DeepType: On-Device Deep Learning for Input Personalization Service with Minimal Privacy Concern

MENGWEI XU, Peking University, China
FENG QIAN, University of Minnesota - Twin Cities, USA
QIAOZHU MEI, University of Michigan, USA
KANG HUANG, Kika Tech, China
XUANZHE LIU*, Peking University, China

Mobile users spend an extensive amount of time on typing. A more efficient text input instrument brings a significant enhancement of user experience. Deep learning techniques have been recently applied to suggesting the next words of input, but to achieve more accurate predictions, these models should be customized for individual users. Personalization is often at the expense of privacy concerns. Existing solutions require users to upload the historical logs of their input text to the cloud so that a deep learning predictor can be trained. In this work, we propose a novel approach, called DeepType, to personalize text input with better privacy. The basic idea is intuitive: training deep learning predictors *on the device* instead of *on the cloud*, so that the model makes personalized and private data never leaves the device to externals. With DeepType, a global model is first trained on the cloud using massive public corpora, and our personalization is done by incrementally customizing the global model with data on individual devices. We further propose a set of techniques that effectively reduce the computation cost of training deep learning models on mobile devices at the cost of negligible accuracy loss. Experiments using real-world text input from millions of users demonstrate that DeepType significantly improves the input efficiency for individual users, and its incurred computation and energy costs are within the performance and battery restrictions of typical COTS mobile devices.

CCS Concepts: • **Security and privacy → Software and application security**; • **Human-centered computing → Ubiquitous and mobile computing**;

Additional Key Words and Phrases: Deep Learning, Mobile Computing, Personalization

ACM Reference Format:
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. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 4, Article 197 (December 2018), 26 pages. https://doi.org/10.1145/3287075

*Corresponding author.

Authors' addresses: Mengwei Xu, Key Laboratory on High-Confidence Software Technologies (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing, China, 100871, the work is done when Mengwei Xu was an intern in Kika Tech, mwx@pku.edu.cn; Feng Qian, University of Minnesota - Twin Cities, Minneapolis, MN, USA, 55455; Qiaozhu Mei, University of Michigan, Ann Arbor, MI, USA, 48109; Kang Huang, Kika Tech, Beijing, China, 100013; Xuanzhe Liu, Key Laboratory on High-Confidence Software Technologies (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing, China, 100871.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
2474-9567/2018/12-ART197 $15.00
https://doi.org/10.1145/3287075

1 INTRODUCTION

An unprecedented large amount of textual content is being generated by the Internet users. Everyday there are 2 million comments posted to Reddit [10], 500 million Tweets [18], 3.5 billion Google search queries [9], at least 100 billion instant messages [3, 7], and 200 billion emails sent and received [1]. A majority of these texts are typed in, and a significant proportion is typed in from mobile devices. No accurate statistics can be found on how much time a mobile device user spends on text input; but no doubt it must be a surprisingly large number. Any instrument that successfully reduces this tedious effort could bring a significantly improved user experience.

Next-word prediction (sometimes also referred to as auto-completion) is one of such instruments. It is a key feature of Input Method Applications (IMA) for improving the input efficiency. An accurate prediction of the next word a user is going to input reduces the number of clicks as well as the probability of typing errors. This feature is especially appealing on mobile devices where keyboards are small and fingers are big.

Techniques of auto-completion have been evolving quite rapidly, all the way from dictionary lookups to predicting the next words through sophisticated machine learning (ML) models trained using large-scale, real-world input logs [24, 63]. A supervised prediction task as it is, one should not be surprised that the state-of-the-art deep learning models have found their way successfully into this application. Typical examples include RNNs (Recurrent Neural Networks) and LSTMs (Long Short-Term Memory) that are known to be powerful for sequence prediction tasks, and they have been recently adopted by popular IMA providers such as Google GBoard [4, 11, 36, 56].

User expectations also evolve with technology innovation. Due to the diverse and dynamic contexts of text input (e.g., instant messaging, emails, and Web search), users are no longer satisfied with generic suggestions of auto-completion; they instead expect it to be intelligent enough so that the predictions are adaptive to the context and to their personalized preferences. For example, given the same prefix "*every Friday night I usually go to the _*", Alice tends to input "*gym*" next, while Bob is likely to input "*club*." An even smarter algorithm knows that Alice would input "*grocery store*" when she types "*Sunday*" instead of "*Friday*."

But personalization is at the expense of privacy sacrifice. ML models can be personalized when they are trained using data generated by individual users, and most of IMA providers upload the text input from individual devices onto the cloud (with or without users' consent) where possibly personalized models can be trained [14]. The trained models are then downloaded to individual devices where the predictions are done locally. Although it is a common practice for IMA providers to de-identify and encrypt the input texts to certain extents, there are still many privacy concerns, many of which are well articulated in the context of commercial search engines [42, 65]. Compared to search engine queries, keyboard inputs are even more sensitive, as they may contain a variety of private information such as credit card numbers, passwords, secret messages, etc., which is impossible to be completely stripped from the text. This leads to a dilemma between personalization and privacy concerns, which is faced by many other online services as well [27, 62]. The privacy expense of intelligent information services raises increasing public concerns, which has consequentially influenced the global policy making, with the recently emerging concerns of General Data Protection Regulation (GDPR) [1] as one of many examples.

Is there a way out of the paradox, towards personalized text prediction input with better privacy? Given the recent advances in mobile computing and deep learning, in this paper, we make a positive response and propose a novel framework, namely DeepType, which finds the best of both worlds. (1) **Deep Personalization**. The deep learning predictors are trained using the historical text input of individual users. As a result, every individual device uses a different, personalized prediction model, which fully adapts to the preference and context of the user. (2) **Better Privacy**. The model training is performed on only individual mobile devices, and thus no personal data needs to be uploaded to the cloud. Such a principled design significantly alleviates the user privacy concerns during the machine/deep learning phase, which complies to the highest privacy level that was defined by Shen et al. [65] (other privacy levels include pseudo identity, group identity, etc).

[1] GDPR. https://www.eugdpr.org/

The key idea of DeepType is intuitive: to *distribute the training of personalized deep learning models to mobile devices, with no personal data being uploaded to the cloud.* Despite such an intriguing idea, it is known that deep learning models rely on a large number of training examples and a significant amount of computation resources, yet a single mobile device falls short on both of these two aspects.

In this paper, we try to answer two research questions in order to make our approach practical.

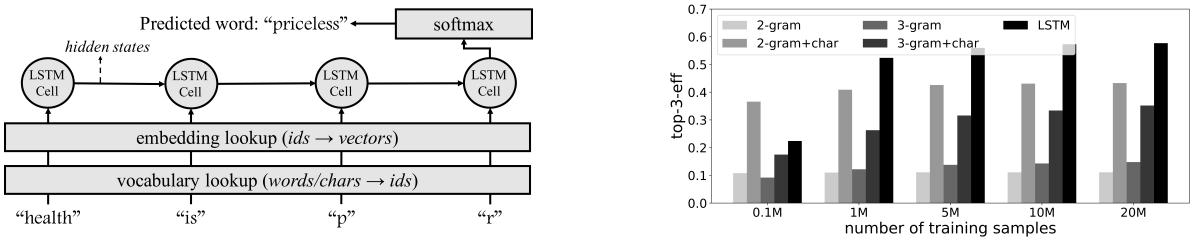
Q1: ***How to effectively train a personalized deep learning model solely based on the text input data available on a mobile device?*** It is commonly believed that training a workable deep learning model requires a large labeled data set, while the size of data generated from a single device is small.

Q2: ***How to efficiently train a personalized deep learning model on a mobile device without compromising user experience?*** Intuitively, training deep learning models requires substantial computation. Although the computation power of today's mobile devices keeps improving, it is still very challenging to run state-of-the-art deep learning models such as multi-layer LSTM on commercial off-the-shelf (COTS) mobile devices without compromising the user experience, prediction accuracy, or devices' battery life.

For **Q1**, we propose to generate a personalized model through *incremental training based on a well-initialized global model.* More specifically, we first pre-train a global model based on publicly available corpora, e.g., news or Tweets. The role of the global model is to learn the general text input behavior of a broad population, and the use of public corpora is free of privacy concerns. Training the global model needs a large amount of data and is costly, but it can be trained offline, only once on the cloud, and then be dispatched to the mobile devices. Based on the global model, we then incrementally train a personalized model on every individual device using its locally accumulated text input of the user. The model structure is also dynamically customized to enhance personalization. Furthermore, the personalized model is continuously updated as the user inputs more text. We evaluate the effectiveness of the incremental training using the largest keyboard input data set to date, which is collected from millions of users of a leading IMA on Google Play (Section 3). The experiments indicate that on-device personalization initialized by the global model trained with public corpora improves the input quite efficiency by around **20%** over a non-personalized model. The result is significantly better than personalized models trained without the global initializer.

For **Q2**, we utilize a set of effective techniques to reduce the computation cost of on-device personalization and integrate them into DeepType. Indeed, some methods are applied from the existing machine learning literature, but we integrate them into a holistic system and demonstrate that they are highly effective for the on-device input personalization, achieving significant performance gain along with little accuracy loss. We implement DeepType and the preceding on-device optimizations upon TensorFlow [16]. Our experiments on commodity mobile devices show that these optimization techniques can reduce **91.6%** of the training time, **90.3%** of device energy consumption, and **81.5%** of the model size of DeepType, at the expense of only **1.8%** accuracy loss. Based on the empirical study of the device usage behaviors of millions of real-world users collected by our industrial partner, we find that mobile devices are in the state of being charged, idle, and high battery level for more than **2.7** hours per day. On-device offline training using the user's historical input text can be done when the device is in such a state, thus causing little impact on the user experience. In addition, the latency of online training, which occurs in realtime as the user is typing, can be as short as 60 ms regardless of the sample length, making it feasible to update the model along with new text inputs.

To further evaluate the usability and effectiveness of DeepType in real-world settings, we conduct an IRB-approved user study involving 34 volunteers at a university in the U.S. During the 18-day user study, the subjects used DeepType to input English texts on their Android smartphones. The user study confirms that DeepType incurs low latency for both the inference and the online training, i.e., its median delays are only 13.3% and 32.6% against the median inter-keystroke time, respectively. We also conduct a survey about the user experience. It is reported that most subjects are satisfied with DeepType in terms of its responsiveness and battery impact.



(a) Overview of proposed LSTM model

(b) Performance of proposed LSTM model

Fig. 1. An overview of our proposed LSTM model and its performance compared to traditional ngram algorithms.

Meanwhile, 78.1% of the subjects confirm that the next-word prediction accuracy exhibits significant or noticeable improvement over time even within the user study window.

To the best of our knowledge, this is the first study that comprehensively explores on-device personalization for next-word prediction based on the large-scale text input data and real-world deployment. We plan to release our trained global model to the research community. Going beyond the application of next-word prediction, we believe our encouraging results motivate more applications that perform deep learning training (as opposed to inference only) on commodity mobile devices. We summarize our contributions of DeepType as follows.

