# EchoWrite 2.0: A Lightweight Zero-Shot Text-Entry System Based on Acoustics

Yongpan Zou, *Member, IEEE*, Zhihong Xiao, Shicong Hong, Zishuo Guo, and Kaishun Wu, *Member, IEEE*

***Abstract*—Limited by size, shape, and other factors, it is rather inconvenient to interact with new smart devices by traditional methods. Acoustic-based methods following a machine learning approach have been put forward to resolve this problem in previous works. But they possess limitations of heavy training overhead, low performance for unseen users, and intensive computation cost. Following our previous work in this area, we further overcome shortcomings of existing work and propose a lightweight and zero-shot text-entry system for unseen users based on acoustic sensing. The key novelty of this work is proposing a new model training strategy including dataset construction and augmentation methods to effectively enhance generalization ability of a simple learning model with as few training data as possible, based on our insight into the problem. We design and implement a real-time Android application system called EchoWrite 2.0 to validate our idea with extensive experiments. Results show that EchoWrite 2.0 can recognize digits, English letters, and words with an accuracy of 85.3%, 73.2%, and 96.9%, respectively, for unseen users without providing any data to the learning model. The comparison with related work in different aspects shows overall superiority of EchoWrite 2.0.**

***Index Terms*—Acoustic signals, text entry, wearable devices.**

## I. Introduction

RECENTLY, novel smart devices have gained enormous popularity among different people. A growing number of people tend to possess at least one wearable device (e.g., smartwatch, smartband, and smartglass) according to a forecast [1]. These devices are mostly of tiny sizes, which incur inconvenience for interaction tasks such as entering texts with widely used soft keyboards. As a result, it is even a rather challenging task to dial telephone numbers on these devices. In addition, since a soft keyboard requires a user to touch the screen with fingers, it will fail to work when fingers are wet or dirty, and result in hidden risks of COVID-19 due to direct interaction with public equipment.

Researchers have explored different novel methods for text entry. Among these methods, speech recognition is a popular choice that allows a user to convey information by speaking. But it possesses shortcomings of inconvenience in certain occasions (e.g., in a conference or in a library), performance degradation in noisy environments, and privacy leakage in public places. Other text-entry techniques are based on either radio-frequency (RF) signals such as WiFi and RFID [2]–[5], or inertial sensors [6]–[10]. However, RF-based techniques need specialized equipment (e.g., RFID readers and tags, multiantenna Wi-Fi transceiver, etc.) and/or require users to attach additional hardware (tags) on them. Hence, these methods are not very applicable to mobile devices. Besides, the inertial sensor-based scheme requires a user to hold a device with hand(s) or wear an additional hardware while writing texts in the air. In a word, all these equipment-dependent schemes appear to be not appealing to users.

Following a device-free manner, some recent works make use of acoustic sensors embedded in smart devices, namely, microphone and/or speaker, to deal with text-entry or related tasks [11]–[18]. Among them, the first five works adopt a passive sensing way that merely utilize a microphone to sense acoustic signals caused by writing digits or letters on a surface such as a table or paper with a pen. By analyzing received signals, the written contents can be inferred. Different from them, AcouDigits [17] makes use of both speaker and microphone in a smart device to emit and receive acoustic signals when writing texts in the air. It mainly takes advantage of Doppler shifts in received signals to differentiate written texts. Despite these differences, they share similar data processing pipelines following machine or deep learning approaches. However, these works possess the following shortcomings. First, they do not cope with the *cross-user problem* well, which makes their methods perform badly for unseen users. This is because different people write texts in different ways, which results in differences between the distribution of training data and that of testing data of unseen users. Although some of them try to resolve this issue by requesting unseen users to provide enough data to retrain classification models, it incurs heavy retraining burden for users and greatly degrades their experience. Second, in order to achieve high recognition performance, they require a large amount of training data to build a complex learning model. This not only increases the training overhead (including data collection effort and model

Manuscript received 11 October 2021; revised 8 January 2022 and 1 April 2022; accepted 2 May 2022. Date of publication 10 June 2022; date of current version 15 November 2022. This research was supported in part by China NSFC under Grant 62172286, Grant 61802264, Grant U2001207, and Grant 61872248, in part by Guangdong NSF under Grant 2022A1515011509 and Grant 2017A030312008, in part by the Shenzhen Science and Technology Foundation under Grant JCYJ20180305124807337, Grant ZDSYS20190902092853047, and Grant R2020A045, in part by the Project of DEGP under Grant 2019KCXTD005 and Grant 2021ZDZX1068, and in part by the Guangdong "Pearl River Talent Recruitment Program" under Grant 2019ZT08X603. This article was recommended by Associate Editor G. Serra. *(Corresponding author: Kaishun Wu.)*

The authors are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, Guangdong 518060, China (e-mail: yongpan@szu.edu.cn; xiaozhihong2019@email.szu.edu.cn; kyodante@outlook.com; sdgzs2018@outlook.com; wu@szu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/THMS.2022.3175416.

Digital Object Identifier 10.1109/THMS.2022.3175416

training time) for a system builder, but also affects real-time response performance on commercial devices. As a result, we ask such a question: *Can we design a lightweight, zero-shot text-entry system that can reduce model complexity, training, and retraining overhead as much as possible?*

Similar to AcouDigits [17], in this work, we also follow an active sensing route with a speaker emitting high-frequency signals to capture finger writing activities. Compared with passive sensing, active sensing has stronger robustness to environmental noises due to higher signal power and customized modulation. But the critical difference from AcouDigits [17] lies in the novel model training strategy proposed in this work. The key idea of our methods is to enrich the diversity of training dataset and expand boundaries of data distribution to cover as many unseen cases as possible, under requirements of decreasing the size of training set and getting rid of retraining process. To overcome conflict between the goal and requirements, we first figure out intrinsic factors that affect the generalization ability of a given learning model. We find that writing speed plays a dominant role since Doppler shift patterns highly depend on the speed of a moving finger during writing. What is more, we discover that users usually write texts at different speeds, and even the same person shows this kind of variance at times. It means that speed is a critical factor that induces user diversity and is the key to cope with *cross-user problem*.

Based on the above insight, we intentionally add diversity into training dataset by requesting participants to write texts at three different speed levels, namely, *slow*, *medium*, and *fast* when constructing a training dataset. Since user diversity mainly depends on writing speed, we only need one participant to collect training data, which can also effectively reduce the number of samples. In addition, to further enrich diversity of the training dataset, we apply data augmentation (DA) techniques on the original dataset similar to image processing. Combining with the above measures, we find that it is rather effective to reduce the training overhead for system builders and retraining cost for unseen users. What is more, proposed methods make it possible to achieve high recognition accuracy and good real-time response even with a very simple learning model LeNet.

In a nutshell, the main contributions of our work can be summarized as follows.

1) We propose a novel model training strategy that combines multiscene training dataset construction and DA, in order to improve data diversity and thus boost generalization capability of a learning model. With our methods, EchoWrite 2.0 greatly reduces training overhead and totally eliminates retraining process while keeping high recognition performance for unseen users. We believe that our methods can be applied to many other related topics such as human activity recognition that encounter similar problems.
2) We design and implement a real-time Android system on mobile devices, and also conduct extensive experiments to evaluate its texts-entry performance. Experimental results show that our system can recognize digits, letters, and words with average accuracies of 85.3%, 73.1%, and 96.9%, respectively, for unseen users even with much less training samples compared with other related works.

The rest of this article is organized as follows. Section II introduces the related works. In Section III, we present specific technical design of EchoWrite 2.0. Section IV introduces the experimental design, and performance evaluation is described in Section V. We conduct discussion in Section VI. Finally, Section VII concludes the article.

## II. Related Work

In this section, we discuss the existing works that are closely related to EchoWrite 2.0.

### A. Commercial Text-Entry Approaches

Due to the increasing popularity, smart devices have gained much attention to develop novel applications such as human–computer interfaces (HCI) [19], [20] and crowd sourcing [21]. To enhance interaction experience of inputting texts, different text-entry methods have been come up with. It is almost the most common way to make use of soft keyboard as an interface on present mobile devices such as smartphones, tablets, and the like. The advantages of this method include low cost, high efficiency, and good convenience. But it is not suitable for those devices with tiny screens such as smartglasses and smartwatches. In order to boost the text-input efficiency on these devices, researchers have made some technical improvements to soft-keyboard methods [19], [20]. Nevertheless, they still cannot resolve intrinsic shortcomings of touch-based text-entry methods. Speech recognition is another promising technique that can be used for inputting texts on smart devices such as Siri [22] and Cortana [23]. However, speech recognition is not appropriate for any usage scenarios in real life. For example, it is to be negatively affected by background noises, which shall degrade the recognition performance. In addition, it has the risk of privacy leakage and causing awkward feelings to users when they use speech assistant in the public occasions.

