks are trained with Algorithm 1 consodering the heavy training overhead of DNN model and compressed models of CGD tasks are trained with Algorithm 2 for a better performance. A total of 25, 17 and 19 compressed models are trained for the HDR, CGD and VD tasks, respectively, to provide compressed models for inference in all possible scenes.

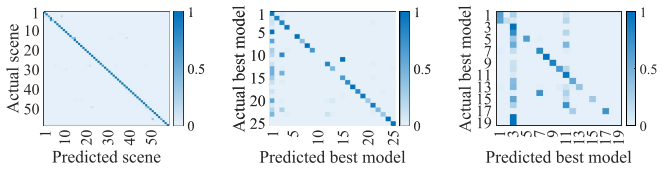
3) Candidate Methods: We compare Anole with the following candidate methods:

TABLE III
DETAILS OF DEPLOYED MODELS, WHERE FLOPS OF THE DEEP MODEL YOLOV3 IS 10× BIGGER THAN YOLOV3-TINY AND RESNET18

Task	Model	Role	FLOPS	Weights
HDR	LeNet	Compress model	31 M	234 KB
	MLP	$\mathcal{M}_{decision}$	3.6 M	935 KB
	Resnet18	Deep model	4.69 Bn	44 MB
	Resnet50	Deep model	9.74 Bn	87 MB
CGP	LinearR	Compress model	0.2 K	0.1 KB
	DecisionTree	$\mathcal{M}_{decision}$	0.1 K	1.5 KB
VD	YOLOv3-tiny	Compress model	5.56 Bn	34 MB
	Resnet18	$\mathcal{M}_{scene}$	4.69 Bn	44 MB
	MLP	$\mathcal{M}_{decision}$	3.6 M	935 KB
	YOLOv3	Deep model	65.86 Bn	237 MB

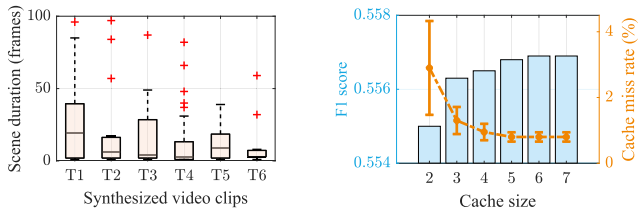
- *Single Deep Model (SDM) [19]:* One single deep model is trained with all training samples for online inference. For the HDR task, a ResNet18 and a ResNet50 are respectively trained for their supreme image classification performance. For the VD task, a fully-fledged YOLOv3 is trained. For the CGP task, no deep models are considered because there is no large model good enough so far for this task.
- *Single Shallow Model (SSM) [35]:* One single compressed model is trained with all training samples for online inference. For the HDR task, a LeNet is trained. For the VD task, a YOLOv3-tiny is trained.
- *Clustering-based Domain Generalization (CDG) [36]:* Compressed models are trained on domains defined by clustering training data samples in the feature space. During online prediction, the compressed model trained on the cluster which has the closest mean compared with the feature of the test sample is selected for use.
- *Dataset-based Multiple Models (DMM):* One separate compressed model is trained on each training dataset, i.e., the original, rotated, and noisy MNIST for the HDR task, the KITTI, BDD100 k, and SHD for the VD task, the human-defined subsets for CGP task. During online prediction, the compressed model corresponding to the same dataset as the test sample is selected for use.

4) Metrics: We evaluate the performance of all candidate methods with respect to inference accuracy and latency. For the HDR task, accuracy is defined as the proportion of correctly predicted samples to the total number of test samples. For the CGD task, we use Error Rate (ER), defined as ER $= \frac{\sum_{n=1}^{N}|\hat{y}_n - y_n|}{\sum_{n=1}^{N} y_n}$, Mean Absolute Error (MAE), defined as MAE $= \frac{\sum_{i=1}^{N}|\hat{y}_i - y_i|}{N}$, Mean Squared Error (MSE), defined as MSE $= \frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N_j}$ and Pearson Correlation Coefficient (PCCS), defined as PCCS $= \frac{\sum_{i=1}^{N}(\hat{y}_i - \hat{\mu})(y_i - \mu)}{\sqrt{\sum_{i=1}^{N}(\hat{y}_i - \hat{\mu})^2}\sqrt{\sum_{i=1}^{N}(y_i - \mu)^2}}$ for evaluation, where N denotes the number of all testing data; $\hat{y}_n$ and y_n are the estimation and the ground truth of the n-th sample; $\hat{\mu}^j$ and μ^j are the mean of $\hat{y}_n$ and y_n, respectively. For the VD task, we use F1 score, defined as F1 $= \frac{2 \cdot p \cdot r}{p+r}$, where p and r denote the precision and recall of detection, respectively. We also consider the end-to-end delay, i.e., the time duration from receiving a test sample to obtaining the corresponding inference result.