- We present a multi-phase model construction approach to building on-device personalized deep learning models for the next-word prediction. The method generates highly personalized models with significant alleviation of user privacy, compared to state-of-the-art approaches of offloading training tasks to the cloud.
- We employ (to the best of our knowledge) the largest real-world input data collected from more than one million users, and quantitatively demonstrate that our personalization approach considerably improves the input efficiency, compared to performing deep learning without personalization. Meanwhile, DeepType's fully local approach is capable of maintaining high prediction accuracy as the "offload everything" approach.
- We develop DeepType with a set of performance optimizations and integrate them into a holistic IMA system. We systematically evaluate their effectiveness and good usability on COTS smartphones.
- We carry out a user study involving 34 voluntary users who used our system for 18 days. The results confirm DeepType's good usability and low runtime overhead.

2 NEXT-WORD PREDICTION: BACKGROUND AND THE STATUS QUO

The experience of Input Method Application (IMA) is extremely crucial on mobile devices due to the touch-centric interaction over limited screen estate. In IMA, it is well known that *next-word prediction* can facilitate users to input text more efficiently. For example, given the current context of a series of words (such as [*"health"*, *"is"*]) and characters ([*"p"*, *"r"*]), and the model is supposed to find out the probability distribution of the next word the user wants to input, e.g., *"priceless"*. The predicted *top-k* results will be displayed as a list of recommended words to users for selection, and the number of k depends on the keyboard layout and device type.

Deep Learning Models for IMA. Recently, advanced deep learning model has been adopted in next-word prediction [11, 36, 56]. We use the long short-term memory (LSTM) model [73], which is the state-of-the-art algorithm for various NLP tasks, to model the text input behavior of keyboard users. Figure 1a illustrates an overview of the model. The input is the combination of the previous input words and characters, which shall be first converted to integer IDs by looking up a static vocabulary and then to vectors via embedding [8]. Each embedding will be fed into an LSTM cell along with the hidden state from previous cell. The output state of

the last LSTM cell needs to go through a softmax layer and the final output is a vector of float numbers, which specifies the probability distribution of the next word.

The overall model architecture and the configurations are consistent with the LSTM model for PTB dataset [13] in TensorFlow official repository [13]. Some parameters are tailored for our specific prediction task: (1) **Vocabulary size** indicates how many words can be recognized by the model. All words out of the vocabulary will be regarded as a unique *"<unk>"* word in both the training and the testing dataset. In this paper, we use a default size 20K of vocabulary to include the most frequently used words. (2) **Step size** is the number of unrolled LSTM cells, representing how many time steps will be considered into the processing. It is dynamically tuned to the maximum length of the training sample within a mini-batch. (3) **Stacked LSTMs** is often used to enhance the capability of LSTM cells to process and memorize the temporal data. We use a 2-layer LSTMs in this paper. (4) **Batch size** is the number of training examples in one training pass. We use 128 as the default batch size.

To train the model, we adopt the mini-batch gradient descent algorithm [48], the dominant method in practice. During the training process, the input characters of one word are bundled into one sample. Thus, the number of training samples is equal to the input words. In other words, 10K data means 10K input words from users, and the number of input sentences is obviously lower than 10K. For example, the word *"is"* in an input sentence *"health is priceless"* is represented by a training sample with step size of 3 (including 1 inputted words and 2 characters). The input vector is [*"health"*, *"i"*, *"s"*] and the output vector is [*"is"*, *"is"*, *"is"*]. This training sample tries to teach our model that under the pre-sequence of [*"health"*], [*"health"*, *"i"*] and [*"health"*, *"i"*, *"s"*], the output is all expected to be the word *"is"*. Similarly, the word *"priceless"* is converted into a training sample with step size of 11 (2 inputted words and 9 characters). The input is [*"health"*, *"is"*, *"p"*, *"r"*, *"i"*, *"c"*, *"e"*, *"l"*, *"e"*, *"s"*, *"s"*], and the output is [*"is"*, *"priceless"* $\times$ 10]. These samples are batched together to feed into our model for the training phase. Since the samples have different lengths, shorter samples in a batch need to be padded to the longest one. During the training, we use mask matrix [5] to filter the impacts from those paddings.

3 HOW MUCH IMPROVEMENT CAN DEEP LEARNING BRING TO NEXT-WORD PREDICTION?

We begin with applying vanilla LSTM models without domain-specific optimizations or per-user customization. We present how the model introduced in Section 2 works on real-world users' input logs and identify challenging issues of building personalized models for next-word prediction.

3.1 The Dataset

All user data used in this paper, including training the LSTM model and user behavior characterization, is collected from real-world users of Kika keyboard[2] Kika is a leading IMA in Google Play, with over billions of accumulative downloads and over 60 millions of daily active users (DAU) throughout the world. The dataset used in this work spans from June to December in 2017, covering the input history data from 1,156,550 active users and over 10 billion messages that are typed in English. The dataset was collected by Kika in purpose of improving user experience, with explicit agreement from users on user-term statements and strict policy in data collection, transmission, and storage. When conducting this study, we take very careful steps to protect the user privacy and preserve the ethics of research. First, our work is approved by the Research Ethical Committee of the institutes (a.k.a, IRB) that the authors are currently affiliated with. Second, the users' identifies are all anonymized. Third, the data are stored and processed on a private, HIPPA-compliant cloud server, with strict access authorized by Kika. The whole process is compliant with the public privacy policy of Kika company.

[2]https://play.google.com/store/apps/details?id=com.qisiemoji.inputmethod, retrieved on October 22, 2018. We plan to release our models and part of the used dataset when this work gets published.

3.2 Applying Off-the-shelf LSTM Model without Personalization

We then apply the preceding LSTM model to the dataset. To evaluate the model accuracy (performance), we use a metric called **top-k input efficiency** (*top-k-eff* for short). For each word x, we first define $\mathcal{R}$(x) as the minimal number of inputted characters after which the model correctly predicts the word x within the top-k list. For example, given the input of two words as [*"health", "is"*], and an expected output word *"priceless"*. When users type *"p"*, the output top-3 words could be [*"priority", "paramount", "precious"*], so it fails to identify the intended word. After a user types another character *"r"* and the output top-3 words become [*"priority", "precious", "priceless"*], the user gets the recommended word that she plans to input. In this way, the user can directly click the word *"priceless"*. In this case, $\mathcal{R}$(*"priceless"*) is equal to 2. Based on the above definition, top-k-eff is calculated as

$$\text{top-k-eff}(\mathcal{D}) = \frac{\Sigma(len(x) - \mathcal{R}(x))}{\Sigma len(x)}, x \in \mathcal{D}$$

where *len* is the number of characters of words, $\mathcal{D}$ is the evaluation dataset, each item of which comprises the inputted words, characters, and the expected output word (label). Intuitively, top-k-eff can be interpreted as *the proportion of typing efforts can be saved with the help from a given model.* A higher top-k-eff indicates a more efficient input procedure. Obviously, the top-k-eff of a dumb keyboard without any prediction model should be 0.

We present some preliminary analysis results of the proposed LSTM model. We train the model via various sizes of datasets and evaluate them on a separated dataset with 100K randomly chosen samples. The training datasets are randomly selected from all users of Kika, spanning from Jun. 2017 to Nov. 2017, while the testing dataset is randomly selected in Dec. 2017.

To illustrate the performance of the LSTM, we use the classical N-gram based approach as comparison. N-gram language models [23] have been widely explored to make the next-word prediction. N-gram models are designed to work at the word-level. In our scenario, the input characters are optionally used to filter the predicted words from N-gram model of which the prefixes do not match those characters. In this way, model accuracy can be further improved. We use the default configuration for our LSTM model as mentioned previously. The results are illustrated in Figure 1b, where different bars represent different models. For example, "2-gram+char" means that we use a N-gram model (N=2) and filtering strategy based on input characters mentioned above. As observed, the LSTM model significantly outperforms the N-gram based approaches when there is enough training data. Such a result is not surprising since LSTM is known to be able to capture long-term dependencies of text information.

We then explore how the size of training data can impact the performance of the LSTM model. As shown in Figure 2, the model converges after around 5 epochs of training from the perspective of both perplexity and top-3-eff. The most intuitive observation is that a larger training set results in a more accurate model. However, it is observed that such an improvement cannot always be significant. For example, the model trained with 10M data exhibits 9.2% higher top-3-eff compared to the model trained with 1M data (0.524 to 0.573). Such an improvement becomes much smaller (only 0.8%) when we further increase the training data from 10M to 20M. The results indicate that even when deep learning is used, having more training data only provides marginal accuracy gains. A key reason for this is a lack of personalized models as to be discussed next.

3.3 Towards Personalized Input Prediction on Deep Learning

In practice, the overall workflow of input prediction, which is adopted by many popular IMAs including Kika, is to train the language model based on the corpora collected from users on cloud, and then to dispatch the model to users. Since the model is trained via the corpora from different users, it can capture only the general input patterns without customization per user. The poor support of personalization can have non-trivial impacts on the model performance, since users can have quite diverse inputs even given the same previous input sequences (we demonstrate it in Section 6.1). Just as shown in preceding results, the LSTM model is useful for input prediction, but the improvement does not always scale with the increasing size of the training data.

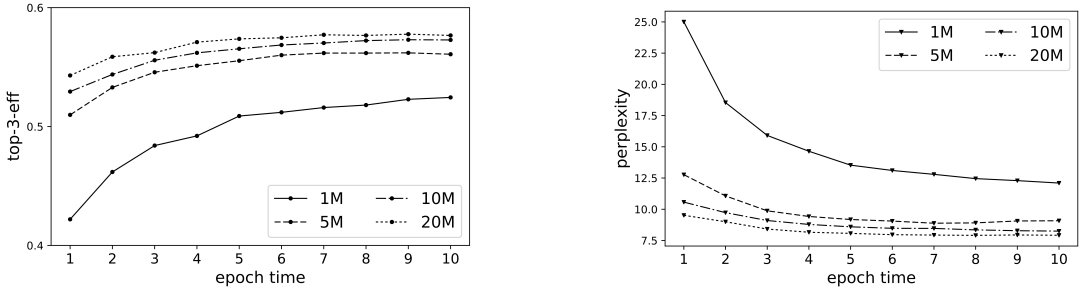


Fig. 2. Performance of proposed LSTM model in two metrics: top-k-eff and perplexity.

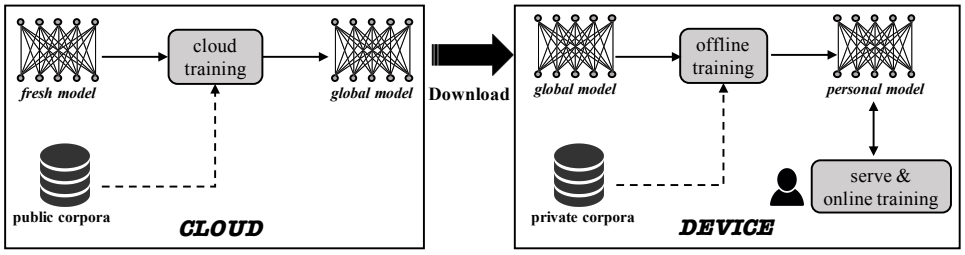


Fig. 3. An illustrated workflow of DeepType approach.