### B. Sensor and RF-Based Text-Entry Systems

In addition to the above, some other techniques have also been put forwarded including sensor-based methods [6], [7], [10], [24] and RF-based schemes [2]–[5]. In [6], the accelerometer in a smartphone is utilized to recognize characters written by a user in the air. In [7] and [10], different sensors such as distance sensor, proximity sensor and accelerometer have been fully used to design middlewares for users to enter texts to other devices. Nevertheless, they have the following shortcomings in comparison with EchoWrite 2.0: 1) in addition to the device to interact with, they require additional hardware/devices; 2) users need to carry a device in the hand while entering texts, which induces inconvenience. The RF-based schemes make use of 60-GHz transceivers, RFID, and Wi-Fi to design precise motion tracking [3], [4] or text-input systems [2], [25]. However, those systems need to deploy specialized RF hardware, which are not applicable for commercial devices. Compared with the above methods, EchoWrite 2.0 enables users to enter texts without

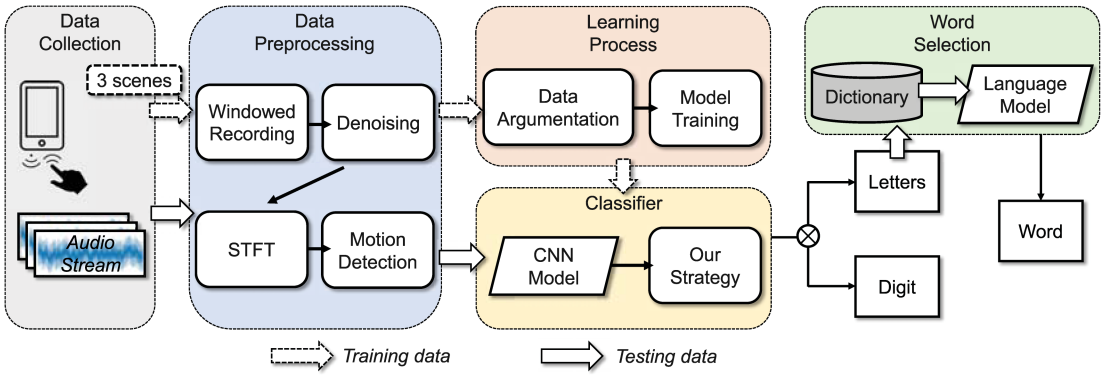



Fig. 1. Data processing workflow of EchoWrite 2.0.

additional hardware and allows them to free hands instead of carrying or wearing devices in hand or on body.

### C. Acoustic Signal-Based HCI

Researchers have also widely made use of acoustic sensors including microphone and speaker to design HCI on mobile devices. Some take advantage of the Doppler effect for gesture recognition [26]–[29] and motion tracking [30]–[33]. But all those systems except [27] and [28] require a user to carry a device in hand to perform gestures, which is rather inconvenient. What is more, the techniques proposed in these previous work cannot be used for recognizing texts. Even though the works [27], [28] can recognize in-air gestures, both of them distinguish a small set of coarse gestures such as "PULL" and "FLICK" as they only use Doppler features induced by hand gestures. Finer recognition granularity is needed to recognize texts written by a finger in the air. The works [34]–[36] are most related to EchoWrite 2.0 that achieves millimeter-level 2-D finger motion tracking in a device-free way. Nevertheless, there exist notable differences between our work and them. First, they deal with the motion tracking problem instead of text entry. Further extensions are needed for them to recognize texts. Moreover, those works require a mobile device to equip multiple speaker–microphone pairs in order to track finger motion. However, a majority of existing smart devices, such as smartphones and smartwatches, do not satisfy this requirement due to the limitation of cost or size. In contrast to them, our system only needs one speaker–microphone pair, which makes it much easier to be implemented on almost all commercial devices. Some other related works [11]–[17], [37] conducted research of text entry with acoustic signals by various means. But our work shows advantages of better performance for unseen users, much less training overhead for system builder, and good real-time response. We conduct a comprehensive comparison with these works in the evaluation part described in Section V-E.

## III. System Description

The overview of EchoWrite 2.0 is displayed in Fig. 1, which exhibits different functional components in line with data processing pipeline. In detail, a speaker in a smart device first emits sin wave-modulated inaudible signals, which is then bounced off by a finger writing texts near the device. At the same time, a microphone in the same device receives bounced echoes at a default sampling rate of 44.1 KHz. The received echoes are first denoised to filter out interfering noise and magnify the signal components of interest. Then, we transform the signal sequence into Doppler spectrograms via short-time Fourier transform (STFT), which uncover Doppler patterns of writing different digits and letters. To cut off unnecessary information in spectrograms, we detect writing events based on Doppler shifts and extract corresponding parts. In the model training stage, we collect training dataset following our proposed strategy to cover cases of different speeds. After that, we apply DA techniques that have been widely used in image processing to the obtained training dataset in order to further improve the dataset. After that, we train a LeNet model to recognize entered texts. In a word, EchoWrite 2.0 is composed of three main modules, i.e., data preprocessor, model training stage, and texts recognition module. In the following, we shall give detailed description about this workflow.

### A. Preprocessing

*1) Denoising:* The received signals are first fed into a preprocessing module to remove noise and boost signal-to-noise ratio. More precisely, we conduct noise removal on the raw signals and then detect finger writing gestures in this stage. To avoid disturbing others, we make use of high-frequency inaudible audio signals with a frequency of 19 KHz, which is beyond the human hearing range. Because of this, we only pay attention to a certain frequency band of received signals, which is centered at 19 KHz with frequency shifts caused by finger motions. Signal components out of this frequency band are regarded as noises and filtered out by a bandpass filter. To determine the filter parameters, we need to estimate the quantity of frequency shifts when a finger writes in the air. Considering a case in which a finger moves at 1 m/s, the resultant maximal frequency shift is about 112 Hz determined by

$$\Delta f = f_0 \cdot \left| 1 - \frac{v_s \pm v_f}{v_s \mp v_f} \right| \tag{1}$$

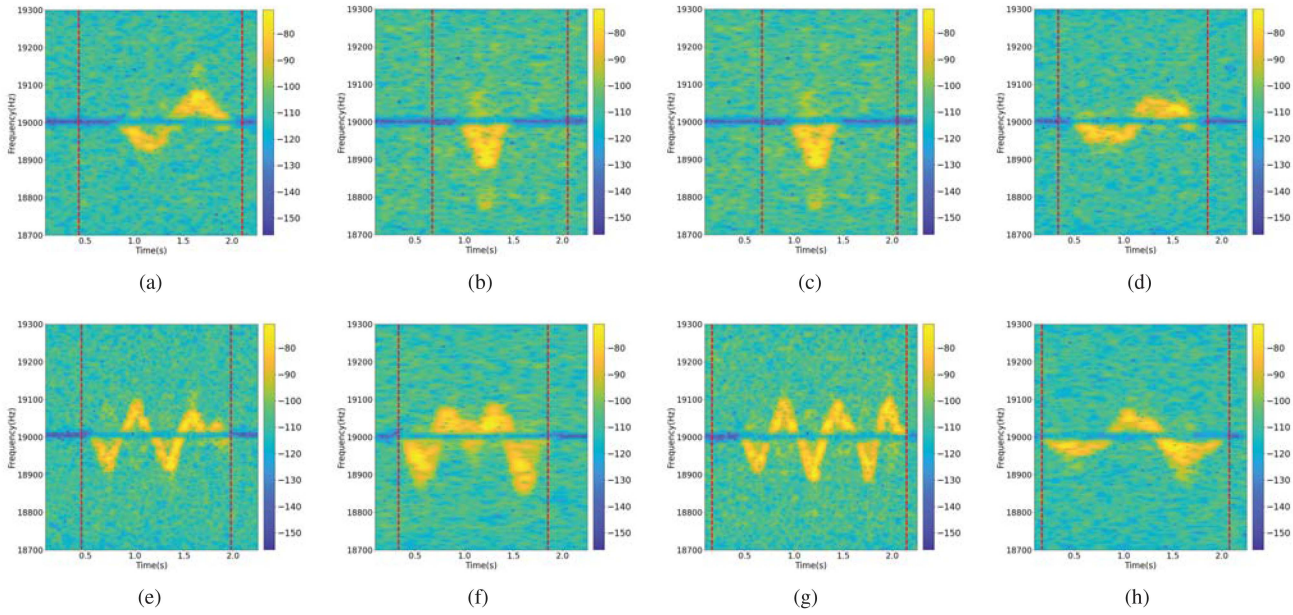



Fig. 2. Illustrations of spectrograms and writing activity detection. In each spectrogram, the color bar indicates the intensity of signals and the red dashed lines represent the detected starting and ending positions with our method. (a) Writing "0." (b) Writing "1." (c) Writing "3." (d) Writing "8." (e) Writing "A." (f) Writing "H." (g) Writing "M." (h) Writing "X."

where $f_0$, $v_s$, and $v_f$ represent the frequency of emitted signals, the speed of sound, and the velocity of finger motion, respectively. Based on this, we apply a six-order Butterworth bandpass filter with a passing band of $[18\,700, 19\,300]$ Hz on raw signals to remove background noises and human voices. Besides, signals also transmit through a direct path from speaker to microphone, which result in a dominant component at central frequency (i.e., 19 KHz). To eliminate it, a three-order band stop filter with a stopband $[18\,985, 19\,015]$ Hz is used to suppress this static component interference.

*2) Event Detection:* Following the denoising operation, writing gesture detection is performed to extract corresponding signal segments. Different from most previous works, we conduct events detection based on spectrograms of original data. It is because that the finger motion induces notable frequency shifts in the time–frequency domain. By transforming signals in time domain into spectrograms, it is easy to figure out the starting and ending points of writing activities. Technically, we apply a sliding Hamming window on a signal sequence and obtain a series of audio frames. After that, we perform FFT on each frame to obtain the spectrum [38], and then check whether there exist frequency shifts around the 19-KHz band. In detail, we set the frame length and nFFT (the length of FFT) to be 8192 sample points (i.e., about 0.1858 s) and the overlapping length between consecutive frames to be 7168 samples (i.e., about 0.1625 s). There are some tricks of setting these parameters. A larger frame length (i.e., nFFT) results in better frequency resolution (i.e., activity analysis granularity) but worse time resolution and real-time response and vice versa. The above parameters are empirically tuned by taking the tradeoff of above factors into account. Then, we can obtain a spectrogram of signal sequence with $x$ sample points, which has $\lceil (x - 8192)/1024 \rceil$ time bins and 4097 frequency bins with a resolution of 5.38 Hz. As the induced Doppler frequency belongs to $[18\,700, 19\,300]$ Hz, we can reduce computational cost by cutting off parts beyond this range.

