InFi: End-to-end Learnable Input Filter for Resource-efficient Mobile-centric Inference

Mu Yuan
ym0813@mail.ustc.edu.cn
University of Science and Technology of China
Hefei, China

Lan Zhang*
zhanglan@ustc.edu.cn
University of Science and Technology of China
Hefei, China

Fengxiang He
hefengxiang@jd.com
JD Explore Academy
Beijing, China

Xueting Tong
tongxueting@mail.ustc.edu.cn
University of Science and Technology of China
Hefei, China

Xiang-Yang Li
xiangyangli@ustc.edu.cn
University of Science and Technology of China
Hefei, China

ABSTRACT

Mobile-centric AI applications put forward high requirements for resource-efficiency of model inference. Input filtering is a promising approach to eliminate the redundancy in the input so as to reduce the cost of inference. Previous efforts have tailored effective solutions for many applications, but left two essential questions unanswered: (1) *theoretical filterability of an inference workload* to guide the application of input filtering techniques, thereby avoiding the trial-and-error cost for resource-constrained mobile applications; (2) *robust discriminability of feature embedding* to allow input filtering to be widely effective for diverse inference tasks and input content. To answer these questions, we first provide a generic formalization of the input filtering problem and theoretically compare the hypothesis complexity of inference models and their input filters to understand the optimization potential of applying input filtering. Then we propose the first end-to-end learnable input filtering framework that covers most state-of-the-art methods and surpasses them in feature embedding with robust discriminability. Based on our framework, we design and implement an input filtering system *InFi* supporting six input modalities. *InFi* is the first to support text and sensor signal inputs and model partitioning deployments widely adopted by under-resourced mobile systems. Comprehensive evaluations confirm our theoretical results and show that *InFi* outperforms strong baselines in applicability, accuracy, and efficiency, owing to its generality and end-to-end learnability. *InFi* can achieve 8.5× throughput and save 95% bandwidth, while keeping over 90% accuracy, for a video analytics app on mobile platforms.

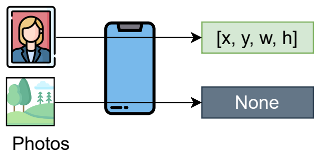
*Lan Zhang is the corresponding author.
This paper has been accepted for publication at MobiCom'22. This version is a preprint identical in content to the version to be published.

ACM Reference Format:
Mu Yuan, Lan Zhang, Fengxiang He, Xueting Tong, and Xiang-Yang Li. 2022. InFi: End-to-end Learnable Input Filter for Resource-efficient Mobile-centric Inference. In *The 28th Annual International Conference On Mobile Computing And Networking (ACM MobiCom '22), October 24–28, 2022, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3495243.3517016

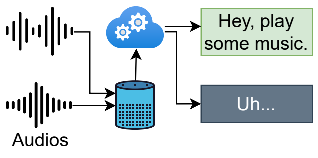
1 INTRODUCTION

The increased computing power of mobile devices and the growing demand for real-time sensor data analytics have created a trend of mobile-centric artificial intelligence (AI) [39, 43, 45, 47, 67]. It is estimated that over 80% of enterprise IoT projects will incorporate AI by 2022 [16]. The on-device inference of computer vision models brings us increasingly rich real-time AR applications on mobile devices [8]. A judicious combination of on-device and edge computing can analyze videos taken by drones in real-time [66]. The resource efficiency of model inference is critical for AI applications, especially for resource-limited mobile devices and latency-sensitive tasks. However, many AI models with state-of-the-art accuracy [4, 15, 30] are too computationally intensive to perform high-throughput inference, even when they are offloaded to edge or cloud servers [73].

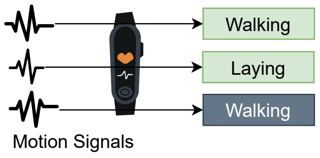
For resource-efficient inference, one direct and popular way is to eliminate the redundancy of the deep model itself via accelerating and compressing techniques [1, 22, 25, 52, 55, 57, 59]. In this work, we follow another series of approaches [5, 9, 19, 20, 33, 40] that attempt to filter the redundancy in the input data. Fig. 1 shows four examples of input redundancy in mobile-centric AI applications. We call this series of approaches input filtering and classify them into two categories: SKIP and REUSE. (1) **SKIP** methods [5, 33] aim to filter input data that will bring useless inference results, e.g., images without faces for a face detector (Fig. 1a) and audios without a valid command for a speech recognizer (Fig. 1b). FilterForward [5] trains a binary classifier and sets a threshold on classification confidence to filter input images. (2) **REUSE** methods [19, 20] attempt to filter input whose results can reuse the previous inference results, e.g., motion signals of the same action (Fig. 1c) and video frames with the same vehicle count (Fig. 1d). FoggyCache [19] maintains a cache of feature embedding and inference results of previous inputs and searches reusable results in the cache for newly arrived data.



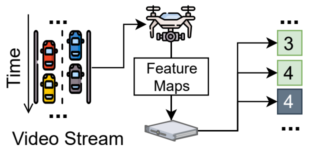
(a) Face detection on mobile phones.



(b) Speech recognition offloaded on cloud.



(c) Action classification on smart-bands.



(d) Vehicle counting partitioned between drones and edges.

Figure 1: Input redundancy in mobile-centric AI applications. Gray squares indicate redundant inference results: (a) no detected face, (b) invalid recognized speech, (c) previous classification result can be reused, (d) latest count result can be reused.

Input filtering usually works as a necessary prelude to inference for under-resourced mobile systems. Moreover, compared with model optimizations, input filtering provides more flexible trade-offs between the accuracy and efficiency, e.g., FilterForward can adjust the threshold in SKIP and FoggyCache can adjust the cache size in REUSE. Although prior efforts have designed effective input filters for a range of applications, two important and challenging questions remain unanswered:

1. Theoretical filterability analysis for the guidance of applying input filtering to mobile-centric inference: Not all inference workloads have the optimization potential by using input filtering. Sometimes, to achieve the required accuracy, a SKIP/REUSE filter is more costly than the original inference. Characterizing the conditions under which the filter has to cost more to be accurate is thus essential to input filtering. Previous efforts study the input filtering problem from an application-oriented perspective. They start from the observation of redundancy and propose bespoke input filtering solutions without further analyzing the relation between their inference workloads and input filters. Without theoretical guidance and explanation, though they delivered accurate and lightweight input filters for specific workloads, the trial-and-error process of designing input filters for other workloads is still very cumbersome and may fail next time, especially for resource-scarce mobile systems.

2. Robust feature discriminability for diverse tasks and modalities in mobile-centric inference: A discriminative feature representation [68] is critical to filtering performance, since it directly determines the accuracy of making SKIP decisions and finding REUSABLE results. The accuracy is critical to most AI inference workloads, especially for anomaly detection tasks [7, 74]. If anomaly events are incorrectly filtered, the efficiency gains will be meaningless. This is a challenge to the discriminability of features. Recent work [40] shows that for different workloads, the discriminability of low-level features is different, e.g., area feature works better for counting while edge feature works better for detection. Most existing filtering methods leverage handcrafted features [19, 20, 40] or pre-trained neural networks as feature embedding [5], and implicitly assume that these features are sufficiently discriminative for the target workloads. However, mobile applications usually have high diversity in input content and inference tasks. The dependency on pre-trained or handcrafted features leads to unguaranteed discriminability to these diversities. Our experiments (§ 6.2) show that, for an action classification workload, neither a SKIP method using the pre-trained feature [5] nor a REUSE method using the handcrafted feature [19] can work effectively. The feature embedding should be obtained in a workload-agnostic and learnable manner, rather than tailored case by case.

To answer these questions, we first provide a generic formalization of the input filtering problem and conditions of valid filters. Then we theoretically define filterability and analyze the filterability of two most common types of inference workloads, namely classification and regression, by comparing the hypothesis complexity [34, 50] of the inference model and its input filter. Instead of designing bespoke solutions for narrowly-defined tasks, we propose the first, to our best knowledge, end-to-end learnable framework which unifies both SKIP and REUSE approaches [5, 19, 40]. The end-to-end learnability provides feature embedding with robust discriminability in a workload-agnostic manner, thus significantly broadens the applicability. Based on the unified framework, we design an input filtering system, named *InFi*, which supports both SKIP and REUSE functions. In addition to image, audio and video inputs, *InFi* complements existing techniques in supporting text, sensor signal, and feature map inputs. Previous methods are typically designed for a certain deployment, e.g., inference offloading [19, 40]. *InFi* flexibly supports common deployments in mobile systems, including on-device inference, offloading, and model partitioning [75]. In summary, our main contributions are as follows:

•We formalize the input filtering problem and provide validity conditions of a filter. We present the analysis on complexity comparisons between hypothesis families of inference workloads and input filters, which can guide and explain the application of input filtering techniques.

•We propose the first end-to-end learnable input filtering framework that unifies SKIP and REUSE methods. Our framework covers most existing methods and surpasses them in feature embedding with robust discriminability, thus supporting more input modalities and inference tasks.

•We design and implement an input filtering system *InFi*. Comprehensive evaluations on workloads with 6 input modalities, blue 12 inference tasks, and 3 types of mobile-centric deployments show that *InFi* has wider applicability and outperforms strong baselines in accuracy and efficiency. For a video analytics application on a mobile platform (NVIDIA JETSON TX2), *InFi* can achieve up to 8.5× throughput and save 95% bandwidth compared with the naive vehicle counting workload, while keeping over 90% accuracy.

2 INPUT FILTERING

This section formalizes the input filtering problem and provides the conditions of a "valid" input filter for resource-efficient mobile-centric inference.

