A Two-Stage Data Selection Framework for On-Device Model Training

Chen Gong
Shanghai Jiao Tong University
Shanghai, China
gongchen@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*—In mobile computing, the demand for model training on devices is escalating due to data privacy and personalized service needs. However, we observe that current 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 significant 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 devices and three typical mobile computing tasks with diverse 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.**

***Index Terms*—mobile computing, on-device machine learning, data selection and utilization**

I. Introduction

Machine learning (ML) models have been widely embedded in mobile applications to provide diverse intelligent services, such as image tagging in Google Lens [1], command recognition in Siri [2], text prediction in Microsoft SwiftKey [3] and 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 [4]–[6], or multiple devices to collaboratively train a global model that can generalize well across different data distributions [7].

A successful ML model training process highly relies on an abundant high-quality dataset for model training [8]–[10]. On one hand, a large-scale training dataset is essential to the generalization capability of the final model [9]. On the other hand, massive high-quality data helps to stabilize the parameter update process of model training and further reduce the number of training rounds to reach a target accuracy [10]. As a result, on the cloud side, it is common to collect extensive training data for iterative model updates, such as 29 million games for training AlphaGo [11] and 500 billion tokens for pre-training ChatGPT-3.5 [12]. Similarly, for the device side, it is also desirable to fully utilize the on-device data resource to achieve satisfactory model training performance.

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 [4], [13], co-using multiple types of computing resources to speed up model inference and training [14]–[17], dynamically adjusting the size of trainable parameters to improve training efficiency [18], [19] and 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 $3.5\times$ 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 §II-B). 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?*

A practical data selection framework for on-device model training has to achieve *effectiveness* and *efficiency* simultaneously, which is challenging: *1) 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 the improvement of model training performance. *2) 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 (or quality) 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 §II-C indicate that conventional cloud-side data selection approaches such as importance sampling [20], [21], heuristic selection [22]–[26] and coreset selection [27]–[29] fail to be applied to device side due to ineffectiveness or inefficiency.

In this work, we address the above challenge by proposing a **t**wo-stage onl**i**ne da**t**a selectio**n** 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.

In summary, our main contributions are as follows:

- To the best of our knowledge, we are the first to point out the severity of data 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.

II. Background and Motivation

A. 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 $\mathcal{P}$:

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

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 [30] 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 $\mathcal{B} = \{(x_i, y_i)|1 \leq i \leq |\mathcal{B}|\}$ 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 alternatively and iteratively.

B. Under-Utilization of On-Device Data Resources

We elaborate three characteristics of edge devices, which lead to the under-utilization of on-device data resources.

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 [31]). Second, the on-device per-sample training time is relatively long due to the limited computing hardware [15]. 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).

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 chathistory 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 [32] and no more than 2GB for a Google Play app [33]. Furthermore, for low-end smartphones or IoT devices like HUAWEI WiFi AX3 [34] 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.

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

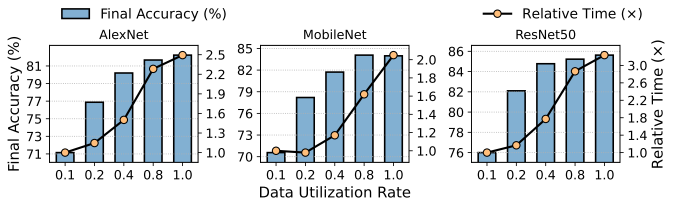


Fig. 1. Final accuracy and normalized time of model training processes over CIFAR-10 dataset using varied proportions of data for parameter update.

training round, which leads to substantial per-round training time and storage overhead [13], [35]. 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 [20]. Our preliminary experiments on representative device Jetson Nano and dataset CIFAR-10 [36] in Fig. 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 $2.05-3.24\times$. 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.

C. Limitation of 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) [20], [21] 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 §III-B. 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 Fig. 2(a). For device-side efficient deployment, Mercury [37] 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 [22], [24], data representativeness measured by closeness to the distribution centroid in feature space [26] and diversity to other samples [25], and etc.

