A Two-Stage Data Selection Framework for Data-Efficient Model Training on Edge Devices

Chen Gong
Shanghai Jiao Tong University
Shanghai, China
gongchen@sjtu.edu.cn

Rui Xing
Shanghai Jiao Tong University
Shanghai, China
cocojess@sjtu.edu.cn

Zhenzhe Zheng
Shanghai Jiao Tong University
Shanghai, China
zhengzhenzhe@sjtu.edu.cn

Fan Wu
Shanghai Jiao Tong University
Shanghai, China
fwu@cs.sjtu.edu.cn

Abstract

The demand for machine learning (ML) model training on edge devices is escalating due to data privacy and personalized service needs. However, we observe that current on-device model training is hampered by the under-utilization of on-device data, due to low training throughput, limited storage and diverse data importance. To improve data resource utilization, we propose a two-stage data selection framework Titan to select the most important data batch from streaming data for model training with guaranteed efficiency and effectiveness. Specifically, in the first stage, Titan filters out a candidate dataset with potentially high importance in a coarse-grained manner. In the second stage of fine-grained selection, we propose a theoretically optimal data selection strategy to identify the data batch with the highest model performance improvement to current training round. To further enhance time-and-resource efficiency, Titan leverages a pipeline to co-execute data selection and model training, and avoids resource conflicts by exploiting idle computing resources. We evaluate Titan on real-world edge devices and three representative edge computing tasks with diverse models and data modalities. Empirical results demonstrate that Titan achieves up to 43% reduction in training time and 6.2% increase in final accuracy with minor system overhead, such as data processing delay, memory footprint and energy consumption.

CCS Concepts

• **Human-centered computing → Mobile computing**; • **Computing methodologies → Machine learning**.

Keywords

On-Device Machine Learning, Data Selection and Utilization

---

*Zhenzhe Zheng is the corresponding author.

1 Introduction

Machine learning (ML) models have been widely embedded in mobile applications to provide diverse intelligent services, such as image tagging in Google Lens [2], command recognition in Siri [1], text prediction in Microsoft SwiftKey [3], etc. With growing concerns over data privacy and higher demands on personalized model performance, on-device model training is becoming necessary to facilitate a single device to adapt the local model to its own data distribution [15, 17, 34, 48, 59, 66], or multiple devices to collaboratively train a global model that can generalize well across different data distributions [16, 18, 35].

The success of ML models highly relies on the abundant valuable data for model training [14, 20, 50]. On one hand, a large-scale training dataset is essential to the generalization performance of ML models [50]. On the other hand, massive data aids in stabilizing the model training process and reducing the training time to reach a target accuracy [20]. As a result, on cloud server side, it is common to collect extensive training data for iterative model updates, such as 29 million games for training AlphaGo [47] and 500 billion tokens for pre-training ChatGPT3 [38]. Similarly, on mobile devices side, it is also desirable to fully utilize the on-device sensor data for model training to achieve a satisfactory model training performance and improve user experience.

Motivation. Previous works for on-device model training mainly focused on the exploitation of limited *hardware resources*, such as optimizing memory allocation to increase batch size during training [15, 54], co-using multiple types of computing resources to speed up model inference and training [26, 52, 53, 58], dynamically adjusting the size of trainable parameters to improve training efficiency [23, 57], etc. However, we observe that the under-utilization of *data resource* is another key bottleneck to on-device model training, which results in up to 3.5× longer training time and 13.3% lower final accuracy in our pilot experiments due to low training throughput, limited storage and diverse data importance (elaborated in §2.2). Therefore, a crucial open problem is: *Is it possible*

to design an on-device data selection framework to concentrate the limited hardware resources on important training data for superior model training performance?

Challenges. The design of an on-device data selection framework needs to achieve *effectiveness* and *efficiency* simultaneously: *i) Effectiveness:* As the on-device model performance directly impacts the quality of application service and user experience, it is necessary for the data selection framework to provide theoretical and empirical guarantees on enhancing model training performance. *ii) Efficiency:* For deployment, the data selection framework is desired to be time-and-resource efficient. First, the application data typically undertakes real-time services like teleconferencing, which requires the data selection process to be low-latency to avoid compromising user experience. Second, the data selection framework needs to avoid intense resource conflict with model training process, which would extend the time of each model update and offset the performance improvement brought by data selection.

It is challenging to satisfy these two properties simultaneously. Higher effectiveness necessitates a more accurate but time-intensive data importance evaluation process over a broader candidate dataset, which inevitably increases per-sample latency and consumes more computing resources. Our theoretical analysis and experimental results in §2.3 indicate that conventional cloud-side data selection approaches such as importance sampling [28, 65], heuristic selection [7, 13, 43, 45, 63] and coreset selection [32, 36, 42] fail to be applied to device side due to ineffectiveness or inefficiency.

Our Solutions. In this work, we address the above challenge by proposing a <u>t</u>wo-stage onl<u>i</u>ne da<u>ta</u> selectio<u>n</u> framework Titan, which simultaneously achieves high effectiveness and efficiency for on-device data utilization. First, to guarantee the effectiveness, we theoretically analyze the correlation between the training data batch and the on-device model training performance, based on which we demonstrate the sub-optimality of the state-of-the-art importance sampling approach due to overlooking a crucial term of class variance during inter-class batch size allocation. Further, we propose a *theoretically optimal data selection strategy* to identify the data batch with the highest improvement to model performance in each training round. Second, to improve time-efficiency, Titan employs a *two-stage architecture*. In the first stage, Titan leverages a carefully designed coarse-grained filter to estimate the potential importance of each streaming data sample within millisecond-level latency, and locally buffers a small candidate dataset. In the second stage, the buffered candidate dataset undergoes our proposed data selection strategy to enhance effectiveness. Third, for higher time-and-resource efficiency, we design a *pipeline* to facilitate the co-execution of model training and data selection, and exploits the idle computing resources commonly seen on devices to mitigate potential resource conflicts.

Contributions of this work are summarized as follows:
- To the best of our knowledge, we are the first to point out the severity of data resource under-utilization in on-device model training process, and conduct in-depth analysis for this issue.
- We perform comprehensive evaluation of existing cloud-side data selection approaches for device-side setting, and provide theoretical and empirical analysis on their failures.
- We propose an on-device data selection framework Titan, consisting of a theoretically optimal data selection strategy, a two-stage

architecture and a pipeline design, to simultaneously achieve high efficiency and effectiveness for on-device data utilization.
- We implement Titan framework on real-world device and demonstrate Titan's superiority across three typical mobile computing tasks with varied data modalities and ML models.

2 Background and Motivation

In this section, we first briefly introduce the background of on-device model training (§2.1). Then, we delve into the under-utilization of on-device data resources to illustrate the motivation of this work (§2.2). Next, we elaborate the limitations of existing cloud-side data selection methods (§2.3).

2.1 On-Device Model Training

Similar to cloud-side ML, the objective of on-device model training can be formulated as minimizing the loss function $L(w, \mathcal{P})$, which represents the prediction error (or loss) of model with parameters w on local data distribution $\mathcal{P}$:

$$w^* = \min_w L(w, \mathcal{P}) \stackrel{\text{def}}{=} \mathbb{E}_{(x,y)\sim\mathcal{P}}\big[l(w, x, y)\big], \tag{1}$$

where $\mathbb{E}_{(x,y)\sim\mathcal{P}}\big[l(w, x, y)\big]$ denotes the expected loss (or error) of model w over data (x, y) following distribution $\mathcal{P}$.

In on-device model training, mini-batch SGD [44] is widely adopted to solve the above optimization problem (1), which involves three steps in each training round t:
1) Data Collection: Data samples $(x, y) \sim \mathcal{P}$ are continuously collected by device in a streaming manner and stored in the local storage. We use $\mathcal{S}$ and $\mathcal{S}_y$ to denote the sets of all the stored data samples and the data samples with class $y \in \mathcal{Y}$.
2) Data Loading: A batch of data samples $\mathcal{B} = \{(x_i, y_i) | 1 \le i \le |\mathcal{B}|\}$ is loaded from storage to memory as training data.
3) Model Update: Current model parameters w_t are updated by the average gradient of the loaded training data batch:

$$w_{t+1} = w_t - \eta_t \cdot \mathbb{E}_{(x,y)\in\mathcal{B}}\big[\nabla_w l(w_t, x, y)\big],$$

where w_t denotes the updated model parameter in training round t and η_t is the corresponding learning rate.

Typically, data collection is conducted concurrently with data loading and model update, both of which are executed iteratively.

2.2 Under-Utilization of On-Device Data

We elaborate the three unique characteristics of mobile devices, which leads to the under-utilization of on-device data and motivates the design of a device-specific data selection framework.

Low data throughput during model training. The limited memory and computing resources of devices restrict the training data throughput. First, the memory size constrains the number of data samples that can be co-processed within a data batch (*e.g.* batch size 16 for common lightweight model MobileNetV1 has reached the limit of high-end devices like MI 9 with 6GB RAM [6]). Second, the on-device per-sample training time is relatively long due to the limited computing hardware [26]. Specifically, the forward-and-backward propagation over modern ML models can be time-consuming (*e.g.* it takes around 20s for the representative device Jetson Nano to train one data batch with size 16 on MobileNetV1).
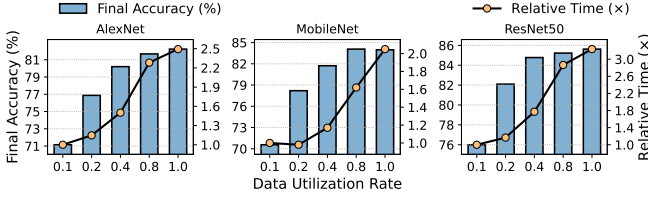
Figure 1: Final inference accuracy and normalized training time of different models and data utilization rates.