In the training stage, we can manually label the starting and ending time of a writing event in the process of collecting training data. However, when a user tries such a real-time system, it requires to detect writing events and extract corresponding *active* segments automatically for the sake of input efficiency. To do this, we check frequency shifts of each frame of spectrogram at different frequencies. When there exists a threshold number ($\text{Thre}_1$) of continuous frequency shifts with high power, the frame is asserted to be *active*; otherwise, it is regarded as an *inactive* frame. As a general stroke writing time is about 450 ms [11] (about three bins), we set the above threshold $\text{Thre}_1$ to be 4. However, frequency shifts may also be caused by some random movements instead of writing activities. Hence, we empirically set a power threshold $\text{Thre}_2$ to be $-80$ to filter out random interference with low energy. The results of detecting writing digits and letters based on spectrograms are shown in Fig. 2. We can note that when writing "0," a user sweeps his/her finger through the microphone and back, which induces sine wave-like Doppler frequency shifts. As for digit "1" a user writes it with finger going away from microphone directly. As a result, a "valley" appears in the spectrogram.

## B. Model Training

In this section, we mainly describe the methods of model training.

*1) Model Training Strategy:* Inspired by the above insight, we redesign data collection strategy to cover more practical situations of unseen users. To be more specific, we request a trainer to collect training data in three cases according to the

level of writing speed, namely, i.e., *slow*, *medium*, and *fast*. In this way, we can construct a dataset covering three different usage scenarios that intrinsically affects Doppler shift patterns of writing different digits or letters. This strategy seems to be simple but is proved to be effective by experiments as demonstrated in Section V. The key insight is that we figure out the intrinsic factors that induce users' diversities and affect classifier's generalization capability to unseen users. We intentionally vary values of these intrinsic factors when collecting training data and thus add variance to the training dataset covering as many usage cases as possible. In our work, the most critical factor is writing speed according to our analysis, which is the reason why we design such a training data collection scheme.

*2) Data Augmentation:* After collecting training data, we apply DA techniques to further improve the generalization ability of classification model based on a small sample set. Inspired by DA in image processing, we make use of some common DA techniques including translation and scaling. Specifically, the translation operation refers to shifting spectrograms to horizontally and vertically, with translation ratios of 0.11 and 0.05, respectively. The scaling operation refers to zooming in or out a spectrogram proportionally by ratios of 1.15 and 1.07 in two directions. These parameters are carefully set to avoid distorting the spectrograms too much in the case of losing intrinsic Doppler shift patterns of writing gestures. Our insights are twofold. For one thing, considering the complex conditions of practical usage scenarios, our detection method may fail to detect writing activities precisely with detected starting and ending points shifting horizontally, as shown in Fig. 2. Translation adds more horizontally shifting samples automatically and improves the data diversity. For another thing, considering the diverse writing speeds of different users, spectrograms of the same digits or letters present compression and stretching effect vertically. Scaling adds more vertical deformations to spectrograms, which simulates virtual users with various writing speeds. In our work, translation and scaling operations are randomly applied to original training dataset and enlarge it to be roughly three times, namely, from 450 samples to 1350 samples for digits, and from 1170 samples to 3510 samples for letters. It is noted that although there are some other DA operations such as flipping, cropping, and rotation, they are not appropriate for our problem since they change Doppler shift patterns.

Fig. 3 clearly shows the effectiveness of our strategy with DA techniques in boosting the generalization capability of a classifier. In this figure, the red points represent samples of an unseen user projected in 2-D feature space after principal component analysis. The black points represent samples of a trainer collected with our strategy at three levels of writing speeds. The blue points represent samples created by DA based on the trainer's medium-speed samples (denoted by black pentagon points). Correspondingly, the black, red, and blue ellipses indicate distributions of the three datasets. As we can see, the red and black points are much closer to each other, which means that both datasets share very similar distribution patterns. In contrast, the red and blue points separate from each other obviously, which indicates the noticeable difference between corresponding data distributions. In other words, our strategy of constructing

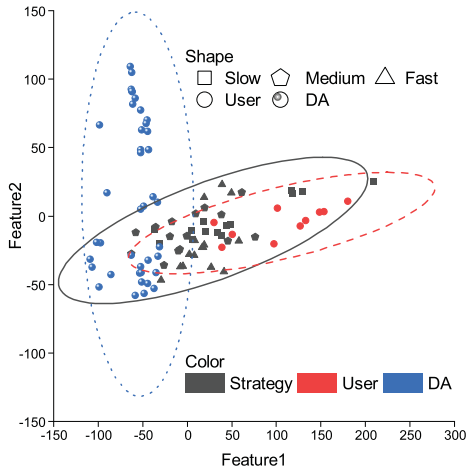



Fig. 3. Demonstration of effectiveness of our strategy with DA techniques.

training dataset is beneficial for aligning sample distributions of a trainer and unseen user, and boosting classification performance finally.

*3) Classification Model:* Following the above, we take advantage of a LeNet model for classification by taking several factors such as accuracy, real-time response, and training time into account. Although there are some other classification models appropriate for mobile devices, LeNet is an relative optimal choice after trying SVM, LSTM, and MobileNet, as described in Section V-A1 in detail. LeNet is a simple and straightforward convolutional neural network (CNN) structure, which has only seven layer besides the input layer. After tuning network parameters, LeNet performs well in terms of key evaluation metrics including accuracy, response time, and training overhead. Table I shows specific architecture and parameters of LeNet utilized in our work. It is noted that we use the same network architecture for digits and letters recognition except for the number of nodes in the last layer, with 30 and 78 nodes, respectively. We train these two models with Adam as optimizer and cross entropy as loss function utilizing corresponding dataset separately and deploy both of them into a mobile device. For the convenience of practical use, we create a shift button in the present version of Android APP such that users can easily switch text-entry functions. Obviously, this can also be accomplished by designing a special gesture if being totally touching-free is required.

*4) Classification Decision:* In this way, our model could predict an unseen user's input in an acceptable accuracy while only trained on one user's small dataset. Noticing that the speed is the core of the Doppler effect, we divide 10 digits and 26 letters into 30 and 78 classes, respectively, according to the three speed levels. We make a sum of different speeds to make the final classification as follows:

$$C = \arg\max_{i \in S}(P(i)), \text{ where } P(i) = \sum_{j \in V} P(i,j) \tag{2}$$

where $C$ is the predicted digit or letter; $S = \{`0', \ldots, `9'\}$ or $\{`A', \ldots, `Z'rbrace$, representing the full set of digits or letters; $V = \{\text{slow, medium, fast}\}$ representing the set of speed levels; and $P(i,j)$ is the probability of digit or letter $i$ with speed $j$.

TABLE I
SPECIFIC ARCHITECTURE AND PARAMETERS OF LENET UTILIZED IN OUR WORK

| Layer index | Layer type | Parameters |
|---|---|---|
| 0 | Input layer | $56 \times 56 \times 3$ |
| 1 | Convolution layer | size: $5 \times 5$, strides: (1,1), num: 6, padding: valid, activation: 'ReLu' |
| 2 | Pooling layer | size: $2 \times 2$, strides: (2,2), padding: same |
| 3 | Convolution layer | size: $5 \times 5$, strides: (1,1), num: 16, padding: valid, activation: 'ReLu' |
| 4 | Pooling layer | size: $2 \times 2$, strides: (2,2), padding: same |
| 5 | Fully connected layer | nodes: 120, activation='ReLu' |
| 6 | Fully connected layer | nodes: 84, activation='ReLu' |
| 7 | Softmax layer | nodes: # of classes, activation='ReLu' |

### C. Words Recognition

Based on the results of letter recognition, we can further accomplish words recognition based on Bayesian inference. Suppose a user writes a word $S$ of length $N$ denoted by $s_1, s_2, \ldots, s_N$. As we know, the classification model outputs a 26-dimensional likelihood vector with each element indicating the probability of recognizing $s_i$ as a certain letter $l_i$. Consequently, for the word $W$, there exists an $N \times 26$ likelihood matrix denoted by $\boldsymbol{A}$. In order to infer the right word that $S$ represents, we need to compute the probability of matching $s_1, s_2, \ldots, s_N$ with each word of length $N$ in a dataset.