[1]The variance represents the average difference between the gradient of the selected training data and the expected gradient of the entire dataset.

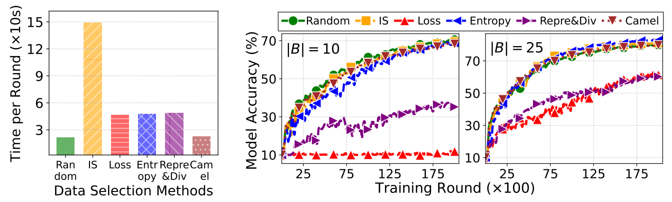

(a) Time per round. (b) Training processes with different batch sizes.

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

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) [27]–[29] 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 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* [29] 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 Fig. 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.

III. Design of Titan

A. Overview

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

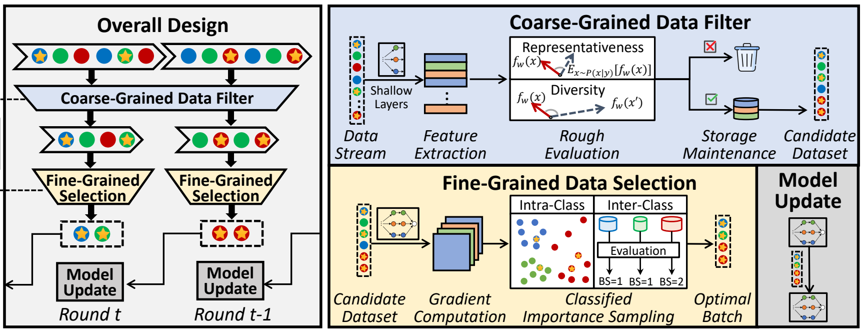


Fig. 3. Overall design and workflow of Titan.

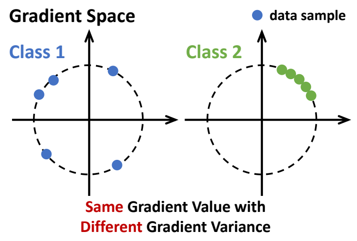


Fig. 4. Simple example to compare IS and C-IS.

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 Fig. 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.

B. 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:

$$I_t(y) \triangleq$$
$$|\mathcal{S}_y| \Big[\underbrace{\mathbb{V}_{(x,y)\sim P_{t,y}}\big[\nabla l(w_t,x,y)\big]}_{\text{variance of gradient}} - \underbrace{\mathbb{V}_{(x,y)\sim P_{t,y}}\big[||\nabla l(w_t,x,y)||_2\big]}_{\text{variance of gradient norm}} \Big]^{\frac{1}{2}} \tag{2}$$

where $|\mathcal{S}_y|$ denotes the size of stored data with class y, $\mathbb{V}_{(x,y)\sim P_{t,y}}[f(x)]$ 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:

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

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 [20], [21] 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 the initial model parameter w_t to the updated model parameter w_{t+1}:

Theorem 1 (Training Performance Measurement [20], [21])**.** *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:*

$$\mathbb{E}_{\mathcal{B}\sim P_t}\Big[\underbrace{||w_t-w^*||_2^2 - ||w_{t+1}-w^*||_2^2}_{\text{reduction in distance to } w^*}\Big] = -\eta_t^2 \cdot \underbrace{\mathbb{V}_{\mathcal{B}\sim P_t}\big[\nabla L(w_t,\mathcal{B}\big]}_{\text{optimized through } P_t}$$
$$+ \underbrace{2\eta_t \cdot (w_t-w^*)^\top \nabla L(w_t,\mathcal{P}) - \eta_t^2 ||\nabla L(w_t,\mathcal{P})||_2^2}_{\text{fixed by initial model parameter } w_t \text{ in each round t}},$$