To customize the model, a straightforward approach is to train a personal model for every single user on the cloud and dispatch the model to its owner. Though appealing, such an "on-cloud" approach has two inherent limitations.

• **Privacy Concerns**: On-cloud personalization raises the privacy concerns since it requires users to upload their input text that often contains very sensitive information such as credit card numbers, bank accounts, and passwords. Although most IMA vendors can encrypt or anonymize the transferred data, the privacy concern is still inevitable.

• **High Computational Cost**: On-cloud personalization is not scalable when training the per-user model for millions of users. We make a simple estimation to train a personalized model for users by the preceding LSTM model on a Tesla K80 GPU [12]. The results indicate that the LSTM averagely needs more than 3 minutes to derive the personalized mode per user. In other words, even we have 50 Tesla K80 machines, we still need more than one month to personalize one million users.

4 THE DEEPTYPE APPROACH

To address the preceding issues, we design DeepType, a novel IMA framework for the next-word prediction problem that can help achieve good personalization, privacy preservation, and feasibility through multi-phase deep learning model construction. In DeepType, the deep learning models are largely trained *locally on mobile devices*. Doing so helps achieve (1) personalization, since the LSTM model for each user is trained from her own input data, (2) privacy-preservation, as the input text is always kept on local device without uploading to the external world, and (3) feasibility, because the heavy work load of training is carried out in a distributed manner instead of cloud-centric.

Despite being conceptually straightforward, there remain two practical challenges. *First, the training data of each user may be insufficient to train a high-quality deep learning model*; *Second, the hardware of mobile devices is often not powerful enough to train deep learning models*. Note most existing work [21, 47, 57] of mobile deep learning only considers performing the inference stage (as opposed to the training stage) on COTS mobile devices.

To address the first challenge, in addition to performing personalized training on users' devices, we also strategically leverage the cloud to "bootstrap" the training without incurring privacy concerns. Specifically, we propose a *multi-phase* model construction framework called DeepType as illustrated in Figure 3. The overall workflow of DeepType includes three major phases.

(1) *On-cloud pre-training*. We first use the publicly available text corpora, such as Twitter and Wikipedia, to train a general LSTM model as described in Section 2. This phase is typically performed on the cloud. The trained model is called a *global model*. This model does not rely on any user's input behaviors but built entirely upon the public knowledge base. The global model is then dispatched to all IMA users' devices.
(2) *On-device offline training*. On each mobile device, a new model called a *personal model* will be trained by synthesizing the user's input behaviors with the global model. The personalization is mainly achieved by incrementally training the global model based on the input data accumulated on the device. After the personalization, the model will be deployed to serve users on future next-word prediction.
(3) *On-device online training*. The deployed personal model will be continuously reinforced during the user input procedure. The training data at this phase is generated and used on-the-fly.

The difference between on-device offline and online training is that offline training is performed based on the dataset accumulated on device (the accumulation may span months or years), while the online training leverages the data generated at runtime. With the on-device training, the model is deeply customized in terms of both model parameters and model structure to fit every single user. Specifically, we utilize two techniques in the customization process: incremental training and dynamic vocabulary.

• **Incremental training** changes only the model parameters. It differs from the normal training from the aspect of how weight parameters are initialized before the training begins. A normal training begins with random (or zero) parameter numbers, while our incremental training will first initialize the parameters derived from the pre-trained global model, which has already learned the general input behavior to some extent. The configuration of the incremental training is almost the same with that of the global model, except smaller batch size, i.e., only 16 samples each batch instead of 128.

• **Dynamic vocabulary** changes only the model structure. It is used to customize the vocabulary per user. As mentioned previously, the vocabulary contains a fixed number of ⟨word, id⟩ pairs. If the word that a user wants to input is out of the vocabulary, this word cannot be correctly predicted. We observe that users usually have their unique input words, e.g., friends' names, that are not included in the vocabulary of the global model. For every single user, we first count the occurrence frequencies of each word from the local input data, and identify the unique words that have been used more than a threshold but not included in the vocabulary of the global model. We use this set of unique words to replace the words in the vocabulary of the global model that has never been used by the user. Such replacement is directly applied on the vocabulary entries.

Multi-phase model construction deals with the aforementioned first challenge where the training data of each user may be insufficient to train a high-quality deep learning model. Since the parameters of the global model are already initialized by learning from the public corpora, the on-device incremental training can immediately take effect on personalization. In Section 6.1, we will demonstrate that DeepType can substantially improve the model performance. Also note that the preceding training processes (including both on-cloud and on-device) follow the standard machine learning paradigm, e.g., mini-batch gradient descent, without any additional efforts on customizing the learning algorithm such as [2, 45]. Hence, it makes our approach easily deployable and generalizable to other applications that need personalization such as advertisement recommendation and application pre-launch.

Recall that the second challenge of DeepType is the relatively weak computation power of COTS mobile devices, especially for the "heavy-fragmented" Android platform [51]. Since the training is performed locally, we need to minimize its impact on users' normal usage of their devices by judiciously determining *when and how* to perform training. To this end, we propose a series of optimizations for both on-device offline and online training. These optimizations significantly improve the efficiency of deep learning training on COTS devices with little sacrifice of the accurcy. They will be detailed in Section 5.

After optimization, we notice that the online training can be actually accomplished within a short time interval when user is typing (i.e., <60ms according to our experiments in Section 6.2.3). The reason is that during the typing procedure the input text samples are generated at a relatively low speed. However, the on-device offline training requires intensive processing of a relatively large dataset accumulated for months or even years. Therefore, we need to carefully choose the timing of performing on-device offline training so that it does not compromise the user experience. Specifically, DeepType provides configurable policies that trigger the offline training process. For example, the default preconditions for on-device offline training are: (a) device is idle, (b) device is being charged, and (c) device is in the high battery level. By analyzing user behavior from real-world users in Section 6.3, we demonstrate that these preconditions can oftentimes be satisfied. A device is able to perform training for 2.7 hours per day on average.

5 PERFORMANCE OPTIMIZATIONS FOR DEEPTYPE

The training phase of a deep learning model, even for a small input dataset, can be quite computation-intensive for today's COTS mobile devices. In this section, we describe a series of techniques to reduce such computation overhead, making DeepType feasible on COTS mobile devices.

5.1 Layer Compression

Layer compression is a popular strategy to reduce the model size and inference computations. DeepType attempts to employ the proper compression algorithm to reduce the training time. Various algorithms have been proposed to compress heavy layers, such as node pruning [39, 52, 70], sparsity regularization [30, 61], and layer-redundancy compression [22, 46, 69]. Based on a set of experiments, we choose the SVD (Singular Value Decomposition)-based layer compression [31, 46] that performs well on mobile devices, as it results in a substantial reduction of training time but with minimal loss of model accuracy.

For a large weight matrix $W \in \mathbb{R}^{m \times n}$ to be compressed, its SVD is defined as $W = USV^\intercal$, where $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$, $S \in \mathbb{R}^{m \times n}$. S is a diagonal matrix with the singular values on the diagonal, and U, V are orthogonal matrices. The singular values of S often decay rapidly, so W can be well approximated by keeping only the $\mathtt{t}$ largest entries of S, resulting in the approximation $W' = U'S'V'^\intercal$, where $U' \in \mathbb{R}^{m \times t}, V' \in \mathbb{R}^{t \times k}, S' \in \mathbb{R}^{t \times t}$. S' and V' can be further combined into one matrix $Y \in \mathbb{R}^{t \times m}$. The impacts of the compression can be illustrated as inserting one intermediate layer into a fully-connected layer. Assume that *i-th* layer has **M** neurons, and the next layer has **N** neurons. By inserting a smaller layer with **t** neurons between them, the number of weight parameters and calculations can be compressed to $(t \cdot M + t \cdot N) \div (M \cdot N)$.

We apply this technique to the last softmax layer of the proposed LSTM model (Section 2). Our experiments show that only 10% of the singular values need to be kept to approximate the original weight matrix (e.g., $m = n = 400, t = 40$), so the compression ratio can reach 89.8%.

Note that the layer compression is performed on the *"pre-trained"* global model. After the compression, we re-train the model to tune the parameters before dispatching it to the user devices. Given the fact that the softmax layers contribute the most to the overall model parameters and calculations, our layer compression can effectively decrease the model size and the training time.

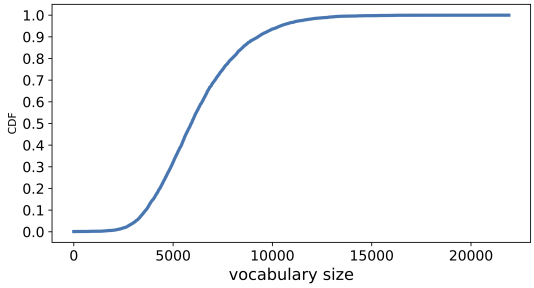


Fig. 4. Vocabulary size used by users within 6 months (Jul. 2017 to Dec. 2017). Mean: 6,214, median: 5,911.

5.2 Vocabulary Compression

The computation cost of our model can be further reduced by compressing the vocabulary. The rationale behind the vocabulary-level compression is that our global model uses a large vocabulary (20K) to cover most word occurrence. But, for a specific individual user, the size of vocabulary is often smaller. It implies that we can replace the vocabulary with a smaller one. Therefore, we first randomly select 100,000 users of Kika, and count the number of words used and their frequencies for every single user during the 6-month (Jul. 2017 to Dec. 2017). We finally identify 54.8M unique words used by the users in total, and the top 20K used words count for 95.5% of the overall word occurrences. Furthermore, we find that the size of words that are used at least once by a specific user can can be much smaller. As shown in Figure 4, the median word number is 5,957 across these users, with more than 94.5% users have less than 10K words used.

Given the above observation, we can eliminate some unnecessary parameters from the softmax layer, which is directly associated with the word vocabulary. The softmax layer can be described as follows,

$$\mathbf{P}_{b\times n} = softmax(\mathbf{S}_{b\times m} * \mathbf{W}_{m\times n} + \mathbf{B}_{b\times n})$$

where $\mathbf{P}$ is the next-word probability output, $\mathbf{S}$ is the states from the LSTM block, $\mathbf{W}$ is the weight matrix, $\mathbf{B}$ is the bias matrix, b is the batch size, m is the hidden state size, and n is the vocabulary size. Our compression technique is achieved by eliminating unnecessary columns from the weight matrix $\mathbf{W}$. More specifically, $\mathbb{S}$ denotes the set of word IDs that are found in the vocabulary of global model but not found in the current user's historical data. The last layer of our model can be compressed as:

$$\mathbf{P}_{b\times n'} = softmax(\mathbf{S}_{b\times m} * \mathbf{W}^*_{m\times n'} + \mathbf{B}^*_{b\times n'}), \mathbf{W}^*_{m\times n'} = \left\{w_k\right\}, \mathbf{B}^*_{m\times n'} = \left\{b_k\right\}, k \in \{1...n\}, k \notin \mathbb{S}$$

where w_i is the i-th column of the original weight matrix $\mathbf{W}$, b_i is the i-th column of the original bias matrix $\mathbf{B}$.