Specifically, for a candidate word $W = l_1, l_2, \ldots, l_N$, the probability of mapping $S$ to it should be the product of matching every $s_i$ with $l_i$, which can be computed in a recursive way as follows:

$$P_{S\to W}(i) = P_{S\to W}(i-1) \times \boldsymbol{A}(s_i, l_i) \times P(l_i|l_1, l_2, \ldots, l_{i-1}),\ i > 1 \tag{3}$$

where $P_{S\to W}(i)$ represents the probability of matching the first $i$ letters of $S$ with those of $W$. $\boldsymbol{A}(s_i,\ l_i)$ represents the probability of recognizing $s_i$ as $l_i$. $P(l_i|l_1, l_2, \ldots, l_{i-1})$ represents the appearance probability of $l_i$ given $l_1, l_2, \ldots, l_{i-1}$, which reflects dependency of letters in words. Nevertheless, it is rather computationally intensive to calculate this probability if we take $n$-gram dependency into account especially when $n$ is a large number. As a result, we reduce it to be a simple form as follows with 2-gram dependency according to Markov chain assumption.

$$P(l_i|l_1, l_2, \ldots, l_{i-1}) \approx P(l_i|l_{i-1}) \tag{4}$$

where $P(l_i|l_{i-1})$ is also called as transition probability, which can be obtained by computing corresponding frequencies within a large words dataset as follows:

$$P(l_i|l_{i-1}) = \frac{C_{(l_{i-1}, l_i)}}{C_{(l_{i-1}, A\to Z)}} \tag{5}$$

where $C_{(l_{i-1}, l_i)}$ and $C_{(l_{i-1}, A\to Z)}$ represent the frequency of letter combination $l_{i-1}l_i$ and combinations $l_{i-1}(A \to Z)$, respectively. When $i$ equals 1, meaning that $l_i$ is the first letter of a word, the corresponding $P(l_i|l_{i-1})$ reduces to be $P(l_1)$. Consequently, (3) can be simplified as follows:

$$P_{S\to W}(i) = P_{S\to W}(i-1) \times \boldsymbol{A}(s_i, l_i) \times \frac{C_{(l_{i-1}, l_i)}}{C_{(l_{i-1}, A\to Z)}},\ i > 1. \tag{6}$$

Thus, for a word of length $N$, the final matching probability can be computed by

$$P_{S\to W}(N) = P(l_1) \times \prod_{i=1}^{N} \boldsymbol{A}(s_i, l_i) \prod_{i=2}^{N} \frac{C_{(l_{i-1}, l_i)}}{C_{(l_{i-1}, A\to Z)}},\ N > 1 \tag{7}$$

when $N$ equals 1, $P_{S\to W}(1)$ equals $\boldsymbol{A}(s_1, l_1) \times P(l_1)$. Some notes should be given about the above equations. First, when we compute (7), $P(l_1)$ and all the $C_{(l_{i-1}, l_i)}$ and $C_{(l_{i-1}, A\to Z)}$ should be counted within the words set of length $N$, instead of the whole words set. Second, all the conditional probabilities in (7) can be precomputed by counting vocabularies and then stored in a database. When a user enters texts, the system continuously queries the database to obtain corresponding conditional probabilities to infer possible words. Following the above procedure, we can obtain probabilities of mapping $S$ to different candidate words with the same length. Finally, the application will recommend several candidates with highest probabilities.

## IV. IMPLEMENTATION AND EXPERIMENTS

### A. System Implementation

To evaluate our method, we implement EchoWrite 2.0 Android application on a Samsung Galaxy Tab S2, which has an Exynos 5433 CPU chipset with a basic frequency of 1.3 GHz and 3 GB ROM. In the application, we modulate one speaker to emit a sinusoidal acoustic wave of 19-KHz frequency that can be supported by most commercial devices. At the same time, a microphone is set to receiving acoustic echoes with a sampling rate of 44.1 KHz simultaneously, which satisfies the Nyquist sampling theorem and is supported by current Android OS. Obtaining echo signals, we make use of Python codes in the Android application to via Chaquopy project [39] to deal with signal processing operations including denoising, transforming, and events detecting. The reason why we utilize Python codes instead of more efficient C codes here is that different programming languages possess different numerical truncation errors that have impact on the final results according to our experiments. In accordance with following classification model, we sacrifice some real-time performance to attain better recognition performance. Nevertheless, we think this is more an engineering problem that can be handled by sparing more development effort in the future. After that, we train classification models with Keras-2.3.1 [40] offline, pack them into h5 files, and then deploy them into mobile devices with TensorFlow Lite [41].

(a) (b)

Fig. 4. Experimental setup and Android APP. (a) Experimental setup. (b) UI of our Android APP.

### B. Experiments

To do experiments, we recruit a total number of 22 participants including 15 males and seven females from our campus with ages from 17 to 24. We pay them by 100 CNY per hour after experiments. The experiments are divided into two stages with the setup as shown in Fig. 4(a), that is, training data collection and system testing. In the former stage, we request four participants (denoted by $P1$, $P2$, $P3$, and $P4$) to write each digit and letter in the air 15 times at three levels of speed, namely, "*fast speed*," "*medium speed*," and "*slow speed*." Thus, each trainer collects 45 samples for each digit and letter used for training. It is to be noted that, although four participants take part in training data collection, we utilize samples from only one of them, that is, 450 samples for digits and 1170 samples for letters, instead of all the collected training samples to train the model and build our real-time system.

After that, we request all the participants (including the above four trainers) to use the system and evaluate its performance from three parts. Different from the training stage, we do not impose any restrictions on writing to participants except for following a set of uniform writing trajectories that are the same as those used in our previous works, namely, AcouDigits [17] (for digits) and EchoWrite [18] (for letters). In other words, they can write digits and letters at any speed, and in any size that they are used to. In this testing stage, participants can write texts in the way that they are used to. The first part is about the performance of recognizing basic digits and letters. We ask testers to write each digit and letter ten times and collect a total number of 2200 and 6760 testing cases for digits and letters, respectively. The second part is to evaluate the performance of entering words with EchoWrite 2.0. To make it more reliable and convincing, we make use of three word sets including the 50 words used for the same purpose in a latest related work [16], randomly selected 50 words in Corpus of Contemporary American English (COCA) [42], randomly selected 105 words in 20 sentences [43]. Additionally, we also evaluate the impact of word length and frequency on word recognition performance. We first categorize words in COCA by length from 1 to 15, and randomly select ten words for evaluation in each category except for the first and last ones as they only contain one word. We also classify words in COCA into ten groups with word frequency ranging from $(0, 500] \cap \mathbb{Z}$ to $(4500, 5000] \cap \mathbb{Z}$. In each group, we also randomly select ten words for evaluation. After picking up the above two word sets, participants write each word within them ten times.

There are several points about the experiments to be pointed out. First, there are some environmental variables that may affect system's performance such as noise level, writing distance, and device orientation. We conduct all the above experiments in a *baseline setting* with noise level within [45, 50) dB, writing distance at [0, 30) cm away from a device, and relative orientation being $0^\circ$. With data collected in such a *baseline setting* by a trainer, we train a *unified model* instead of specific models to build our system, and also test its performance under different conditions by varying values of the above environmental factors. Note that a *unified model* indicates that our system can handle unseen usage scenarios involving diverse users, heterogeneous devices, and different noise levels without collecting new training samples and updating the recognition model. Nevertheless, considering the overwhelming experimental overhead of full test, we request two instead of all participants to do this part of experiments. In Sections V-B3–V-B5, we shall give detailed evaluation of the impact of those variables. Second, besides the above, we also conduct experiments to evaluate the system's real-time running performance including *response time*, *CPU* and *memory occupation*, and *battery consumption*. We shall give detailed description about this in Section V-D.

## V. Evaluation

In this section, we evaluate the performance of EchoWrite 2.0 from different aspects. Besides the most frequently used metric (i.e., *accuracy*), we also adopt *top-k accuracy* to evaluate the performance of recognizing letters and words. For a letter or word entered $N$ times in total, if it occurs $n$ times in the lists with top $k$ candidates, then the *top-k accuracy* is defined as $\frac{n}{N}$.

### A. Real-Time System Building

In this section, we shall determine several key components of our real-time system by experiments, which include classification method, training subject and strategy, and DA techniques. To reduce the overhead of building a system, we request four out of the whole participants to collect 15 samples at different speeds, and each of them constructs a dataset with 450 samples.

*1) Classification Methods:* We first aim to determine a proper classification method by testing the following classifiers in recognizing digits.

1) *AcouDigits:* We implement the complete data processing pipeline including preprocessing, segmentation, feature selection, and SVM classifier following the work [17]. Then, we train and test the system with "leave-one-subject-out" policy with data of the four participants.
2) *MobileNet:* We replace the "CNN model" block in Fig. 1 with a MobileNet and train it with spectrograms [44].
3) *LSTM raw:* In this method, we directly feed the obtained acoustic signals into a fine-tuned LSTM model [45].
4) *LSTM spec.:* In this method, we feed the obtained spectrograms after preprocessing as described in Fig. 1 into an LSTM model.
5) *LeNet:* We replace the "CNN model" block in Fig. 1 with a LeNet [46].

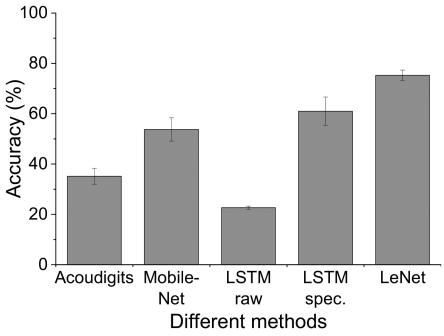



Fig. 5. Performance of recognizing digits with different methods.

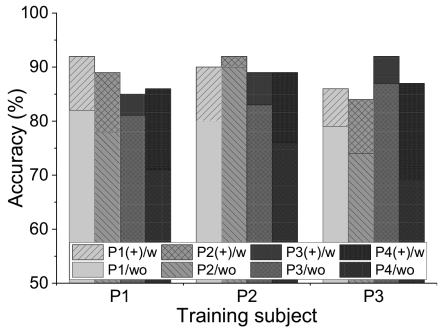



Fig. 6. System performance with/without training strategy on different subjects.

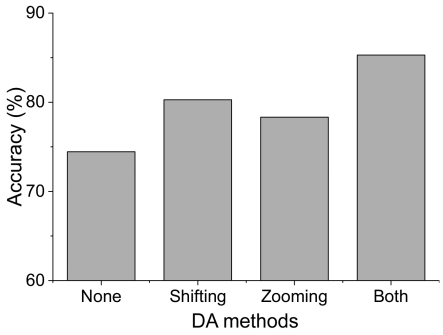



Fig. 7. System performance with/out DA techniques.