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

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):*

$$
\begin{aligned}
P_t^*(x,y) &\triangleq \arg\min_{P_t} \mathbb{V}_{\mathcal{B}\sim P_t}\left[\nabla L(w_t,\mathcal{B})\right] \\
&= \arg\min_{P_t} \frac{1}{|\mathcal{B}|}\mathbb{V}_{(x,y)\sim P_t}\left[\nabla l(w_t,x,y)\right] \qquad \text{(a)} \\
&= \left|\left|\nabla l(w_t,x,y)\right|\right|_2 \Big/ \sum_{(x',y')\in\mathcal{S}} \left|\left|\nabla l(w_t,x',y')\right|\right|_2. \qquad \text{(b)}
\end{aligned}
$$

Proof. We observe that Eq.(a) *implicitly assumes an independent selection process for each data sample (x,y) in data batch $\mathcal{B}$*. Eq.(b) was induced by previous IS work [20], [21] according to Cauchy-Schwarz inequality. □

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 $y\in\mathcal{Y}$:*

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

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

where $\mathcal{S}_y \subseteq \mathcal{S}$ and $\mathcal{B}_y \subseteq \mathcal{B}$ are the candidate data and selected data for each class $y\in\mathcal{Y}$.

Proof. The detailed proof please refer to Appendix VI. □

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 Fig. 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 with varying batch sizes.

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 Eq.(2) and Eq.(3), respectively.

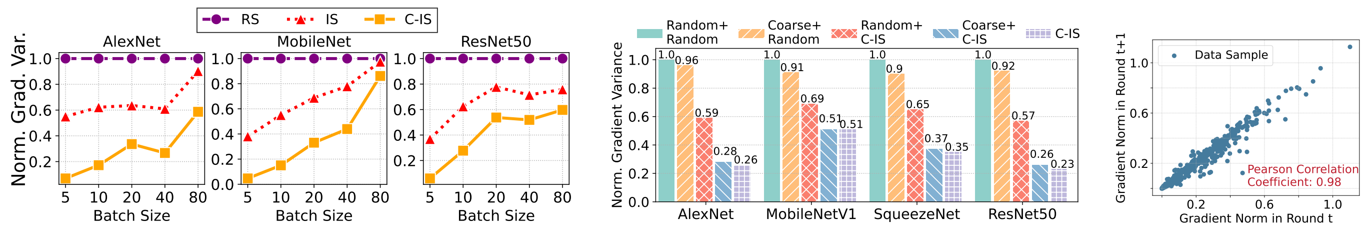
Proof. According to our previous analysis, term β_y for each class $y\in\mathcal{Y}$ is uniquely determined by its intra-class data selection strategy $P_{t,y}$ while term γ_y is a fixed value. Therefore, we can minimize the overall gradient variance as follows: *1)* Derive the minimal β_y^* by optimizing $P_{t,y}$, which can be directly solved using Cauchy-Schwarz inequality. *2)* Given $(\beta_y^* - \gamma_y)$ for each class, minimize the overall objective $\sum_y \alpha_y(\beta_y^* - \gamma_y)$ by optimizing $|\mathcal{B}_y|$, the analytical expression of which can also be computed through Cauchy-Schwarz inequality. □

Intuitive Understanding. The sample importance $I_t(x,y)$ in Eq.(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 Eq.(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 Fig. 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.

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], [29], [38].

C. 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 [39], Titan leverages a two-stage architecture to guarantee both efficiency and effectiveness.



(a) Optimality of C-IS over random sampling (RS) and IS. (b) Effect of coarse-grained filter on C-IS performance. (c) Data importance variation.

Fig. 5. Preliminary experiments on CIFAR-10 dataset and different models with learning rate 0.1 to support some claims.

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 fine-grained selection (*i.e.*, C-IS).

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:

$$\mathrm{Rep}(x,y) = -\Big|\Big|f_w(x) - \underbrace{\mathbb{E}_{\mathcal{P}(x'|y)}\big[f_w(x')\big]}_{\text{class } y'\text{s centroid}}\Big|\Big|_2^2,$$

where $f_w(x)$ denotes the feature extracted by current model w and $\mathcal{P}(x'|y)$ denotes the data distribution of 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:

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

We evaluate the impact of coarse-grained filter on C-IS's ability in reducing gradient variance in Fig. 5(b), where A+B denotes leveraging A to filter out 30 candidate data samples out of 100 samples and employing B to further select 10 samples as data batch. 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 $\mathbb{E}_{\mathcal{P}(x'|y)}\big[f_w(x')\big]$ and average feature norm $\mathbb{E}_{\mathcal{P}(x'|y)}\big|\big|f_w(x')\big|\big|_2^2$ using each streaming data (x, y). Based on these estimators, Titan could realize online coarse-grained filtering by buffering data with the highest $\mathrm{Rep}(x, y) + \mathrm{Div}(x, y)$. 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 §IV-C.