5.3 Planning Fine-grained Bucketing

For training RNN models, the bucketing [43] technique is often used to handle variable lengths of the input. A naive solution without bucketing sets a fixed length of every single batch such as 30, and then pads the shorter input with zero or cuts the longer ones. Such a solution, however, requires that short input samples are processed as a longer (fixed) length. As a result, the redundant computations lead to more training time. In contrast, bucketing separates input samples that have similar lengths into the same bucket, and for each bucket pads the samples to the length of the longest sample in that bucket with zero. Theoretically, more fine-grained bucket sizes can make more efficient training. However, the size of the bucket may not be a multiple of the batch size (128 for default) since the data samples are more sparsely distributed. In these cases, updates will not be based on a full batch. Thus, the pre-training phase uses only 4 bucket sizes: 5, 10, 20, and 30.

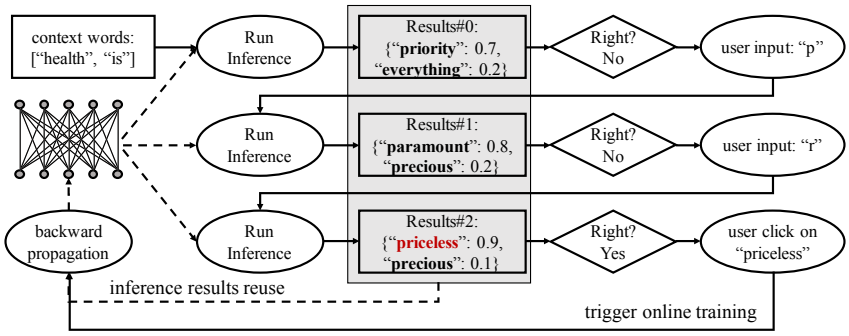


Fig. 5. The workflow of *reusing inference results* to optimize on-device online training.

For the on-device training, however, since we use only small batch size (16 for default), we aggressively use more fine-grained bucket when separating input samples into buckets of exactly the same length. In such a case, no input samples will be padded so no additional processing cost will occur.

5.4 Fine-tune Training

We borrow the idea that is widely adopted in the transfer learning [72] by training only the last layer (softmax) of the LSTM model to reduce the overall training workloads. Such a process is often known as the *fine-tune training*. The basic motivation behind the fine-tune is treating the layers before the last layer in a pre-trained model as fixed feature extractors, so that we can use the extracted features to carry out prediction tasks by tuning only a small portion of layers. The idea fits well into our next-word prediction scenario for two reasons.

First, we already have a well-initialized global model, and the training datasets for the global model and personal model are both representative of users' input patterns to some extent.

Second, the features extracted from a sequence of LSTM blocks are mostly generalized. These features may include grammars-related contexts (e.g., singular or plural subject), emotion-related contexts, content-related contexts, etc. In other words, the learned weights of word embeddings and LSTM blocks in the global model are representative enough for input patterns, so that we can focus on the fine-tuning in terms of only the weights of last softmax layer.

5.5 Reusing Inference Results

To save the online on-device training time, we can reuse the computation results from the inference process, i.e., the probability results of next-word prediction. As we know, the backward propagation (BP) algorithm, which is the standard technique to train deep learning models, involves two main steps: performing the inference to get the error (forward pass), and updating the weight parameters to minimize the error (backward pass). The first step of the BP procedure is equal to the forward inference and can be used to predict next-word in our case. Figure 5 shows a complete procedure of how our model facilitates user to input word *"priceless"* and reinforce itself afterwards. After users input the characters *"p"* and *"r"*, our model correctly recommends the word *"priceless"*, so users directly click on the word as the next input word. With this feedback, we obtain a data sample for on-device online training. The training of this data does not go through the forward propagation. Instead, we reuse the results cached during the previous inference, and directly compute the loss function based on the prediction probability distribution. In practice, the data will not be immediately used for training. It will be buffered and then used when the size of accumulated input samples reaches a pre-defined batch size (16).

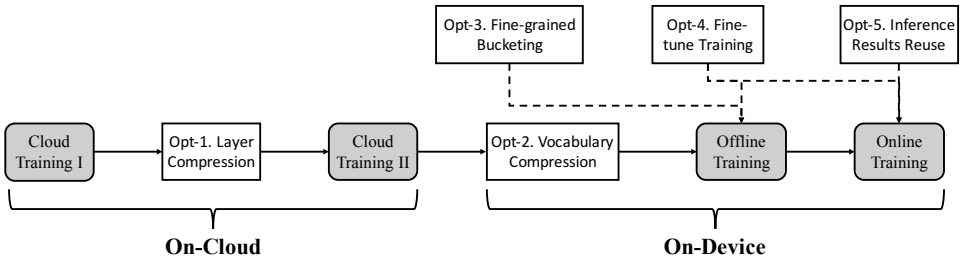


Fig. 6. The overall workflow of DeepType after embedding our optimization techniques. Here, "opt" is short for "optimization".

To preserve the reuse technique to be correct, the training phase should be finished before the next inference is activated, otherwise the inference results will be invalid for the next-round training. This is the reason why the reuse technique cannot be applied on the on-device offline training, since the cached results can expire when the model parameters are updated.

5.6 Putting Everything Together

Figure 6 illustrates the workflow of our approach after integrating the preceding techniques. On the cloud, the global model is re-trained to tune the parameters after the layer compression. After dispatched to mobile devices, the global model will be further cut down by taking into account the vocabulary compression. This step cannot be performed on the cloud since it needs to inspect the word usage of each single user. The dynamic vocabulary technique mentioned in Section 4 is also performed at the same time. Next, the model will be personalized, and the fine-grained bucketing and fine-tune training techniques are used during the on-device offline training. Finally, the personal model is continuously reinforced during the on-device online training. Fine-tune training and inference result reuse techniques are also applied to optimize the on-device online training.

6 EVALUATION

In this section, we evaluate DeepType from three folds: (1) the accuracy improvement gained by personalization, (2) the cost of on-device personalization, and (3) the feasibility of deploying DeepType on mobile devices.

6.1 Accuracy of Personalized Model

We first evaluate the accuracy improved by the personal model derived by DeepType. For simplicity, we conduct the experiments on a GPU-powered server (Tesla K80) with TensorFlow [16] (version 1.2) installed.

6.1.1 Experiment Setups. **Selected Compared Alternatives.** We comprehensively compare the model performance of DeepType to other alternatives, based on three guidelines: (1) training the global models with different public corpora crawled from the Internet (i.e., BBC News and Twitter) and the private corpora collected from Kika users, (2) training the personal model with user's input data or not, (3) training the personal model from scratch without a global model. Finally, we have 6 alternatives for comparison, as listed in Table 1.

Training and testing datasets. First, to train the global model, we try three different datasets, two public corporaes from Twitter corpora and BBC News crawled from the Internet, and the private corpora collected from Kika users as described in Section 2. For fairness, each dataset contains around 10M samples, since we have already demonstrated that larger dataset does not improve the accuracy in Section 3.

Second, to train the personal model (incremental training), we use the private corpora collected from Kika users (Jun. 2017 to Nov. 2017). For each user, the training dataset size contains 100K entries of text inputs. Later in Section 8, we will demonstrate that such a volume of data accumulated on every single device is reasonable.

Table 1. The summarized performance of proposed model trained in different ways (medium value reported). We train the global model and personal models for 5 epoches.

corpora to train the global model	personalization?	top-3-eff
public corpora crawled from Twitter	✓(DeepType)	0.616
	✗	0.513
public corpora crawled from BBC News	✓	0.508
	✗	0.325
private corpora collected from Kika users	✓	0.624
	✗	0.568
no global model	✓	0.331

Last, the testing dataset is also collected from Kika users but at a different period of time (Dec. 2017). For every single user, we have 20K data for testing. Indeed, we cannot perform the experiments for all users due to the time and cost limit. Instead, we randomly select 100 users for testing. In fact, it takes more than one month to run all the experiments mentioned in this section on the GPU server. Although the user number is limited in this experiment, the overall testing dataset contains 2M samples in total and the results are representative.

6.1.2 Experiment Results. We first summarize the accuracy gained from the personal model of DeepType and other alternatives in Table 1. According to our experiments, both the global model and the personal model take around 5 epochs to converge. From the results we can have several key observations as follows.

• The most straightforward and significant finding is that the personal model results in substantial improvements on the accuracy, indicating that our approach can more accurately predict the word that a user tends to type. Considering the case that DeepType uses the Twitter corpora to train the global model and personalizes it based on users' own input data. Then the personal model improves the top-3-eff from 0.513 to 0.616 (↑20.1%). Similarly, even if the global model is trained via the private corpora from Kika, the personal model still improves the top-3-eff from 0.568 to 0.624 (↑9.9%). Such a result demonstrates the significant improvement brought by the personal model, considering the fact that training the global model with much more data (20M) can boost the top-3-eff to only 0.571 (↑0.5%).

• Another key finding is that our approach performs better than the status quo approach that is adopted by IMAs such as Kika, i.e., training a global model over private corpora from all users (0.616 vs. 0.568). In other words, our approach performs much better than the status quo in terms of both accuracy and privacy.

• The results of our approach is comparable to the most "aggressive" one, i.e., training the global model via the private corpora from all users, and personalizing the per user. Indeed, the latter performs slightly better than DeepType (0.624 vs. 0.616), but our approach performs the training on device only without uploading the data to the cloud, hereby the user privacy is better preserved.

• Additionally, the source of public corpora needs to be properly selected for a better model performance. For example, the global model trained via the BBC News corpora has quite poor top-3-eff (0.325), and the personalization result is not satisfactory (0.508), either. In contrast, the results based on Twitter are much better. Such a result is reasonable as the Twitter corpora is more representative to users' personal input behavior.

• Finally, personalizing the model from scratch (i.e., no global model) results in the worst performance (0.331). The reason is that the data volume of each user is insufficient to train a deep learning model. This result testifies the necessity of training a well-initialized global model.