(a) $\mathcal{M}_{scene}$ for VD (b) $\mathcal{M}_{decision}$ for HDR (c) $\mathcal{M}_{decision}$ for VD

Fig. 8. Confusion matrices of scene profiling models, showing high accuracy for scene encoding and model decision, respectively.



(a) Scene duration distributions on synthesized video clips (b) Cache miss rate and F1 score obtained with varying cache size

Fig. 9. (a) Boxplot of scene duration, measured as the number of frames without model switching; (b) cache miss rate and F1 score as functions of varying cache sizes.

B. Effect of Scene Profiling Models

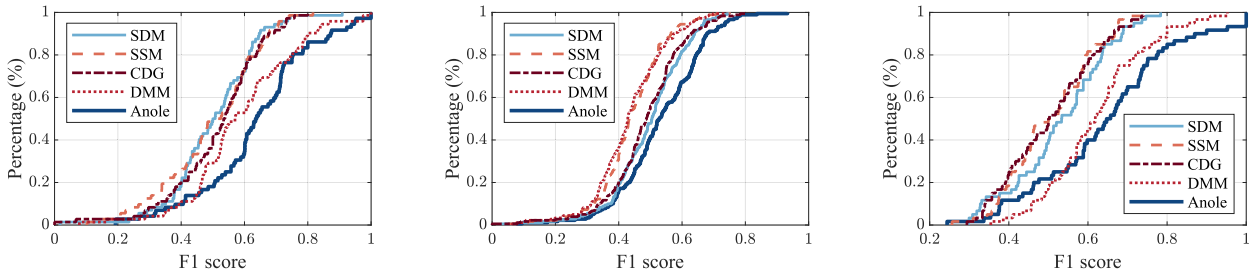
1) Scene Encoder $\mathcal{M}_{scene}$: For the VD task, we test $\mathcal{M}_{scene}$ on classifying scenes on the validation set of seen scenes. Scenes are defined based on the multi-level clustering results. Fig. 8(a) shows the scene classification confusion matrix of scene encoder $\mathcal{M}_{scene}$ on the validation set. It can be seen that $\mathcal{M}_{scene}$ works well among almost all scenes. There also exist some exceptional scenes that are confusing to $\mathcal{M}_{scene}$. We merge similar scenes in the feature space before training compressed models.

2) Decision Model $\mathcal{M}_{decision}$: We evaluate the ability of $\mathcal{M}_{decision}$ in selecting the top-one model on the validation set of seen data. Fig. 8(b) and (c) show the confusion matrix of the $\mathcal{M}_{decision}$ models predicting best models versus true best models for the HDR and VD tasks, respectively. It can be seen that $\mathcal{M}_{decision}$ have basic model selection ability. This is because the decision of model selection is based on the well-trained $\mathcal{M}_{scene}$, with one scene corresponding to a group of suitable models. We can also see that $\mathcal{M}_{decision}$ may make mistakes on some models. This is because the top one model may not be significantly better than other models.

C. Effect of Cache-Based Model Update Strategy

To effectively evaluate the effect of our cache-based model update strategy, we synthesize six fast-changing video clips, denoted as T1-T6, for the VD task. Specifically, for each synthesized video clip, we randomly select 5 clips from the 64 clips in the dataset. For each selected clip, we randomly cut a video segment of 100 frames (from the testing set for a seen clip) and then splice all video segments, resulting a synthesized video clip of 500 frames. We then conduct model inference using Anole on T1-T6.

1) Scene Duration: Fig. 9(a) plots the boxplot of scene duration measured as the number of frames without model switching



(a) KITTI (b) BDD100k (c) SHD

Fig. 10. CDFs of F1 score of all candidate methods on each source dataset, demonstrating the advantage of Anole over candidate methods, including the versatile large SDM. Note that the more the line leans towards the *bottom right corner*, the better the performance.

TABLE IV
INFERENCE ACCURACY OF CANDIDATE METHODS ACHIEVED IN CROSS-SCENE AND NEW-SCENE SCENARIOS ON THE HDR TASK

Method	Model	GPU Mem. (MB)	Cross-scene		New-scene	
			Noisy	Rota.	Noisy	Rota.
SDM	Res.18	45.27	0.773	0.828	0.777	0.826
	Res.50	98.31	0.704	0.810	0.702	0.808
SSM	LeNet	0.18	0.755	0.751	0.754	0.752
CDG	LeNet	0.18	0.099	0.721	0.099	0.721
DMM	LeNet	0.18	0.620	0.755	0.619	0.758
Anole	LeNet	0.18	**0.822**	**0.844**	**0.821**	**0.846**

TABLE V
PERFORMANCE OF DIFFERENT METHODS ON THE CGD TASK

Method[1]	Latency	ER	MAE	MSE	PCCS
SSM	0.87s	2.28	1.48	2.69	0.17
CDG	0.87s	5.05	3.27	14.16	0.12
DMM	0.90s	2.14	1.38	2.34	0.16
Anole	1.05s	**1.38**	**0.90**	**1.40**	**0.40**

[1] There is no effective deep model (*i.e.*, SDM) so far for the CGD task owing to the lacking of large-scale datasets.

on all six synthesized video clips. It can be seen that scenes change rapidly in the perspective of $\mathcal{M}_{decision}$, with over 80% of scenes lasting fewer than 40 frames and the mean scene duration less than 20 frames.