D. 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 Fig. 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 [15], [19]. 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 Fig. 6 in §IV-C.

IV. Evaluation

We first introduce our experiment setup (§IV-A). Then we present the overall performance of Titan (§IV-B) and conduct component-wise analysis (§IV-C). Further, we test the applicability of Titan to different scenarios (§IV-D).

A. 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 [36] consists of $60,000$ images of 10 objects. We train four representative ML models for this task, including the classic dense model AlexNet [40], lightweight models MobileNetV1 [41] and SqueezeNet [42] as well as a larger model ResNet50 [43].
2) Audio Recognition (AR): Google Speech Commands [44] includes $100,000$ sound files of 20 commands collected from $2,000$ users, and we train ResNet34 [43] for this task.
3) Human Activity Recognition (HAR): HARBOX [45] 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 [46] 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 [47] with 4GB RAM, 4 A57 CPU cores and a Maxwell GPU, which has similar hardware and running environment with mainstream devices [48], [49]. 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 cloud-side data selection methods, including: *1) Random selection* (RS) selects random data for model training; *2) Importance sampling* (IS) [20] 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 [22] and low loss LL [51]), cross entropy of the model output logits (CE [24]) or data representativeness and diversity (OCS [25]); *4) Coreset selection* (Camel [29]) 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. *Final inference accuracy* denotes the test accuracy of the finally trained model. *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. *Processing latency* quantifies the time cost for processing each streaming data. *Memory and energy consumption* measure the peak memory footprint and overall energy cost of Titan framework.

[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 [15], [16], [50], *2)* CPUs are more supported by today's on-device training libraries [13], 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 §IV-D.

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 §II-B. 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.

B. Overall Performance

Titan **significantly reduces the wall-clock time to reach target accuracy.** Table I 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 also observe that most baselines have significantly longer training time, caused by the additional delay of computing each streaming sample's importance and inferior improvement in model training performance. In contrast, the data selection strategy of Titan is theoretically guaranteed to optimize the model performance in each round and the pipeline design further overlaps the extra time cost, as elaborated in §IV-C.

Titan **improves the inference accuracy of final model.** Table I 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.

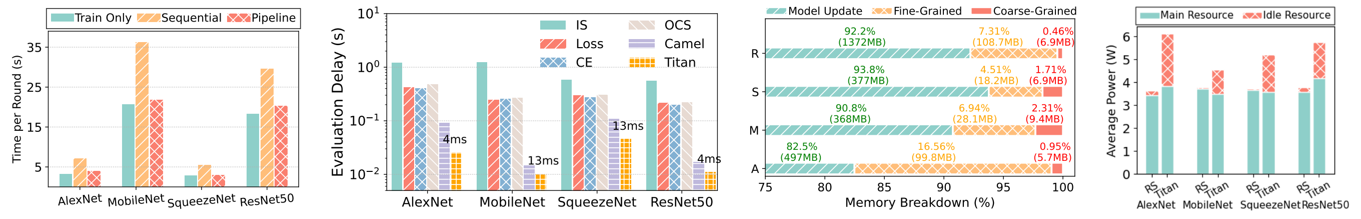
Titan **reduces the processing time of each streaming data to millisecond-level.** As shown in Fig. 6(b), Titan achieves the lowest processing time and highest throughput for data importance computation, with only $4-13$ms across different model structures. The millisecond-level latency is attributed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

Titan **introduces marginal extra memory and energy overhead.** Fig. 6(c) breaks down the memory footprint of Titan, indicating that the co-execution of data selection and model update mostly incurs less than 10% memory cost compared with original model update, such as 105MB for AlexNet, 37MB for MobileNet, 25MB for SqueezeNet and 114MB for 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 due to avoiding the cumbersome model back-propagation process. Fig. 6(d) compares the average device power of only model training

[3]Current ML models typically consists of several blocks with similar structures and each block is composed of several neural network layers.