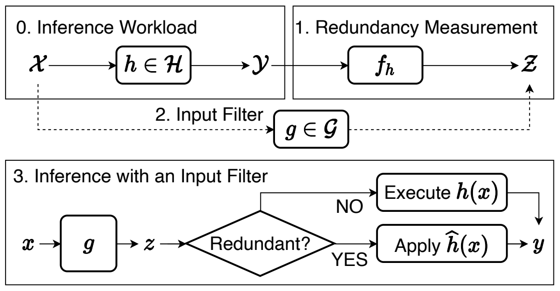


Figure 2: Overview of input filtering for inference workload.

2.1 Problem Definition

An input filtering problem needs to determine what input is redundant and should be filtered for a given inference model. First, the definition of an input filtering problem is based on its target inference model. Let $\mathcal{X}, \mathcal{Y}$ denote the input space and the label space of the target model, respectively. Define $c : \mathcal{X} \to \mathcal{Y}$, named the target concept [64], which provides the ground-truth label for each input. Then training a target model is to search for a function h from a hypothesis family [64] $\mathcal{H}$ using a set of training samples $S = \{(x_i, y_i)\}_{i=1}^{m}$, where $(x_1, ..., x_m)$ are sampled independently from $\mathcal{X}$ with an identical distribution D and $y_i = c(x_i)$. Using the above notations, we define the learning problem of the target inference model h by $(\mathcal{X}, \mathcal{Y}, c, \mathcal{H}, D, S)$. Step 0 in Fig. 2 shows the original inference workflow of a trained model h, which takes an input from $\mathcal{X}$ and returns an inference result $y \in \mathcal{Y}$.

Next, given a trained inference model h, its redundancy measurement function can be defined as:

Definition 1 (Redundancy Measurement). A redundancy measurement $f_h : \mathcal{Y} \to \mathcal{Z}$ of a model h is a function that takes **only** the output of h as input and returns a score that indicates whether the inference computation is redundant.

Such measurements are common in practice. For example, based on the output of a face detector the inference computation that returns no detected face is redundant and can be skipped, and we can set the score $z = 0$; Otherwise, $z = 1$. Formally, $y \mapsto 1(|y| > 0)$, where y is the output set of detected faces, $1(\cdot)$ is the indicator function. For REUSE cases, if the inference result of an action classifier on a new query is the same as previously cached, the computation is redundant, and we can define $f_h(y) = 1(y \notin Y_{cached})$. Note that, this definition of redundancy measurement does not depend on ground-truth labels, since our focus is not the accuracy but to optimize the resource efficiency of a deployment-ready target model with trusted accuracy by eliminating its redundant inference. Step 1 in Fig. 2 shows how redundancy measurement works.

Given the inference workload h and redundancy measurement f_h, as Step 2 in Fig. 2, learning an input filter is defined as searching for a function g from a hypothesis family $\mathcal{G}$ using a set of training samples $S' = \{(x_i, z_i)\}_{i=1}^{n}$, where $(x_1, ..., x_n)$ are sampled independently with a distribution D' and $z_i = f_h(h(x_i))$. This learning problem is denoted by $(\mathcal{X}, \mathcal{Z}, f_h \circ h, \mathcal{G}, D', S')$, i.e., g's target concept is the composite function of f_h and h.

Inference with an input filter. Once an input filter g is trained, the inference workflow changes from Step 0 to Step 3 in Fig. 2. The input filter g becomes the entrance of the workload, which predicts the redundancy score z of each input x. If not redundant, the inference model h will be directly executed on the input. Otherwise, we apply the $\hat{h}(x)$, which has two typical implementations: 1) **SKIP**: skipping the inference computation on the input x and returning a NONE result; 2) **REUSE**: reusing previously cached inference results. Sec. 3 will analyze SKIP and REUSE approaches theoretically, which can guide the usage of them. Sec. 4 will introduce our framework that unifies SKIP and REUSE approaches, and show how to determine whether it is redundant based on the predicted score and how to reuse previous results. Sec. 5 will present the detailed design of our input filtering system.

2.2 Validity Conditions

After defining an input filter, we now give the conditions that a "*valid*" input filter needs to meet for resource-efficient mobile inference. The input filter is designed to balance the resource and accuracy: filtering more inputs can save more resources, but it also brings a higher risk of incorrect inference results.

Inference accuracy. With an input filter, the inference result y for input x is returned either by executing $h(x)$ or applying $\hat{h}(x)$. Following previous work [5, 19, 40], the correctness of the result y refers to its consistency with the exact inference result by $h(x)$, rather than the ground-truth label. An input filter's inference accuracy Acc is defined as the ratio of correct results obtained by the inference workload with the filter.

Filtering rate. The filtering rate, denoted by r, is defined as the ratio of filtered inputs (i.e., the ratio of results obtained by applying $\hat{h}$), which is also an important performance metric considered in previous work [5, 19, 40].

Overall cost. The overhead of an inference workload with an input filter needs to take g, h and $\hat{h}$ into consideration. Let $C(\cdot)$ denote the cost of a certain function. For the cost of computation (e.g., runtime), the average cost per input changes from $C(h)$ into $C(g) + (1 - r)C(h) + rC(\hat{h})$. The communication cost (e.g., bandwidth) depends on the deployment of the mobile-centric inference workload. On-device inference does not involve communication, while the overall bandwidth cost of offloading [5, 40] and model partitioning [75] deployments becomes the original cost multiplied by $(1 - r) < 1$.

Based on the above metrics, we define an input filter as "*valid*" if it satisfies two conditions: 1) **Accurate enough**: $Acc > T_{Acc}$, where T_{Acc} is the threshold of acceptable inference accuracy. 2) **Reduced overhead**: the overall cost with an input filter is lower than the original cost. If we aim to reduce the computation cost, we need $(C(g)+(1-r)C(h)+rC(\hat{h}))/C(h) < 1$, i.e., $r > C(g)/(C(h)-C(\hat{h}))$; If we aim to reduce the communication cost, we only need $r > 0$.

3 FILTERABILITY ANALYSIS

As mentioned in Sec.1, not all inference workloads have the optimization potential by using input filtering techniques. Given an inference workload in a mobile-centric AI application, is there a valid input filter? To answer this question, based on our formalization of the input filtering problem, we first define the *filterability* of an inference workload. Then we analyze filterability in three typical inference cases in SKIP settings, and discuss uncovered cases.

3.1 Definition of Filterability

Given the learning problem $(\mathcal{X}, \mathcal{Y}, c, \mathcal{H}, D, S)$ of an inference model and the learning problem $(\mathcal{X}, \mathcal{Z}, f_h \circ h, \mathcal{G}, D', S')$ of its input filter, to simplify the analysis, we make assumptions as follows: (1) $D = D'$ i.e., the training samples follow the identical distribution; (2) $S' = \{(x_i, z_i)\}_{x_i \in S}$, i.e., the two learning problems share the same inputs in their training samples. But they are supervised under different labels. The inference model h is supervised by $y_i = c(x_i)$, while the input filter g is supervised by $z_i = (f_h \circ h)(x_i)$. Our intuitive idea for filterability is that, if an inference workload is filterable, the learning problem of its input filter should have lower complexity than the learning problem of its inference model. Formally, we define *filterability* as follows:

Definition 2 (Filterability). Let $Complex(\cdot)$ denote the complexity measurement of a hypothesis family. We say that the inference workload is filterable, if $Complex(\mathcal{G}) \leq Complex(\mathcal{H})$, where $h \in \mathcal{H}$ and $(f_h \circ h) \in \mathcal{G}$.

Since the hypothesis family cannot be determined based only on input and output spaces, we use the family of the input filter's target concept $f_h \circ h$ as $\mathcal{G}$.

Now we can characterize the theoretically achievable accuracy and overhead of the input filter for a given inference model by leveraging computational learning theory [50]. It has been proven that, the more complex the hypothesis family is, the worse the bounds of generalization error. On the other hand, the hypothesis complexity of neural networks has a positive correlation with the number of parameters. For example, let W, L denote the number of weights and the number of layers in a deep neural network. The VC-dimension [65] (a measurement of the hypothesis complexity) is $O(WL\log(W))$ [23]. In the case of the same layer structure, the more parameters the higher the inference overhead of neural networks. The generalization error bound and the number of parameters correspond to the accuracy and efficiency metrics in validity conditions (§ 2.2), respectively, although they are not strict quantification. Therefore, if an inference workload is filterable, whose input filter has lower hypothesis complexity, we are confident to obtain a valid filter with sufficiently high accuracy and lower overhead than the inference model. Next, we will analyze the complexities of the hypothesis family of inference workload h and its input filter g in different cases.

3.2 Low-confidence Classification as Redundancy

Considering an inference workload, where the inference model is a binary classifier h that returns the classification confidence, and the redundancy measurement regards the classification result with confidence lower than a threshold t as redundant, i.e., $f_h(y) = \text{sign}(y > t)$. Confidence-based classification is very common in mobile AI applications, such as speaker verification. We adopt the empirical Rademacher complexity [34], denoted by $\widehat{\mathfrak{R}}_S(\cdot)$, as the complexity measurement, which derives the following generalization bounds [50]:

Theorem 3 (Rademacher complexity bounds). *Let $\mathcal{H}$ be a family of hypothesis taking values in $\{-1, +1\}$. Then for any $\delta > 0$, with probability at least $1 - \delta$, the following holds for all $h \in \mathcal{H}$:*

$$R(h) \leq \widehat{R}(h) + \widehat{\mathfrak{R}}_S\mathcal{H}) + 3\sqrt{\frac{\log(2/\delta)}{2m}}, \tag{1}$$

where $R(h)$ and $\widehat{R}(h)$ denote the empirical and generalization errors, and m is the number of training samples.

This theorem shows that the higher a hypothesis family's empirical Rademacher complexity, the worse the bounds of its generalization error. The classification confidence-based redundancy measurement creates two hyperplanes parallel to $h = 0$: points between them are considered redundant, and points outside them are considered not redundant. Thus, the hypothesis family of the input filter's target concept has the form: $\mathcal{G} = \{\text{sign}(h(x)(h(x) + b))\}$, where $h \in \mathcal{H}$ and $b \in \mathbb{R}$. Then we have proven the following lemma, which shows that the discussed inference workload is **not filterable**.