2) Cache Miss Rate: Fig. 9(b) depicts the cache miss rate and the F1 score as functions of cache size in the unit of compressed model size. It can be seen that a cache capable of loading up to 5 models can sustain a low cache miss rate and a stable inference accuracy. This observation aligns with the observation of the long-tail model utility distribution as shown in Fig. 6(b). It is also observed that the inference accuracy remains satisfactory even for a cache size of 2 models, demonstrating the feasibility of Anole on devices with extremely limited GPU memory.

D. Cross-Scene Experiments

In this experiment, we investigate the performance of all candidate methods cross fast-changing scenes, using samples in the test set of seen data. For the VD task, F1 score is calculated every ten frames to show the instantaneous performance changes.

1) Performance Comparison: Table IV lists the accuracy of all candidate methods over all cross-scene HDR test images. Table V list the considered metrics of all candidate methods on the testing samples of CGD task. Fig. 10 plots the CDFs of F1 score of all candidate methods on each test set of seen data selected from KITTI, BDD100 k and SHD, respectively. For both tasks, Anole outwits other methods in terms of accuracy. For example, on the HDR task, Anole can even outwits SDM by 6.3%; on the CGD task, Anole can achieve a prediction PCCS increase by 135.5%; on the VD task, ANole can outperform SDM in terms of SDM by over 15%. Moreover, other methods exhibit inconsistent performance across different datasets. For example of the VD task, DMM gains good performance on the KITTI and SHD datasets, while SDM only performs well on the BDD100 K dataset. This discrepancy arises because DMM fits simpler datasets whereas SDM is biased towards BDD100 k due to the overwhelming number of training samples.

2) Effect of Data Segmentation and Model Adaptation: It is a common practice to train an individual model on each datasets (i.e., DMM) or to segment a dataset according to feature similarity and train respective models (i.e., CDG). It can be seen that DMM performs similarly to Anole for simple training datasets such as all MNIST datasets for the HDR task, and the KITTI and the SHD datasets for the VD task, but DMM performs poorly on large and complex datasets like BDD100 k. In contrast, CDG trains and selects models on similar data samples. However, the inference accuracy of CDG is not as good as that of Anole over all test sets for both tasks. This demonstrates Proposition 1, which states that a model trained on a scene may not always perform well on that scene. In contrast, Anole employs a decision model to learn the appropriate scenes and determines which model is most suitable for online prediction, resulting in stable performance. Furthermore, although deep-model-based method SDM is generally assumed to have better performance, we surprisingly find that Anole outwits SDM on all test sets. This implies that training a single large DNN model for cross-scene inference is more difficult than training and choosing from a set of specialized compressed models.

E. New-Scene Experiments

In this experiment, we examine the performance of all candidate methods in new scenes, using unseen data. Particularly, for the VD task, six unseen video clips include one clip from KITTI with attributes of {*Street*, *Day*}, four scenes from BDD100 k with attributes of {*Urban*, *Night*}, {*Urban*, *Day*}, {*Highway*,

TABLE VI
INFERENCE ACCURACY OF ALL CANDIDATE METHODS OBTAINED ON UNSEEN DATA

Method	KITTI	BDD100k				SHD	Average
	Street, Day	Urban, Night	Urban, Day	Highway, Dusk	Street, Night	Tunnel, Night	
SDM[1]	0.437	0.531	**0.477**	**0.476**	**0.468**	0.409	0.466
SSM	0.387	0.514	0.335	0.404	0.454	0.370	0.411
CDG	0.459	0.537	0.453	0.410	0.440	0.401	0.450
DMM	0.407	0.482	0.382	0.388	0.384	0.374	0.403
Anole	**0.506**	**0.590**	0.453	0.440	0.461	**0.470**	**0.487**

[1] SDM uses a deep model, resulting in higher latency, larger memory usage (Table VII), and higher power consumption (Figure 13).The best results are indicated in bold while the Second-best results are marked in blue.

(a) Implementation of Anole on Jetson TX2 NX deployed on a UAV (b) Results in a typical night scenario

Fig. 11. (a) Implementation of Anole on a Jetson TX2 NX connected with a 1080p HD camera. (b) Visualization of vehicles detected in a night scenario.