(a) Per-round training time. (b) Training processes with batch sizes 10 and 25.

Figure 2: Per-round training time and training curves of different data selection methods on MobileNetV1 and CIFAR-10, tested on real device Jetson Nano.

Limited on-device storage for training data. In numerous mobile applications, on-device data is continuously collected in a streaming manner, but devices typically have quite limited storage for training data due to user preference as well as software and hardware limitation. On one hand, users usually prioritize reserving storage space for personal files like photos, documents and chat history instead of each application's training data. On the other hand, both iOS and Android platforms impose size limitations on applications, such as less than 4GB for an iOS app [10] and no more than 2GB for a Google Play app [19]. Furthermore, for low-end devices like HUAWEI WiFi AX3 [24] with less than 1GB storage, it is impractical to save all the collected data for model training.

Diverse data importance to training performance. The on-device data samples have diverse importance (or quality) for model training, stemming from *1)* the wide range of on-device data distribution caused by varied user behaviors and application services at different times of a day, and *2)* heterogeneous data quality due to sensors from different producers and unstable network environments. Consequently, the involvement of low-importance data in model training will further reduce the on-device data utilization.

Motivating Experiments. The aforementioned properties restrict the on-device data utilization and hinder the success of on-device model training processes. On one hand, if we attempt to utilize all the collected data for parameter update to achieve superior training performance, all the samples need to be incorporated in each training round, which leads to substantial per-round training time and storage overhead [49, 54]. On the other hand, if we leverage only partial data for higher efficiency of model training and data storage, the parameter update computed from partial data tend to deviate from the expected update computed from all data, thereby degrading the final model accuracy [28]. Our preliminary experiments on representative device Jetson Nano and dataset CIFAR-10 [29] in Figure 1 show that leveraging only partial data resource can reduce the final accuracy by 9.6−13.4% while the utilization of full data will prolong total training time by 2.05–3.24×. Therefore, an on-device data selection framework is necessary to focus limited hardware resources on partial but important data resources for higher data utilization efficiency.

2.3 Existing Data Selection Approaches

Existing cloud-side data selection approaches can hardly be applied to the device side due to ineffectiveness or inefficiency.

Importance sampling (IS) [28, 65] is the state-of-the-art data selection approach, which selects each training data sample according to its importance to model training performance. Previous
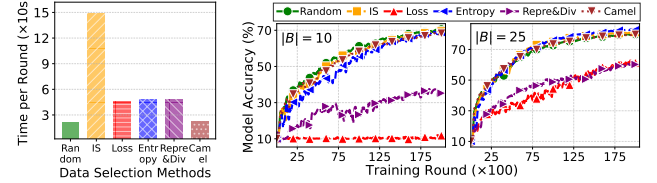
research has demonstrated a negative correlation between the gradient variance of the training data batch[1] and the model training performance. As a result, IS defines the per-sample importance as its gradient norm to minimize such gradient variance and optimize the training performance.

However, IS is neither effective nor efficient for devices. On one hand, we identify that the theoretical optimality of IS relies on an underlying assumption that each sample in the training data batch is independently selected, which leads to sub-optimal performance for batch-level selection, especially for small training batches on devices. The detailed theoretical analysis and verification results are provided in §3.2. On the other hand, IS requires computing each sample's gradient over model parameters, which can prolong the per-round training time by up to 7× shown in Figure 2(a). For device-side efficient deployment, Mercury [64] proposed to divide the dataset into multiple subsets and recompute only one subset's importance per training round, which however, is not applicable for real-world mobile computing tasks involving streaming data.

Heuristic data selection (HDS) enhances model training efficiency by selecting the training data with various intuitive metrics, such as model uncertainty quantified by loss or entropy of model output logits [7, 45], data representativeness measured by closeness to the distribution centroid in feature space [13] and diversity to other samples [63], etc.

We find that HDS is efficient but lacks effectiveness from both theoretical and empirical aspects. Theoretically, existing HDS fails to directly correlate the data importance metric with model training performance, thereby essentially optimizing a proxy objective of intuitively defined metrics rather than the fundamental objective (1) of model training performance. Therefore, practical implementation of HDS often involves cumbersome trial-and-error processes to explore the appropriate metrics that could bring the highest model performance improvement. Empirically, Figures 2(b) reveals that HDS (*i.e. Loss*, *Entropy* and *Repre&Div*) even leads to degraded training performance compared with random selection when the batch size is small. This is because traditional HDS relies on large batch sizes to mitigate the distribution deviation and parameter update bias of the heuristically selected training data batch.

Coreset Selection (CS) [32, 36, 42] aims to select a small weighted data subset, *i.e.* coreset, to approximate the entire dataset in terms of gradient computation, thereby reducing the training data scale

---

[1]The variance represents the average difference between the gradient of the selected training data and the expected gradient of the entire dataset.
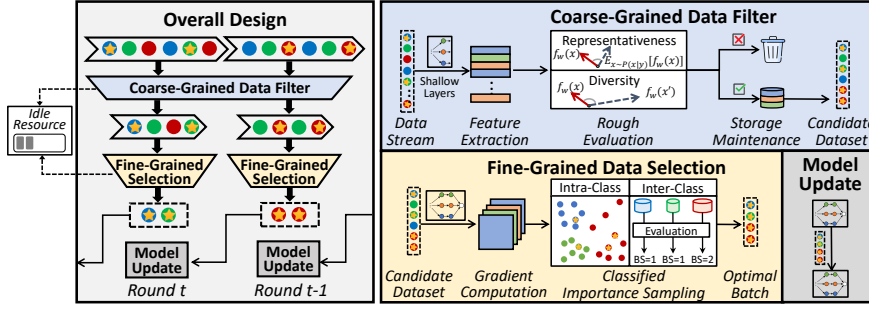
Figure 3: Overall design and workflow of Titan.

Figure 4: Comparison between IS and C-IS. IS will select an equal number of data samples from classes 1 and 2, while C-IS will select more samples from class 1 by considering its larger gradient variance.

without significant deviation in parameter update direction. Previous research formulated the gradient estimation error of the coreset as a sub-modular function, and derived the optimal coreset by minimizing such error.

We observe that CS is either inefficient or ineffective for devices. To select a coreset with size $|\mathcal{B}|$ from $|\mathcal{S}|$ data samples, CS requires computing the gradients of all $|\mathcal{S}|$ samples to solve the error minimization problem, incurring high computation overhead similar to IS. Instead of directly minimizing the gradient distance between coreset and the entire dataset, *Camel* [32] uppers bound the gradient distance by raw input distance to avoid the cumbersome model backpropagation process. However, this approximation compromises the theoretical guarantee of CS and also exhibits inferior performance in our prior experiments in Figure 2(b). This is because the complex structures of modern ML models and the wide distribution of on-device data make the raw data distance unable to accurately reflect the gradient distance.

3 Design of Titan

In this section, we first give an overview of Titan framework (§3.1), and then we elaborate each key component in Titan (§3.2-§3.4).

3.1 Overview

Design Rationale. Titan aims to exploit on-device data resources effectively and efficiently by incorporating three key designs: *1) a theoretically optimal strategy* for training data batch selection, which operates as a fine-grained selection component to identify the data batch that brings the highest improvement to model performance, *2) a coarse-grained filter* to filter out a small candidate dataset from streaming data in real time through heuristic metrics specially co-designed with the optimal fine-grained selection component, *3) a pipeline design* to co-execute the processes of data selection and model training and mitigate their potential resource conflict by utilizing on-device idle computing resources.

Workflow. As depicted in Figure 3, Titan adopts a two-stage architecture and forks three concurrent processes to steadily select the optimal training data batch from real-time data streams for each round of model training:

1) Coarse-grained filter: Whenever a data sample is collected by device, Titan extracts its feature by inputting the data into shallow layers of ML models, and estimates its potential importance within milliseconds through two specially designed heuristic metrics. Then, Titan maintains a candidate dataset in local buffer with

a priority queue to facilitate the subsequent fine-grained selection. *2) Fine-grained selection*: During each round t, Titan computes the gradient for each buffered data sample over the final model layer, and identifies the ideal data batch for next round $t + 1$ through the proposed optimal data selection strategy.

3) Model update: Simultaneously in round t, current model parameter w_t is updated by the data batch chosen in preceding round $t - 1$, which forms a seamless pipeline with the fine-grained selection process for the upcoming round.

3.2 Fine-Grained Data Selection

To optimize the effectiveness of on-device data selection, we propose a new data batch selection strategy for mini-batch SGD, namely classified importance sampling (C-IS), which consists of inter-class batch size allocation and intra-class data selection. We first provide the definitions of *class importance* and *sample importance*, which are used to determine *how many* and *which* data samples to select for each class. Then, we analyze C-IS's theoretical optimality in improving on-device model training performance and provide an intuitive explanation for better understanding.