There are several points to be noted. First, we consider these models instead of other more complex and powerful ones since they can be more easily supported by mobile devices to guarantee favorable real-time response. Second, for fair comparison, all the above methods make use of the same dataset, namely, a total number of 450 samples with 45 ones for each class (i.e., training strategy is not applied) from the same subject for model training. And then, we test them with samples from the other three subjects. We conduct the above training and testing process with "leave-one-subject-out" policy among the four training subjects. Third, we have carefully tuned each method including input size, model structure, and hyperparameters with best effort to achieve their optimal performance. The average results of different methods are shown in Fig. 5 . As we can see, LeNet performs the best with an accuracy of 75.3%, better than the suboptimal method *LSTM spec.* by about 14.3%. We envision that this is mainly because LeNet is a simpler network and has stronger generalization ability especially when the training dataset is not large enough. *LSTM raw* achieves the worst performance since time series do not reveal the patterns of writing different texts obviously in contrast with spectrograms. We can also see that *Acoudigits* does not work well on our new dataset with an accuracy of 35.2%. We suspect that this is because when we construct the training datasets of $P_1$, $P_2$, and $P_3$, we request them to write digits with different speeds and scales intentionally, which exaggerates the writing diversities of different users. Additionally, we note that in our evaluation the size of input spectrograms results in a tradeoff between recognition accuracy and parameters size (or training time) with LeNet. With a larger input size, the accuracy and parameter size increase. We set this parameter to be 56 for a better balance of accuracy and real-time response. Based on the above results, we finally build our system with LeNet and conduct the following evaluations.

*2) Effectiveness of Strategy:* Following the above, we also evaluate the effectiveness of our proposed training strategy by responding to *how much gain we can obtain from this strategy* and *whether it goes for different training subjects*. To do this, we test the system performance by considering the strategy is used or not. In the case of with strategy, we train *LeNet* with one trainer's augmented dataset in each time, which has 45 samples for each of the 30 classes, and is obtained by applying DA techniques on the original dataset introduced in Section III-B2. In the case of without strategy, we make use of the same dataset with the above but merge samples of writing the same digit with different speeds into one group. As a result, there are actually 135 samples for each of the ten classes in this case. In both cases, we make use of "leave-one-subject-out" policy train and test classification models of which parameters are carefully tuned in each time. The results are shown in Fig. 6 in which $Px/wo$ represents the accuracy when the system is trained on $Px$'s dataset without applying strategy, while $Px(+)/w$ indicates the accuracy gain brought by the strategy. As we can see, the average accuracies across three training subjects in two cases are 88.4% and 79.2%, respectively. This means that the proposed strategy can indeed bring about a nearly 9.2% accuracy gain without increasing the overhead of collecting training data. As a result, we take advantage of training strategy when building the system by default. We can also notice that the strategy is similarly effective for different training subjects, with accuracies ranging from 87.3% ($P3$) to 90% ($P2$). This indicates that the proposed method applies to different training subjects without noticeable performance variations. Thus, we select $P1$ as the training subject and utilize its data to train a real-time system for the following evaluation.

*3) Gain of DA:* With strategy used, we also evaluate the impact of DA techniques by training LeNet with original dataset of $P1$ without augmentation (i.e., 30 classes and 450 samples in total) and its augmented versions (i.e., 30 classes and 1350 samples in total) with different augmentation operations including shifting, zooming, and both of them. After that, we test the obtained four models with data of three other subjects (i.e., $P2$, $P3$, and $P4$). We can see from Fig. 7 that the accuracies of

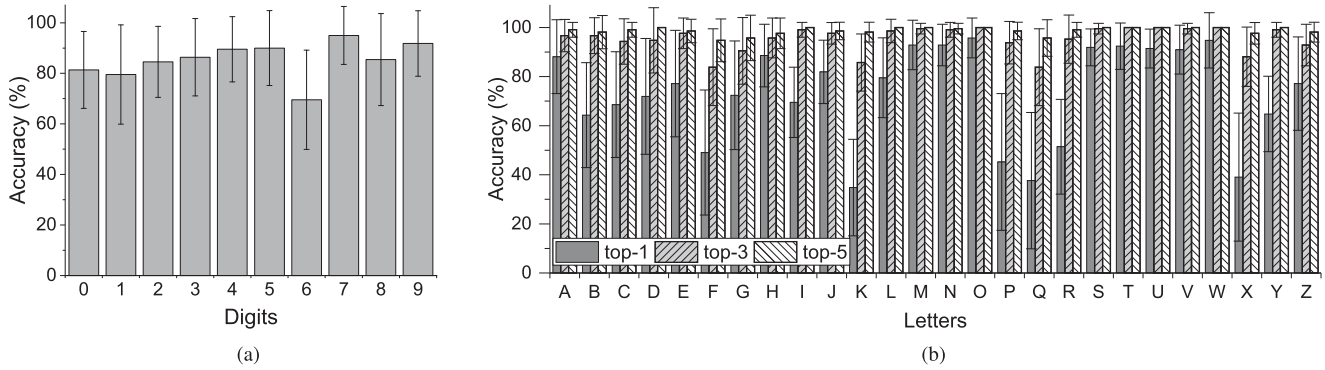



Fig. 8. Overall accuracies of recognizing different digits and letters. (a) Average accuracies of different digits. (b) Average accuracies of different letters.

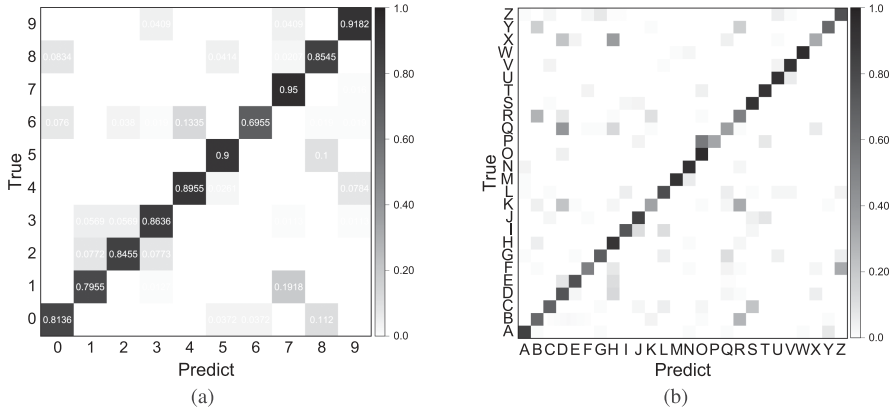



Fig. 9. Confusion matrices of recognizing digits and letters: (a) confusion matrix of recognizing digits; (b) confusion matrix of recognizing letters.

recognizing digits in the above four cases are 74.4%, 80.3%, 78.3%, and 85.3%, respectively. In other words, DA with both shifting and zooming brings about the maximum accuracy gain of 10.9% when training data collection strategy is used. Nevertheless, comparing the results shown in Figs. 5 and 6, we can find that when strategy is not applied, DA can improve system performance by only 2.7%. This in turn proves the effectiveness of proposed strategy.

### *B. Digit and Letter Recognition*

Based on the above results, we build a real-time system by making use of the augmented dataset of training subject $P1$ to train a fine-tuned LeNet with proposed training strategy. After that, we conduct comprehensive evaluation of its performance in recognizing digits and letters. Without specification, all the 22 participants take part in the evaluation experiments in this stage.

*1) Overall Performance:* Fig. 8(a) and (b) displays the overall performance of EchoWrite 2.0 in recognizing digits and letters, respectively. As we can see, the average accuracy of digits recognition over all the experimenters is 85.3% with the lowest accuracy of 70% for digit "6" and highest accuracy of 95% for digit "7." This indicates that different digits vary in recognition accuracy, which is due to the similar Doppler patterns in spectrograms of some digits. For example, as digit "4" has rather similar writing trajectory to that of digit "6," their spectrograms present similar Doppler patterns, which make them easier misclassified. Similarly, "0" and "6," "1" and "7" also have similar Doppler patterns. These can be verified by Fig. 9(a), which shows that over 13% samples of digit "6" are misclassified to be digit "4"; nearly 20% samples of digit "1" are misclassified to "7." Different from digits, we display top-1, top-3, and top-5 accuracies for English letters, since top-$k$ accuracy is meaningful for the following words input according to (7). As we can see from Fig. 8(b), the average top-1, top-3, and top-5 accuracies of all the letters are 73.2%, 95.4%, and 98.8%, respectively. Similar to digits, due to the similarities of writing patterns, some letters have relatively low recognition accuracies such as "K," "P," and "Q." This is because "D" and "K," "O" and "P," "Q" and "D" share similar Doppler shift patterns in spectrograms. Consequently, as shown in Fig. 9(b), "K," "P," and "Q" are easily miscategorized to "D," "O," and "D," respectively.