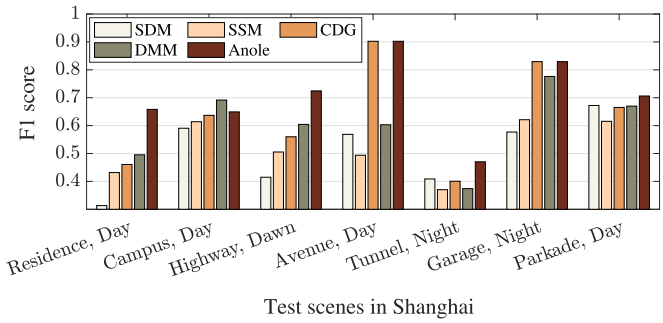


Fig. 12. F1 score of all methods on test scenes in Shanghai, where Anole exceeds other methods with a latency of less than 20 ms on Jetson TX2 NX.

Dusk}, and {*Street*, *Night*}, and one scene from SHD with attributes of {*Tunnel*, *Night*}. Tables IV and VI list the accuracy results for the HDR and VD tasks, respectively. It can be seen that though SDM with a much larger model size is expected to excel other shallow-model-based methods on unseen scenes, Anole demonstrates supreme generalization ability and even outperforms SDM on all unseen data. As for unseen scenes from BDD100 k, Anole can still achieve high accuracy comparable to that of SDM.

F. Real-World Experiments

As depicted in Fig. 11(a), we implement all methods on the Nvidia Jetson TX2 NX on a UAV to conduct real-world experiments. The UAV used in experiments integrates a flight controller with an MPU6500 gyro sensor for stabilization, four RS2205 propulsion motors, an ELRS radio transmitter and a 4S 1550mAh LiPo battery. We control the UAV to fly at an altitude of about 3 meters and a constant speed of about 10 km/h with sudden acceleration and deceleration at the begining and the end of each flight. The traffic on seven types of roads is taped at 1080p@30FPS at different time in a day from March 1 to March 7, 2022. Well-trained compressed models and the decision model are downloaded to the Jetson device. LabelImg [22] is used to label all recorded frames as the ground truth for offline analysis.

Fig. 12 plots the F1 score of all methods. Anole outperforms all other candidate methods in all test scenarios. We visualize the car detection results of Anole (white solid frames) and SDM (red dashed frames) in a typical night driving scenario in Fig. 11(b). The inference results obtained using SDM frequently contain errors, especially false negative errors as shown in the enlarged subgraph.

TABLE VII
INFERENCE LATENCY AND MEMORY CONSUMPTION ON MOBILE DEVICES

Model \ Metric	Latency (ms)			GPU Memory (MB)	
	Nano	TX2	Lap.	Loading	Execution
$\mathcal{M}_{sce.} + \mathcal{M}_{dec.}$	23.2	3.1	20.8	44	584
YOLOv3	313.8	42.9	62.2	$240\times n$[1]	1,730
YOLOv3-tiny	37.8	10.8	32.2	$40\times n$	1,120

[1] n denotes the number of compressed models to load.

G. Inference Latency

We evaluate the inference latency of the decision model $\mathcal{M}_{decision}$ and compressed models on different mobile devices. The results are shown in Table VII. The results reveal that YOLOv3-tiny exhibits significantly lower latency when compared to deep YOLOv3, which is generally deemed unsuitable for deployment on devices. For instance, the latency of YOLOv3-tiny on Jetson Nano is 87.9% lower than that of YOLOv3. This highlights the substantial potential for accelerating inference using shallow models. It is also evident that $\mathcal{M}_{decision}$ can be executed in real-time on embedded mobile devices such as Jetson Nano, with a latency as low as 23.2 ms, making it suitable for online inference applications.

H. Memory and Power Consumptions

We investigate the memory consumption of different models from the following two aspects, i.e., loading model only, and the memory consumption during inference with a batch size of 1. Table VII demonstrates that memory consumption for loading model is significantly lower than that during inference, owing to the presence of hidden parameters during inference. We also examine the impact of different power configurations adopted

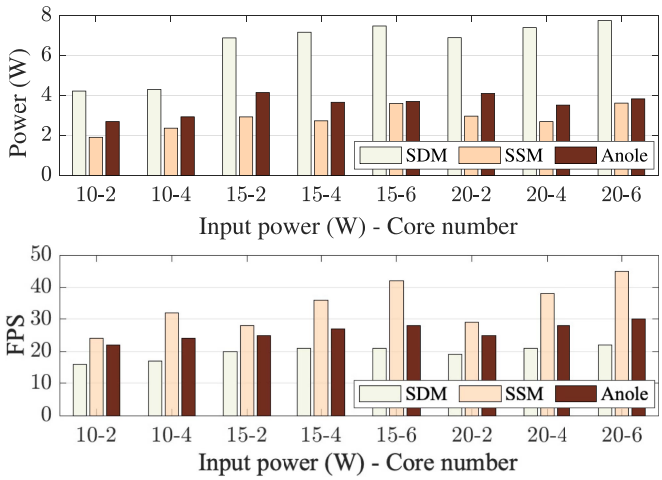


Fig. 13. Power consumption and inference speed of different methods in various power modes.

by Jetson TX2 NX to the performance of Anole. The power consumption and inference speed of Anole and baselines under different power modes are shown in Fig. 13, respectively. Anole achieves a 45.1% reduction in power consumption compared with SDM and a inference speed of over 30 FPS with an input power of 20 W running 6 cores.