Inter-Class Batch Size Allocation. To select a batch of training data with size $|\mathcal{B}|$, C-IS determines the data selection size $|\mathcal{B}_y|$ for each class $y \in \mathcal{Y}$ according to the class importance $I_t(y)$ in current round t, which is defined as:

$$I_t(y) \overset{\text{def}}{=}$$
$$|\mathcal{S}_y|\left[\underbrace{\mathbb{V}_{(x,y)\sim P_{t,y}}[\nabla l(w_t, x, y)]}_{\text{variance of gradient}} - \underbrace{\mathbb{V}_{(x,y)\sim P_{t,y}}[||\nabla l(w_t, x, y)||_2]}_{\text{variance of gradient norm}}\right]^{\frac{1}{2}} \quad (2)$$

where $|\mathcal{S}_y|$ denotes the total amount of stored data samples with class y and $\mathbb{V}_{(x,y)\sim P_{t,y}}[f(x)]$ denotes the variance of function $f(x)$ with selection probability $P_{t,y}(x)$ for each data sample x in class y and $|| * ||_2$ denotes the l_2-norm.

Intra-Class Data Selection. To select $|\mathcal{B}_y|$ important data samples from the stored data $\mathcal{S}_y$ of class $y \in \mathcal{Y}$, we select each sample (x, y) with probability proportional to its sample importance $I_t(x, y)$, which is defined as:

$$I_t(x, y) \triangleq \underbrace{\left\|\nabla l(w_t, x, y)\right\|_2}_{\text{gradient norm}}. \quad (3)$$

Theoretical Analysis. To demonstrate the theoretical optimality of C-IS, we first present Theorem 1 and Lemma 1 to analyze the
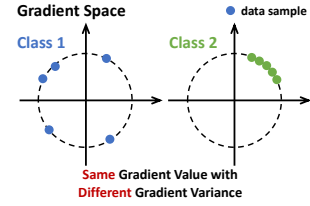
impact of training data batch on model training performance as well as the sub-optimality of state-of-the-art IS in data batch selection. Further, we present Theorem 2 and Lemma 2 to demonstrate the optimality of our proposed C-IS in improving model training performance.

Previous studies [28, 65] have demonstrated a negative correlation between the gradient variance of the training data batch $\mathcal{B}$ and model training performance, where the performance of model with parameter w is quantified by its distance to the optimal parameter w^* (*i.e.* $||w - w^*||_2^2$). Therefore, the model training performance in round t can be measured by the decrease in the distance to w^* from initial model parameter w_t to updated model parameter w_{t+1}:

Theorem 1 (Training Performance Measurement [28, 65]). *The model training performance in round t is negatively correlated with the gradient variance of the training data batch $\mathcal{B}$ selected by data selection strategy P_t:*

$$\mathbb{E}_{\mathcal{B} \sim P_t} \left[\underbrace{||w_t - w^*||_2^2 - ||w_{t+1} - w^*||_2^2}_{\text{reduction in distance to } w^*} \right] = -\eta_t^2 \cdot \underbrace{\mathbb{V}_{\mathcal{B} \sim P_t} \left[\nabla L(w_t, \mathcal{B}) \right]}_{\text{optimized through } P_t}$$

$$+ \underbrace{2\eta_t \cdot (w_t - w^*)^\top \nabla L(w_t, \mathcal{B}) - \eta_t^2 ||\nabla L(w_t, \mathcal{B})||_2^2}_{\text{fixed by initial model parameter } w_t \text{ in each round } t},$$

where $\mathbb{V}_{\mathcal{B} \sim P_t} \left[\nabla L(w_t, \mathcal{B}) \right]$ denotes the gradient variance of $\mathcal{B}$.

Proof. The detailed proof please refer to Appendix A.1. □

Accordingly, IS proposed to minimize such variance and maximize training performance by optimizing the selection probability of each data sample, i.e. $P_t(x, y)$. However, we identify in Lemma 1 that IS potentially assumes that each data sample in the training data batch is independently selected, resulting in optimal sample-level data selection but sub-optimal batch-level data selection for mini-batch SGD.

Lemma 1 (Optimal Sample-Level Selection). *To minimize the gradient variance of selected data batch $\mathcal{B}$, IS computes the optimal selection probability P_t^* for each data sample (x, y):*

$$P_t^*(x, y) \triangleq \arg\min_{P_t} \mathbb{V}_{\mathcal{B} \sim P_t} \left[\nabla L(w_t, \mathcal{B}) \right]$$

$$\overset{(a)}{=} \arg\min_{P_t} \frac{1}{|\mathcal{B}|} \mathbb{V}_{(x,y) \sim P_t} \left[\nabla l(w_t, x, y) \right]$$

$$\overset{(b)}{=} \left\| \nabla l(w_t, x, y) \right\|_2 / \sum_{(x',y') \in \mathcal{S}} \left\| \nabla l(w_t, x', y') \right\|_2,$$

where we observe that Equation (a) implicitly assumes an independent selection process for each data sample (x, y) in data batch $\mathcal{B}$. Equation (b) holds according to Cauchy-Schwarz inequality [28, 65].

To analyze the sub-optimality of IS in on-device settings and provide theoretical insight for designing optimal batch-level data selection strategy, we decompose the gradient variance of the selected data batch into three terms in Theorem 2.

Theorem 2 (Gradient Variance Decomposition). *The gradient variance of data batch $\mathcal{B}$ selected from candidate dataset $\mathcal{S}$ using selection strategy P_t can be decomposed into the weighted sum of*

terms α_y, β_y and γ_y for each class $y \in \mathcal{Y}$:

$$\mathbb{V}_{\mathcal{B} \sim P_t} \left[\nabla L(w_t, \mathcal{B}) \right] = \sum_{y \in \mathcal{Y}} \alpha_y \cdot (\beta_y - \gamma_y), \text{where } \alpha_y = \frac{|\mathcal{S}_y|^2}{|\mathcal{S}|^2 \cdot |\mathcal{B}_y|},$$

$$\beta_y = \sum_{(x,y) \in \mathcal{S}_y} \frac{\left\| \nabla l(w_t, x, y) \right\|^2}{|\mathcal{S}_y|^2 \cdot P_{t,y}(x)}, \gamma_y = \left\| \mathbb{E}_{(x,y) \in \mathcal{S}_y} \left[\nabla l(w_t, x, y) \right] \right\|^2,$$

where $\mathcal{S}_y$ and $\mathcal{B}_y$ are candidate data and selected data for each class.

Proof. The detailed proof please refer to Appendix A.2. □

We identify that the gradient variance of the selected data batch is composed of three terms of each class y: 1) α_y is impacted by batch size allocation $|\mathcal{B}_y|$ across classes and the other two terms, 2) β_y is determined by intra-class data selection strategy $P_{t,y}$, and 3) γ_y is a constant that varies for different classes. As a result, traditional IS can be regarded as conducting optimal intra-class data selection to minimize β_y, but executing sub-optimal inter-class batch size allocation based on solely β_y rather than $(\beta_y - \gamma_y)$. Furthermore, the overlooked term $-\alpha_y \gamma_y$ can become significant for on-device settings with limited memory, as α_y increases with smaller batch sizes. This is also verified by our empirical results in Figure 5(a), which indicates that 1) the gradient variance gap between existing IS and our proposed C-IS becomes wider with smaller batch sizes and 2) C-IS consistently achieves the optimal performance.

To optimize on-device model training performance, we propose a new optimal batch-level data selection strategy C-IS, which keeps using IS for optimal intra-class data selection while taking the integral term $(\beta_y - \gamma_y)$ into consideration when allocating batch size to different classes $y \in \mathcal{Y}$.

Lemma 2 (Optimal Batch-Level Selection). *To maximize the training performance of mini-batch SGD, given batch size $|\mathcal{B}|$ and dataset $\mathcal{S}_y$ for each class $y \in \mathcal{Y}$, the optimal selection size for each class (i.e. $|\mathcal{B}_y|^*$) and the optimal selection probability for each sample within the class (i.e. $P_{t,y}^*(x)$) are:*

$$|\mathcal{B}_y|^* \propto I_t(y), P_{t,y}^*(x) \propto I_t(x, y),$$

where $I_t(y)$ and $I_t(x, y)$ are the class importance and sample importance defined in Equation (2) and Equation (3), respectively.

Proof. The detailed proof please refer to Appendix A.3. □

Intuitive Understanding. The sample importance $I_t(x, y)$ in Equation (3) is exactly the norm of sample gradient over model parameters, reflecting the contribution of each sample to parameter update. The class importance $I_t(y)$ in Equation (2) essentially quantifies the overall diversity of each class, (*i.e.* gradient variance minus gradient norm variance). Higher class importance indicates that data samples within this class have diverse gradients but similar gradient norms. Naturally, more samples are needed to thoroughly represent the gradient distribution of such class. However, conventional IS distributes batch size to each class solely based on average gradient norm, focusing on classes with high gradient value rather than diversity. A simple example is provided in Figure 4 for better comparison, where IS will select the same number of samples from classes 1 and 2 but C-IS will select more samples from class 1 by considering variance, which is obviously more reasonable.

(a) Optimality of C-IS over random sampling (RS) and IS.