*2) Impact of User Diversity:* We also display experimental results from the perspective of user diversity, considering that different people have varied writing habits. To do this, we calculate the average recognition accuracies over all digits and letters for each participant. As shown in Fig. 10, the recognition accuracies of different participants vary from 77% to 92% with a mean value of 85.3% and standard deviation of 3.7% for digits, from 65.4% to 78.9% (top-1 accuracy) with a mean value of 73.2% and standard deviation of 3.7% for letters. It is noted

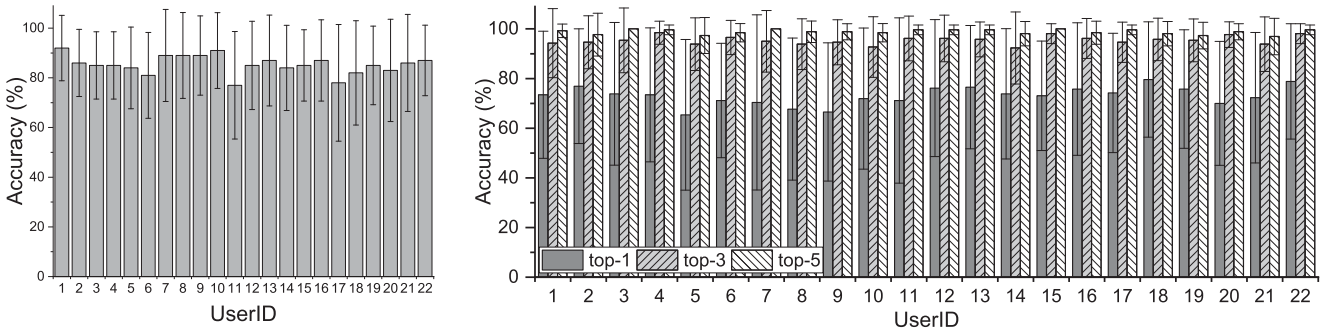
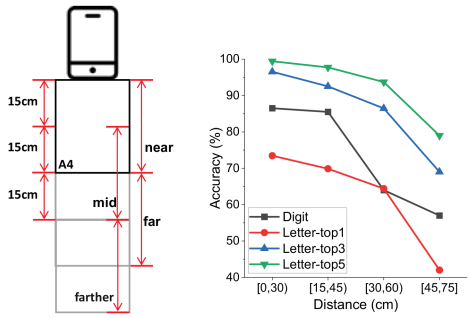
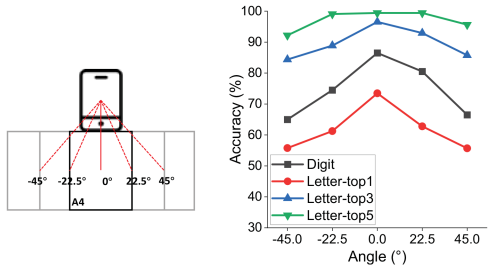



Fig. 10. Average accuracy of recognizing digits and letters for different participants.




(a) (b)

Fig. 11. Impact of relative distance. (a) Experimental setting. (b) Experimental results.




(a) (b)

Fig. 12. Impact of relative orientation. (a) Experimental setting. (b) Experimental results.

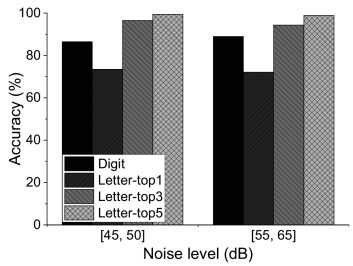



Fig. 13. Impact of noise level.

that the tested system is built only with training data from subject $P1$. This implies that even though there exist variations of recognition performance among different participants, the system based on our method shows high generalization ability across different users.

*3) Impact of Relative Distance:* As aforementioned, the above evaluation experiments are conducted in *baseline setting*. In this section, we evaluate the impact of relative distance between writing finger and the device. Since this distance varies during writing and is hard to be precisely measured, we consider four different ranges, namely, *near* (i.e., [0, 30] cm), *medium* (i.e., [15, 45] cm), *far* (i.e., [30, 60] cm), and *farther* (i.e., [45, 75] cm), as shown in Fig. 11(a). We can see that when the relative distance increases, the performance decreases whether for digits or letters. When the writing distance belongs to the *far range*, the average accuracies of digits and letters reduce to 64.4% and 64%, respectively. The underlying reason is that with an larger writing distance, the received acoustic signals become weaker, which causes the Doppler shifts to be not obvious enough.

*4) Impact of Relative Orientation:* We also evaluate the impact of relative orientation between a writing finger and the device with an experimental setting, as shown in Fig. 12(a). We divide the relative orientation from $-45°$ to $45°$ with a step of $22.5°$. In each case, we request participants to test the system with ten trails and obtain the results, as shown in Fig. 12(b). As we can see, relative orientation has obvious impact on the system performance. When a writing finger deviates from the device with an angle of $45°$, the performance decreases to 66.5% and 55.7% for digits and letters, respectively, which are lower by about 20% compared with the case of $0°$. The reason is that Doppler shifts not only depend on writing speed but also are closely related to relative orientation. Since our system is trained in the *baseline setting*, when writing orientation changes, the Doppler shift patterns of digits and letters become different, which makes it more difficult to recognize them correctly.

*5) Impact of Noise:* We also evaluate the impact of environmental noise by considering two normal cases with noise level belonging to [45, 50] dB and [55, 60] dB, which cover most daily scenarios. To control the noise level, we play different kinds of noise audio files including traffic noise, human speech, TV programs, music, etc. with precisely controlled volume. Then, we ask participants to write digits in different noise settings. We summarize the results according to the noise level, as shown in Fig. 13. We can see that the average accuracy of digits and letters in two noise levels are very close, with a

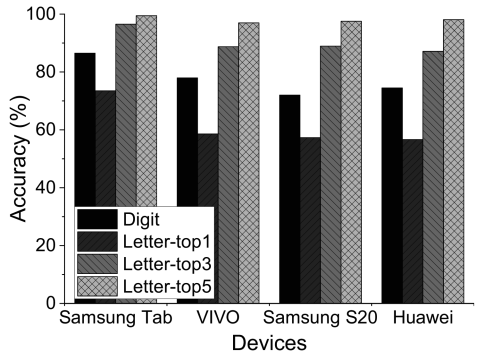



Fig. 14. System performance on different smart devices.

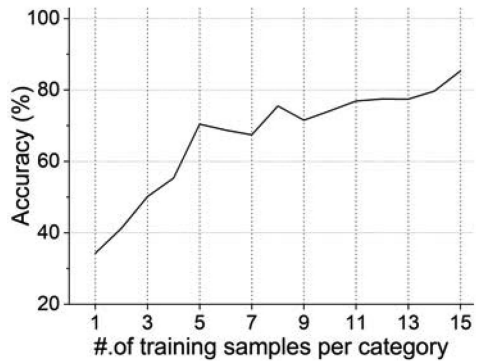



Fig. 15. System performance on different smart devices.

negligible difference of 2.5% and 1.2%, respectively. This is mainly because we make use of high-frequency acoustic signals (i.e., 19 KHz), which is far beyond normal environmental noise level.

*6) Impact of Devices:* In addition, we also evaluate the impact of different smart devices, considering that microphones and speakers equipped in devices show different response patterns. We deploy our mobile application on three other smartphones besides Samsung Tab S2, including Vivo X20, Samsung S20, and Huawei P40 Pro. It is noted that the deployed applications on other devices are exactly the same, which means the model is trained with data collected with Samsung Tab S2 in *baseline setting*. We can see from Fig. 14 that the average accuracies on the other three devices of recognizing digits and letters are (78%, 58.6%), (72%, 57.3%), and (74.5%, 56.7%). Compared with the performance on Samsung Tab S2, the performance decreases by about 11.7% and 15.9% for digits and letters, respectively. This indicates that the differences of sensors indeed cause noticeable performance variation.

*7) Impact of Training Dataset Size:* At last, to gain better knowledge of the impact of training dataset, we run testing experiments for digits recognition by varying the number of training samples per category from 1 to 15, indicating the total number of training samples from 30 to 450. The results are shown in Fig. 15. We can see that the recognition accuracy increases from 34.3% to 85.3%, which means that increasing training dataset size can boost the system performance. We can also notice that when the dataset size is below 330, its impact is more noticeable with accuracy increasing obviously. After that, adding more samples increases recognition accuracy more slightly. The above results prove that our method only relay on a very small training dataset, which is a good property for constructing a practical system.

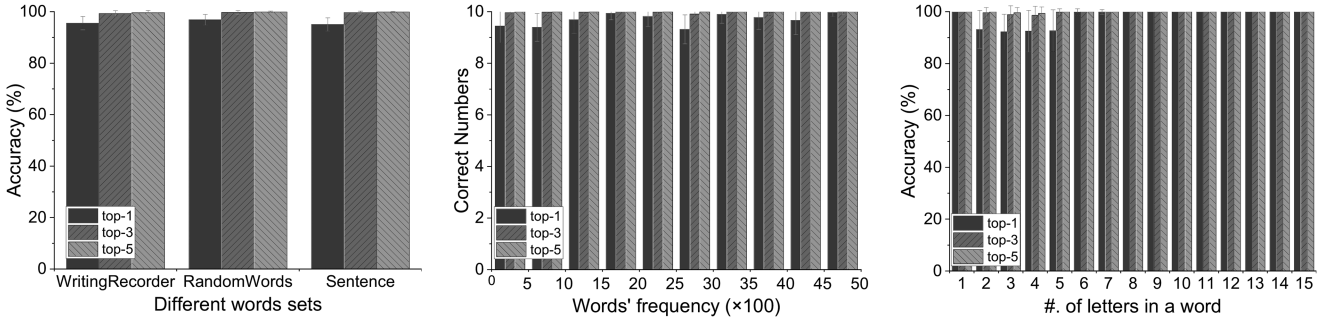
### C. Words Input

We also evaluate the performance of recognizing words. Fig. 16(a) shows its top-1, top-3, and top-5 accuracies when tested on three different words sets described in Section IV-B. The average top-1 accuracies in three cases are 95.5%, 96.9%, and 95.0%, respectively. We can see that even with different words sets, our system maintains stable and high performance. As for the top-3 and top-5 accuracies, the differences on three words sets are more negligible. As for the impact of word's frequency, we can see from Fig. 16(b) that the standard deviations of top-$k$ accuracies over different frequency ranges are 2.3%, 0.28%, and 0.04%, respectively. Compared with the mean values more than 96.9%, the mean of standard deviations 0.44% shows highly stable and satisfactory performance in words recognition, which also indicates that our method applies to words with different frequencies. Similarly, we can see from Fig. 16(c) that no matter how long words are, EchoWrite 2.0 can recognize them with high accuracy. In other words, the words recognition performance has little to do with length.