Lemma 4. *Let $\mathcal{H}$ be a family of binary classifiers taking values in $\{-1, +1\}$. For $\mathcal{G} = \{sign(h(h + b))\}$ where $h \in \mathcal{H}, b \in \mathbb{R}$:*

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) \geq \widehat{\mathfrak{R}}_S(\mathcal{H}). \tag{2}$$

Proof. By definition, $\widehat{\mathfrak{R}}_S(\mathcal{H}) = E_\sigma[\sup_{h \in \mathcal{H}}(\frac{1}{m}\sum_{i=1}^m \sigma_i h(x_i))]$ and $\widehat{\mathfrak{R}}_S(\mathcal{G}) = E_\sigma[\sup_{h \in \mathcal{H}, b \in \mathbb{R}}(\frac{1}{m}\sum_{i=1}^m \sigma_i \text{sign}(h(x_i)(h(x_i) + b))]$, where Rademacher variables $\sigma_i \in \{-1, +1\}$. By fixing $b = 2$,

$$\begin{aligned}\widehat{\mathfrak{R}}_S(\mathcal{G}) &\geq E_\sigma[\sup_{h \in \mathcal{H}, x_i \in S}(\frac{1}{m}\sum_{i=1}^m \sigma_i \text{sign}(h(x_i)(h(x_i) + 2))] \\ &= E_\sigma[\sup_{h \in \mathcal{H}, x_i \in S}(\frac{1}{m}\sum_{i=1}^m \sigma_i \text{sign}(h(x_i))] = \widehat{\mathfrak{R}}_S(\mathcal{H}),\end{aligned}$$

where we used the fact that $\text{sign}(h(x_i) + 2) \equiv 1$. □

Multi-class classifiers can be treated as a set of confidence scoring functions, one for each class. The above lemma can also be applied to derive that multi-class classifiers using such a confidence-based redundancy measurement are not filterable either.

3.3 Class Subset as Redundancy

Considering the inference model h as a multi-class mono-label classifier and $\mathcal{Y} = \{y_1, ..., y_l\}$. Then its hypothesis family $\mathcal{H}$ has the form: $\mathcal{H} = \{\max(h_1, ..., h_l) : h_i \in \mathcal{H}_i, i \in [1, l]\}$, where h_i returns the probability of the i-th class. The redundancy measurement checks whether the predicted class belongs to a specific subset, i.e., $f_h(y) = 1(y \in \mathcal{Y}')$, where $\mathcal{Y}' \subseteq \mathcal{Y}$. It is common in mobile applications to select only a subset of labels for use. For example, when deploying a pre-trained common object detector [41] on a drone for traffic monitoring, we only care about the labels of vehicles and pedestrians, while considering other labels like animals and trees as redundancy. With the class subset-based redundancy measurement, the hypothesis family of the input filter's target concept has the form: $\mathcal{G} = \{\max(h_i) : y_i \in Y'\}$. We have proven the following lemma, which shows that the discussed inference workload is **filterable**:

Lemma 5. *Let $\mathcal{H}_1, ..., \mathcal{H}_l$ be l hypothesis sets in $\mathbb{R}^{\mathcal{X}}$, $l \geq 1$, and let $\mathcal{H} = \{\max(h_1, ..., h_l) : h_i \in \mathcal{H}_i, i = 1, ..., l\}$. For $\mathcal{G} = \{\max(h_i) : i \in$*

$J\}$, where $J \subseteq \{1, ..., l\}$:

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) \leq \widehat{\mathfrak{R}}_S(\mathcal{H}). \tag{3}$$

Proof. For any $j = 1, ..., l$:

$$\begin{aligned}\widehat{\mathfrak{R}}_S(\mathcal{H}) &= \frac{1}{m} \underset{\sigma}{E}[\sup_{x_i \in S} \sigma_i \max_{h_k \in \mathcal{H}_k} (h_k(x_i))] \\ &\geq \frac{1}{m} \underset{\sigma}{E}[\sup_{x_i \in S} \sigma_i \max_{j \in J}(h_j(x_i))] = \widehat{\mathfrak{R}}_S(\mathcal{G}).\end{aligned}$$

□

The equation holds only if the max-value scoring function is in the selected subset for all $x_i \in S$, which means that without loss of inference accuracy, the optimal filterable ratio in the data is 0. Except in this extreme case, we can think that the complexity of learning the input filter is strictly lower.

3.4 Regression Bound as Redundancy

Considering a bounded regression model h, whose outputs are bounded by $M \in \mathbb{R}$ that $|h(x) - c(x)| \leq M$ (recall that c is the target concept) for all $x \in X$. The redundancy measurement checks whether the returned value is larger than a threshold, i.e., $f_h(y) = 1(y > T)$. As an example, face authentication on mobile devices usually requires the coordinates of the detected face to be within the specified range. Then learning the target concept of input filter becomes learning a regression model whose outputs are bounded by T, where $T < M$. We also adopt the empirical Rademacher complexity and have the following theorem [50]:

Theorem 6. *Let $p \geq 1$ and $\mathcal{H} = \{x \mapsto |h(x) - c(x)|^p : h \in H\}$. Assume that $|h(x) - c(x)| \leq M$ for all $x \in X$ and $h \in H$. Then the following inequality holds: $\widehat{\mathfrak{R}}_S(\mathcal{H}) \leq pM^{p-1}\widehat{\mathfrak{R}}_S(H)$.*

Since $M > T$, this theorem shows that the upper bound of $\widehat{\mathfrak{R}}_S(\mathcal{G})$ is tighter than the upper bound of $\widehat{\mathfrak{R}}_S(\mathcal{H})$. So we can be confident that the bounded regression inference workload discussed is **filterable**.

3.5 Discussions

Guidance for applying SKIP. The above results are derived in SKIP settings. Our experiments (§ 6.3) show that the theoretically filterable cases achieve significantly better filtering performance than the cases that are not filterable. So our filterability analysis can reveal the optimization potential by using SKIP filters, and explain the filtering performance.

Other inference tasks. Classification and regression are the most common inference tasks, and the three redundancy measurements discussed are widely adopted [5, 71, 72]. However, there are some inference tasks that are challenging to measure the hypothesis complexity, like reinforcement learning [13] and structured learning [12]. Besides, their redundancy measurements are typically ill-defined. We believe that our problem formalization and analysis approach are general, based on which we will analyze the filterability of other tasks in the future work.

Characteristics of REUSE. For REUSE approach, we cannot determine the hypothesis family of the input filter's target concept. Here we only give one necessary condition: the inference result is discrete or can be discretized. For example, classification and counting models return discrete outputs. But continuous localization coordinates of detection models cannot be reused directly, unless reusing detection results with high IoU are regarded as correct, which is equivalent to discretizing the outputs.

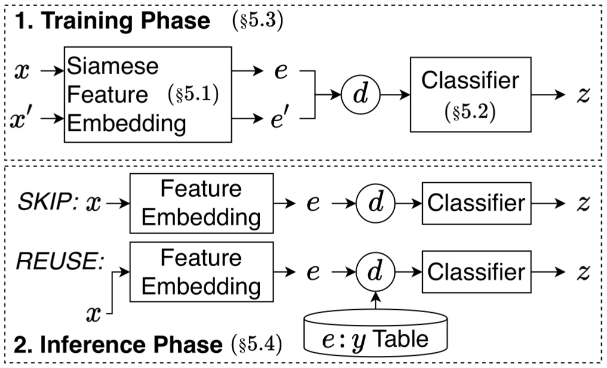


Figure 3: Unified and end-to-end learnable framework for both SKIP and REUSE input filtering.

4 FRAMEWORK

In this section, we first propose a novel input filtering framework that unifies SKIP and REUSE approaches. Then we discuss how existing approaches are covered by our framework and their limitations. Finally we present the key design, end-to-end learnability, and advantages it brings.

4.1 SKIP as REUSE

We unify SKIP and REUSE approaches based on the idea that:

SKIP equals to REUSE the NONE output of $h(\vec{0})$.

Suppose we have an all-zero input $\vec{0}$ and apparently its inference result can be interpreted as NONE. Then given a new input x, if it is similar to $\vec{0}$ in the feature space, we can REUSE the cached NONE result, i.e., we SKIP the inference computation. The key to reuse is to measure the semantic similarity between the current input and previously cached ones. However, it is difficult to accurately measure semantic similarity directly based on the raw input. As Step 1 in Fig. 3 illustrated, our framework first computes the feature embedding of each raw input. Taking a pair of inputs x, x', then our framework applies a difference function d on their corresponding embeddings e, e' and feeds the result into a classifier that predicts a single scalar z. Under this framework, for SKIP, we fix x' as an all-zero input $\vec{0}$, then the process degenerates to a binary classification task that takes x as input and returns the prediction z. In this way, our framework unifies SKIP and REUSE approaches, with only difference in interpretation of the value z. For REUSE, we interpret z as the distance between two inputs. For SKIP, we interpret z as the probability that input x is not redundant.

4.2 Inference with an Input Filter

For the inference phase, as shown in Step 2 in Fig. 3, SKIP and REUSE filters only differ in the inputs of the difference function d. (1) **SKIP:** Inference with a SKIP filter is the same as serving a binary classifier. We can set a threshold on the predicted redundancy score z to determine whether to skip. (2) **REUSE:** Inference with a REUSE filter needs to maintain a key-value table, where a key is a feature embedding and its value is the corresponding inference result. For an arrived input x, the trained feature embedding network returns its embedding e and the distances z between e and cached keys are computed by the difference function d and the trained classifier. Then we can leverage classification algorithms, e.g., KNN, to obtain the reusable cached results.