We also measure how the size of training data and the consideration of dynamic vocabulary can affect the personalization result. The results are reported in Figure 7. The left and right plots use a global model (G-M) trained from the public Twitter corpora and the private corpora, respectively, base the comparison baseline. Based on that, we enhance the model using the personalization (P-M) by varying training data size (50K and 100K) as

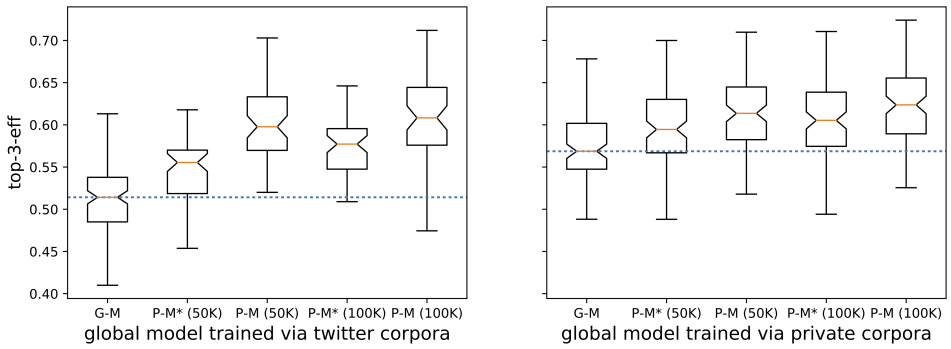


Fig. 7. The performance of models trained via different corpora for global model (twitter corpora, private corpora), different dataset size for personalization (50K or 100K), and whether dynamic vocabulary technique is used. "G-M" stands for global model, and "P-M" stands for personal models. The models marked with (*) are trained w/o dynamic vocabulary.

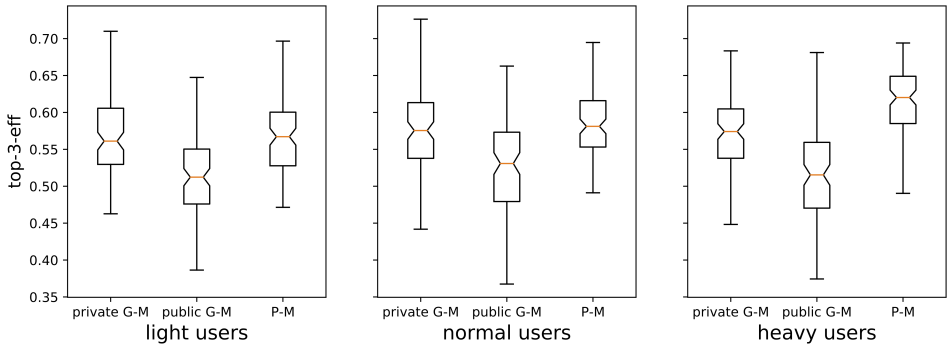


Fig. 8. The performance of models for different types of users. "private G-M" stands for global model trained via private corpora, "public G-M" stands for global model trained via twitter corpora, and "P-M" stands for personal models (DeepType).

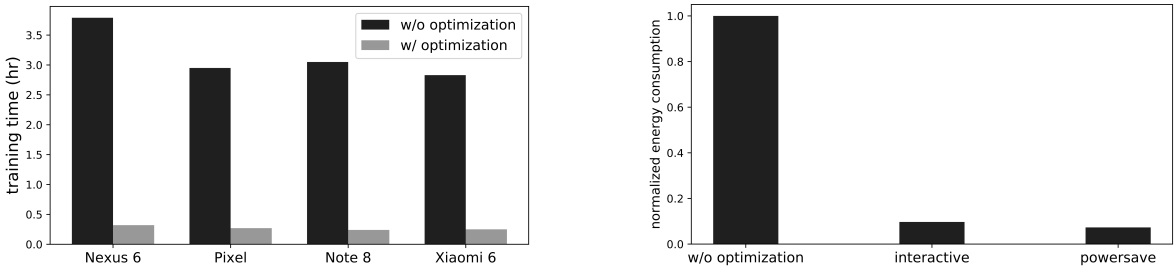
well as dynamic vocabulary ("*" indicates that dynamic vocabulary is not used). The Y axis presents the top-3-eff results. As shown, increasing the training data size can be quite helpful for personalization improvement. In the left figure, using 50K user data for personalization boosts the top-3-eff from 0.513 (G-M) to 0.595 (P-M 50K), while using 100K data further makes the result grow up to 0.616 (P-M 100K). Similar observations can be also found in the right figure.

Another finding is the effectiveness introduced by the dynamic vocabulary. When using 100K data to personalize a global model over the the Twitter corpora, dynamic vocabulary makes a much higher accuracy (0.616 for P-M 100K) compared to the result without it involved (0.577 for P-M* 100K).

Furthermore, we evaluate the improvement of personal model on different kinds of users. We divide users into three groups based on the size of their input words during June to December in 2017: heavy users (top 25%), light users (least 25%), and normal users (the rest). We then randomly select 100 users from each of the three groups, and evaluate the performance of two global models (trained via the Kika private corpora and the Twitter corpora) and personal model (DeepType). The personal model is incrementally trained by the users' input data during 5 months. The results are illustrated in Figure 8. As observed, the global models perform similarly

Table 2. A summary of achieved improvements by applying proposed techniques to train 100K samples on Nexus 6.

	Personal Model (Baseline)	Fine-tune Training	Layer Compression	Vocabulary Compression	Fine-grained Bucketing	**Combine Together**
top-3-eff	0.616	0.617	0.612	0.614	0.616	**0.605**
training time (hr)	3.79	2.17	2.97	2.63	1.35	**0.32**
model size (MB)	74.5	/	17.0	42.4	/	**13.8**



(a) Time to train 100K data on different devices.

(b) Energy consumption w/ and w/o optimization.

Fig. 9. Evaluation of proposed optimization techniques for on-device offline training.

on different kinds of users, but the performances of personal models vary a lot. Such a finding evidences that the size of training dataset can affect how much benefits can be gained by the personal model. In addition, the improvement of the personal model is significant. For light users, personalization boosts the top-3-eff from 0.515 to 0.572 (↑11.1%), and for heavy users, the top-3-eff is improved from 0.512 to 0.627 (↑22.5%).

Accuracy Improvement of Online Training. The preceding evaluations are done with offline training, i.e., train the model using all (personalized) historical input data and then test the model against future input. In contrast, online training incrementally updates the model on the fly as the user types, and makes the model update immediately "visible" and "applied" to users. We next emulate the online training process to show its benefits. We consider personalizing a global model trained via the Twitter corpora for 100 users that are randomly selected. The personalized dataset is 100K for a user. For every single user, we sequentially feed the samples in the dataset into our emulator to mimic the words typed by the user. The emulator first runs an inference on each sample. Then per the online training paradigm, the emulator immediately trains the model if the accumulated samples reach a batch of 16 words. We then compare the accuracy obtained with and without online training. Compared to the case without online training (top-3-eff around 0.51), the online training improves the overall accuracy by around 9.5%. Such an improvement is much higher on later stages when the model has already been well personalized by pervious efforts from the online training. For example, the accuracy improvement brought by the last 10K of data is considerably higher, i.e., 16.9% compared to the overall improvement of 9.5%.

6.2 Cost of On-Device Training

In this subsection, we use various commodity mobile devices to evaluate the cost of on-device training, which is crucial to impact the feasibility of our approach. The results shall demonstrate that our proposed techniques mentioned in Section 5 can be practically deployed in terms of training time and energy consumption.

6.2.1 Implementations of On-Device Training. We have implemented the DeepType prototype along with the techniques mentioned in Section 5 based on the popular TensorFlow [17]. To support the fine-grained bucketing,

we keep multiple unrolled LSTM graphs with different step sizes in memory, and let them share the parameters of LSTM cells. For the fine-tune training, we leverage the internal feature of TensorFlow [15] that can fix some parameters and only allow the others to be updated during training. We implement our current prototype only at the Android platform.

We use the Nexus 6 smartphone model (Android 6.0, Quad-core 2.7 GHz Krait 450) to conduct our experiments. The energy consumption is obtained via the Qualcomm Snapdragon Profiler [6].

6.2.2 Cost of On-Device Offline Training. Table 2 summarizes the improvements attained from our proposed techniques in the on-device offline training. We take into account three metrics: the accuracy (top-k-eff defined in Section 3), the training time (one epoch), and the model size. Here, we take the DeepType without using any on-device optimization as the baseline, i.e., incrementally training the proposed LSTM model without any compression, bucketing, or cache reuse.

Our primary observation is that the proposed optimization techniques in DeepType can dramatically improve the training time and model size with few accuracy loss. The training time is reduced by 91.6% and the model size is reduced by 81.5%, while the accuracy loss is only 1.8% compared to the baseline. We then explore accuracy, training time, model size, and energy consumption, respectively.

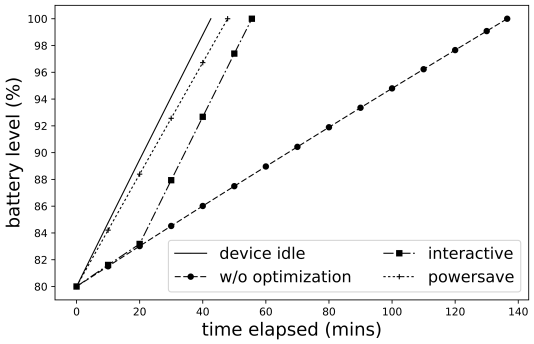
Accuracy. We observe that DeepType has very marginal impacts on the overall accuracy of the LSTM model. For example, the layer compression and the vocabulary compression reduce the top-3-eff from 0.616 to 0.612 and 0.614, respectively. Fine-grained bucketing changes only how data is batched, thus inherently has no effects on the model performance. Surprisingly, the fine-tune training, i.e., updating only the parameters of the last layer during our incremental training, can reach even higher accuracy than the baseline. With all those techniques applied, the model performance drops from 0.616 to 0.605 (↓1.8%). This accuracy loss is quite marginal compared to the gained improvement and the alleviated privacy concerns.

Training time. As observed from Table 2, the proposed techniques have substantial improvements of on-device training time. For example, the fine-tune training reduces the training time of 100K data on Nexus 6 from 3.79 hours to 2.17 hours, while the fine-trained bucketing reduces it to 1.35 hours. Combining all these techniques together results in a 91.6% reduction of training time (i.e., from 3.79 hours to 0.32 hours). As we will show in Section 6.3, the empirical device usage results show that a device has 2.7 hours available for on-device training on average per day, the training of 100K samples can be done within one day. This result indicates that our approach is feasible on commodity mobile devices.

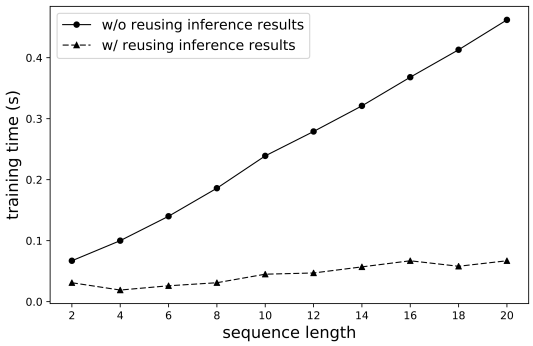
To make the evaluation more comprehensive, we also evaluate the speedup of on-device training on other popular commodity device models: Google Pixel, Galaxy Note 8, and Xiaomi 6. As observed from Figure 9a, our DeepType can effectively reduce the training time on all these devices. For example, about 92.1% of the training time can be saved on Note 8.

Model size. As shown in Table 2, the layer compression and vocabulary compression can reduce 77.2% and 43.1% of the model size, respectively, while other two techniques have no impacts since they change only the training process. These reduction comes from the last fully-connected layer before softmax, which is the bottleneck of the overall model size. Compared to the processing procedure, the sequence of LSTM blocks has nothing to do with the model size since they all share same weight parameters. Such a reduction of model size is crucial to user experience, as it can save the local storage, memory usage at runtime, and the time to load the model when our application is launched.