TABLE I
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) Memory footprint. (d) Energy consumption.

Fig. 6. System overhead analysis of Titan, including per-round training time, processing delay of streaming data, memory footprint and energy consumption.

(*i.e.* RS) and Titan framework. We notice that Titan increases the power by $20-67\%$ because of utilizing idle computing resources for data selection. However, the overall energy consumption, which can be measured by $power \times time$, is increased by only -31% to $+18\%$ due to the model training speedup brought by Titan. Accordingly, although Titan is not specially designed for energy conservation, it leads to only marginal or even reduced energy costs but achieves less training time and higher model accuracy.

C. Component-Wise Analysis

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 Fig. 7. Across various model structures, C-IS consistently achieves the best model training performance, with 5.8% increase in final accuracy and $1.59\times$ reduction in training time on AlexNet, 4.8% and $1.62\times$ on MobileNetV1, 3.1% and $1.43\times$ on SqueezeNet, 4.9% and $1.72\times$ on ResNet50, which coincides the theoretical optimality of C-IS analyzed in §III-B.

Coarse-Grained Filter. We further conduct a comparison between individual fine-grained selection (C-IS) and Titan with the shallowest n model blocks for feature extraction in coarse-grained filter (Titan-n). Empirical results in Fig. 9 reveal the following results: *1)* Compared with individual C-IS, coarse-grained filter significantly reduces the processing delay of each streaming data, achieving speedup of $32\times$ on AlexNet, $40\times$ on MobileNetV1, $6.5\times$ on SqueezeNet, and $94\times$ on ResNet50. This enhances the generality of Titan to applications with varied data velocities and delay-tolerant levels; *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 0.4% model accuracy drop in AlexNet, 0.1% in MobileNetV1, 0.8% in SqueezeNet and 0.5% in ResNet50, compared with the ideal case of conducting C-IS on all streaming data; *3)* When leveraging more model blocks for feature extracton, 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 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 Fig. 6(a), which demonstrates that pipeline incurs negligible time for synchronizing the model and data between different processes compared with practical model training time.

D. Extended Experiments

To further demonstrate the generality of Titan framework, we also evaluate its performance in scenarios of fluctuant on-device idle computing resources and federated learning.

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 Fig. 10 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 further evaluate the performance of Titan in a federated learning (FL) [52] setting with CIFAR-10 and MobileNetV1, where we distribute the dataset to 50 devices following a similar non-IID class pattern with previous work [53]. In each round, 20% devices are randomly selected to participate in model training, which locally update the

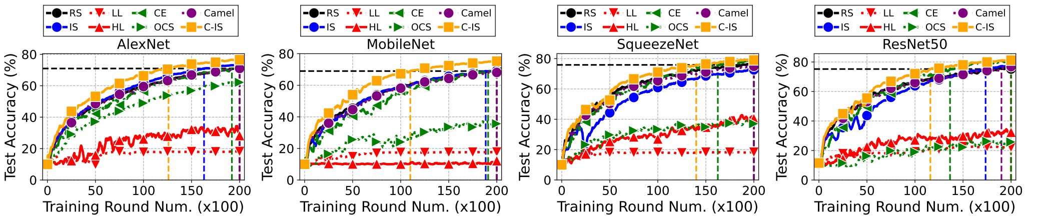


Fig. 7. Training curves of different data selection methods. The horizontal line denotes target accuracy, and vertical lines indicate the required number of rounds to reach target accuracy.

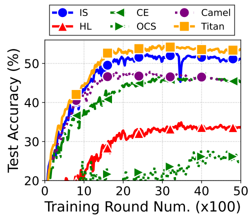


Fig. 8. Performance in federated learning scenario.