4.3 Sub-instance Approaches

Here we explain how our framework covers three state-of-the-art input filtering methods [5, 19, 40] that will be used for comparison in our evaluations.

Sub-instance1: FilterForward (FF) [5] is a SKIP method for image input. FF uses a pre-trained MobileNet's intermediate output as the feature embedding. Then it trains a "micro-classifier" that consists of convolution blocks to make the binary decision for filtering.

Sub-instance 2: FoggyCache (FC) [19] is a REUSE method for image and audio input. FC uses low-level features (SIFT for image, MFCC for audio) and applies locality-sensitive hashing (LSH) for embedding. Then FC uses L2 norm as the difference function and applies KNN to get the reusable inference results from previously cached ones.

Sub-instance 3: Reducto [40] is a variant of SKIP method for video input. It measures low-level feature (pixel, edge, corner, area) difference between successive frames. If they are similar enough, Reducto skips the current frame and returns the latest result. Formally, let x be the current frame and x' be the previous frame. Reducto defines $d(e, e') = (e - e')/e'$, where e, e' are low-level features of x, x'. It uses a threshold function as the classifier, i.e., $\mathbb{1}(d(e, e') > T)$.

Limitations of sub-instances. First, the existing non-end-to-end methods lack theoretical abstraction of the problem. They split the filtering task into sub-modules and propose bespoke solutions for each module, leading to difficulty in performing theoretical analysis. Second, though the handcrafted features (like Pixel and SIFT [19, 40]) or pre-trained feature embeddings (like MobileNet [5]) are very efficient, they are not optimal in accuracy [17] and result in limited support for input modalities and unguaranteed discriminability to diverse tasks. Input modalities like text and sensor signals are also very common for mobile AI applications, which, however, haven't been supported by existing filters yet. Our experiments also show that these non-end-to-end learned features are poorly discriminative in many cases.

4.4 End-to-end Learnability

To obtain feature with robust discriminability for diverse data modalities and inference tasks in mobile applications, a key design principle of our framework is the end-to-end learnability. End-to-end learning system casts complex processing components into coherent connections in deep neural networks [17] and optimizes itself by applying gradient-based back-propagation algorithms all through the networks. Deep end-to-end models have shown state-of-the-art performance on various tasks including autonomous driving [54] and speech recognition [2]. As aforementioned, a main component of our unified framework is to measure the semantic similarity between two inputs. To make our framework end-to-end learnable, we leverage the metric learning [37] paradigm, whose goal is to learn a task-specific distance function on two objects. The metric learning paradigm turns the fixed difference function d (e.g., Euclidean distance and L2 norm) used by existing methods into an end-to-end learnable network. Within the metric learning paradigm, we adopt Siamese network structure [36] for feature embedding to support two inputs and flexible input modalities. Siamese network uses the same weights while working on two different inputs to compute comparable output vectors, and has been successfully applied in face verification [58], pedestrian tracking [38], etc. We can flexibly implement the Siamese feature embedding by incorporating different neural network blocks to learn modality-specific features in an end-to end manner, instead of tailoring handcrafted or pre-trained feature modules. Our experimental results show that our end-to-end learned features have robust discriminability to diverse inference workloads in mobile-centric AI applications.

5 DESIGN OF INFI

Based on our input filtering framework, in this section, we present the concrete design of *InFi* (INput FIlter), which supports both SKIP and REUSE functions, named *InFi*-Skip and *InFi*-Reuse. The design of *InFi* has four key components: feature embedding, classifier, training mechanism, and inference algorithm. We also discuss diverse deployments of *InFi* in AI applications on mobile, edge, and cloud devices.

5.1 Feature Networks for Diverse Input Modalities in Mobile-centric AI

InFi supports filtering inference workloads with six typical input modalities in mobile applications: text, image, video, audio, sensor signal, and feature map. We develop a collection of modality-specific feature networks as building blocks for learning feature embedding. Our major consideration in designing these feature networks is resource efficiency on mobile devices.

Text modality (g_{text}). Text is tokenized into a sequence of integers, where each integer refers to the index of a token. We adopt the word-embedding layer to map the sequence to a fixed-length vector by a transformation matrix and use a densely connected layer with a Sigmoid activation to learn the text features.

Image modality (g_{image}). We use depth-wise separable convolution [11], denoted by *SepConv*, to learn visual features. *SepConv* is a parameter-efficient and computation-efficient variant of the traditional convolution which performs a depth-wise spatial convolution on each feature channel separately and a point-wise convolution mixing all output channels. Then we build residual convolution blocks [24] *ConvRes* as follows:

$$\begin{aligned} &ConvStep(x) = LN(SepConv(ReLU(x))),\\ &c_1(x) = ConvStep(x), c_2(x) = ConvStep(c_1(x)),\\ &ConvRes(x) = MaxPool2D(c_2(x)) + ConvStep(x), \end{aligned}$$

where *ReLU* denotes the rectified linear unit, *LN* denotes the layer normalization and *MaxPool2D* denotes the 2D max-pooling layer. Finally, we build the image feature network with two residual blocks followed by a global max-pooling layer and a Sigmoid-activated dense layer.

Video modality (g_{video}). For video modality, we need to represent not only the spatial but also the temporal features. Given a window of frames, we stack one residual block for each frame and then concatenate their resulting feature maps. Except for the first residual block, the video feature network performs the same operation as the image feature network.

Audio modality (g_{audio}). We consider audio inputs in the form of either a 1D waveform or a 2D spectrogram and use the same structure as image feature networks to learn features from audio.

Sensor signal and feature map modality (g_{vec}). Motion sensors are widely used in mobile devices and play a key role in many smart applications, e.g., gyroscope for augmented reality [29] and accelerator for activity analysis [3]. Feature maps refer to the intermediate outputs of deep models and need to be transmitted in workloads that involve model partitioning [75]. We consider these two types of input as a vector with fixed shape and use two densely connected layers to learn the feature embedding from the flattened vector.

Flexible support for input modalities. Our design provides a flexible support for diverse input modalities in mobile-centric AI applications. We can easily integrate a modality-specific neural network from advanced machine learning research as the feature network block into our framework, so as to learn feature embeddings in the end-to-end way.

5.2 Task-agnostic Classifier

Each feature network $g_{modality}$, where *modality* belongs to {text, image, video, audio, vec}, takes x as input and output the embedding emb. We add a dropout layer after the last dense layer of feature networks to reduce overfitting. Following previous design of Siamese network [36], we use the absolute difference as the function d. Let emb_1, emb_2 denote the embedding outputs of two inputs x_1, x_2. The classifier is defined as $g_{cls} = \sigma(\sum_j w_j |emb_1^{(j)} - emb_2^{(j)}| + b)$, where $emb^{(j)}$ denotes the j-th element in the embedding vector. To sum up, the input filter function $g : \mathcal{X} \rightarrow \mathcal{Z}$ can be defined as $g(x) = (g_{cls} \circ g_{modality})(x)$. With a proper implementation, the modality of input data can be automatically detected without manually setting.

5.3 End-to-end Training

InFi-Skip and *InFi*-Reuse share the same model architecture, but have different formats of training data. 1) Learning an *InFi*-Skip filter uses the same paradigm as training a binary classifier. Thus its training samples are $(x_i, f_h(h(x_i)))_{i=1}^n$. In practice, we can use the original training set of h or data collected during serving h. Since f_h only depends on the inference result, the supervision labels can be collected automatically. We use the binary cross-entropy loss function $\mathcal{L} = z \log(g(x)) + (1 - z) \log(1 - g(x))$, where z denotes the redundancy measurement label. 2) *InFi*-Reuse filters are trained using the contrastive loss [21] with a margin parameter of one. Given a set of input and their discrete inference results, the redundancy measurement is defined as the distance metric between a pair of inputs. Formally, a training sample consists of a pair of inputs and their distance label $(x_i, x_j, 1(y_i \neq y_j))$.

Algorithm 1: Inference with an *InFi* Filter

```
input: input source $src$, redundancy threshold $T$, cache size $s$, KNN
       parameter $K$, homogeneity threshold $\theta_T$
1  def InFiSkip(src):
2      while $x \leftarrow$ read($src$) do
3          if $g(x) > T$ then $y \leftarrow$ inference($x$);
4          else $y \leftarrow None$;
5  def InFiReuse(src):
6      Initialize empty $cache$;
7      while $x \leftarrow$ read($src$) do
8          if Len($cache$) $< s$ then
9              $y \leftarrow$ inference($x$);
10             $cache[g_{modality}(x)] \leftarrow y$;
11         else
12             $y, \theta \leftarrow$ HKNN($cache, g_{modality}(x), g_{cls}, K$);
13             if $\theta < \theta_T$ then
14                 $y \leftarrow$ inference($x$);
15                 replace ($cache$, {$g_{modality}(x) : y$});
```

We can optimize all trainable parameters end-to-end, using standard back-propagation algorithms. In this paper, we focuses on the potential of end-to-end learnability and regard the training of filters as an offline process. In the future, we will consider the dynamic nature of data content and explore online update of *InFi*.

5.4 Inference Phase

After training an *InFi* filter, we integrate it into the original inference workload using Alg. 1.

***InFi*-Skip.** We set a redundancy threshold for *InFi*-Skip to determine whether to skip the current input. And if we skip the input, *InFi*-Skip will return a NONE result, whose interpretation depends on the redundancy measurement in specific applications. For example, NONE means no face detected in face detection, 0 vehicle in vehicle counting application, meaningless speech in speech recognition, etc.