Energy Consumption. We then evaluate the energy consumption of our approach on Nexus 6. The previous experiments all run the device in the default CPU governor, namely *interactive*, which boosts the CPU frequency to a high level. In practice, there are cases that the CPU is running in a low frequency (*powersave* governor) for reasons like high chip temperature or low battery remained. Figure 9b shows the normalized energy consumption of applying our optimization techniques to train a 100K data under different CPU statuses. As observed, we can significantly reduce the consumed energy for 90.3% on default governor. Such a saving is consistent with the



(a) Battery charging under different device status.



(b) Training time of one batch with different data lengths.

Fig. 10. Evaluation of on-device training: impacts on battery charging and online latency.

reduced processing time. In addition, running in the power-saving state leads to even less energy consumption, although it requires more time to train the model. This finding indicates that we can adaptively turn down the CPU frequency to save energy if it's not urgent to deploy a new model.

We further evaluate how our model training affects the device charging. For each round, we charge the device battery from 80% remaining and log the battery level every 10 minutes during the charging process. We carry out this experiment under four different contexts: *device idle*, train the model on 100K data *w/o optimization*, train the model on 100K data with optimization enabled with *interactive* governor and *powersave* governor. In this experiment we train the data for only one epoch. The results of these four rounds are illustrated as four lines in Figure 10a. As observed, it takes around 40 minutes to fully charge the device from 80% to 100% when the device is idle, but it takes about 2.5 times more (140 minutes) to charge when the model is trained in background. Applying our optimization techniques with the default governor doesn't improve much at the beginning, since they both keep the CPU busy around 75% usage as recorded. But after around 20 minutes the device is charged faster because the training is finished, and in overall it takes around 55 minutes to full charged. In another case, we train the model on the powersave governor with optimization enabled, and it takes around 48 minutes until fully charged, a little quicker than the above case. But at this point the training is not already finished actually. It should be noted that we do not expect the slow charging to be a big issue, because the offline training needs to be performed only *once* after the model update, i.e., when the IMA providers have engineered a better model to replace the old one. For example, currently the Kika engineers determined to update the model (without personalization and local training) roughly every 3 months.

6.2.3 Cost of On-Device Online Training. The on-device online training is expected to exhibit similar performance compared to the on-device offline training, since they use the same training algorithm. The difference is that we reuse inference results to continuously tune the model during the user input procedure. As mentioned previously, the latency of on-device online training is more critical as it has to be done within the interval of user input.

We show the performance of training one batch (16) of samples within different sequence length in Figure 10b. As observed, without reusing the inference results, the training time is proportional to the sequence length, and the training time exceeds 0.4s when the sequence length is longer than 18. This result is reasonable, as the training algorithm (BP) always makes a forward pass of each LSTM blocking. Thus, longer sequences mean more computations. However, with our technique of reusing inference results, the training time is quite constant and efficient (20ms~60ms). This result demonstrates that the sequence length of training data has no effects on the

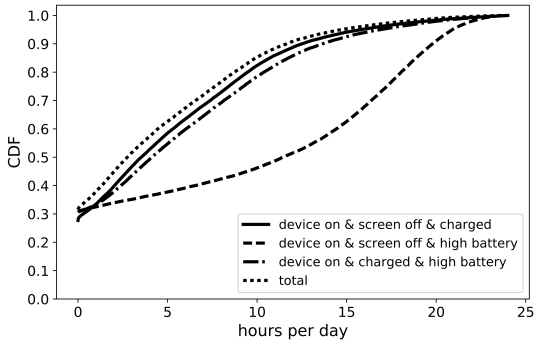


Fig. 11. Classification of mobile device usage. Medium values reported.

overall training time. The reason of such stable training time is twofold. First, during the forward pass, we do not need to go through the LSTM blocks since we directly reuse the results from previous inference. Second, during the backward pass, we update only the parameters of the last fully-connected layer without propagating the gradients to the former layers (due to the fine-tune training technique). As a summary, our experiments show that the on-device online training has very few impacts on the user experience.

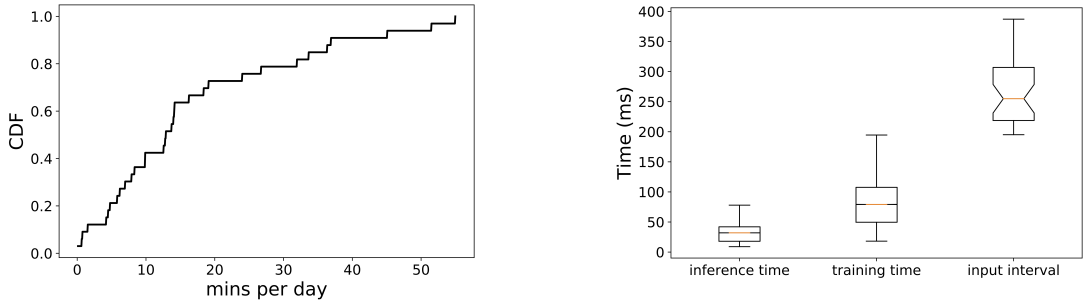
Finally, we evaluate the energy overhead of on-device online training. We measure the energy drain of typing 300 words using the Facebook app. We repeat the typing procedure with and without DeepType's online training (each repeated for 5 times with average values reported), respectively. The results indicate that the online training increases the energy consumption of our DeepType app by only **7.4%**. The *system-wide* energy overhead incurred by online training is much lower – only **1.3%**. Note that although training is more computationally intensive than inference, it runs at a low frequency in our case: the inference (prediction) is performed every time when the user types a character, while the online training is invoked only when there is a batch of samples (16 words).

6.3 Timing of On-Device Offline Training

The preceding sections demonstrate that DeepType can achieve satisfactory performance for input prediction. Since we essentially rely on the on-device offline training that can occupy the local computation cost, we need to carefully consider the timing of this phase to make the approach practically feasible. One straightforward concern is how much time is available for model training on each mobile device per day. To answer this question, we analyze the device usage behavior of real-world mobile users to support the feasibility of on-device offline training.

We have collected four kinds of events and the corresponding timestamps from mobile devices: device turned on/off, device screen turned on/off, device battery charged on/off, and device switched into high/low battery states. To collect the information, we add corresponding event receivers in Kika app. We distribute the Kika app with those event receivers to part of the users of Kika, and finally receive 4.7GB data from 45,000 users during one week (Mar. 1st to Mar. 7th in 2018).

We investigate into how much time is available for the model personalization per day. To ensure that our approach has acceptable impacts on the user experience or even be user-unaware, we tend to perform the on-device offline training when many (even all) of such states are satisfied: device turned on, screen turned off, battery charged, and high battery remained. The reason is that 1) model training is possible only when device is turned on, 2) screen turned off indicates the users is not interacting with the device, thus no affects imposed on the user experience such as user-perceived latency, 3) being charged and high-remaining battery status indicate that the on-device training is not likely to have significant impacts on the battery life of the device.



(a) Time spent on typing each day

(b) DeepType latency and input interval

Fig. 12. Quantitative analysis results of user study

As illustrated in Figure 11, the key observation is that, more than 50% users spend around 2.7 hours on favored states per day. If we alleviate the restriction of battery charged, more time can be potentially utilized (11.4 hours). Recall that our experiments in Section 6.2 demonstrate that the offline training takes no more than 2 hours (100K data, 5 epochs), so the personalization can be completed within one day since a device downloads a new global model. It is worth mentioning that the favored states are not necessarily continuous, and the non-continuity does not affect the offline training since the training can be paused when the state switches. Recall that the basic unit of training is a single batch of 16 words, whose training can be accomplished within one second. In this way, pausing the training task can be executed quickly.

7 USER STUDY FOR ASSESSING DEEPTYPE IN THE WILD

To further assess the usability and effectiveness of DeepType in real-world settings, we conducted an IRB-approved user study at a large university in the U.S.

7.1 Organization of User Study

We recruited 34 voluntary subjects who are students, staff, or faculty members. For diversity concerns, the distribution of involved subjects is: 38% are female; 85% are students; 15% are staff or faculty members. The age of the subjects ranges from 18 to 39, with 82% below 30. All subjects are native English speakers and use English as the primary language on their mobile devices. The subjects use diverse Android smartphones manufactured by 6 vendors (Google, Samsung, Huawei, etc.) along with 15 models (Pixel, Galaxy, One Plus, etc.). We asked the subjects to install DeepType on their smartphones, configure it as the default keyboard, and use it when typing English. Each subject gets a small amount of compensation for his/her participation.

The user study was launched from late July to middle August, 2018. Till the submission of this paper, we obtained and analyzed a 18-day dataset from 07/28/2018 to 08/14/2018. Since all users are new comers for DeepType, they do not have any historical input data. As a result, DeepType performs only the online training during the study. Also, the vocabulary compression optimization (Section 5) was not used due to the lack of the input history. Indeed, measuring the accuracy relies on a long-term historical data accumulation. As we have demonstrated in Section 6, the gained accuracy from DeepType over large-scale users' input data is promising. Hence, in this paper, we report user study results from two folds: quantitative (by characterizing user behaviors and the runtime overhead) and qualitative (by conducting a user survey from all subjects).

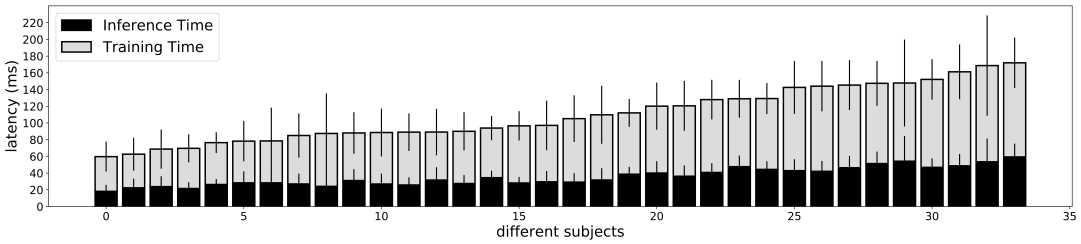


Fig. 13. Runtime latency (both inference and training time) across different users.