(b) Significance of coarse-grained filter on C-IS performance.
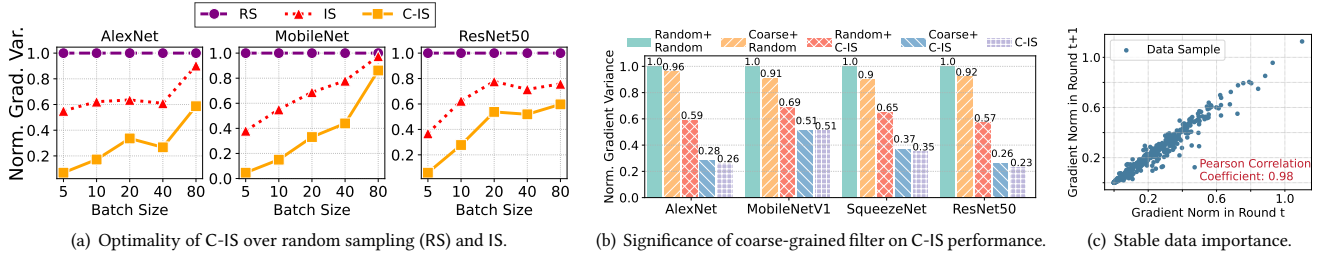
(c) Stable data importance.

Figure 5: Preliminary experiments on CIFAR-10 dataset and different models with learning rate 0.1 to support some claims, including (a) to empirically demonstrate C-IS's optimality over previous methods, (b) to reveal the efficiency and efficacy of coarse-grained filter, and (c) to show the stable importance scores of data samples across consecutive training rounds.

Practical Implementation. For on-device implementation, we propose to substitute the gradient of each data sample over entire model parameters with the gradient over only last model layer, which avoids the cumbersome backpropagation process and saves computation and memory costs. Such simplification relies on the phenomenon that partial model gradient can reflect the trend of full model gradient, as analyzed theoretically and empirically by existing works [27, 32, 36].

3.3 Coarse-Grained Data Filter

While C-IS enables identifying the data batch with the highest effectiveness on model performance improvement, it also incurs substantial delay due to calculating the accurate data importance (*i.e.* gradient and its norm) for each streaming data. A straightforward remedy is to reduce the frequency of importance computation, which in turn constrains the size of candidate data for C-IS and compromises its effectiveness.

Inspired by the billion-scale item ranking process of online recommendation system [11], Titan leverages a two-stage architecture to guarantee both efficiency and effectiveness. Specifically, Titan introduces an additional stage of coarse-grained filter and designs two heuristic metrics to filter out a candidate dataset that could facilitate the processes of inter-class batch size allocation and intra-class data selection in the subsequent fine-grained selection.

1) Representativeness: To enable an accurate measurement of *class importance* during inter-class batch size allocation, the filtered data is expected to *represent* the characteristics of the majority of data samples in each class. Therefore, the representativeness of each data(x, y) can be measured by its closeness to the class centroid in feature space:

$$\text{Rep}(x, y) = -\left\| f_w(x) - \mathbb{E}_{\mathcal{P}(x'|y)}\left[f_w(x') \right] \right\|_2^2,$$

where $f_w(x)$ denotes the feature extracted by current model w and $\mathbb{E}_{\mathcal{P}(x'|y)}\left[f_w(x') \right]$ denotes the feature centroid of data in class y.

2) Diversity: To identify more high-importance data samples, the filtered data needs to be *diverse* enough to cover the data distribution. Therefore, the *diversity* of each data (x, y) can be quantified as its average distance to the other data within the same class in the feature space:

$$\text{Div}(x, y) = \mathbb{E}_{\mathcal{P}(x'|y)}\left[\left\| f_w(x) - f_w(x') \right\|_2^2 \right]$$
$$= \left\| f_w(x) \right\|_2^2 + \mathbb{E}_{\mathcal{P}(x'|y)} \left\| f_w(x') \right\|_2^2 - 2\left\langle f_w(x), \mathbb{E}_{\mathcal{P}(x'|y)} \left[f_w(x') \right] \right\rangle.$$

We evaluate the impact of coarse-grained filter on C-IS's ability in reducing gradient variance in Figure 5(b), where $A+B$ denotes leveraging method A to filter out $0.3v$ candidate data samples out of v streaming samples and employing method B to further select $0.1v$ data samples as data batch, where v is set to 100 for preliminary experiments. The result shows that compared with the ideal case of performing C-IS on all data, coarse-grained filter can reduce the candidate data size by 70% with less than 3% degradation of gradient variance reduction degree.

Practical Implementation. For efficient implementation of coarse-grained filter, Titan only needs to dynamically maintain two running-sum estimators for average feature $\mathbb{E}_{\mathcal{P}(x'|y)}\left[f_w(x') \right]$ and average feature norm $\mathbb{E}_{\mathcal{P}(x'|y)}\left\| f_w(x') \right\|_2^2$ using each streaming data (x, y). Based on these estimators, Titan could realize online coarse-grained filtering by buffering data with the highest $\text{Rep}(x, y) + \text{Div}(x, y)$. For the feature extraction function $f_w(x)$, we propose to input the raw data x into the first few network layers of model w and regard the layer outputs as features, which is according to our empirical observation that *1)* the features extracted by shallow layers are sufficient to filter out an effective candidate data for subsequent fine-grained data selection, and *2)* forward pass through a few model layers only introduces minor latency and memory footprint. A detailed empirical analysis is presented in §4.3.

3.4 Pipeline Design

Although the two-stage architecture of Titan achieves higher time-efficiency, the wall-clock time per training round still increases significantly due to the model dependency and resource preemption between data selection and model update: *1) Model dependency*: As data selection relies on the latest model parameter to compute the accurate importance of each sample and class, the processes of model update and data selection have to be executed alternately and sequentially. *2) Resource preemption*: The limited computing resource is shared and preempted by data selection and model update, which will slow down the original model update process.

Titan overcomes the above challenges by leveraging a pipeline design to enable the co-execution of model update and data selection. To eliminate *model dependency*, Titan proposes a simple but effective "one-round-delay" scheme, where each model parameter w_t is updated by the data batch selected in the previous round using the slightly outdated model w_{t-1}. Such approximation enables the co-execution of parameter update for the current round and data

selection for the next round, and its feasibility is supported by our observation in Figure 5(c) that per-sample importance (*i.e.* gradient norm) typically does not change significantly in consecutive training rounds.

To avoid *resource preemption*, Titan offloads the data selection process to commonly seen idle computing resources. Despite that mobile devices are typically equipped with multiple types of computing resources (*e.g.* CPU, GPU and NPU), current devices mainly use one type of resource type for parameter update, due to the high synchronization overhead of sharing each layer's outputs and gradients across different hardware per each parameter update [26, 57]. By offloading only data selection to other available computing resource, Titan prevents its resource conflict with model update and incurs low cost by synchronizing model parameters and the small selected data batch only once per model update. A breakdown analysis of the system cost is provided in Figure 6 in §4.3.

Further, to handle dynamic idle computing resources on edge devices, Titan system automatically adjusts the candidate data size chosen by coarse-grained filter. Specifically, it continuously evaluates the importance of each stored data sample using idle resources. For each new training round, these evaluated samples naturally become candidate data for fine-grained selection, without the need to predefine a candidate data size. Also, our evaluation already accounts for such resource fluctuations, as the computation speed is impacted by dynamic factors like device power, heat dissipation and charging.

4 Evaluation

We first introduce our experiment setup (§4.1). Then we present the overall performance of Titan (§4.2) and conduct component-wise analysis (§4.3). Further, we test the applicability of Titan to different scenarios in Appendix B, including fluctuant idle computing resources, federated learning and noisy on-device data.

4.1 Experiment Setup

Tasks, Datasets and Models. To demonstrate Titan's generality, we evaluate it on three typical mobile computing tasks with three data modalities and six model structures: *1) Image Classification (IC)*: CIFAR-10 [29] consists of 60, 000 images of 10 objects. We train four representative ML models for this task, including the classic dense model AlexNet [30], lightweight models MobileNetV1 [22] and SqueezeNet [25] as well as a larger model ResNet50 [21]. *2) Audio Recognition (AR)*: Google Speech Commands [56] includes 100, 000 sound files of 20 commands collected from 2, 000 users, and we train ResNet34 [21] for this task. *3) Human Activity Recognition (HAR)*: HARBOX [39] contains IMU data collected from 6 activities of 121 users. According to previous work, we resample with a sliding time window of 2s at 50Hz and obtain 34, 115 data samples with 900-dimension features. An MLP [41] with two fully-connected layers and a SoftMax layer is trained for this task.

Hardware Setup. We implement Titan framework on the real-world mobile platform NVIDIA Jetson Nano [37] with 4GB RAM, 4 A57 CPU cores and a Maxwell GPU, which has similar hardware and running environment with mainstream devices [31, 62]. For pipeline implementation, Titan forks three processes using different

computation hardware[2]: *Process 1* conducts coarse-grained filtering with mobile GPU to filter out a small candidate dataset from data streams; *Process 2* executes fine-grained selection with mobile GPU to identify the optimal data batch from the candidate data; *Process 3* steadily updates the model parameter with mobile CPU using the data batch shared by process 2.

Baselines. We compare Titan with existing data selection methods, including: *1) Random selection* (RS) selects random data for model training; *2) Importance sampling* (IS) [28] selects each training data sample according to gradient norm over the final model layer; *3) Heuristic data selection* selects training data batch according to per-sample training loss (high loss HL [7] and low loss LL [46]), cross entropy of the model output logits (CE [45]) or data representativeness and diversity (OCS [63]); *4) Coreset selection* (Camel [32]) greedily selects the sample that minimizes the input distance between the currently selected data batch and entire dataset.