VIII. Related Work

A. DNN Prediction on Mobile Devices

To perform DNN inference on mobile devices, new DNNs are specially designed [6], [7], [8], [9] or existing DNNs are compressed to match the computing capability of a mobile device [10], [11]. First, model structure can be optimized to reduce complexity [6], [7], [37]. Second, quantization precision can be reduced to minimize computational cost, e.g., use integers instead of floating-point numbers [38], [39]. Third, the neural network model can also be accelerated by pruning, i.e., deleting some neurons in the neural network [10], [11], [40]. Scene information is also utilized for model compression on edge/mobile devices [41], [42], [43], [44]. Finally, model distillation can distill the knowledge of large models into small models [8], [9]. Such schemes ensure real-time model inference at the expense of compromised accuracy.

Another direction is to divide DNNs and perform collaborative inference on both edge devices and the cloud [13], [14], [45], [46], [47], [48], or to transmit compressed sensory data to the cloud for data recovery and model inference [16], [17]. Neurosurgeon [14] partitions the computation of each DNN inference task in a layer granularity. CLIO [49] addresses the instability of network conditions and optimizes inference under different network states. These approaches need coordination with the cloud for each inference, leading to unpredictable inference delays when the communication link is unstable or disconnected. However, they prove inadequate for cross-scene mobile inference scenarios where even deep models are unable to cope.

B. Cross-Scene DNN Prediction

Data-driven machine learning models face challenges in maintaining robust inference performance when dealing with cross-scene inference [50]. One natural approach for scene partitioning is to partition the scene based on prior knowledge or historical samples. First, based on prior knowledge, a similarity graph is constructed to cluster similar domains together [51]. However, obtaining such prior knowledge based on domain expertise can be challenging. Second, the original data or their extracted features can be utilized for more automated scene partitioning [52]. However, these methods may result in the loss of critical information in complex systems [36]. Cross-scene DNN prediction can also be enhanced given a golden model in the cloud for online sample labelling [53], [54]. However, the existence of a perfect or golden model is not always feasible.

C. Mixture of Experts

In recent years, we have witnessed the success of Mixture of Experts (MoE) [55], [56], especially in efficient training of large language models (LLM). MoE employs multiple experts for model training, each for one domain. Then, a gate network will be used to determine the correspondence between samples and experts. Though inspired by MoE, Anole differs from MoE in the following 2 aspects. First, experts in MoE are diversified by constraints of losses, but they themselves cannot be related to the scene. In fact, the main purpose of MoE is to expand the number of model parameters, rather than to customize and select scene-specific models. Second, MoE is just a model architecture, and models based on MoE architecture still need to deploy the entire model during deployment. Therefore, MoE-based models often require a significant amount of memory, which is unacceptable for mobile agents like UAVs. In contrast, Anole employs multiple compressed models for online model inference, each designed for one scene. Only a few compressed models are needed to be deployed during online inference. Therefore, Anole is more suitable for mobile devices only with limited resources.

IX. Discussion

A. How to Choose the CMT-DM and CMT-SM Algorithm?

The effect of Anole relies on an army of compact DNN models, which can be given in advance or trained by the two algorithms (i.e., Algorithm 1, CMT-DM and Algorithm 2, CMT-SM) proposed in Section V-A2. We compare the differences of CMT-DM and CMT-SM, which can guide the choice of the two models training algorithms.

First, in general, CMT-SM is better in terms of inference accuracy than CMT-DM because CMT-SM combines multiple scenes based on both semantics and model training performances, while CMT-SM only considers the similarity of semantics of different scenes. Therefore, with adequate computing resources, we should first choose the CMT-SM algorithm.

However, each combination of scenes requires one round of model training. In fact, the upper bound complexity of the CMT-SM algorithm can be computed by considering the case where no combination of tasks is conducted and the number of

scenes never reduces. In this case, the complexity of CMT-SM with n scenes at the ith iteration is $O(T_R \cdot (n-i+1))$, where T_R denotes the time for conducting a training session. Thus, the overall complexity $\mathcal{O}(T_R \cdot (\sum_1^{n-1} i)) \in \mathcal{O}(T_R \cdot (n \cdot (n-1)/2)) \in \mathcal{O}(T_R \cdot n^2)$. Such a complexity is too large if T_R is a large number, e.g., the training time of a typical DNN. In these cases (e.g., the HDR task and the VD task), choosing the CMT-DM algorithm is a practical option.