***InFi*-Reuse.** To reuse previous inference results, we need to maintain a cache whose entry is a key-value pair of an input embedding and its inference results. Following the previous RESUE approach [19], we adopt K-Nearest Neighbors (KNN) algorithm to reuse cached results. But it is possible that a new input is not similar with any cached entries, i.e., a cache miss. We adopt Homogenized KNN (H-KNN) [19] algorithm to handle this problem, which calculates a homogeneity score θ of the found K nearest neighbors and sets a threshold θ_T on the homogeneity score to detect the cache miss. Then we can replace entries using policies like least frequently used (LFU), denoted by `replace` in Alg. 1. Different from original KNN that typically uses Euclidean distance, which is non-parametric, we set the distance measurement as the trained g_{cls}. We denote HKNN($cache, emb, g_{cls}, K$) as the H-KNN function which returns the majority inference result y of K nearest neighbors of emb in $cache.keys$ using the g_{cls} to calculate the distance between embeddings, and computes θ. We focus on taking the advantage of end-to-end learnability, and other subtle optimization opportunities such as cache warm-up are out of the scope of this work.

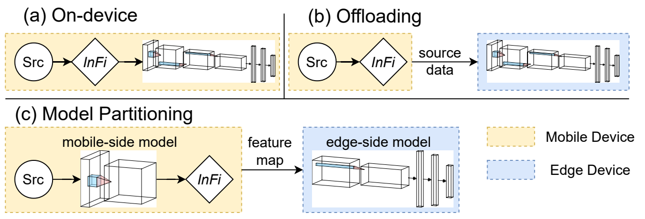


Figure 4: Three deployments of inference workload and *InFi* on mobile-edge systems.

5.5 Mobile-centric Deployments

Unlike existing work tailored for specific deployment, e.g., inference offloading [5, 19, 40], *InFi* supports diverse mobile-centric deployments: (1) **On-device:** both inference model and input filter are deployed on one device; (2) **Offloading:** the input filter is deployed on one device, and the inference model is deployed on another device. (3) **Model Partitioning (MP)** [75]: the inference model is partitioned across two devices, and the input filter is deployed with the first part. MP is a promising approach to collaboratively make use of the computing resources of mobile and edge devices [44, 61] and better protect the privacy of mobile data [51]. For MP deployment, the filter's input is the feature map, so existing filtering approaches [5, 19, 40] cannot be applied. Due to the support of feature map modality, *InFi* is the first input filter that can be applied in model partitioning workloads. Fig. 4 illustrates the three typical deployments on mobile and edge devices. Note that *InFi* is not limited to systems with a single mobile and edge node. For example, training one filter per server, or changing one filter's binary classifier into a multi-category one (one bit per server), *InFi*-Skip can be used in the multi-tenancy context [5].

6 EVALUATION

In this section, we evaluate *InFi* in 12 mobile-centric AI applications, covering six input modalities. The comparisons with three strong baselines [5, 19, 40] show that *InFi* has wider applicability and outperforms them in both accuracy and efficiency. Compared with a native vehicle counting workload in a city-scale video analytics application, *InFi*-Skip / *InFi*-Reuse can achieve 1.59-2.9× / 1.87-8.5× throughput and save 66.5-70.7% / 91.1-95.0% bandwidth, respectively, while keeping over 90% inference accuracy (see Tab. 5). Experiments also confirm the theoretical results in Sec. 3.

6.1 Implementation and Configurations

We implemented *InFi* [1] in Python. We build all feature networks and classifiers with TensorFlow 2.4 [60], and train them using the RMSprop [26] optimizer. Learning rate is set as 0.001, batch size is 32, and the number of training epochs is 20. In the text feature network, the output dimension of embedding layer is 32. In image, video, and audio feature networks, we use 32 and 64 convolution kernels in the two residual blocks. We use 128 units in the first dense layer in vector feature networks. The last dense layer of all feature networks has 200 units and 0.5 dropout probability. We train models on Ubuntu 16.04 and use TFLite [42] to transform saved checkpoints into .tflite files. For Android, we use Java API to run inference using the `Interpreter` object that loads the .tflite file.

[1]https://github.com/yuanmu97/infi

Table 1: Datasets and Inference Workloads

Dataset	Modality	Inference Task
Hollywood2	Video Clip	Action Classification (AC)
	Image	Face Detection (FD) Pose Estimation (PE) Gender Classification (GC)
	Audio	Speech Recognition (SR)
	Text	Named Entity Recognition (NER) Sentiment Classification (SC)
ESC-10	Audio	Anomaly Detection (AD)
UCI HAR	Motion Signal	Activity Recognition (HAR)
MoCap	Motion Signal	User Identification (UI)
City Traffic	Video Stream	Vehicle Counting (VC)
	Feature Map	Vehicle Counting (VC-MP)

Datasets and inference models. To evaluate *InFi*'s wide applicability, we choose 10 inference workloads that cover six input modalities and three deployments (see Tab. 1). Five datasets are used: (1) We reprocessed a standard video dataset, Hollywood2 [46], to create four different input modalities: video clip, image, audio and text. An action classification model [62] is deployed on the original video clips. Images are sampled from the video clips and a face detection [56], a pose estimation [6] and a gender classification [56] models are deployed. Audio is extracted from each video clip and we deploy a speech recognition model [2]. Text is the caption generated on sampled images by an image captioning model [69]. A named entity recognition model [27] and a sentiment classification model [35] are deployed. (2) We use ESC-10 dataset [53] for audio anomaly detection and deploy an transformer-based model [18]. (3) We use UCI HAR dataset [3] for motion signal-based human activity recognition and deploy a LSTM-based model [10]. (4) We use MoCap dataset [14] for training a motion signal-based user identification (12 users) model, using a LSTM-based architecture [10], and deploy it as the inference workload. (5) We collected a video dataset, named City Traffic, from a real city-scale video analytics platform. We collected 48 hours of videos (1FPS) from 10 cameras at road intersections and deploy YOLOv3 re-implemented with TensorFlow 2.0 [76] to count the number of vehicles in video frames. All deployed inference models load publicly released pretrained weights. And we split each dataset for training and testing by 1:1 (Hollywood2 and UCI HAR are split randomly, while City Traffic is split by time on each camera).

Devices and deployments. We use an edge server with one NVIDIA 2080Ti GPU and three mobile platforms: (1) NVIDIA JETSON TX2 [63], (2) XIAOMI Mi 5, and (3) HUAWEI WATCH. All device-independent metrics are tested on the edge. For vehicle counting, we test three deployments: on-device, offloading, and model partitioning (see Sec. 5.5).

Baselines. We adopt three strong baselines: FilterForward (FF) [5], Reducto [40], and FoggyCache (FC) [19]. See Sec. 4.3 for details of

Table 2: Filtering rate @ 90% inference accuracy of SKIP methods.

Method	FD	PE	GC	AC	VC	AD
FF	0.0%	14.5%	0.0%	0.0%	48.0%	/
Reducto	/	/	/	/	48.6%	/
InFi-Skip	36.1%	18.9%	33.1%	56.0%	66.5%	75.4%
Optimal	64.8%	34.4%	71.8%	91.2%	77.7%	86.8%
Method	**SR**	**NER**	**HAR**	**UI**	**SC**	**VC-MP**
InFi-Skip	44.1%	26.8%	91.2%	72.4%	22.5%	70.7%
Optimal	59.9%	34.4%	91.8%	79.8%	63.8%	77.7%

Table 3: Filtering rate @ 90% inference accuracy of REUSE methods.

Method	GC	AC	HAR	SC	VC-MP	VC
FC	66.1%	13.2%	/	/	/	59.4%
InFi-Reuse	**98.8%**	**32.1%**	**98.3%**	**43.4%**	**95.0%**	**91.1%**

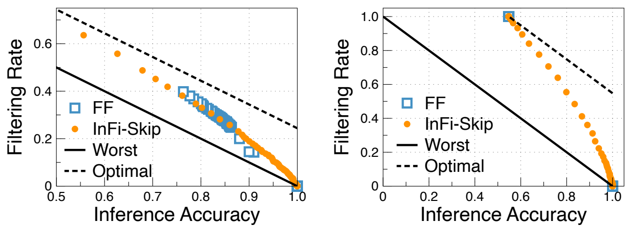
baselines. For workloads with no existing method presented (to our best knowledge), we tested a method dubbed *Low-level* that first computes low-level embedding for inputs (MFCC for audio, Bag-of-Words for text, raw data for motion signal and feature map). Then *Low-level* uses K-nearest neighbors vote (K=10) for both SKIP and REUSE cases. We also deployed YOLOv3-tiny [1] model for vehicle counting and a lightweight pose estimation model [52] to compare input filtering and model compression techniques.

6.2 Inference Accuracy vs. Filtering Rate

First, we test two device-independent metrics (inference accuracy and filtering rate) on the ten inference workloads. We adjust the confidence threshold in FF, Reducto and *InFi*-Skip, and the ratio of cached inputs in FC and *InFi*-Reuse, from 0 to 1 with 0.01 interval.

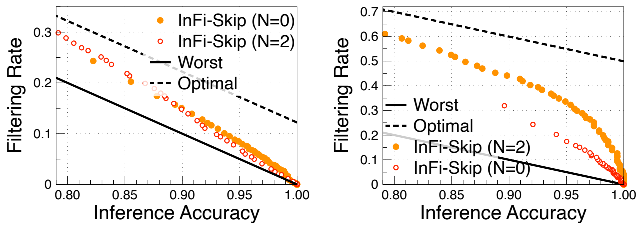
Redundancy measurements. (1) SKIP: For FD (PE), outputs with no detected face (person keypoints) are redundant. For GC (SC), outputs with classification confidence less than a threshold, CONF, are redundant. For AC, outputs that are not in a subset of classes, Sub, are redundant. For SR, outputs with the number of recognized words less than a threshold, N, are redundant. For NER, outputs without entity label "PERSON" are redundant. For HAR, outputs that are not "LAYING" are redundant. For UI, outputs that do not belong to the first 6 users are redundant. For AD, outputs that are not in {"Cry, Sneeze, Firing"} (anomaly events) are redundant. For VC and VC-MP, outputs with zero count are redundant. (2) REUSE: Experimental results show that cache miss happens rarely, so the homogeneity threshold is set as 0.5. We regard inputs that hit the cache as redundant. For the VC (-MP), since we have 86K images from each camera, a fixed cache ratio can lead to serious inefficiency in the KNN algorithm. We fix the cache size as 1000 and reinitialize the cache every 5000 frames. For other inference workloads, we set a fixed cache size according to the cache ratio.