Evaluation Metrics. We use five metrics to evaluate the overall performance of data selection. *1) Final inference accuracy* denotes the test accuracy of the finally trained model. *2) Time-to-accuracy* measures the wall-clock time required for each method to reach the target accuracy. For simplicity, the target accuracy is set as the final accuracy of RS. *3) Processing latency* quantifies the time cost for processing each streaming data. *4) Memory and 5) energy consumption* measure the peak memory footprint and overall energy cost of Titan framework for evaluating system overheads.

Parameter Configuration. The default learning rates are 0.1 for AlexNet, MobileNet and SqueezeNet and 0.005 for other larger models, reduced by a factor of 0.95 per 100 training rounds. The training batch size is 10 to satisfy the memory constraint of common devices as elaborated in §2.2. The velocity of on-device data stream is set to 100 samples per training round, indicating that 10 out of 100 streaming samples are selected as training data batch for model update in each round. For coarse-grained filter, we use the first model block[3] for feature extraction, and set the size budget for the buffered candidate dataset to 30 samples.

4.2 Overall Performance

We begin by comparing the overall performance of Titan with all baselines on three representative mobile computing tasks.
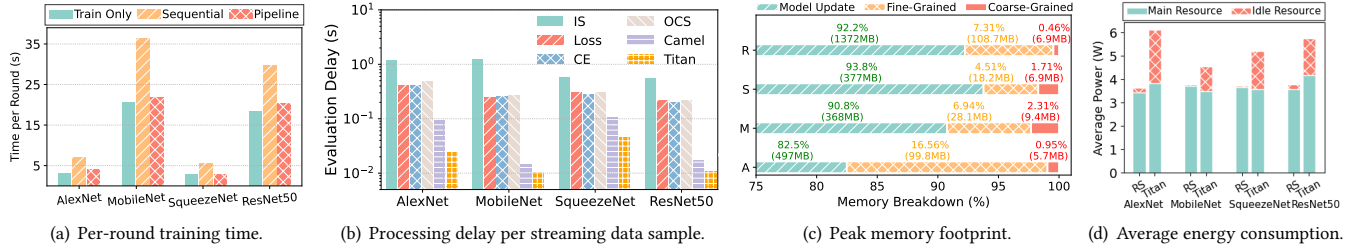
Titan **significantly reduces the wall-clock time to reach target accuracy.** Table 1 summarizes the time taken by different methods to reach target accuracy, which is normalized by the time of RS for clearer comparison. Compared with the most lightweight baseline, Titan reduces the training time by 30–43% for IC task, 23% for AR task, and 29% for HAR task. We observe that most baselines have significantly longer training time than Titan, caused by the extra delay of computing each streaming sample's importance and inferior improvement in model training performance. In contrast,

---

[2]We use mobile CPU for model update and GPU for data selection as *1)* CPUs train models faster than GPUs on current mobile devices [9, 26, 53], *2)* CPUs are more supported by today's on-device training libraries [54], and *3)* using GPU for model update and CPU for data selection can be viewed as cases with varied amounts of idle computing resources, analyzed in Appendix B.
[3]Current ML models typically consists of several blocks with similar structures and each block is composed of several neural network layers.

Table 1: Overall performance of Titan **and baselines, where** blue **highlights the top value. For baselines failing to reach target accuracy, we simply present the normalized time of entire model training process.**

Task	Model	Normalized Time-to-Accuracy (×)								Final Model Accuracy (%)							
		RS	IS	LL	HL	CE	OCS	Camel	Titan	RS	IS	LL	HL	CE	OCS	Camel	Titan
IC	AlexNet	1.00	3.25	3.98	3.98	3.59	4.06	2.07	0.70	71.2	73.5	18.2	34.3	71.6	62.3	71.3	74.5
	MobileNet	1.00	3.22	3.45	3.45	3.41	3.67	1.15	0.57	69.2	69.5	17.7	13.9	69.6	38.1	68.7	75.4
	SqueezeNet	1.00	3.96	3.97	3.97	3.04	4.06	2.07	0.69	76.2	73.0	18.3	45.0	78.0	40.7	75.6	79.0
	ResNet50	1.00	2.32	3.14	3.14	2.20	2.18	1.11	0.66	76.5	78.0	22.3	34.9	81.7	27.3	76.8	81.1
AR	ResNet34	1.00	2.04	3.14	3.14	2.96	3.19	0.81	0.77	76.0	78.7	14.7	58.8	73.2	59.4	76.5	79.8
HAR	MLP	1.00	3.56	6.30	6.47	5.28	14.4	12.5	0.71	75.5	77.5	45.5	21.8	60.9	68.0	75.6	76.7

(a) Per-round training time.

(b) Processing delay per streaming data sample.

(c) Peak memory footprint.

(d) Average energy consumption.

Figure 6: Overall analysis of Titan**'s system overhead, including time, memory and energy.**

Titan's data selection method is theoretically guaranteed to optimize the model performance in each round and its pipeline design further overlaps the extra time cost.

Titan **maintains or improves the final inference accuracy of on-device models.** Table 1 shows that Titan achieves the highest final accuracy for most ML models, including AlexNet, MobileNetV1, SqueezeNet and ResNet34, and achieves the second-best accuracy on other ML models with only marginal accuracy drop compared to the top baseline, such as 0.6% drop compared to CE on ResNet50 and 0.8% drop compared to IS on MLP.

Titan **reduces the processing time of each streaming data to millisecond-level.** As shown in Figure 6(b), Titan achieves the lowest processing time and highest throughput for data importance computation, with only $4-13$ms across different model structures. The millisecond-level latency is attributed to the time-efficiency of coarse-grained filter, and further facilitates the practical deployment of Titan in common applications without compromising the quality of real-time service.

Titan **introduces marginal system overheads, including peak memory footprint and overall energy consumption.** Figure 6(c) breaks down the memory footprint of Titan, indicating that the pipeline design incurs less than 10% extra memory footprint compared with original model training process, such as 105MB, 37MB, 25MB and 114MB for AlexNet, MobileNet, SqueezeNet and ResNet50. The high memory costs for AlexNet and ResNet50 are attributed to their large parameter sizes, and for lightweight models like MobileNet and SqueezeNet, Titan incurs less than 40MB memory overhead. Figure 6(d) compares the average device power of original model training (*i.e.* RS) and Titan framework. We notice that while our system increases device power (1.68×, 1.21×, 1.39×, and 1.53× higher than RS for four models), it reduces the wall-clock training time (0.7×, 0.57×, 0.69×, and 0.66×). As a result, the total

energy consumption becomes 1.17×, 0.69×, 0.96×, and 1.01× compared to RS, reflecting a 31% reduction to 18% increase. Accordingly, Titan leads to only marginal or even reduced energy costs, but achieves much less training time and higher model accuracy.

4.3 Component-Wise Analysis

We then analyze the role of each key component in Titan, including fine-grained selection, coarse-grained filter and pipeline design.

Fine-Grained Selection. To show the individual impact of fine-grained selection strategy C-IS, we compare the training processes of different data selection methods in Figure 7. Across various model structures, C-IS consistently achieves the best model training performance, with 5.8% increase in final accuracy and 1.59× speedup in model convergence rate on AlexNet, 4.8% and 1.62× on MobileNetV1, 3.1% and 1.43× on SqueezeNet, 4.9% and 1.72× on ResNet50, which coincides the theoretical optimality of C-IS analyzed in §3.2.

Coarse-Grained Filter. We further conduct a comparison between only individual fine-grained selection (C-IS) and Titan with different number n of model blocks for feature extraction in coarse-grained filter (Titan-n). Empirical results in Figure 8 reveal the following results: *1)* Compared with C-IS, coarse-grained filter significantly reduces the processing delay of each streaming data, achieving speedup of 32×, 40×, 6.5×, 94× on AlexNet, MobileNetV1, SqueezeNet and ResNet50; *2)* The shallow features extracted by the first model block exhibit satisfactory performance in selecting a candidate dataset with potential high importance for fine-grained data selection, with only 0.1%–0.4% model accuracy drop in various model structures, compared with the ideal case of conducting C-IS on all streaming data; *3)* When leveraging more model blocks for feature extraction, the effectiveness of Titan seems to gradually degrade. This is because deeper model layers tend to extract more concentrated and similar features for data samples within the same
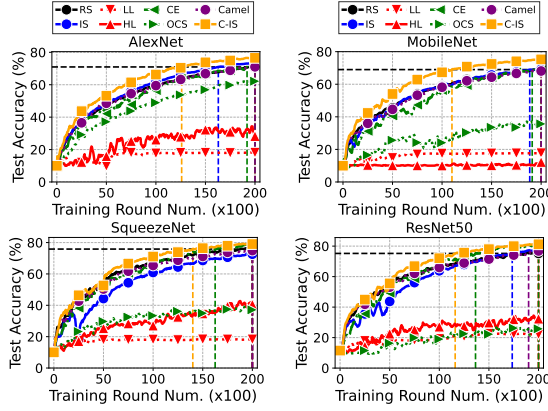
Figure 7: Training curves of different methods. Horizontal line denotes target accuracy, and vertical lines denote numbers of rounds to reach such accuracy.
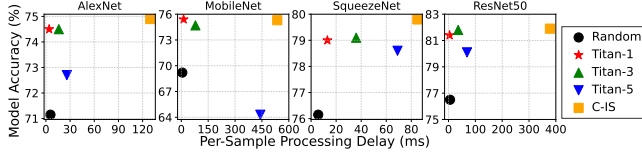

Figure 8: Impact of coarse-grained filter on data processing delay and final model accuracy across different models and model block numbers for feature extraction.

class, making it more difficult to filter out diverse data for intra-class data selection. Consequently, we propose to leverage the first model block in practice for high time-efficiency and stable training performance improvement.