B. Limitation of Anole

Anole has the limitation that, Anole may encounter new scenes which it cannot handle. The performance of Anole relies on a dataset containing all possible scenes and the goal of Anole is to achieve stable model inference across the seen scenes and try our best to generalize to new scenes. The reason why Anole may generalize to unseen new scenes is that the unseen new scenes may be similar to the seen scenes in the view of compressed models. However, we cannot always guarantee this. There may be some new scenes that Anole cannot handle. Fortunately, experiments have shown that such situations are not common. When encountering new scenes that cannot be processed, Anole can make timely reminders and switch to the large model on the cloud to ensure basic inference response.

C. Can Mobile Foundation Model (FM) Replace Anole?

Mobile Foundation Model [57] inspired by Large Language Model (LLM) such as ChatGPT also proves to be potential to solve the cross-scene online model inference (OMI) problem on mobile devices. Mobile FM also shows the amazing generalization performance on typical mobile tasks. However, Anole can exceed the mobile FM in the following 2 aspects. First, Anole has extremely low memory requirements. Anole only needs to load a cache of compressed models to memory or GPU memory (usually $<$ 2GB), while mobile FM needs to load the whole FM to mobile devices ($>$ 7.5 GB). As a result, Anole is more suitable for the low-end devices with limited memories. Second, the power consumption of Anole is much less than mobile FM. Despite multiple models are utilized in Anole, each time Anole only use one model for inference. On the contrary, mobile FM needs a whole inference of a large neural network, consuming tens of times more power than Anole. Therefore, Anole is also more suitable for the battery-powered mobile devices such as UAVs. Second, mobile FM cannot work well on all tasks on mobile devices. Without scene-related information, for tasks with less well-annotated data (such as the CGD task), mobile FM cannot solve these task well.

D. Model Selection Cost

Recent research [58] has pointed out the heavy cold start cost of model inference. For the proposed Anole scheme, the best compressed model is automatically selected by the decision model on each test sample basis, which is extremely fast as after a one-time feature extration, the decision model can rank all compressed models according to the feature similarty and select the best one or multiple models.

As for the cost of model switching between disk and memory, it depends on the memory size of a specific mobile device. If all compressed models can be preloaded into the memory, the model switching incurs no additional cost. Typical mobile devices, such as Jetson Orin Nano, Raspberry Pi 5, and Xiaomi 14 smartphone, may have a memory capacity of 8-16 GB. In contrast, a compressed model can be quite light (e.g., YOLO-tiny and ResNet-18 are about 40-45 MB). As a result, it is common to preload about 200 compressed models in a mobile device. According to our study as depicted in Fig. 6(b), the utility of compressed models follows a power-law distribution, which means that deploying only a small portion of compressed models is sufficient. In addition, we also apply the Least Frequently Used (LFU) cache strategy to deal with the extreme cases where the selected models are not present in the memory.

X. Conclusion

In this paper, we have proposed Anole, an online model inference scheme on mobile devices. Anole employs a rich set of compressed models trained on a wide variety of human-defined scenes and offline learns the implicit mode-defined scenes characterized by these compressed models via a decision model. Moreover, the most suitable compressed models can be dynamically identified according to the current testing samples and used for online model inference. As a result, Anole can deal with unseen samples, mitigating the impact of OOD problem to the reliable inference of statistical models. Anole is lightweight and does not need network connection during online inference. It can be easily implemented on various mobile devices at a low cost. Extensive experiment results demonstrate that Anole can achieve the best inference accuracy at a low latency.

References

[1] K. Wang, X. Fu, Y. Huang, C. Cao, G. Shi, and Z.-J. Zha, "Generalized UAV object detection via frequency domain disentanglement," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 1064–1073.
[2] Y. Zhou, Y. He, H. Zhu, C. Wang, H. Li, and Q. Jiang, "MonoEF: Extrinsic parameter free monocular 3D object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 12, pp. 10114–10128, Dec. 2022.
[3] Z. Zheng et al., "Contextual anomaly detection in solder paste inspection with multi-task learning," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 6, pp. 1–17, 2020.
[4] A. Dalmia et al., "Pest management in cotton farms: An AI-system case study from the global south," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 3119–3127.
[5] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, "On-edge multi-task transfer learning: Model and practice with data-driven task allocation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1357–1371, Jun. 2020.
[6] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, arXiv: *1704.04861*.
[7] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.
[8] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," 2019, arXiv: *1910.01108*.
[9] X. Jiao et al., "TinyBERT: Distilling BERT for natural language understanding," 2019, arXiv: *1909.10351*.
[10] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, arXiv: *1506.02626*.
[11] Y. Wang et al., "Pruning from scratch," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 12273–12280.