Overview of results. Tab. 2 and Tab. 3 summarize the results of SKIP and REUSE methods. Following related work [40], we report the filtering rates at 90% inference accuracy. The optimal results are computed by (1-0.9)+r_N where r_N denotes the ratio of redundant inputs in the test dataset. Results show that *InFi*-Skip outperforms FF and Reducto on all 10 workloads with significantly higher filtering rate and wider applicability. Similarly, *InFi*-Reuse significantly outperforms FC on all 6 applicable workloads. *InFi*-Skip can filters 18.9%-91.2% inputs and *InFi*-Reuse can filters 32.1%-98.8% inputs, while keeping more than 90% inference accuracy. For all workloads, *Low-level* method cannot achieve 90% inference accuracy unless no input is filtered (i.e. 0.0% filtering rate), and we omit these results in the tables.



(a) Pose Estimation (b) Face Detection

Figure 5: Comparison between FF and *InFi*-Skip filters on visual detection workloads.



(a) SR (N=0) (b) SR (N=2)

Figure 6: *InFi*-Skip filters on speech recognition workloads. N is the minimal number of recognized words.

Feature discriminability. By comparing FF and *InFi* on FD, PE, GC, and AC workloads, we evaluate the discriminability of our end-to-end learned features. As shown in Fig. 5, FF works on the pose estimation workload, but not on the face detection workload. The "Worst" case is calculated by $r = 1 - Acc$. The reason may be that there is a "person" label in the ImageNet dataset, so the pretrained feature embedding in FF is discriminative for determining whether there is a human pose. However, on other tasks (e.g., FD, GC and AC), the pretrained features are not discriminative and FF can only provide two extreme filtering policies: either filtering all input or filtering nothing, which is useless in practice. On the contrary, *InFi*-Skip learns feature embedding with robust discriminability and performs well on all four workloads. With over 90% inference accuracy, *InFi*-Skip can filter 18.9% and 36.1% inputs for PE and FD workloads, respectively.

Transferability. One interesting question is, how transferable is the trained filter to workloads with a looser or tighter redundancy measurement? We set the minimal number of recognized words,

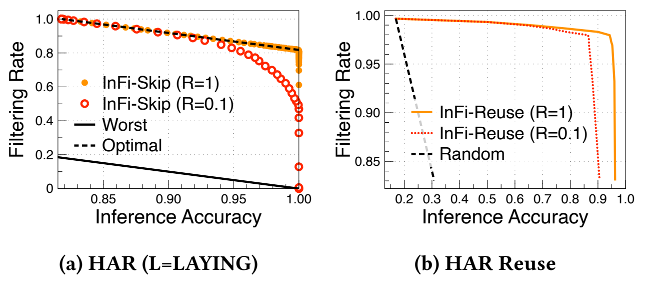


Figure 7: *InFi* filters on HAR inference workloads. R denote the ratio of training samples used. The "Random" case labels each input randomly.

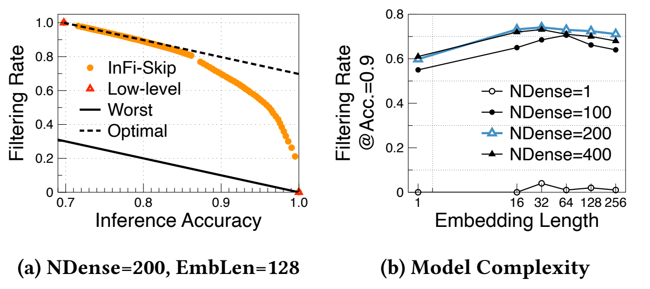


Figure 8: *InFi*-Skip on UI workload. NDense is the number of dense units. EmbLen is the length of embedding.

N, as 0 and 2 and train two *InFi*-Skip filters. Then we test the two filters on two test sets with different N. As shown in Fig. 6, the performance of *InFi*-Skip (N=2) is closed to *InFi*-Skip (N=0) when tested with N=0, however, the performance of *InFi*-Skip (N=0) is apparently worse when tested with N=2. An intuitive explain is that the learned feature with a looser redundancy measurement covers the one with a tighter redundancy measurement, while the opposite is not true.

Sensitivity to training size. We further divide training splits to sets with different size. As shown in Fig. 7, using only 10% samples from the training set, *InFi* can still achieve near-optimal performance on HAR workload. Let R denote the ratio of training samples used for training. When achieving over 95% inference accuracy, *InFi*-Skip (R=1) filters 86.4% inputs while *InFi*-Skip (R=0.1) still filters 81.1%. For high-accuracy reuse, the impact of training size is relatively greater. When filtering 90% inputs, *InFi*-Reuse (R=1) can achieve 95.9% inference accuracy, while the accuracy of *InFi*-Reuse (R=0.1) decreases to 88.1%.

Sensitivity to model complexity. To explore the relationship between the complexity and performance of input filters, we trained *InFi*-Skip filters for the UI workload using different length of embedding (1, 16, 32, 64, 128, 256) and number of dense units (1, 100, 200, 400) in the classifier. And we measure the performance by the maximum filtering rate when achieving 90% inference accuracy. As shown in Fig. 8b, except for extreme cases (e.g., single dense or embedding unit), the filtering performance is relatively robust.

Sensitivity to K in KNN. The parameter K in KNN affects the classification accuracy. We vary K from 1 to 20 and test the REUSE

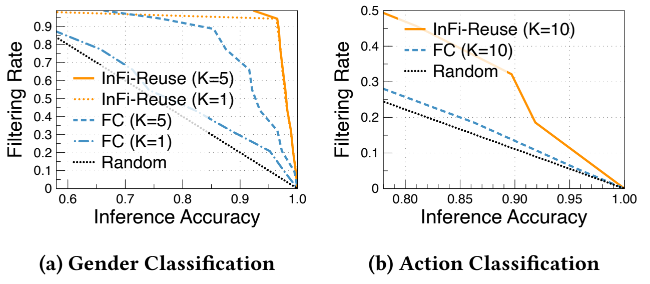


Figure 9: Comparison of FC and *InFi*-Reuse on visual classification workloads. K is the parameter in KNN.

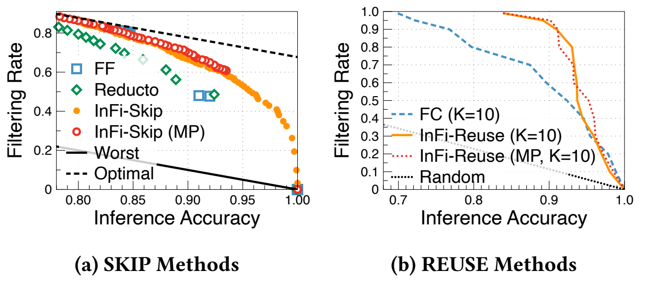


Figure 10: Comparisons of filters on VC and VC-MP.

filters' performance. As shown in Fig. 9, on GC workload, *InFi*-Reuse is robust to varied K parameters, while FC suffers serious performance degradation. For example, with 90% inference accuracy, FC (K=5) can filter 68.4% inputs, while FC (K=1) can only filter 27.3% which is slightly higher than the random guess (20%). On the contrary, *InFi*-Reuse (K=1,5) can all achieve a 94.3% filtering rate with more than 95% inference accuracy. For the AC workload, the results show that the handcrafted feature SIFT is not discriminative, and all tested K parameters lead to similar performance with random labeling. *InFi*-Reuse can learn an action-related discriminative feature, it can filter 18.6% inputs and keep more than 90% inference accuracy (K=10).

Comparisons on VC(-MP) workloads. Unlike other datasets, the video frames arrive in time order rather than randomly. For VC-MP, we partition the YOLOv3 model to mobile-side (the first 39 layers) and edge-side (the rest layers). As shown in Fig. 10, *InFi* outperforms FF, Reducto, and FC, and also is the only applicable method for the VC-MP workload. With over 90% inference accuracy, *InFi*-Skip achieves 66.5% filtering rate, while FF and Reducto achieve 48.0% and 48.6%, respectively; *InFi*-Reuse filters 31.7% more inputs than FC when K=10. The results show the superiority of end-to-end learned features over handcrafted and pre-trained ones.

6.3 Filterability

In Sec. 3, we compare the hypothesis complexity of the inference and filter models. Let "Conf.>T" denote the low-confidence classification case (§ 3.2), "Class Subset" denote the redundant class subset case (§ 3.3), and "Reg.>T" denote the bounded regression case (§ 3.4). GC and SC belong to the "Conf.>T" case, where T is 0.9. AC, NER, and HAR belong to the "Class Subset" case, where

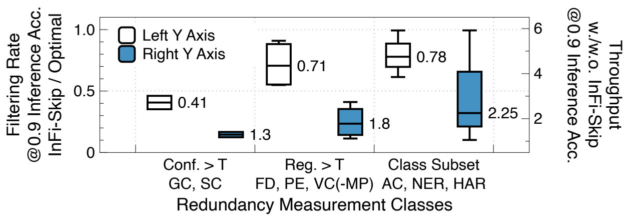


Figure 11: Comparison of filterable and non-filterable cases.

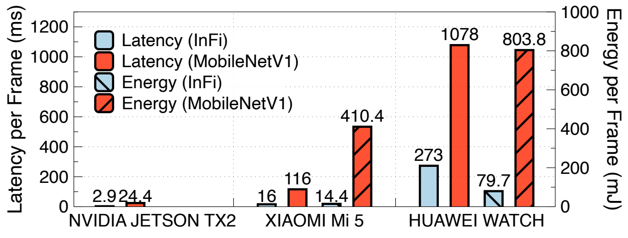


Figure 12: Latency and energy costs of *InFi* (image modality) and MobileNetV1 on mobile platforms.