Pipeline Design. To show the role of the pipeline design in reducing time overhead and resource conflict, we visualize the per-round time of only model training, sequential execution and co-execution of model update and data selection in Figure 6(a), which demonstrates that pipeline incurs negligible time for synchronizing the model and data between different processes compared with practical model training time.

4.4 Application to Various Scenarios.

In Appendix B, we provide the empirical results when applying Titan to various practical scenarios, including fluctuant idle computing resources, federated learning and noisy on-device data streams.

5 Other Related Work

In §2.3, we provided a thorough introduction of other data selection approaches and here we review other relevant works.

On-Device Model Training. Recently, there has been a trend towards moving model training from cloud servers to the resource-constrained devices. Prior works focused on improving the utilization of hardware resource (*e.g.*, memory, storage and computing resource) to enhance training efficiency, such as optimizing memory allocation to increase training batch size [15, 54], exploring the co-execution of multiple types of computing resources to accelerate computation [26, 53, 58], and offloading computation to cloud server [40, 55, 61]. However, only a few works noticed the

under-utilization of on-device data resource, which address such issue through cloud-side data distribution estimation and model pre-training [33, 60] before model deployment. Therefore, Titan is complementary to previous works.

Two-Stage Architecture. The design of two-stage system has been widely adopted in industrial recommendation system [5, 8, 11, 12] to recommend highly personalized items from a vast item space in real-time. In the first stage, one or multiple efficient retrieval models are used to produce a candidate set that contains thousands of items from the whole item space. Then, in the second stage, a more powerful model re-ranks the candidate items and recommends the top few items to the user. Such design allows for a trade-off between the system scalability and performance. In the area of on-device data selection, to the best of our knowledge, we are the first to consider leveraging the two-stage design to simultaneously enhance the effectiveness and efficiency of data utilization to improve model training performance.

Difference with Federated Learning (FL). For motivation and setting, we target model training on a single device using local data, while FL trains model across multiple devices with heterogeneous data distributions. As a result, FL focuses on mitigating the negative impact of cross-device data heterogeneity on global model performance, and does not fully consider the data utilization for local model training on each single device. In contrast, we tackle data under-utilization problem in a broader scenario of on-device model training with both theoretical and empirical analysis, which has not been studied before to the best of our knowledge. In Appendix B, we extend our evaluation to an FL setting with 50 devices and heterogeneous local data distributions, where we achieve a 3.17× speedup in global model convergence and a 2.03% increase in accuracy. However, FL-specific data utilization methods are not applicable to our single-device setting, as they aim to tackle non-i.i.d data distribution across devices, which rely on the presence of multiple devices.

6 Conclusion

In this work, we identify that the under-utilization of on-device data resource hinders the successful model training process for edge computing tasks. To address this issue, we propose an on-device data selection framework Titan to simultaneously achieve high effectiveness and time-and-resource efficiency for on-device data utilization through an optimal data selection strategy, a two-stage architecture and a pipeline design. Extensive evaluation on real-world device and representative edge computing tasks demonstrate the remarkable advantages of Titan in final model accuracy and wall-clock training time compared with conventional cloud-side data selection approaches, with minor additional system costs.

References

[1] 2019. Siri - Apple. https://www.apple.com/siri//.

[2] 2024. Google Lens - Search What You See. https://lens.google/.

[3] 2024. Microsoft SwiftKey Keyboard. https://www.microsoft.com/en-us/swiftkey.

[4] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *Annual Conference on Machine Learning and Systems (MLSys)*. 374–388.

[5] Fedor Borisyuk, Krishnaram Kenthapadi, David Stein, and Bo Zhao. 2016. CaS-MoS: A Framework for Learning Candidate Selection Models over Structured Queries and Documents. In *SIGKDD*. 441–450.

[6] Dongqi Cai, Qipeng Wang, Yuanqiang Liu, Yunxin Liu, Shangguang Wang, and Mengwei Xu. 2021. Towards ubiquitous learning: A first measurement of on-device training performance. In *EMDL*. 31–36.

[7] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. 2020. Selection via Proxy: Efficient Data Selection for Deep Learning. In *International Conference on Learning Representations (ICLR)*.

[8] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys*. 191–198.

[9] Anish Das, Young D Kwon, Jagmohan Chauhan, and Cecilia Mascolo. 2022. Enabling on-device smartphone gpu based training: Lessons learned. In *PerCom Workshops*. 533–538.

[10] Apple Developer. 2023. Maximum build file sizes. https://developer.apple.com/help/app-store-connect/reference/maximum-build-file-sizes/.

[11] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *ACM The Web Conference (WWW)*. 1775–1784.

[12] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time. In *ACM The Web Conference (WWW)*. 1775–1784.

[13] Dante Everaert and Christopher Potts. 2024. GIO: Gradient Information Optimization for Training Dataset Selection. In *International Conference on Learning Representations (ICLR)*.

[14] Amirata Ghorbani and James Y. Zou. 2019. Data Shapley: Equitable Valuation of Data for Machine Learning. In *International Conference on Machine Learning (ICML)*. 2242–2251.

[15] In Gim and JeongGil Ko. 2022. Memory-efficient DNN training on mobile devices. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 464–476.

[16] Chen Gong, Zhenzhe Zheng, Yunfeng Shao, Bingshuai Li, Fan Wu, and Guihai Chen. 2024. ODE: An Online Data Selection Framework for Federated Learning With Limited Storage. *IEEE/ACM Transactions on Networking (TON)* 32, 4 (2024), 2794–2809.

[17] Chen Gong, Zhenzhe Zheng, Fan Wu, Xiaofeng Jia, and Guihai Chen. 2024. Delta: A Cloud-assisted Data Enrichment Framework for On-Device Continual Learning. In *International Conference on Mobile Computing and Networking (MobiCom)*. 1408–1423.

[18] Chen Gong, Zhenzhe Zheng, Fan Wu, Yunfeng Shao, Bingshuai Li, and Guihai Chen. 2023. To Store or Not? Online Data Selection for Federated Learning with Limited Storage. In *ACM The Web Conference (WWW)*. 3044–3055.

[19] Google. [n. d.]. Android Developers: APK Expansion Files. https://developer.android.com/google/play/expansion-files.

[20] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv:1706.02677* (2017).

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE / CVF Computer Vision and Pattern Recognition Conference (CVPR)*. 770–778.

[22] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[23] Kai Huang, Boyuan Yang, and Wei Gao. 2023. ElasticTrainer: Speeding Up On-Device Training with Runtime Elastic Tensor Selection. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 56–69.

[24] HUAWEI. 2023. HUAWEI WiFi AX3 Pro. https://consumer.huawei.com/en/routers/ax3-pro/specs/.

[25] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv:1602.07360* (2016).

[26] Fucheng Jia, Deyu Zhang, Ting Cao, Shiqi Jiang, Yunxin Liu, Ju Ren, and Yaoxue Zhang. 2022. CoDL: efficient CPU-GPU co-execution for deep learning inference on mobile devices. In *ACM International Conference on Mobile Systems,* *Applications, and Services (MobiSys)*. 209–221.

[27] Angelos Katharopoulos and François Fleuret. 2017. Biased Importance Sampling for Deep Neural Network Training. (2017). arXiv:1706.00043

[28] Angelos Katharopoulos and François Fleuret. 2018. Not All Samples Are Created Equal: Deep Learning with Importance Sampling. In *International Conference on Machine Learning (ICML)*. 2530–2539.

[29] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*.

[31] Chenning Li, Xiao Zeng, Mi Zhang, and Zhichao Cao. 2022. PyramidFL: a fine-grained client selection framework for efficient federated learning. In *Annual International Conference on Mobile Computing and Networking (MobiCom)*. 158–171.

[32] Yiming Li, Yanyan Shen, and Lei Chen. 2022. Camel: Managing Data for Efficient Stream Learning. In *SIGMOD*. 1271–1285.

[33] Bingyan Liu, Yuanchun Li, Yunxin Liu, Yao Guo, and Xiangqun Chen. 2020. Pmc: A privacy-preserving deep learning model customization framework for edge computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)* 4, 4 (2020), 1–25.

[34] Haibo Liu, Chen Gong, Zhenzhe Zheng, Shengzhong Liu, and Fan Wu. 2025. Enabling Real-Time Inference in Online Continual Learning via Device-Cloud Collaboration. In *Proceedings of the ACM on Web Conference(WWW)*. 2043–2052.

[35] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. [n. d.]. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Artificial Intelligence and Statistics (AISTATS)*.

[36] Baharan Mirzasoleiman, Jeff A. Bilmes, and Jure Leskovec. 2020. Coresets for Data-efficient Training of Machine Learning Models. In *International Conference on Machine Learning (ICML)*. 6950–6960.

[37] NVIDIA. 2023. Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[38] OpenAI. 2023. ChatGPT General FAQ. https://help.openai.com/en/articles/6783457-chatgpt-general-faq.

[39] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. 2021. Clusterfl: a similarity-aware federated learning system for human activity recognition. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 54–66.