[12] Z. Zheng, P. Luo, Y. Li, S. Luo, J. Jian, and Z. Huang, "Towards lifelong thermal comfort prediction with KubeEdge-sedna: Online multi-task learning with metaknowledge base," in *Proc. 13th ACM Int. Conf. Future Energy Syst.*, 2022, Art. no. 101.
[13] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1423–1431.
[14] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017.
[15] Y. Fang, Z. Jin, and R. Zheng, "TeamNet: A collaborative inference framework on the edge," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1487–1496.
[16] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2019, Art. no. 25.
[17] W. Zhang et al., "Elf: Accelerate high-resolution mobile deep vision with content-aware parallel offloading," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 201–214.
[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[19] P. Adarsh, P. Rathi, and M. Kumar, "YOLO v3-tiny: Object detection and recognition using one stage improved model," in *Proc. IEEE Int. Conf. Adv. Comput. Commun. Syst.*, 2020, pp. 687–694.
[20] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3354–3361.
[21] F. Yu et al., "BDD100k: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2633–2642.
[22] D. Tzutalin, "Labelimg," GitHub Repository, 2015. [Online]. Available: https://github.com/tzutalin/labelImg
[23] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.
[24] O. Chapelle and L. Li, "An empirical evaluation of thompson sampling," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 2249–2257.
[25] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Ann. Operations Res.*, vol. 134, no. 1, pp. 19–67, 2005.
[26] Z. Li, K. Ren, X. Jiang, B. Li, H. Zhang, and D. Li, "Domain generalization using pretrained models without fine-tuning," 2022, arXiv: *2203.04600*.
[27] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ, USA: Wiley, 2018.
[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[29] H. Vanholder, "Efficient inference with tensorrt," in *Proc. GPU Technol. Conf.*, 2016, Art. no. 2.
[30] R. Kruse, S. Mostaghim, C. Borgelt, C. Braune, and M. Steinbrecher, "Multi-layer perceptrons," in *Computational Intelligence: A Methodological Introduction*. Berlin, Germany: Springer, 2022, pp. 53–124.
[31] L. Breiman, *Classification and Regression Trees*. Evanston, IL, USA: Routledge, 2017.
[32] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[33] R. M. Rifkin and R. A. Lippert, "Notes on regularized least squares," *Comput. Sci. Artif. Intell. Lab.*, Cambridge, MA, USA, Tech. Rep. CBCL-268, 2007, pp. 1–10.
[34] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.
[35] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, arXiv: *1804.02767*.
[36] Z. Zheng, Y. Wang, Q. Dai, H. Zheng, and D. Wang, "Metadata-driven task relation discovery for multi-task learning," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 4426–4432.
[37] H. Chen et al., "AdderNet: Do we really need multiplications in deep learning?," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1468–1477.
[38] X. Jiang et al., "MNN: A universal and efficient inference engine," in *Proc. Conf. Mach. Learn. Syst.*, 2020, pp. 1–13.
[39] E. Elsen, M. Dukhan, T. Gale, and K. Simonyan, "Fast sparse ConvNets," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 14629–14638.
[40] Z. Liu et al., "Rethinking the value of network pruning," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–21.
[41] B. Feng, Y. Wang, G. Li, Y. Xie, and Y. Ding, "Palleon: A runtime system for efficient video processing toward dynamic class skew," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 427–441.
[42] R. Xu, J. Lee, P. Wang, S. Bagchi, Y. Li, and S. Chaterji, "LiteReconfig: Cost and content aware reconfiguration of video object detection systems for mobile GPUs," in *Proc. 17th Eur. Conf. Comput. Syst.*, 2022, pp. 334–351.
[43] Y. Wang, W. Wang, D. Liu, X. Jin, J. Jiang, and K. Chen, "Enabling edge-cloud video analytics for robotics applications," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1500–1513, Second Quarter 2023.
[44] S. Jiang, Z. Lin, Y. Li, Y. Shu, and Y. Liu, "Flexible high-resolution object detection on edge devices with tunable latency," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 559–572.
[45] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, and Y. Zhang, "Auto-split: A general framework of collaborative edge-cloud AI," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2021, pp. 2543–2553.
[46] Z. Zheng, Y. Li, H. Song, L. Wang, and F. Xia, "Towards edge-cloud collaborative machine learning: A quality-aware task partition framework," in *Proc. 31st ACM Int. Conf. Inf. Knowl. Manage.*, 2022, pp. 3705–3714.
[47] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: Synergistic progressive inference of neural networks over device and cloud," in *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, Art. no. 37.
[48] P. Guo, B. Hu, and W. Hu, "Mistify: Automating DNN model porting for on-device inference at the edge," in *Proc. 18th USENIX Symp. Netw. Syst. Des. Implementation*, 2021, pp. 705–719.
[49] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon, "CLIO: Enabling automatic compilation of deep learning pipelines across IoT and cloud," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 58:1–58:12.
[50] S. Zhai et al., "RISE: Robust wireless sensing using probabilistic and statistical assessments," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 309–322.
[51] L. Han, Y. Zhang, G. Song, and K. Xie, "Encoding tree sparsity in multi-task learning: A probabilistic framework," in *Proc. AAAI Conf. Artif. Intell.*, 2014, pp. 1854–1860.
[52] W. Lu et al., "Out-of-distribution representation learning for time series classification," in *Proc. Int. Conf. Learn. Representations*, 2023, pp. 1–21.
[53] M. Khani et al., "RECL: Responsive resource-efficient continuous learning for video analytics," in *Proc. 20th USENIX Symp. Netw. Syst. Des. Implementation*, 2023, pp. 917–932.
[54] R. Bhardwaj et al., "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 119–135.
[55] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.
[56] N. Shazeer et al., "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 1–19.
[57] J. Yuan et al., "Mobile foundation model as firmware," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2024, pp. 279–295.
[58] K. Zhao, Z. Zhou, L. Jiao, S. Cai, F. Xu, and X. Chen, "Taming serverless cold start of cloud model inference with edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 8, pp. 8111–8128, Aug. 2024.