Table 4: Computation Complexity of Visual Models

Architecture	Param#	MFLOPs
VGG19	1.44E+08	39,285
ResNet101	4.47E+07	15,158
ResNet50	2.56E+07	7,728
MobileNetV1	4.25E+06	1,137
FF (full frame)	3.12E+05	592
InFi-Skip (image modality)	2.47E+04	351

AC selects 2 action labels, NER selects "PERSON" label, and HAR selects "LAYING" label. FD, PE and VC(MP) belong to the 'Reg.>T' case, where T is 0. SR is a sequence-to-sequence model, which cannot perfectly fits any of these three cases. We compute the ratio of resulting filtering rate to the optimal filtering rate at 90% inference accuracy to compare the filterability of different cases. From a practical perspective, we evaluate the overall throughput with and without *InFi*-Skip filters. As shown in Fig. 11, the "Conf.>T" case which we proved that the filter's complexity is not less than the inference model's achieves obviously lower filtering ratio (0.41 median), while other cases which we proved that the filter tends to be less complex achieve apparently higher ratios (0.71/0.78 medians). On the other hand, the overall throughput improved by *InFi*-Skip filters on filterable cases is more significant than the non-filterable cases. In the non-filterable cases, GC and SC, *InFi* achieves around 1.3× throughput, while in the filterable cases, it can improve the throughput up to 5.92× and achieves 1.8 and 2.25 medians for regression and subset-class cases, respectively. These results show the guiding significance of our proved filterability in real applications.

6.4 Computation and Resource Efficiency

As we discussed in Sec. 2.2, a "valid" filter should be both accurate and lightweight. The above results have shown that *InFi* can filter significant amount of inputs while keeping accurate inference. Now, we consider two device-independent metrics: Float Operations (FLOPs) and the number of Parameters (Param#). We choose standard CNNs models, FF and *InFi* with image feature network for comparison. The FLOPs is calculated by TensorFlow profiler API `float_operation`. As shown in Tab. 4, even compared with MobileNetV1 which is designed for efficiency, *InFi* reduces about 70% FLOPs and 99.4% parameters. FF (full frame) denotes the overall model that consists of the first 43 layers of MobileNetV1 and the full-frame micro-classifier porposed by FF. In the training phase, *InFi* (image modality) takes around 710 ms per batch (batch size is 32) and requires 5337 MB GPU memory which most commercial GPUs can meet. *InFi* for other input modalities requires far less resources, e.g., *InFi* (vector) tasks 3 ms per batch and needs only 435 MB memory. Next, we test the latency and energy in the inference phase on mobile platforms. As a fair comparison, we chose the TFLite-optimized MobileNetV1 [49], which is one of the most efficient CNNs on mobile devices. As shown in Fig. 12, on three mobile platforms, *InFi* with the image feature network costs only 12-25% runtime of MobileNetV1. The average energy costs of *InFi* are 14.4/79.7 mJ per frame, which are much lower than MobileNetV1 (410.4/803.8 mJ per frame) on the phone/smartwatch. We implement *InFi* with MindSpore [48] and the results show that *InFi*'s low-energy consumption and low-latency execution do not depend on the implementation framework.

6.5 Different Mobile-centric Deployments

Now we evaluate the overall performance of inference workloads in real systems with three ways of deployments. First, we consider the vehicle counting workload: 1) on-device: *InFi* (image) and YOLOv3 model on TX2; 2) offload: *InFi* (image) on TX2 and YOLOv3 model on edge; 3) model partitioning (MP): first 39 layers (10 convolution blocks) of YOLOv3 and *InFi* (feature map) on TX2, rest of YOLOv3 on edge server. The average throughput of YOLOv3 model on TX2 and edge is 3.2 FPS and 22.0 FPS, respectively. For MP deployment, the edge-side model serves 24.5 FPS. We report the average throughput and the bandwidth saving of using *InFi*-Skip and *InFi*-Reuse, with over 90% inference accuracy, in Tab. 5. As a fair comparison, we test the throughput of YOLOv3-tiny [1] model, a compressed version for YOLOv3. The inference accuracy of YOLOv3-tiny is only 67.9% which does not meet the 90% target. Breaking down the overheads, *InFi*'s inference costs around 3 ms per frame, and the average latency of KNN is 6 ms per frame with K=10 and cache size=1000. Achieving over 90% inference accuracy, *InFi*-Skip improves the throughput to 9.3/55.2/39.0 FPS for on-device/offload/MP deployments, respectively. Apparently, in vehicle counting workloads, there are more filtering opportunities for *InFi*-Reuse. *InFi*-Reuse improves the throughput to 27.2/77.2/46.0 FPS for these three deployments. Except the on-device deployment that does not involve cross-device data transmission, *InFi*-Skip / *InFi*-Reuse also save 66.5% / 91.1% and 70.7% / 95.0% bandwidth for offloading and MP workloads. Unlike YOLOv3-tiny which trades a significant and fixed loss of accuracy for efficiency, *InFi* provides

Table 5: Throughput (FPS) / Bandwidth saving (%) of vehicle counting workloads. Acc. denotes inference accuracy, compared with vehicle count results of YOLOv3.

Workload	YOLOv3	*InFi*-Skip	*InFi*-Reuse	YOLOv3-tiny
Acc. (%)	100	90.3	90.5	67.9
On-device	3.2/-	9.3/-	27.2/-	20.4/-
Offloading	22.0/-	55.2/66.5	77.2/91.1	225.3/-
MP	24.5/-	39.0/70.7	46.0/95.0	230.4/-

Table 6: Throughput (FPS) / Bandwidth saving (%) of pose estimation workloads.

Workload	OpenPose	*InFi*-Skip	OpenPose-light
Inference Accuracy (%)	100	90.1	76.5
On-device	15.4/-	18.0/-	28.1/-
Offloading	27.7/-	31.5/18.9	98.5/-
MP	29.2/-	33.1/20.2	102.4/-

a flexible trade-off between the inference accuracy and overheads. Second, we evaluate the pose estimation workload: 1) on-device: *InFi* (image) and OpenPose model on TX2; 2) offload: *InFi* (image) on TX2 and OpenPose model on edge; 3) model partitioning (MP): first 39 layers (10 convolution blocks) of OpenPose and *InFi* (feature map) on TX2, rest of OpenPose on edge server. Also, we test the throughput of OpenPose-light [52] model, a lightweight version of OpenPose. Experimental results are shown in Tab. 6. Similar to the vehicle counting workload, the lightweight model cannot achieve our target 90% inference accuracy, although its throughput boosts significantly. *InFi*-Skip can flexibly balance the inference accuracy and throughput. For example, for the on-device deployment, the throughput improves to 1.17× after using *InFi*-Skip and the inference accuracy keeps over 90%.

Temporal robustness of selected threshold. A question worth considering is whether the filtering performance of the threshold selected through experiments is robust during online use of *InFi* filters. We divided videos into 10 equal-length clips in the time order to evaluate the temporal robustness of filtering threshold. For the VC workload, when setting the threshold as 0.2, *InFi*-Skip filters out 53.6% frames and achieves 94.9% inference accuracy. The standard deviation of the inference accuracy in the 10 clips is 2.7% and the minimal inference accuracy is 91%, which shows the temporal robustness of the filtering threshold.

7 RELATED WORK

InFi focuses on filtering redundant inputs in the inference stage. We discuss two closely related methods: frame filtering [5, 33, 40] and inference caching [9, 19, 20, 70], and several techniques for specific inference pipelines [28, 31, 32, 73].

Frame filtering. NoScope [33] trains task-specific difference detectors to choose necessary frames for object queries in the video database. FilterForward [5] leverages MobileNet and trains a binary micro-classifier on the intermediate output of a selected layer to determine whether to transmit the input image to the server with offloaded model. Reducto [40] performs on-device frame filtering by thresholding difference of low-level features between successive frames. Through elaborate selection for different tasks, low-level features can efficiently and accurately measure the difference.

Inference caching. Potluck [20] stores and shares inference results between augmented reality applications. It dynamically tunes the threshold of input similarity and manages cache based on the reuse opportunities. FoggyCache [19] is more general and can be applied to both image and audio inputs. It designs adaptive LSH and homogenized KNN algorithms to address practical challenges in inference caching. Instead of caching the final inference results, DeepCache [70] stores the intermediate feature maps to achieve more granular reuse. For object recognition, Glimpse [9] maintains a cache of video frames on mobile devices. It uses cached results to perform on-device object tracking and sends only trigger frames to the server with offloaded recognition model.

Approaches tailored for specific pipelines. Focus [28] is designed for querying detected objects in video database and uses compressed CNN to index possible object classes at ingest stage and reduces the query latency by clustering similar objects. Blazeit [32] develops neural networks-based methods to optimize approximate aggregation queries of detected objects in video database. Focusing on the object detection in video streams, Chameleon [31] proposes to adaptively select a suitable pipeline configuration including the resolution and frame rate of videos, backbone neural networks for inference, etc. Elf [73] is designed for mobile video analytic where the input data is pre-processed by a lightweight on-device model and then offloaded in parallel to multiple servers with the same subsequent inference functionality.

Our proposed input-filtering framework unifies the frame filtering and inference caching approaches. And we complement existing work in theoretical analysis and flexible supports for more input modalities and deployments.

8 CONCLUSION

In this paper, we study the input filtering problem and provide theoretical results on complexity comparisons between the hypothesis families of inference models and their input filters. We propose the first end-to-end learnable framework that unifies both SKIP and REUSE methods and supports multiple input modalities and deployments. We design and implement an input filter system *InFi* based on our framework. Comprehensive evaluations confirm our proven results and show that *InFi* has wider applicability and outperforms strong baselines on accuracy and efficiency.