### D. Real-Time Running Performance

As a real-time system, it is crucial to evaluate the real-time running performance of our system. In this section, we shall consider three aspects including response time, CPU and memory occupation, and battery consumption.

*1) Running Time on Different Devices:* When evaluating the real-time response performance, we measure the running time of different parts constituting the system, which mainly include producing PCM file, processing of raw signals, model predication, and running language model (only when entering words). Moreover, since devices possess different computing capability, which has impact on the evaluation results, we conduct measurement on different smartphones, as done in Section V-B6. The results are shown in Fig. 17, which demonstrate two key points. For one thing, in terms of device, Samsung S20 and Huawei P40 Pro takes the least total response time compared with other two devices, which are 266.1 and 339.3 ms, respectively. The reason is kind of straightforward. These two kinds of devices are the most latest ones that came into market in the early 2020, with CPU clock speed more than 2.8 GHz and no less than 8 GB RAM. In contrast, Samsung Tab S2 is a product five years ago with a 1.3 GHz CPU and 3 GB RAM. As a result, the total response time of our system on Samsung Tab S2 is about 412 ms less than that on Samsung S2. On the other hand, from the perspective of different system parts, we can see from the figure that processing raw signals takes most of the time on either device. This is because in the raw signal processing stage, the system needs to perform continuous STFT in order to obtain Doppler shifts spectrograms, which consumes a great




(a) (b) (c)

Fig. 16. Words recognition performance under different testing conditions. (a) Words recogniiton performance on different words sets. (b) Words recognition performance by word's frequency. (c) Words recognition performance by word's length.

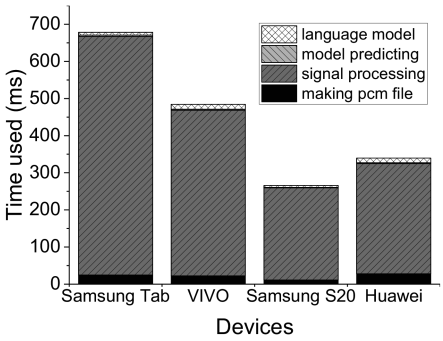



Fig. 17. Running time of different parts on different devices.

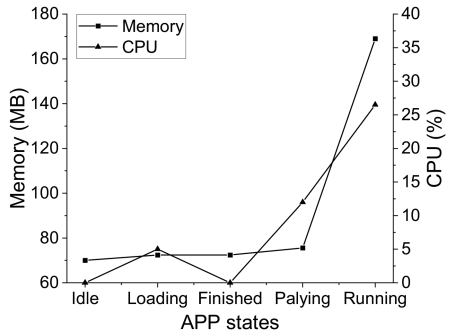



Fig. 18. CPU and memory resources occupation of different APP states.

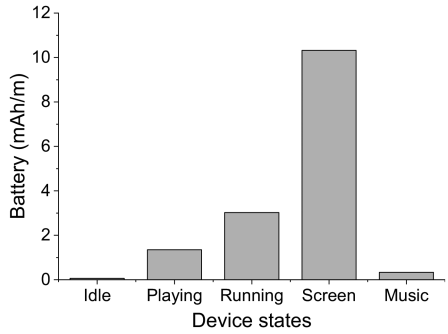



Fig. 19. Battery consumption of our system.

amount of time. In addition, as we implement this operation with Python codes and embed them into the Android program, the running performance can be further optimized with some engineering work such as replacing the signal processing codes with C programs. We leave this as part of the future work.

*2) CPU and Memory Occupation:* We also test the CPU and memory occupation of different APP running states, which include *idle* (launching but not running the APP), *loading* (loading the model), *finished* (finishing loading), *playing* (emitting high-frequency acoustics), and *running* (running all functions). To do this, we make use of Android profiler tool, which can monitor how an application uses CPU and memory resources in real time. In this experiment, taking the impact of word's length into account, we request one participant to write 15 words in total of lengths from 1 to 15 with the APP and log resources occupation data when writing each word. After that, we average the obtained results of different words. As we can see from Fig. 18, the maximum CPU and memory occupation happen when running all the functions, with an average value of 26.5% and 169 MB, respectively. It is to be noted that the memory occupation depends on the lengths of words, which varies between 112 and 273 MB according to our experiments.

*3) Battery Consumption:* Finally, we also measure the battery consumption with Android debug bridge tool when the system is running. For better comparison, we also measure the battery consumption with battery fully charged when a smart device stays in different states, including *Idle* (no any programs are running), *Playing* (just emitting 19-KHz acoustic signals), *Running* (running the whole APP program), *Screen* (lighting the screen with medium intensity), and *Music* (playing music). Each state has been tested five times, each of which lasts for 10 min. It is to be noted that the battery consumption of each state has excluded the impact of screen. As we can see from Fig. 19, light screen with medium intensity consumes the most battery energy, much more than that of our application. Compared with playing music, a daily used service, our system consumes 2.69 mAh/min more.

### *E. Comparison With Previous Work*

In the last part of evaluation, we conduct a comprehensive comparison with related work, in order to show the strengths

TABLE II
COMPARISON WITH OTHER RELATED WORK IN DIFFERENT ASPECTS

| Work | Active/passive | Trainers | Tr. samples | Retr. samples | Acc.digit | Acc.letter | Acc.word | Mobile/Server |
|---|---|---|---|---|---|---|---|---|
| WordRecorder [11] | passive | 8 | 142960 (l) | 0 | - | 88% | 74.8% | mobile |
| Pentelligence [12] | passive | 21 | 6169 (d) | 90 | 98.3% | - | - | server |
| WritingHacker [13] | passive, active | 2 | 1040 (l) | 20 | - | 67.3% | 50%∼87% | server |
| UbiWriter [14] | passive | 10 | 2080 (l) | 8 | - | 69.2% | 91.8% | server |
| Ipanel [15] | passive | 57 | 28500 (d)+74100 (l) | 0 | 85% | 85% | 91% | desktop |
| WritingRecorder [16] | passive | 19 | 3952 (l) | 0 | - | 65.0% | 86.8% | both, offloading |
| AcouDigits [17] | active | 1 | 2000(d) | 0 | 75.4% | - | - | mobile |
| **Ours** | **active** | **1** | **450 (d)+1170 (l)** | **0** | **85.3%** | **73.1%** | **96.9%** | **mobile** |

The bold entities represent the results of our system proposed in this article in comparison with other related works.

and weaknesses of different work. To make it clearer, we compare several previous related work from the aspects of sensing way (*active* or *passive*), the number of trainers, training, and retraining samples to build/update a system, the accuracies of digits, letters and words recognition, and the platform (*mobile* or *server*) on which the system is implemented and tested. All the accuracies refer to cross-user performance. The results are shown in Table II. We can see that our system can achieve comparable high performance with much fewer trainers and training samples, compared with previous work. It indicates that the methods proposed in this work can effectively reduce the training overhead while maintaining the performance at the same time. In addition, our method also reduces the system's complexity, which runs on mobile devices in real time without the aid of a remote server. As a result, we can claim that we have proposed a *lightweight*, *zero-shot* text-entry system that largely reduces model complexity, training and retraining overhead.

## VI. DISCUSSION

### A. Robustness of EchoWrite 2.0

EchoWrite 2.0 is in nature robust against ordinary noise interference such as human voice since it is built with high-frequency acoustic signals. This is also verified by our experimental results as aforementioned. But there exists another kind of interference, which is caused by irrelevant motions along with writing activities such as nearby persons' walking. Similar to writing activities, those irrelevant motions also induce Doppler frequency shifts to received signals. However, we notice that controlling emission power of acoustic signals shall effectively narrow down the propagation range, beyond which other motions nearly have little influence on the signals. In addition, to handle the interference from the same systems running simultaneously nearby, EchoWrite 2.0 can adopt an additional module responsible for sensing the working frequencies of existing running systems and choosing an nonoverlapping central frequency for emitted acoustics.

### B. Cross-Orientation Problem

As indicated by results in Section V-B4, relative orientation between a device and writing finger has noticeable impact on system performance. This is easy to understand since Doppler shift depends on relative radial velocity, which is determined by both writing speed and normal angle. As a result, changing relative orientation results in differences of Doppler shift patterns even when writing the same text. Moreover, different from speed, which only affects magnitude of Doppler shift, orientation may also alter its sign, which makes it more complex to resolve. A straightforward way to tackle this issue is to utilize the proposed training strategy, that is, dividing relative orientation into different levels and collecting some samples in each case, then applying DA techniques to dataset. Another possible way is to transform Doppler shift at different orientations from the perspective of physical model. We leave these trials as one of our future work.

### C. Uniform Writing Trajectories

In the present version of EchoWrite 2.0, users need to follow predefined writing trajectories of digits and letters in order to guarantee uniform Doppler shift patterns. This results in some inconvenience for users with unique writing habits. Fortunately, our user investigation results show that most people share very similar writing habits and are well used to the writing scheme used in our work. These people also achieve higher performance than those with writing uniqueness, as shown in Fig. 10. It is more favorable and flexible to allow users with uniqueness to customize their own writing trajectories and achieve high performance as well. A possible way is to adopt active learning approach to update the model with a new user's feedback during using this system.