Yunzhe Li (Student Member, IEEE) received the BS degree in computer science from Wuhan University, in 2021. He is currently working toward the PhD degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include mobile sensing and edge computing. He received the Rising Star Award in 2023 from the KubeEdge of Cloud Native Computing Foundation.

Hongzi Zhu (Senior Member, IEEE) received the PhD degree in computer science from Shanghai Jiao Tong University, in 2009. He was a post-doctoral fellow with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, and the Department of Electrical and Computer Engineering, University of Waterloo, in 2009 and 2010, respectively. He is now a professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include mobile sensing, mobile computing, and Internet of Things. He received the Best Paper Award from IEEE Globecom 2016. He is an associate editor of the *IEEE Transactions on Vehicular Technology* and *IEEE Internet of Things Journal*. For more information, please visit http://lion.sjtu.edu.cn.

Zhuohong Deng received the BE degree in computer science and technology from Beijing Forestry University, China, in 2021, and the MS degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, in 2025. He is now an engineer with China Telecom Corporation Limited. His research interest include artificial intelligence system.

Yunlong Cheng (Student Member, IEEE) received the BE degree in computer science and technology from Shanghai Jiao Tong University, China, in 2021. He is currently working toward the PhD degree in computer science and technology with Shanghai Jiao Tong University. His research focuses on data mining, prediction, and scheduling algorithms within the context of cloud services. He has published papers in IPDPS, TCS, DASFAA, etc.

Zimu Zheng (Member, IEEE) received the BEng degree from the South China University of Technology and the PhD degree from the Hong Kong Polytechnic University. He is currently a head research engineer with Huawei Cloud. His research interest lies in edge intelligence, benchmarking, multi-task learning, and AIoT. He received the Best Paper Award of ACM e-Energy 2018 and the Best Paper Award of ACM BuildSys 2018. He is an IEEE member with publications in the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Internet of Things Journal*, IEEE INFOCOM, IEEE ICDCS, *IEEE Transactions on Sustainable Computing*, etc. He has also received several awards for outstanding technical contributions in Huawei and serves as a co-chair in the KubeEdge SIG AI of Cloud Native Computing Foundation.

Liang Zhang (Student Member, IEEE) received the BE degree in computer science and technology from Northeastern University, China, in 2018, and the PhD degree in computer science from Shanghai Jiao Tong University, China, in 2024. She is currently a lecturer with the Department of Computer Science and Technology, Donghua University. Her research interests include stream processing and resource scheduling in the cloud or edge computing environment. For more information, please visit https://zl-cs.github.io/.

Shan Chang (Member, IEEE) received the PhD degree in computer software and theory from Xian Jiaotong University, in 2013. From 2009 to 2010, she was a visiting scholar with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. She was also a visiting scholar with the BBCR Research Lab, University of Waterloo, from 2010 to 2011. She is now a professor with the Department of Computer Science and Technology, Donghua University, Shanghai. Her research interests include security and privacy in mobile networks and sensor networks.

Minyi Guo (Fellow, IEEE) received the PhD degree in computer science from the University of Tsukuba, Japan. He is a Zhiyuan chair professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. He is currently Zhiyuan chair professor. His present research interests include parallel/distributed computing, compiler optimizations, embedded systems, pervasive computing, Big Data, and cloud computing. He is now on the editorial board of the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing* and *Journal of Parallel and Distributed Computing*.