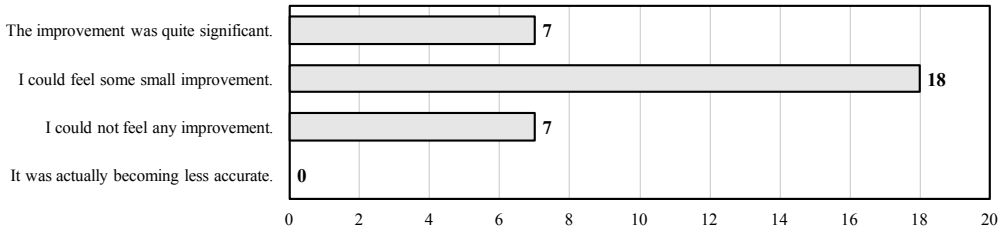


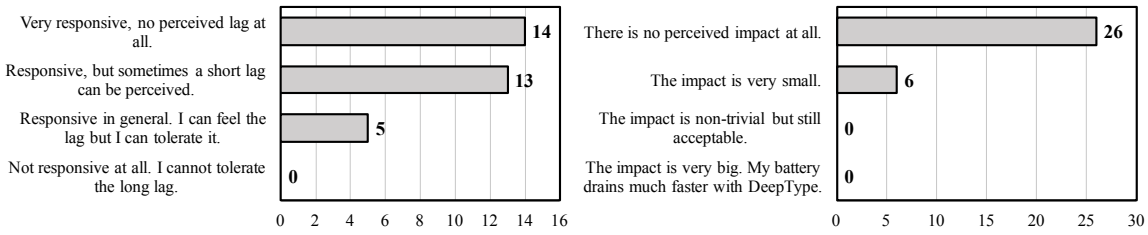
Fig. 14. Subjects' responses of whether the next-word prediction accuracy was improving as time goes by ($Q1$).

7.2 Quantitative Analysis: Usage Behaviors and Runtime Performance

We instrument our DeepType software by logging the following data: the users' typing actions, the inference time, and the training time. The logged data is periodically (everyday or so) uploaded to our secure server when the phone is in charging and idle state. All collected data items were clearly described in the IRB consent form. For privacy consideration, we collect neither the typed texts or subjects' personally identifiable information (PII).

We use the collected data to quantitatively characterize the subjects' usage behaviors and DeepType's performance. First, we explore how much time our subjects use DeepType. We define a typing session as consecutive keystrokes whose inter-keystorke time is shorter than a threshold (1 second). The typing time of a user is then calculated as the sum of the duration of all her typing sessions. As shown in Figure 12a, more than 50% of the subjects spend at least 13.8 minutes on typing (using DeepType) on an average day, and some users spend up to 55 minutes. For such heavy users, DeepType can save 11 minutes every day (or 5.5 hours per month) according to our analysis in Section 6.1 if the users had used DeepType for a sufficient period of time.

We then investigate the runtime performance of DeepType. As shown in Figure 12b, the median inference (prediction) time for one keystroke is around 35ms, and the median training time for one batch is around 86ms. These numbers are quite consistent with the results of our controlled experiments presented in Section 6.2.3. In contrast, the users' inter-keystroke time is much longer, with the median being 264ms (**7.5x** against the median inference time and **3.1x** against the median training time), according to our collected samples across all subjects. In particular, even the lowest median inter-keystroke time across all subjects is 196ms (**5.6x** of the inference time and **2.3x** of the training time). Figure 13 details the inference and training time across users. We observe some performance variations, which is largely due to the devices' diverse hardware configurations and workloads. Despite such variations, the inference (training) latency is always lower than 59ms (127ms). In addition, on 50% of all devices, the inference (training) latency is lower than 33ms (75ms). Overall, the above results demonstrate the low runtime overhead of DeepType on diverse mobile devices in real-world usage scenarios.



(a) Survey of DeepType's latency overhead ($Q2$). (b) Survey of DeepType's energy overhead ($Q3$).

Fig. 15. Subjects' responses of DeepType's overhead on latency (responsiveness) and battery life.

7.3 Qualitative Analysis

To better understand the user experience with DeepType, we make qualitative analysis by sending a survey to all subjects near the end of the data collection period. In the survey, the subjects are asked to answer the following questions through multiple choices.

• ***Q1***: "Over the past days as you were using DeepType, could you feel that the next-word prediction accuracy was improving?"

• ***Q2***: "How do you feel DeepType affects the responsiveness of the next-word prediction? ('responsive' means that the suggested word appear quickly after your tap)"

• ***Q3***: "How do you feel DeepType affects your phone's battery life?"

Out of the 34 users, we got 32 responses for the questionarie. For $Q1$, as shown in Figure 14, 7 (21.9%) users reported significant improvements of the next-word prediction accuracy; 18 (56.2%) users noticed small improvements; 7 (21.9%) users did not feel any improvement; no one felt the prediction was becoming less accurate. The results indicate that DeepType's online training helps improve the prediction, but the improvement was somewhat limited in this particular study. It is not surprising due to the short duration of this study (only 18 days), allowing a bit limited historical data for personalizing the model on each subject's device. We further discuss the limitations of deploying DeepType on fresh users in Section 8.

Regarding the responsiveness and battery life impact of DeepType ($Q2$ and $Q3$), we report the results in Figure 15. Overall the results are quite positive. 27 (84.4%) subjects reported that DeepType was (highly) responsive; no subject complained that DeepType incurred intolerable lags. Regarding the energy consumption, all subjects agreed that the battery life impact of DeepType was either none or little. 26 (81.3%) subjects reported they perceived no impacts from DeepType at all.

7.4 Summary of User Study

Although our user study is conducted in a moderate scale (34 subjects) and for a short duration (18 days), the aforementioned results have already provided promising evidence that DeepType can provide satisfactory user experiences in terms of low runtime overhead, fast responsiveness, and high prediction accuracy.

8 DISCUSSION AND FUTURE WORK

Finally, we discuss some issues and future work of DeepType.

• **Generalization** Alhough our work focuses on one specific scenario: next-word prediction, our proposed approach of on-device personalization can be generalized to other scenarios where user feedbacks can be continuously generated. Those potential scenarios include semi-autonomous driving, mobile app pre-launch,

advertisement recommendation, virtual digital assistants, etc. In addition, some of our proposed techniques to optimize on-device deep learning training can be applied on other models besides RNN.

• **Usability** Our preceding experiments assume that each mobile device stores a training dataset of input text. According to our collected data from Kika users, the average (median) number of data samples generated within one month on each device is 11,140 (8,865). In other words, it takes around half a year to generate 50K data samples for personalization. Fortunately, in practice, most users do not change their IMA frequently: it is reasonable to assume that users keep using one specific IMA for a long time. Hence, it is practical to deploy DeepType. For the small part of users that don't have enough data, we seek to borrow the idea from user clustering which has been widely adopted in building recommendation system [71, 75]. More specifically, we can cluster users into different groups based on their personal information such as gender and age. For users who are temporarily lack in history input data, they can directly share the model weights from those in the same group.

There is also a concern of the possibility where the offline training slows down other background workloads even during "favored states". Typically, such background workloads (e.g., music streaming and health tracking) are delay-tolerant and/or computationally lightweight. To demonstrate that is indeed the case in real-world settings, we log the device states and CPU utilization in our user study. The results show that when devices are under the "favored states", more than 96.2% of the time the devices have lower than 10% CPU usage. In addition, to ensure uncompromised user experience, we can assign the training task with a low priority and/or throttle its CPU utilization when other concurrent processes are running in the background. Finally, recall that described in Section 6.2.2, the offline training is performed infrequently, e.g., every 3 months in current Kika app.

• **DeepType vs. Desktop/Edge Offloading** Indeed, offloading the training task to nearby compute resources such as the user's personal desktop/laptop or edge servers, is another possible solution for improving user privacy. Compared to it, DeepType's local-only approach has various unique advantages. First, the major application context of DeepType is on mobile devices. Under mobile computing environment, not every user has an always-on desktop connection. In addition, a user may not be willing to install additional programs on his/her own desktop PC for the training offloading. Second, compared to DeepType, offloading the training to an external desktop/edge increases the risk of privacy and security concerns. Third, even with reliable and secure desktop connection, offloading works for the offline training only. Offloading online training to the desktop requires additional network connection and transition, and thus can incur much higher latency overhead compared to DeepType.

• **Privacy-preservation** The privacy-preservation claim of DeepType is built upon that user input data never leaves the personal device for model personalization. Clearly, this alone cannot guarantee no privacy violations in other links in the chain. We notice at least two possibilities: the user-typed texts may be stolen by on-device malware; the text may be sent to the cloud by the host app (e.g., Facebook or a messenger app) anyway. The first attack is well studied in the literature and is out of scope of our work (see Section 9). The second one may not be an actual privacy leak when sending the text out is required by an application's functionality. However, it is important to note that an IMA is vastly different from a normal app such as a messenger app in terms of two privacy-related aspects. First, an IMA has the visibility of *all* texts entered by the user across *all* apps. Second, an IMA typically does not take into account the semantics of the text so it may blindly leak *everything* the user typed to the external world; in contrast, a thoughtfully designed app may only upload the minimum amount of information that is essential for realizing the application functionality. Therefore, we believe keeping all text input data of IMA on a local device is a key enhancement towards preserving users' privacy.

• **Comparison with other IMAs**. Our current user study does not involve a baseline keyboard app for comparison. This is because the main purpose of this user study is to characterize DeepType's satisfactory accuracy, deployability with minimal privacy concerns, and runtime performance, rather than the accuracy improvement that has already been quantified in Section 6.1. Indeed, we agree that comparing DeepType with other IMAs such as GBoard is an important step to further evaluate DeepType's benefits in practice. We plan to carry out such comparisons in the future work.

9 RELATED WORK

In this section, we discuss existing literature studies that relate to our work in this paper.

• **Language models and LSTM** Language Modeling (LM) has been a central task in NLP. The goal of LM is to learn a probability distribution over sequences of symbols pertaining to a language. The problem of next-word prediction is a classical LM problem. In this paper, we choose the Long-Short Term Memory (LSTM) model [40] for its ability to retain long term dependencies during user input procedure. Besides LSTM, recent studies have proposed various models to enhance LSTM such as attention-based model [53] and gated recurrent unit (GRU) [20]. These advanced models can potentially bring improvements to our approach, and are compatible with the proposed techniques mentioned in Section 5.

•**Mobile deep learning** Some efforts have been proposed to make deep learning practical on resource-constrained devices, such as model compression [32, 38, 46, 67] and customized hardware architecture support for deep learning [28, 29, 37, 74]. Some of these efforts are adopted in our work (e.g., layer compression), but mostly they target at only inference of deep learning algorithms. In comparison, we propose novel techniques to optimize the training phase of deep learning models on mobile devices.

• **Decentralized machine learning** Different from standard ML approaches, decentralized machine learning [56, 66, 76] distributes the learning phase on various nodes. For example, Google proposed *federated learning* [2, 45], where mobile devices train a model locally, summarize the changes as a small focused update, and send the update to cloud where it is immediately converged with other updates to improve the shared model. In our work, the locally trained models or parameter updates will not be uploaded to the cloud as we have already trained the global model. This not only helps save the data communication between devices and cloud, but also prevents potential attacks based on updates of model parameters.

• **Privacy in mobile computing** Protecting data on mobile devices is a critical research field. For applications that need to upload sensitive data to remote clouds, e.g., for on-cloud model training, researchers have been utilizing de-identification [25, 55] and noise addition [33, 44, 58] in privacy protection. One of the most popular noise addition approaches is differential privacy [19, 26, 56] that introduces randomness into the results of queries on the underlying confidential data. Different from those efforts, DeepType preserves privacy by keeping all data on local devices. Some other approaches aim to protect local private data (e.g., login password) via secure execution environment [34, 41, 60, 64], cloud-based security services [35, 49, 59, 68], and hiding raw data after pre-processing [50, 54]. Those approaches are orthogonal to and can be leveraged by DeepType to protect, for example, the local input history for future personalization.