ACKNOWLEDGMENTS

This research is supported by the National Key R&D Program of China 2018YFB0803400, China National Natural Science Foundation with No. 61822209, No. 62132018, No. 61932016, No.61625205, Key Research Program of Frontier Sciences, CAS. No. QYZDY-SSW-JSC002. This work is partially sponsored by CAAI-Huawei MindSpore Open Fund.

REFERENCES

[1] Pranav Adarsh, Pratibha Rathi, and Manoj Kumar. 2020. YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE, 687–694.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*. PMLR, 173–182.

[3] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. 2013. A public domain dataset for human activity recognition using smartphones.. In *Esann*, Vol. 3. 3.

[4] Adrian Bulat, Jean Kossaifi, Georgios Tzimiropoulos, and Maja Pantic. 2020. Toward fast and accurate human pose estimation via soft-gated skip connections. In *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020)*. IEEE, 8–15.

[5] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya Dulloor. 2019. Scaling Video Analytics on Constrained Edge Nodes. In *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia (Eds.), Vol. 1. 406–417. https://proceedings.mlsys.org/paper/2019/file/85d8ce590ad8981ca2c8286f79f59954-Paper.pdf

[6] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. 2019. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.

[8] Dimitris Chatzopoulos, Carlos Bermejo, Zhanpeng Huang, and Pan Hui. 2017. Mobile augmented reality survey: From where we are to where we go. *IEEE Access* 5 (2017), 6917–6950.

[9] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. 155–168.

[10] Guillaume Chevalier. 2016. LSTMs for human activity recognition.

[11] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1251–1258.

[12] Corinna Cortes, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. 2016. Structured prediction theory based on factor graph complexity. *Advances in Neural Information Processing Systems* 29 (2016), 2514–2522.

[13] Yaqi Duan, Chi Jin, and Zhiyuan Li. 2021. Risk bounds and rademacher complexity in batch reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2892–2902.

[14] Andrew Gardner, Jinko Kanno, Christian A Duncan, and Rastko Selmic. 2014. Measuring distance between unordered sets of different sizes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 137–143.

[15] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. 2021. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2918–2928.

[16] Iman Ghosh. 2020. https://www.visualcapitalist.com/aiot-when-ai-meets-iot-technology/

[17] Tobias Glasmachers. 2017. Limits of end-to-end learning. In *Asian Conference on Machine Learning*. PMLR, 17–32.

[18] Yuan Gong, Yu-An Chung, and James Glass. 2021. AST: Audio Spectrogram Transformer. In *Proc. Interspeech 2021*. 571–575. https://doi.org/10.21437/Interspeech.2021-698

[19] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. 2018. FoggyCache: Cross-device approximate computation reuse. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 19–34.

[20] Peizhen Guo and Wenjun Hu. 2018. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 271–284.

[21] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, Vol. 2. IEEE, 1735–1742.

[22] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. 123–136.

[23] Nick Harvey, Christopher Liaw, and Abbas Mehrabian. 2017. Nearly-tight VC-dimension bounds for piecewise linear neural networks. In *Conference on learning theory*. PMLR, 1064–1068.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[25] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*. 784–800.

[26] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on* 14, 8 (2012), 2.

[27] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. *spaCy: Industrial-strength Natural Language Processing in Python*. https://doi.org/10.5281/zenodo.1212303

[28] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying large video datasets with low latency and low cost. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 269–286.

[29] I Indrawan, I Bayupati, and Desy Purnami Singgih Putri. 2018. Markerless Augmented Reality Utilizing Gyroscope to Demonstrate the Position of Dewata Nawa Sanga. *International Journal of Interactive Mobile Technologies* 12, 1 (2018).

[30] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2177–2190.

[31] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 253–266.

[32] Daniel Kang, Peter Bailis, and Matei Zaharia. [n.d.]. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *Proceedings of the VLDB Endowment* 13, 4 ([n. d.]).

[33] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proceedings of the VLDB Endowment* 10, 11 (2017).

[34] Michael J Kearns, Umesh Virkumar Vazirani, and Umesh Vazirani. 1994. *An introduction to computational learning theory*. MIT press.

[35] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1746–1751. https://doi.org/10.3115/v1/D14-1181

[36] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, Vol. 2. Lille.

[37] Brian Kulis et al. 2012. Metric learning: A survey. *Foundations and trends in machine learning* 5, 4 (2012), 287–364.

[38] Laura Leal-Taixé, Cristian Canton-Ferrer, and Konrad Schindler. 2016. Learning by tracking: Siamese CNN for robust target association. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 33–40.

[39] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2019. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications* 19, 1 (2019), 447–457.

[40] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 359–376. https://doi.org/10.1145/3387514.3405874

[41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[42] TensorFlow Lite. 2021. https://www.tensorflow.org/lite

[43] Cihang Liu, Lan Zhang, Zongqian Liu, Kebin Liu, Xiangyang Li, and Yunhao Liu. 2016. Lasagna: Towards Deep Hierarchical Understanding and Searching over Mobile Sensing Data. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking* (New York City, New York) *(MobiCom '16)*. Association for Computing Machinery, New York, NY, USA, 334–347. https://doi.org/10.1145/2973750.2973752

[44] Jianhui Liu and Qi Zhang. 2019. Code-partitioning offloading schemes in mobile edge computing for augmented reality. *Ieee Access* 7 (2019), 11222–11236.

[45] Chunjie Luo, Fan Zhang, Cheng Huang, Xingwang Xiong, Jianan Chen, Lei Wang, Wanling Gao, Hainan Ye, Tong Wu, Runsong Zhou, et al. 2018. AIoT bench: towards comprehensive benchmarking mobile and embedded device intelligence. In *International Symposium on Benchmarking, Measuring and Optimization*. Springer, 31–35.

[46] Marcin Marszałek, Ivan Laptev, and Cordelia Schmid. 2009. Actions in Context. In *IEEE Conference on Computer Vision & Pattern Recognition*.

[47] Arnab Neelim Mazumder, Jian Meng, Hasib-Al Rashid, Utteja Kallakuri, Xin Zhang, Jae-Sun Seo, and Tinoosh Mohsenin. 2021. A survey on the optimization of neural network accelerators for micro-AI on-device inference. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11, 4 (2021), 532–547.
[48] mindspore ai. 2021. https://github.com/mindspore-ai/mindspore
[49] TensorFlow Lite Hosted Models. 2021. https://www.tensorflow.org/lite/guide/hosted_models
[50] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. *Foundations of machine learning*. MIT press.
[51] Seyed Ali Osia, Ali Shahin Shamsabadi, Sina Sajadmanesh, Ali Taheri, Kleomenis Katevas, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. 2020. A hybrid deep learning architecture for privacy-preserving mobile analytics. *IEEE Internet of Things Journal* 7, 5 (2020), 4505–4518.
[52] D Osokin. 2019. Real-time 2D multi-person pose estimation on CPU: Lightweight OpenPose. In *ICPRAM 2019-Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods*. 744–748.
[53] Karol J. Piczak. [n.d.]. ESC: Dataset for Environmental Sound Classification. In *Proceedings of the 23rd Annual ACM Conference on Multimedia* (Brisbane, Australia, 2015-10-13). ACM Press, 1015–1018. https://doi.org/10.1145/2733373.2806390
[54] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. 2021. Multi-Modal Fusion Transformer for End-to-End Autonomous Driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7077–7087.
[55] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.
[56] Sefik Ilkin Serengil and Alper Ozpinar. 2020. LightFace: A Hybrid Deep Face Recognition Framework. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*. IEEE, 23–27. https://doi.org/10.1109/ASYU50717.2020.9259802
[57] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2158–2170.
[58] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1701–1708.
[59] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2820–2828.
[60] tensorflow. 2021. https://github.com/tensorflow/tensorflow
[61] Xianzhong Tian, Juan Zhu, Ting Xu, and Yanjun Li. 2021. Mobility-included DNN partition offloading from mobile devices to edge clouds. *Sensors* 21, 1 (2021), 229.
[62] An Tran and Loong-Fah Cheong. 2017. Two-stream Flow-guided Convolutional Attention Networks for Action Recognition. In *The IEEE International Conference on Computer Vision Workshop (ICCVW)*.
[63] NVIDIA Jetson TX2. 2021. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/
[64] Leslie Valiant. 2013. *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World*. Basic Books (AZ).
[65] VN Vapnik and A Ya Chervonenkis. 1971. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability & Its Applications* 16, 2 (1971), 264–280.
[66] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. 2018. Bandwidth-efficient live video analytics for drones via edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 159–173.
[67] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. 2019. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network* 33, 5 (2019), 156–165.
[68] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. 2016. A discriminative feature learning approach for deep face recognition. In *European conference on computer vision*. Springer, 499–515.
[69] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*. PMLR, 2048–2057.
[70] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. 2018. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 129–144.
[71] Mu Yuan, Lan Zhang, Xiang-Yang Li, and Hui Xiong. 2020. Comprehensive and efficient data labeling via adaptive model scheduling. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1858–1861.
[72] Mu Yuan, Lan Zhang, Xiang-Yang Li, Lin-Zhuo Yang, and Hui Xiong. 2022. Adaptive Model Scheduling for Resource-Efficient Data Labeling. *ACM Trans. Knowl. Discov. Data* 16, 4, Article 71 (jan 2022), 22 pages. https://doi.org/10.1145/3494559
[73] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. 2021. Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 201–214.
[74] Chong Zhou and Randy C Paffenroth. 2017. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 665–674.
[75] Li Zhou, Hao Wen, Radu Teodorescu, and David HC Du. 2019. Distributing deep neural networks with containerized partitions at the edge. In *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*.
[76] zzh8829. 2021. https://github.com/zzh8829/yolov3-tf2