[40] Xiaoyi Pang, Zhibo Wang, Jingxin Li, Ruiting Zhou, Ju Ren, and Zhetao Li. 2022. Towards online privacy-preserving computation offloading in mobile edge computing. In *IEEE International Conference on Computer Communications (INFOCOM)*. 1179–1188.

[41] Allan Pinkus. 1999. Approximation theory of the MLP model in neural networks. *Acta numerica* 8 (1999), 143–195.

[42] Omead Pooladzandi, David Davini, and Baharan Mirzasoleiman. 2022. Adaptive second order coresets for data-efficient machine learning. In *International Conference on Machine Learning (ICML)*. 17848–17869.

[43] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. 2017. iCaRL: Incremental Classifier and Representation Learning. In *IEEE / CVF Computer Vision and Pattern Recognition Conference (CVPR)*. 5533–5542.

[44] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The Annals of Mathematical Statistics* (1951), 400–407.

[45] Burr Settles. 2009. Active learning literature survey. (2009).

[46] Vatsal Shah, Xiaoxia Wu, and Sujay Sanghavi. [n. d.]. Choosing the Sample with Lowest Loss makes SGD Robust. In *Artificial Intelligence and Statistics (AISTATS)*.

[47] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354–359.

[48] Prashanthi SK, Sai Anuroop Kesanapalli, and Yogesh Simmhan. 2022. Characterizing the performance of accelerated Jetson edge devices for training deep learning models. *SIGMETRICS* 6, 3 (2022), 1–26.

[49] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. 2018. Don't Decay the Learning Rate, Increase the Batch Size. In *International Conference on Learning Representations (ICLR)*.

[50] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *International Conference on Computer Vision (ICCV)*. 843–852.

[51] Ammar Tahir, Yongzhou Chen, and Prashanti Nilayam. 2022. FedSS: Federated learning with smart selection of clients. *arXiv preprint arXiv:2207.04569* (2022).

[52] Tianxiang Tan and Guohong Cao. 2022. Deep learning on mobile devices through neural processing units and edge computing. In *IEEE International Conference on Computer Communications (INFOCOM)*. 1209–1218.

[53] Manni Wang, Shaohua Ding, Ting Cao, Yunxin Liu, and Fengyuan Xu. 2021. AsyMo: scalable and efficient deep-learning inference on asymmetric mobile CPUs. In *Annual International Conference on Mobile Computing and Networking (MobiCom)*. 215–228.

[54] Qipeng Wang, Mengwei Xu, Chao Jin, Xinran Dong, Jinliang Yuan, Xin Jin, Gang Huang, Yunxin Liu, and Xuanzhe Liu. 2022. Melon: breaking the memory wall for resource-efficient on-device machine learning. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 450–463.

[55] Shibo Wang, Shusen Yang, and Cong Zhao. 2020. SurveilEdge: Real-time video query based on collaborative cloud-edge deep learning. In *IEEE International Conference on Computer Communications (INFOCOM)*. 2519–2528.

[56] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[57] Jianyu Wei, Ting Cao, Shijie Cao, Shiqi Jiang, Shaowei Fu, Mao Yang, Yanyong Zhang, and Yunxin Liu. 2023. NN-Stretch: Automatic Neural Network Branching for Parallel Inference on Heterogeneous Multi-Processors. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 70–83.

[58] Daliang Xu, Mengwei Xu, Qipeng Wang, Shangguang Wang, Yun Ma, Kang Huang, Gang Huang, Xin Jin, and Xuanzhe Liu. 2022. Mandheling: mixed-precision on-device DNN training with DSP offloading. In *Annual International Conference on Mobile Computing and Networking (MobiCom)*. 214–227.

[59] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. In *ACM The Web Conference (WWW)*. 2125–2136.

[60] Mengwei Xu, Feng Qian, Qiaozhu Mei, Kang Huang, and Xuanzhe Liu. 2018. DeepType: On-Device Deep Learning for Input Personalization Service with Minimal Privacy Concern. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)* (2018), 197:1–197:26.

[61] Dixi Yao, Liyao Xiang, Zifan Wang, Jiayu Xu, Chao Li, and Xinbing Wang. 2021. Context-aware compilation of dnn training pipelines across edge and cloud. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)* 5, 4 (2021), 1–27.

[62] Rongjie Yi, Ting Cao, Ao Zhou, Xiao Ma, Shangguang Wang, and Mengwei Xu. 2023. Boosting DNN Cold Inference on Edge Devices. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 516–529.

[63] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. 2022. Online Coreset Selection for Rehearsal-based Continual Learning. In *International Conference on Learning Representations (ICLR)*.

[64] Xiao Zeng, Ming Yan, and Mi Zhang. 2021. Mercury: Efficient on-device distributed dnn training via stochastic importance sampling. In *Sensys*. 29–41.

[65] Peilin Zhao and Tong Zhang. 2015. Stochastic Optimization with Importance Sampling for Regularized Loss Minimization. In *International Conference on Machine Learning (ICML)*. 1–9.

[66] Yan Zhuang, Zhenzhe Zheng, Fan Wu, and Guihai Chen. 2024. LiteMoE: Customizing On-device LLM Serving via Proxy Submodel Tuning. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 521–534.

A Proofs

A.1 Proof of Theorem 1

According to the model update formula in Equation (1), we have $w_{t+1} = w_t - \eta_t \nabla L(\mathcal{B}, w_t)$, which leads to:

$$
\mathbb{E}_{\mathcal{B} \sim P_t(\mathcal{S})} \left[||w_t - w^*||_2^2 - ||w_{t+1} - w^*||_2^2 \right]
$$
$$
= \mathbb{E}_{\mathcal{B} \sim P_t(\mathcal{S})} \left[w_t^\top w_t - 2 w_t^\top w^* + 2 w_{t+1}^\top w^* - w_{t+1}^\top w_{t+1} \right]
$$
$$
= \mathbb{E}_{\mathcal{B} \sim P_t(\mathcal{S})} \left[w_t^\top w_t - 2 w_t^\top w^* + 2 \left(w_t - \eta_t \nabla L(\mathcal{B}, w_t) \right)^\top w^* \right.
$$
$$
\left. - \left(w_t - \eta_t \nabla L(\mathcal{B}, w_t) \right)^\top \left(w_t - \eta_t \nabla L(\mathcal{B}, w_t) \right) \right]
$$
$$
= \mathbb{E}_{\mathcal{B} \sim P_t(\mathcal{S})} \left[-2 \eta_t \cdot \nabla L(\mathcal{B}, w_t)^\top (w^* - w_t) - \eta_t^2 \left\| \nabla L(\mathcal{B}, w_t) \right\|^2 \right]
$$
$$
\overset{(a)}{=} -2 \eta_t \cdot (w^* - w_t)^\top \nabla L(\mathcal{S}, w_t) + \eta_t^2 \cdot \mathbb{E}_{\mathcal{B} \sim P_t(\mathcal{S})} \left\| \nabla L(\mathcal{B}, w_t) \right\|^2,
$$

where equality (a) holds because of the unbiased data sampling $\mathcal{B} \sim P_t(\mathcal{S})$. Next, we leverage the definition of variance, i.e., $\mathbb{V}[x] = \mathbb{E}[x^2] - (\mathbb{E}[x])^2$ to further decompose the final term:

$$
\mathbb{E}_{\mathcal{B} \sim P_t(\mathcal{S})} \left\| \nabla L(\mathcal{B}, w_t) \right\|^2
$$
$$
= \mathbb{V}_{\mathcal{B} \sim P_t(\mathcal{S})} \left[\nabla L(\mathcal{B}, w_t) \right] + \left\| \mathbb{E}_{\mathcal{B} \sim P_t(\mathcal{S})} \left[\nabla L(\mathcal{B}, w_t) \right] \right\|^2
$$
$$
= \mathbb{V}_{\mathcal{B} \sim P_t(\mathcal{S})} \left[\nabla L(\mathcal{B}, w_t) \right] + \left\| \nabla L(\mathcal{S}, w_t) \right\|^2,
$$

which leads to the conclusion in Theorem 1.

A.2 Proof of Theorem 2

We first decompose the gradient variance of the data batch $\mathcal{B}$ selected from dataset $\mathcal{S}$ into the weighted variances of sub-batch $\mathcal{B}_y$ selected from data-subset $\mathcal{S}_y$ for each class $y \in \mathcal{Y}$:

$$
\mathbb{V}_{\mathcal{B} \sim P_t(\mathcal{S})} \left[\nabla L(w, \mathcal{B}) \right] = \mathbb{V}_{\mathcal{B} \sim P_t(\mathcal{S})} \left[\sum_{y \in \mathcal{Y}} \frac{|\mathcal{S}_y|}{|\mathcal{S}|} \mathbb{E}_{(x,y) \in \mathcal{B}_y} \left[\nabla l(w, x, y) \right] \right]
$$
$$
= \sum_{y \in \mathcal{Y}} \frac{|\mathcal{S}_y|^2}{|\mathcal{S}|^2} \mathbb{V}_{\mathcal{B}_y \sim P_{t,y}(\mathcal{S}_y)} \left[\mathbb{E}_{(x,y) \in \mathcal{B}_y} \left[\nabla l(w, x, y) \right] \right] \quad (d)
$$
$$
= \sum_{y \in \mathcal{Y}} \frac{|\mathcal{S}_y|^2}{|\mathcal{S}|^2 \cdot |\mathcal{B}_y|} \mathbb{V}_{(x,y) \sim P_{t,y}(\mathcal{S}_y)} \left[\nabla l(w, x, y) \right] \quad (e).
$$