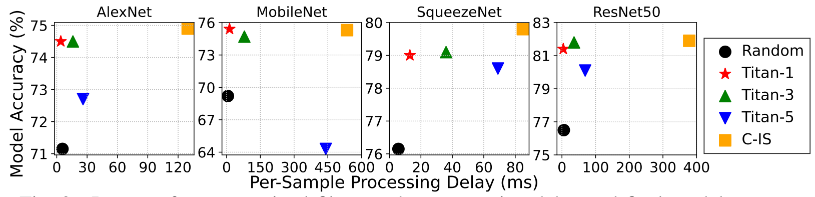


Fig. 9. Impact of coarse-grained filter on data processing delay and final model accuracy.

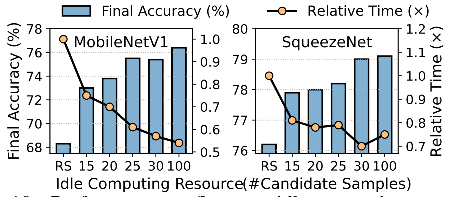


Fig. 10. Performance on fluctuant idle computing resource.

model parameters for 5 iterations and upload the parameter update to server for model aggregation. As shown in Fig. 8, compared with the second best approach, Titan achieves 2.03% increase in final accuracy and 3.17× speedup in time-to-target accuracy, which highlights the potential of applying Titan to the distributed model training processes like FL.

V. Other Related Work

In §II-C, we have provided a thorough introduction of existing data selection approaches and here we review other relevant works and clarify their key differences from ours.

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 [4], [13], exploring the co-execution of multiple types of computing resources to accelerate computation [14]–[16], and offloading computation to cloud server [54]–[56]. 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 [57], [58] before model deployment. Therefore, previous works are orthogonal to our work and Titan is complementary to them.

Two-Stage Architecture. The design of two-stage system has been widely adopted in industrial recommendation system [39], [59] 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 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.

VI. Conclusion

In this work, we identify that the under-utilization of on-device data resource hinders the successful model training process for mobile 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 typical mobile 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.

Appendix: 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 $y \in \mathcal{Y}$:

$$\mathbb{V}_{\mathcal{B}\sim P_t(\mathcal{S})}[\nabla L(w,\mathcal{B})] = \mathbb{V}_{\mathcal{B}\sim P_t(\mathcal{S})}\Bigg[\sum_{y\in\mathcal{Y}}\frac{|\mathcal{S}_y|}{|\mathcal{S}|}\mathbb{E}_{(x,y)\in\mathcal{B}_y}[\nabla l(w,x,y)]\Bigg]$$

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

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

Eq.(c) decomposes the overall batch selection process into sub-processes for each class, and Eq.(d) 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.*$\mathbb{V}[x]=\mathbb{E}[x^2]-\left(\mathbb{E}[x]\right)^2$), we can further decompose the gradient variance:

$$\text{(d)} = \sum_{y\in\mathcal{Y}} \frac{|\mathcal{S}_y|^2}{|\mathcal{S}|^2 \cdot |\mathcal{B}_y|} \cdot \Bigg[\sum_{(x,y)\in\mathcal{S}_y} P_{t,y}(x) \cdot \frac{\big|\big|\nabla l(w,x,y)\big|\big|^2}{\big[P_{t,y}(x)\cdot|\mathcal{S}_y|\big]^2} - \Bigg|\Bigg| \sum_{(x,y)\in\mathcal{S}_y} P_{t,y}(x) \cdot \frac{\nabla l(w,x,y)}{P_{t,y}(x)\cdot|\mathcal{S}_y|} \Bigg|\Bigg|^2 \Bigg] \quad \text{(e)}$$

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

Eq.(e) 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 $\mathbb{E}_{(x,y)\sim P(\mathcal{S})}[f(x)] = \sum_{(x,y)\in\mathcal{S}} P(x) \cdot \frac{f(x)}{P(x)\cdot|\mathcal{S}|} = \mathbb{E}_{(x,y)\sim\mathcal{S}}[f(x)]$.