## VII. CONCLUSION

Inspired by the growing popularity of tiny smart devices, we propose a touch-free text-entry system called EchoWrite 2.0 using acoustic sensors embedded in commercial devices. EchoWrite 2.0 enables a user to enter texts including digits, letters and words in the air using a finger without wearing any additional hardware. To reduce training overhead of unseen users, we put forward a novel model training strategy based on our analysis of intrinsic impact factors. Experimental results show that our method can not only completely eliminate the retraining burden of unseen users, but also greatly reduce the training effort of a system builder. With such an interface, a user can not only resolve text-entry problem of small-size devices such as giving phone calls and entering PIN codes, but also overcome awkward situations like changing TV channel when clean hands are not available.

## REFERENCES


[1] T. Page, "A forecast of the adoption of wearable technology," *Int. J. Technol. Diffusion*, vol. 6, no. 2, pp. 12–29, 2015.

[2] L. Sun, S. Sen, D. Koutsonikolas, and K.-H. Kim, "Widraw: Enabling hands-free drawing in the air on commodity WiFi devices," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 77–89.

[3] J. Wang, D. Vasisht, and D. Katabi, "RF-IDraw: Virtual touch screen in the air using RF signals," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, 2014, pp. 235–246.

[4] L. Yang, Y. Chen, X.-Y. Li, C. Xiao, M. Li, and Y. Liu, "Tagoram: Real-time tracking of mobile RFID tags to high precision using cots devices," in *Proc. 20th Annu. Int. Conf. Mobile Comput. Netw.*, 2014, pp. 237–248.

[5] T. Wei and X. Zhang, "mTrack: High-precision passive tracking using millimeter wave radios," in *Proc. 21th Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 117–129.

[6] S. Agrawal, I. Constandache, S. Gaonkar, R. Roy Choudhury, K. Caves, and F. DeRuyter, "Using mobile phones to write in air," in *Proc. 9th Int. Conf. Mobile Syst., Appl., Serv.*, 2011, pp. 15–28.

[7] C. Amma, M. Georgi, and T. Schultz, "Airwriting: Hands-free mobile text input by spotting and continuous recognition of 3D-space handwriting with inertial sensors," in *Proc. IEEE 16th Int. Symp. Wearable Comput.*, 2012, pp. 52–59.

[8] M. Goel, L. Findlater, and J. Wobbrock, "WalkType: Using accelerometer data to accommodate situational impairments in mobile touch screen text entry," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2012, pp. 2687–2696.

[9] T. Ni, D. Bowman, and C. North, "AirStroke: Bringing unistroke text entry to freehand gesture interfaces," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2011, pp. 2473–2476.

[10] S. Nirjon, J. Gummeson, D. Gelb, and K.-H. Kim, "TypingRing: A wearable ring platform for text input," in *Proc. 13th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2015, pp. 227–239.

[11] H. Du, P. Li, H. Zhou, W. Gong, G. Luo, and P. Yang, "WordRecorder: Accurate acoustic-based handwriting recognition using deep learning," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1448–1456.

[12] M. Schrapel, M.-L. Stadler, and M. Rohs, "Pentelligence: Combining pen tip motion and writing sounds for handwritten digit recognition," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2018, pp. 1–11.

[13] T. Yu, H. Jin, and K. Nahrstedt, "WritingHacker: Audio based eavesdropping of handwriting via mobile devices," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2016, pp. 463–473.

[14] H. Yin, A. Zhou, L. Liu, N. Wang, and H. Ma, "Ubiquitous writer: Robust text input for small mobile devices via acoustic sensing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5285–5296, Jun. 2019.

[15] M. Chen *et al.*, "Your table can be an input panel: Acoustic-based device-free interaction recognition," *Proc. ACM Interactive, Mobile, Wearable, Ubiquitous Technol.*, vol. 3, no. 1, pp. 1–21, 2019.

[16] H. Yin, A. Zhou, G. Su, B. Chen, L. Liu, and H. Ma, "Learning to recognize handwriting input with acoustic features," *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 4, no. 2, pp. 1–26, 2020.

[17] Y. Zou, Q. Yang, Y. Han, D. Wang, J. Cao, and K. Wu, "AcouDigits: Enabling users to input digits in the air," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2019, pp. 1–9.

[18] K. Wu, Q. Yang, B. Yuan, Y. Zou, R. Ruby, and M. Li, "EchoWrite: An acoustic-based finger input system without training," *IEEE Trans. Mobile Comput.*, vol. 20, no. 5, pp. 1789–1803, May 2021.

[19] X. Yi, C. Yu, W. Xu, X. Bi, and Y. Shi, "COMPASS: Rotational keyboard on non-touch smartwatches," in *Proc. ACM CHI Conf. Hum. Factors Comput. Syst.*, 2017, pp. 705–715.

[20] C. Yu, K. Sun, M. Zhong, X. Li, P. Zhao, and Y. Shi, "One-dimensional handwriting: Inputting letters and words on smart glasses," in *Proc. ACM CHI Conf. Hum. Factors Comput. Syst.*, 2016, pp. 71–82.

[21] C. Xiang *et al.*, "Reusing delivery drones for urban crowdsensing," *IEEE Trans. Mobile Comput.*, 2021, doi: 10.1109/TMC.2021.3127212.

[22] Apple Inc., "Siri APP," Accessed: Sep. 3, 2017. [Online]. Available: https://www.apple.com/ios/siri/

[23] Microsoft Inc., "APP Cortana," Accessed: Sep. 3, 2017. [Online]. Available: https://www.microsoft.com/en-us/windows/cortana

[24] X. Fan *et al.*, "BuildSenSys: Reusing building sensing data for traffic prediction with cross-domain learning," *IEEE Trans. Mobile Comput.*, vol. 20, no. 6, pp. 2154–2171, Jun. 2021.

[25] K. Ali, A. X. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using WiFi signals," in *Proc. ACM 21th Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 90–102.

[26] M. T. I. Aumi, S. Gupta, M. Goel, E. Larson, and S. Patel, "DopLink: Using the Doppler effect for multi-device interaction," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Computi.*, 2013, pp. 583–586.

[27] K.-Y. Chen, D. Ashbrook, M. Goel, S.-H. Lee, and S. Patel, "AirLink: Sharing files between multiple devices using in-air gestures," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2014, pp. 565–569.

[28] S. Gupta, D. Morris, S. Patel, and D. Tan, "SoundWave: Using the Doppler effect to sense gestures," in *Proc. ACM SIGCHI Conf. Hum. Factors Comput. Syst.*, 2012, pp. 1911–1914.

[29] W. Ruan, Q. Z. Sheng, L. Yang, T. Gu, P. Xu, and L. Shangguan, "AudioGest: Enabling fine-grained hand gesture detection by decoding echo signal," in *Proc. 2016 ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2016, pp. 474–485.

[30] S. Yun, Y.-C. Chen, and L. Qiu, "Turning a mobile device into a mouse in the air," in *Proc. ACM 13th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2015, pp. 15–29.

[31] Z. Sun, A. Purohit, R. Bose, and P. Zhang, "Spartacus: Spatially-aware interaction for mobile devices through energy-efficient audio sensing," in *Proc. ACM 11th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2013, pp. 263–276.

[32] Z. Zhang, D. Chu, X. Chen, and T. Moscibroda, "SwordFight: Enabling a new class of phone-to-phone action games on commodity phones," in *Proc. ACM 10th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2012, pp. 1–14.

[33] W. Mao, J. He, and L. Qiu, "CAT: High-precision acoustic motion tracking," in *Proc. ACM 22th Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 69–81.

[34] R. Nandakumar, V. Iyer, D. Tan, and S. Gollakota, "FingerIO: Using active sonar for fine-grained finger tracking," in *Proc. ACM CHI Conf. Hum. Factors Comput. Syst.*, 2016, pp. 1515–1525.

[35] W. Wang, A. X. Liu, and K. Sun, "Device-free gesture tracking using acoustic signals," in *Proc. ACM 22th Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 82–94.

[36] S. Yun, Y.-C. Chen, H. Zheng, L. Qiu, and W. Mao, "Strata: Fine-grained acoustic-based device-free tracking," in *Proc. ACM 15th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2017, pp. 15–28.

[37] M. Zhang, Q. Dai, P. Yang, J. Xiong, C. Tian, and C. Xiang, "iDial: Enabling a virtual dial plate on the hand back for around-device interaction," *Proc. ACM Interactive, Mobile, Wearable, Ubiquitous Technol.*, vol. 2, no. 1, pp. 1–20, 2018.

[38] S. W. Smith *et al.*, *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, CA: California Technical Publishing, 1997.

[39] Chaquo Ltd., "Chaquopy: Python SDK for Android," Accessed: Feb. 18, 2020. [Online]. Available: https://chaquo.com/chaquopy/

[40] F. Chollet, "Keras: The Python deep learning API," Accessed: Feb. 18, 2020. [Online]. Available: https://keras.io/

[41] G. Inc., "TensorFlow Lite: Deploy machine learning models on mobile and IoT devices," Accessed: Feb. 18, 2020. [Online]. Available: https://www.tensorflow.org/lite

[42] M. Davies, "The corpus of contemporary american english (COCA)," Accessed: Feb. 18, 2020. [Online]. Available: https://www.english-corpora.org/coca/

[43] I. S. MacKenzie and R. W. Soukoreff, "Phrase sets for evaluating text entry techniques," in *Proc. CHI Extended Abstr. Hum. Factors Comput. Syst.*, 2003, pp. 754–755.

[44] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[45] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[46] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.