Equation(d) decomposes the overall batch selection process into sub-processes for each class, and Equation(e) holds because each sample in sub-batch $\mathcal{B}_y$ is selected from $\mathcal{S}_y$ with strategy $P_{t,y}$. According to the variance definition (i.e. $\mathbb{V}[x] = \mathbb{E}[x^2] - (\mathbb{E}[x])^2$), we can further decompose the gradient variance:

$$
(e) = \sum_{y \in \mathcal{Y}} \frac{|\mathcal{S}_y|^2}{|\mathcal{S}|^2 \cdot |\mathcal{B}_y|} \cdot \left[\sum_{(x,y) \in \mathcal{S}_y} P_{t,y}(x) \cdot \frac{\left\| \nabla l(w, x, y) \right\|^2}{\left[P_{t,y}(x) \cdot |\mathcal{S}_y| \right]^2} - \right.
$$
$$
\left. \left\| \sum_{(x,y) \in \mathcal{S}_y} P_{t,y}(x) \cdot \frac{\nabla l(w, x, y)}{P_{t,y}(x) \cdot |\mathcal{S}_y|} \right\|^2 \right] \quad (f)
$$
$$
= \sum_{y \in \mathcal{Y}} \frac{|\mathcal{S}_y|^2}{|\mathcal{S}|^2 \cdot |\mathcal{B}_y|} \cdot \left[\sum_{(x,y) \in \mathcal{S}_y} \frac{\left\| \nabla l(w, x, t) \right\|^2}{|\mathcal{S}_y|^2 \cdot P_{t,y}(x)} - \right.
$$
$$
\left. \left\| \frac{\sum_{(x,y) \in \mathcal{S}_y} \nabla l(w, x, y)}{|\mathcal{S}_y|} \right\|^2 \right] = \sum_{y \in \mathcal{Y}} \alpha_y \cdot (\beta_y - \gamma_y).
$$

Equation(f) holds because in data selection, to ensure the unbiasedness of selected data for model convergence, each selected sample will be weighted by $\frac{1}{probability \times data\ size}$ to achieve $\mathbb{E}_{(x,y) \sim P(\mathcal{S})}[f(x)] = \sum_{(x,y) \in \mathcal{S}} P(x) \cdot \frac{f(x)}{P(x) \cdot |\mathcal{S}|} = \mathbb{E}_{(x,y) \sim \mathcal{S}}[f(x)]$.

A.3 Proof of Lemma 2

According to our previous analysis, term β_y for each class $y \in \mathcal{Y}$ is uniquely determined by its intra-class data selection strategy $P_{t,y}$ while term γ_y is a fixed value. Therefore, we can minimize the overall gradient variance as follows:

First, we derive the minimal β_y^* by optimizing $P_{t,y}$, which can be directly solved using Cauchy-Schwarz inequality:

$$
\beta_y \geq \sum_{(x,y) \in \mathcal{S}_y} \frac{\left\| \nabla l(w_t, x, y) \right\|^2}{|\mathcal{S}_y|^2 \cdot \frac{\left\| \nabla l(w_t, x, y) \right\|}{\sum_{(x',y') \in \mathcal{S}_y} \left\| \nabla l(w_t, x', y') \right\|}}
$$
$$
= \left[\sum_{(x,y) \in \mathcal{S}_y} \frac{\left\| \nabla l(w_t, x, y) \right\|}{|\mathcal{S}_y|} \right]^2,
$$

where the equality holds when $P_{t,y}(x) = \frac{\left\| \nabla l(w_t, x, y) \right\|}{\sum_{(x',y') \in \mathcal{S}_y} \left\| \nabla l(w_t, x', y') \right\|}$ based on the Cauchy-Schwarz inequality.

Second, given $(\beta_y^* - \gamma_y)$ for each class, we minimize the overall objective $\sum_y \alpha_y (\beta_y^* - \gamma_y)$ by optimizing $|\mathcal{B}_y|$, the analytical expression
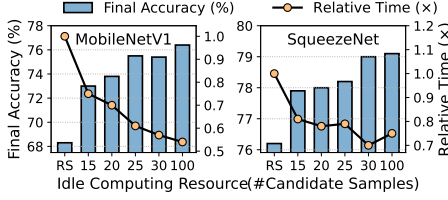
Figure 9: Performance on fluctuant idle computing resource.

of which can also be computed through Cauchy-Schwarz inequality:

$$\sum_{y \in \mathcal{Y}} \alpha_y \cdot (\beta_y^* - \gamma_y) = \sum_{y \in \mathcal{Y}} \frac{|\mathcal{S}_y|^2}{|\mathcal{S}|^2 \cdot |\mathcal{B}_y|} \cdot (\beta_y^* - \gamma_y)$$
$$= \frac{1}{|\mathcal{S}|^2} \cdot \sum_{y \in \mathcal{Y}} \frac{|\mathcal{S}_y|^2 \cdot (\beta_y^* - \gamma_y)}{|\mathcal{B}_y|},$$

which is minimized when $\mathcal{B}_y \propto |\mathcal{S}_y|\sqrt{\beta_y^* - \gamma_y}$ according to the Cauchy-Schwarz inequality and leads to Lemma 2.

B Supplementary Evaluation

To further demonstrate the generality of Titan framework, we also evaluate its performance in diverse practical scenarios to demonstrate the robustness and generality of Titan, such as fluctuant on-device idle computing resources, federated learning scenario and noisy on-device data. The experiments are mainly conducted on image classification task.

Fluctuant Idle Computing Resources. In practice, the co-running applications may occupy varied proportions of computing resources. When there exists more idle computing resource, a larger candidate dataset can be filtered out by coarse-grained filter to facilitate a higher-quality fine-grained data selection process. Experimental results in Figure 9 show that when the candidate dataset size rises from 15 to 100, the final accuracy of Titan is increased from $73.0-77.9\%$ to $76.4-79.1\%$ and the training time reduction also rises from $19-25\%$ to $25-46\%$. The consistent improvement in model training performance demonstrates Titan's robustness to devices with varying idle computing resources.

Federated Learning. We evaluate the performance of Titan in a federated setting setting using CIFAR-10 and MobileNetV1. In this scenario, we initialize the training process with 50 devices, and the data distribution of across these devices follow a non-IID pattern as described in previous work [4, 51]. The data on each device covers only 5 classes. In each training round, random 20% devices participated in the model training process, which independently select $|\mathcal{B}|$ samples from a pool of v real-time samples, update their local models for 3 local training iterations, and upload the updated model to the centralized server for model aggregation. As shown in Figure 10, compared with the second best best approach, Titan achieves an increase of 2.03% in final accuracy and speeds up the training time (i.e., number of communication rounds) to target accuracy by a factor of 3.17×. These findings highlight the potential of applying Titan to federated learning for improving the training performance of global model.

Noisy Data Streams. Generally, Titan framework can improve the model training performance for arbitrary on-device data distribution, as the importance of each data sample in our work is measured by its theoretical contribution to model performance over on-device personal data (elaborated in Theorem 1 and Lemma 2). Specifically, for streaming data with noisy or shifted data distributions, Titan aims to minimize the discrepancy between the optimal model (for noisy or dynamic data distribution) and current model (trained with the selected training data batch, which essentially reflect the model's robustness to noisy data and generalization performance to domain shift. Furthermore, we conduct extensive experiments to demonstrate the applicability of Titan framework to noisy and dynamic data streams.

For noisy data, we incorporate two distinct type of noise: (i) feature noise to emulate the noise introduced by an unstable environment during data collection, where we randomly select 40% of data samples and add Gaussian noise to their input feature x; (ii) label noise to simulate the incorrect labels arising from automatic-labeling where we randomly revise the label y of 40% data samples. Experiment results on CIFAR-10 with MobileNetV1 are presented in Figure 11, which demonstrate that: (i) In various noise settings, Titan consistently outperforms all the baselines, achieving 6% and 3.4% higher final model accuracy in the settings of feature noise and label noise. Moreover, Titan accelerates the time-to-accuracy by a factor of 1.83× and 1.35× in these two settings; (ii) Titan exhibits higher robustness to feature noise compared to label noise, because label noise introduces larger errors to the sample gradients and leads to inaccurate data evaluation.
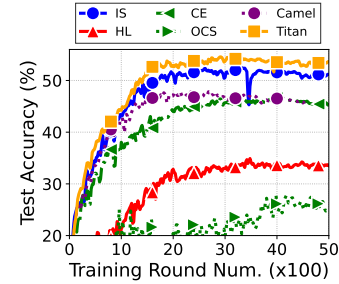

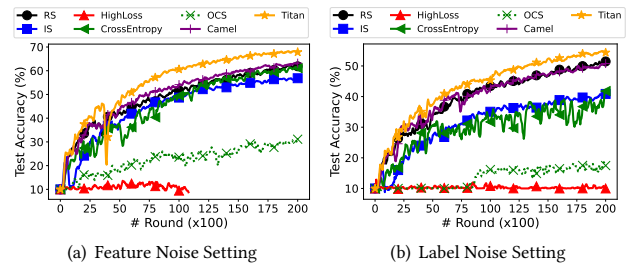**Figure 10: Performance in federated learning scenario.**

(a) Feature Noise Setting　　　(b) Label Noise Setting

Figure 11: Titan **Performance on noisy data distribution.**