References

[1] "Google lens - search what you see." https://lens.google/, 2024.

[2] "Siri - apple." https://www.apple.com/siri//, 2019.

[3] "Microsoft swiftkey keyboard." https://www.microsoft.com/en-us/swiftkey, 2024.

[4] I. Gim and J. Ko, "Memory-efficient DNN training on mobile devices," in *MobiSys*, pp. 464–476, 2022.

[5] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in *WWW*, pp. 2125–2136, 2019.

[6] P. SK, S. A. Kesanapalli, and Y. Simmhan, "Characterizing the performance of accelerated jetson edge devices for training deep learning models," *SIGMETRICS*, vol. 6, no. 3, pp. 1–26, 2022.

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, pp. 1273–1282, 2017.

[8] A. Ghorbani and J. Y. Zou, "Data shapley: Equitable valuation of data for machine learning," in *ICML*, pp. 2242–2251, 2019.

[9] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *ICCV*, pp. 843–852, 2017.

[10] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv:1706.02677*, 2017.

[11] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[12] OpenAI, "Chatgpt general faq." https://help.openai.com/en/articles/6783457-chatgpt-general-faq, 2023.

[13] Q. Wang, M. Xu, C. Jin, X. Dong, J. Yuan, X. Jin, G. Huang, Y. Liu, and X. Liu, "Melon: breaking the memory wall for resource-efficient on-device machine learning," in *MobiSys*, pp. 450–463, 2022.

[14] D. Xu, M. Xu, Q. Wang, S. Wang, Y. Ma, K. Huang, G. Huang, X. Jin, and X. Liu, "Mandheling: mixed-precision on-device DNN training with DSP offloading," in *MobiCom*, pp. 214–227, 2022.

[15] F. Jia, D. Zhang, T. Cao, S. Jiang, Y. Liu, J. Ren, and Y. Zhang, "Codl: efficient CPU-GPU co-execution for deep learning inference on mobile devices," in *MobiSys*, pp. 209–221, 2022.

[16] M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu, "Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus," in *MobiCom*, pp. 215–228, 2021.

[17] T. Tan and G. Cao, "Deep learning on mobile devices through neural processing units and edge computing," in *INFOCOM*, pp. 1209–1218, 2022.

[18] K. Huang, B. Yang, and W. Gao, "Elastictrainer: Speeding up on-device training with runtime elastic tensor selection," in *MobiSys*, pp. 56–69, 2023.

[19] J. Wei, T. Cao, S. Cao, S. Jiang, S. Fu, M. Yang, Y. Zhang, and Y. Liu, "Nn-stretch: Automatic neural network branching for parallel inference on heterogeneous multi-processors," in *MobiSys*, pp. 70–83, 2023.

[20] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," in *ICML*, pp. 2530–2539, 2018.

[21] P. Zhao and T. Zhang, "Stochastic optimization with importance sampling for regularized loss minimization," in *ICML*, pp. 1–9, 2015.

[22] C. Coleman, C. Yeh, S. Mussmann, B. Mirzasoleiman, P. Bailis, P. Liang, J. Leskovec, and M. Zaharia, "Selection via proxy: Efficient data selection for deep learning," in *ICLR*, 2020.

[23] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *CVPR*, pp. 5533–5542, 2017.

[24] B. Settles, "Active learning literature survey," 2009.

[25] J. Yoon, D. Madaan, E. Yang, and S. J. Hwang, "Online coreset selection for rehearsal-based continual learning," in *ICLR*, 2022.

[26] D. Everaert and C. Potts, "Gio: Gradient information optimization for training dataset selection," in *ICLR*, 2024.

[27] B. Mirzasoleiman, J. A. Bilmes, and J. Leskovec, "Coresets for data-efficient training of machine learning models," in *ICML*, pp. 6950–6960, 2020.

[28] O. Pooladzandi, D. Davini, and B. Mirzasoleiman, "Adaptive second order coresets for data-efficient machine learning," in *ICML*, pp. 17848–17869, 2022.

[29] Y. Li, Y. Shen, and L. Chen, "Camel: Managing data for efficient stream learning," in *SIGMOD*, pp. 1271–1285, 2022.