10 CONCLUSION

In this paper, we have addressed the fundamental conflict between personalized machine learning and user privacy. We have demonstrated the solution through DeepType, a novel approach to personalized keyboard input recommendation on mobile devices with minimum privacy concerns. The core idea of DeepType is to distribute the training phase of deep learning models on mobile devices for personalization. To tackle the insufficiency of training data on each mobile device, we pre-train a global model using public corpora on cloud, and then personalize the model for each individual user on device. To make on-device personalization feasible, we present a set of techniques to optimize the training of deep learning models on mobile devices. We evaluate the performance of DeepType both empirically, based on a large amount of behavioral data collected from commodity Android devices, and experimentally, based on a sizable user study. The results demonstrate that DeepType improves input efficiency substantially while reduces computational cost dramatically.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program under the grant number 2018YFB1004801, the National Natural Science Foundation of China under grant numbers 61725201. Feng Qian's research was supported

in part by NSF Award #1629347. Qiaozhu Mei's research was supported in part by the National Science Foundation under grant numbers 1633370 and 1620319.

REFERENCES

[1] Email Statistics Report, 2015-2019. https://www.radicati.com/wp/wp-content/uploads/2015/02/Email-Statistics-Report-2015-2019-Executive-Summary.pdf, 2015.
[2] Federated Learning: Collaborative Machine Learning without Centralized Training Data. https://research.googleblog.com/2017/04/federated-learning-collaborative.html, 2016.
[3] Messenger and WhatsApp process 60 billion messages a day, three times more than SMS. https://www.theverge.com/2016/4/12/11415198/facebook-messenger-whatsapp-number-messages-vs-sms-f8-2016, 2016.
[4] The Machine Intelligence Behind Gboard. https://research.googleblog.com/2017/05/the-machine-intelligence-behind-gboard.html, 2016.
[5] Masks to handle Variable Sequence Lengths input to RNN. https://danijar.com/variable-sequence-lengths-in-tensorflow/, 2017.
[6] Snapdragon Profiler. https://developer.qualcomm.com/software/snapdragon-profiler, 2017.
[7] WeChat users send 38 billion messages daily. http://www.chinadaily.com.cn/china/2017-11/10/content_34372396.htm, 2017.
[8] Deep Learning Embedding. https://www.tensorflow.org/versions/master/programmers_guide/embedding, 2018.
[9] Google search statistics. http://www.internetlivestats.com/google-search-statistics/, 2018.
[10] How many comments are made on reddit each day. https://www.quora.com/How-many-comments-are-made-on-reddit-each-day, 2018.
[11] Neural Network in SwiftKey. https://blog.swiftkey.com/neural-networks-a-meaningful-leap-for-mobile-typing/, 2018.
[12] NVIDIA TESLA K80. https://www.nvidia.com/en-us/data-center/tesla-k80/, 2018.
[13] Penn Tree Bank Dataset. https://catalog.ldc.upenn.edu/ldc99t42, 2018.
[14] Personalization in SwitfKey. https://goo.gl/8XaKrK, 2018.
[15] Selective Parameter Update in TensorFlow. https://www.tensorflow.org/api_docs/python/tf/train/GradientDescentOptimizer, 2018.
[16] TensorFlow. https://www.tensorflow.org/, 2018.
[17] TensorFlow Android Camera Demo. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android, 2018.
[18] Twitter Usage Statistics. http://www.internetlivestats.com/twitter-statistics/, 2018.
[19] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.
[20] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
[21] M. Barz and D. Sonntag. Gaze-guided object classification using deep neural networks for attention-based computing. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'16)*, pages 253–256, 2016.
[22] S. Bhattacharya and N. D. Lane. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems (SenSys'16)*, pages 176–189, 2016.
[23] P. F. Brown, V. J. D. Pietra, P. V. de Souza, J. C. Lai, and R. L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.
[24] F. Cai, M. De Rijke, et al. A survey of query auto completion in information retrieval. *Foundations and Trends® in Information Retrieval*, 10(4):273–363, 2016.
[25] M. Callahan. Handbook for safeguarding sensitive personally identifiable information. *Washington, DC: Department of Homeland Security. Retrieved February*, 23:2014, 2012.
[26] T.-H. H. Chan, M. Li, E. Shi, and W. Xu. Differentially private continual monitoring of heavy hitters from distributed streams. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 140–159, 2012.
[27] R. K. Chellappa and R. G. Sin. Personalization versus privacy: An empirical examination of the online consumer's dilemma. *Information technology and management*, 6(2-3):181–202, 2005.
[28] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*, pages 269–284, 2014.
[29] Y. Chen, J. S. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture, (ISCA'16)*, pages 367–379, 2016.
[30] Y. Chen, L. Mou, Y. Xu, G. Li, and Z. Jin. Compressing neural language models by sparse word representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL'16)*, 2016.
[31] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.
[32] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'14)*, pages 1269–1277, 2014.

[33] J. Domingo-Ferrer, F. Sebé, and J. Castella-Roca. On the security of noise addition for privacy in statistical databases. In *International Workshop on Privacy in Statistical Databases*, pages 149–161, 2004.
[34] J.-E. Ekberg, K. Kostiainen, and N. Asokan. Trusted execution environments on mobile devices. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1497–1498, 2013.
[35] R. Geambasu, J. P. John, S. D. Gribble, T. Kohno, and H. M. Levy. Keypad: An auditing file system for theft-prone devices. In *Proceedings of the sixth conference on Computer systems*, pages 1–16, 2011.
[36] S. Ghosh and P. O. Kristensson. Neural networks for text correction and completion in keyboard decoding. *arXiv preprint arXiv:1709.06429*, 2017.
[37] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. EIE: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture, (ISCA'16)*, pages 243–254, 2016.
[38] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. MCDNN: an approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'16)*, pages 123–136, 2016.
[39] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu. Reshaping deep neural network for fast decoding by node-pruning. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014*, pages 245–249, 2014.
[40] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
[41] S. Jana, A. Narayanan, and V. Shmatikov. A scanner darkly: Protecting user privacy from perceptual applications. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 349–363, 2013.
[42] R. Jones, R. Kumar, B. Pang, and A. Tomkins. I know what you did last summer: query logs and user privacy. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 909–914. ACM, 2007.
[43] V. Khomenko, O. Shyshkov, O. Radyvonenko, and K. Bokhan. Accelerating recurrent neural network training using sequence bucketing and multi-gpu data parallelization. In *Data Stream Mining & Processing (DSMP), IEEE First International Conference on*, pages 100–103, 2016.
[44] J. J. Kim. A method for limiting disclosure in microdata based on random noise and transformation. In *Proceedings of the section on survey research methods*, pages 303–308, 1986.
[45] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.
[46] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2016)*, pages 23:1–23:12, 2016.
[47] N. D. Lane, P. Georgiev, and L. Qendro. Deepear: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'15)*, pages 283–294, 2015.
[48] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th international conference on Knowledge discovery and data mining (KDD'14)*, pages 661–670. ACM, 2014.
[49] W. Li, M. Ma, J. Han, Y. Xia, B. Zang, C.-K. Chu, and T. Li. Building trusted path on untrusted device drivers for mobile devices. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, page 8, 2014.
[50] Y. Li, F. Chen, T. J.-J. Li, Y. Guo, G. Huang, M. Fredrikson, Y. Agarwal, and J. I. Hong. Privacystreams: Enabling transparency in personal data processing for mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT'17)*, 1(3):76, 2017.
[51] X. Lu, X. Liu, H. Li, T. Xie, Q. Mei, G. Huang, and F. Feng. PRADA: prioritizing Android devices for apps by mining large-scale usage data. In *Proceedings of 38th International Conference on Software Engineering, ICSE 2016*, pages 3–13, 2016.
[52] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017.
[53] T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP'15)*, pages 1412–1421, 2015.
[54] M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi. Privacy-preserving sensor data analysis for edge computing.
[55] E. McCallister. *Guide to protecting the confidentiality of personally identifiable information*. Diane Publishing, 2010.
[56] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. In *Sixth International Conference on Learning Representations (ICLR'18)*, pages 0–0, 2018.
[57] G. Mittal, K. B. Yagnik, M. Garg, and N. C. Krishnan. Spotgarbage: Smartphone app to detect garbage using deep learning. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'16)*, pages 940–945, 2016.
[58] K. Mivule. Utilizing noise addition for data privacy, an overview. *arXiv preprint arXiv:1309.3958*, 2013.
[59] J. Oberheide, E. Cooke, and F. Jahanian. Cloudav: N-version antivirus in the network cloud. In *USENIX Security Symposium*, pages 91–106, 2008.

[60] K. Onarlioglu, C. Mulliner, W. Robertson, and E. Kirda. Privexec: Private execution as an operating system service. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 206–220, 2013.
[61] M. Ranzato, F. J. Huang, Y. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*, 2007.
[62] J. Riedl. Personalization and privacy. *IEEE Internet Computing*, 5(6):29–31, 2001.
[63] B. A. Robinson and M. R. Longé. Keyboard system with automatic correction, Aug. 8 2006. US Patent 7,088,345.
[64] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using arm trustzone to build a trusted language runtime for mobile applications. *ACM SIGARCH Computer Architecture News*, 42(1):67–80, 2014.
[65] X. Shen, B. Tan, and C. Zhai. Privacy protection in personalized search. In *ACM SIGIR Forum*, volume 41, pages 4–17. ACM, 2007.
[66] P. Vanhaesebrouck, A. Bellet, and M. Tommasi. Decentralized collaborative learning of personalized models over networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS'17)*, 2017.
[67] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, (CVPR'16)*, pages 4820–4828, 2016.
[68] Y. Xia, Y. Liu, C. Tan, M. Ma, H. Guan, B. Zang, and H. Chen. Tinman: eliminating confidential mobile data exposure with security oriented offloading. In *Proceedings of the Tenth European Conference on Computer Systems*, page 27, 2015.
[69] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013*, pages 2365–2369, 2013.
[70] T. Yang, Y. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint arXiv:1611.05128*, 2016.
[71] L. Yanxiang, G. Deke, C. Fei, and C. Honghui. User-based clustering with top-n recommendation on cold-start problem. In *Intelligent System Design and Engineering Applications (ISDEA), 2013*, pages 1585–1589, 2013.
[72] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3320–3328, 2014.
[73] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.
[74] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'15)*, pages 161–170, 2015.
[75] D. Zhang, C. Hsu, M. Chen, Q. Chen, N. Xiong, and J. L. Mauri. Cold-start recommendation using bi-clustering and fusion for large-scale social recommender systems. *IEEE Trans. Emerging Topics Comput.*, 2(2):239–250, 2014.
[76] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, pages 685–693, 2015.

Received May 2018; revised June 2018; accepted October 2018