[30] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

[31] D. Cai, Q. Wang, Y. Liu, Y. Liu, S. Wang, and M. Xu, "Towards ubiquitous learning: A first measurement of on-device training performance," in *EMDL*, pp. 31–36, 2021.

[32] A. Developer, "Maximum build file sizes.." https://developer.apple.com/help/app-store-connect/reference/maximum-build-file-sizes/, 2023.

[33] Google, "Android developers: Apk expansion files." https://developer.android.com/google/play/expansion-files.

[34] HUAWEI, "Huawei wifi ax3 pro." https://consumer.huawei.com/en/routers/ax3-pro/specs/, 2023.

[35] S. L. Smith, P. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," in *ICLR*, 2018.

[36] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[37] X. Zeng, M. Yan, and M. Zhang, "Mercury: Efficient on-device distributed dnn training via stochastic importance sampling," in *Sensys*, pp. 29–41, 2021.

[38] A. Katharopoulos and F. Fleuret, "Biased importance sampling for deep neural network training," 2017.

[39] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec, "Pixie: A system for recommending 3+ billion items to 200+ million users in real-time," in *WWW*, pp. 1775–1784, 2018.

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[41] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[42] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv:1602.07360*, 2016.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, pp. 770–778, 2016.

[44] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[45] X. Ouyang, Z. Xie, J. Zhou, J. Huang, and G. Xing, "Clusterfl: a similarity-aware federated learning system for human activity recognition," in *MobiSys*, pp. 54–66, 2021.

[46] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta numerica*, vol. 8, pp. 143–195, 1999.

[47] NVIDIA, "Jetson nano developer kit." https://developer.nvidia.com/embedded/jetson-nano-developer-kit, 2023.

[48] C. Li, X. Zeng, M. Zhang, and Z. Cao, "Pyramidfl: a fine-grained client selection framework for efficient federated learning," in *MobiCom*, pp. 158–171, 2022.

[49] R. Yi, T. Cao, A. Zhou, X. Ma, S. Wang, and M. Xu, "Boosting dnn cold inference on edge devices," in *MobiSys*, pp. 516–529, 2023.

[50] A. Das, Y. D. Kwon, J. Chauhan, and C. Mascolo, "Enabling on-device smartphone gpu based training: Lessons learned," in *PerCom Workshops*, pp. 533–538, 2022.

[51] V. Shah, X. Wu, and S. Sanghavi, "Choosing the sample with lowest loss makes SGD robust," in *AISTATS*, pp. 2120–2130, 2020.

[52] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, pp. 1273–1282, 2017.

[53] A. Li, L. Zhang, J. Tan, Y. Qin, J. Wang, and X.-Y. Li, "Sample-level data selection for federated learning," in *INFOCOM*, pp. 1–10, 2021.

[54] D. Yao, L. Xiang, Z. Wang, J. Xu, C. Li, and X. Wang, "Context-aware compilation of dnn training pipelines across edge and cloud," *IMWUT*, vol. 5, no. 4, pp. 1–27, 2021.

[55] S. Wang, S. Yang, and C. Zhao, "Surveiledge: Real-time video query based on collaborative cloud-edge deep learning," in *INFOCOM*, pp. 2519–2528, 2020.

[56] X. Pang, Z. Wang, J. Li, R. Zhou, J. Ren, and Z. Li, "Towards online privacy-preserving computation offloading in mobile edge computing," in *INFOCOM*, pp. 1179–1188, 2022.

[57] M. Xu, F. Qian, Q. Mei, K. Huang, and X. Liu, "Deeptype: On-device deep learning for input personalization service with minimal privacy concern," *IMWUT*, pp. 197:1–197:26, 2018.

[58] B. Liu, Y. Li, Y. Liu, Y. Guo, and X. Chen, "Pmc: A privacy-preserving deep learning model customization framework for edge computing," *IMWUT*, vol. 4, no. 4, pp. 1–25, 2020.

[59] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *RecSys*, pp. 191–198, 2016.