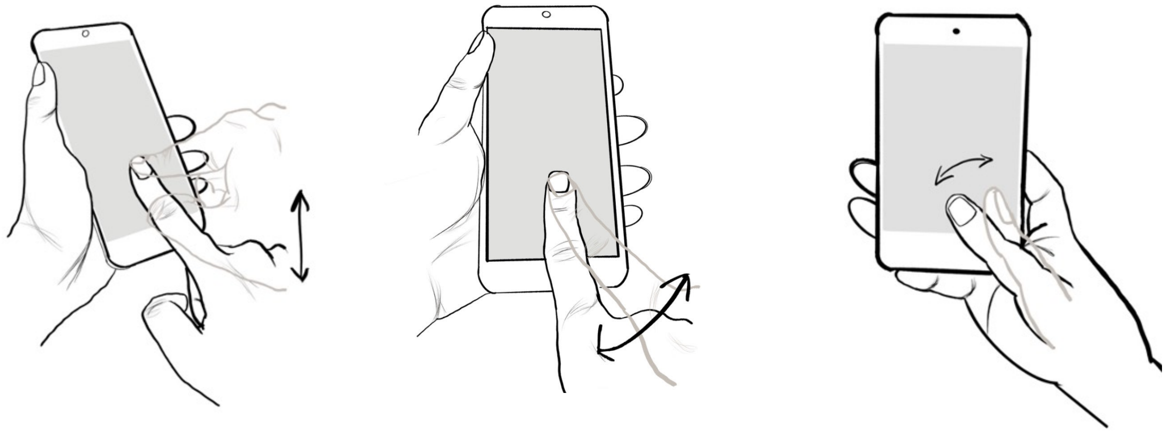
SwivelTouch: Boosting Touchscreen Input with 3D Finger Rotation Gesture

CHENTAO LI, Department of Automation, BNRist, Tsinghua University, China
JINYANG YU, Department of Automation, BNRist, Tsinghua University, China
KE HE, Department of Automation, BNRist, Tsinghua University, China
JIANJIANG FENG*, Department of Automation, BNRist, Tsinghua University, China
JIE ZHOU, Department of Automation, BNRist, Tsinghua University, China



(a) Rotation along pitch angle (b) Rotation along yaw angle (c) Rotation along roll angle

Fig. 1. A schematic diagram of SwivelTouch Gesture Set: (a) Fingers lifting or pressing down along the pitch angle; (b) fingers rotating left or right along the yaw angle; (c) the thumb executing left and right movements along the roll angle during one-handed operation of the device. Our gesture set is designed to support operations using the thumb and index finger in both one-handed and two-handed modes, offering versatility and adaptability in user interactions.

Today, touchscreens stand as the most prevalent input devices of mobile computing devices (smartphones, tablets, smartwatches). Yet, compared with desktop or laptop computers, the limited shortcut keys and physical buttons on touchscreen

*Corresponding author

Authors' Contact Information: Chentao Li, lict23@mails.tsinghua.edu.cn, Department of Automation, BNRist, Tsinghua University, Beijing, China; Jinyang Yu, jy-yu20@mails.tsinghua.edu.cn, Department of Automation, BNRist, Tsinghua University, Beijing, China; Ke He, hek18@mails.tsinghua.edu.cn, Department of Automation, BNRist, Tsinghua University, Beijing, China; Jianjiang Feng, jfeng@tsinghua.edu.cn, Department of Automation, BNRist, Tsinghua University, Beijing, China; Jie Zhou, jzhou@tsinghua.edu.cn, Department of Automation, BNRist, Tsinghua University, Beijing, China.

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 third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2024 Copyright held by the owner/author(s).
ACM 2474-9567/2024/5-ART53
https://doi.org/10.1145/3659584

devices, coupled with the fat finger problem, often lead to slower and more error-prone input and navigation, especially when dealing with text editing and other complex interaction tasks. We introduce an innovative gesture set based on finger rotations in the yaw, pitch, and roll directions on a touchscreen, diverging significantly from traditional two-dimensional interactions and promising to expand the gesture library. Despite active research in estimation of finger angles, however, the previous work faces substantial challenges, including significant estimation errors and unstable sequential outputs. Variability in user behavior further complicates the isolation of movements to a single rotational axis, leading to accidental disturbances and screen coordinate shifts that interfere with the existing sliding gestures. Consequently, the direct application of finger angle estimation algorithms for recognizing three-dimensional rotational gestures is impractical. SwivelTouch leverages the analysis of finger movement characteristics on the touchscreen captured through original capacitive image sequences, which aims to rapidly and accurately identify these advanced 3D gestures, clearly differentiating them from conventional touch interactions like tapping and sliding, thus enhancing user interaction with touch devices and meanwhile compatible with existing 2D gestures. User study further confirms that the implementation of SwivelTouch significantly enhances the efficiency of text editing on smartphones.

CCS Concepts: • **Human-centered computing → Touch screens**.

Additional Key Words and Phrases: touch gesture, text editing, deep neural network, 3D rotation gesture

ACM Reference Format:
Chentao Li, Jinyang Yu, Ke He, Jianjiang Feng, and Jie Zhou. 2024. SwivelTouch: Boosting Touchscreen Input with 3D Finger Rotation Gesture. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 8, 2, Article 53 (May 2024), 30 pages. https://doi.org/10.1145/3659584

1 INTRODUCTION

In the realm of modern communication and interaction design, traditional touch gestures have become an indispensable element, predominantly showcased in the widespread use of smartphones, tablets, and other touchscreen devices. These conventional gestures [41], including taps (double-taps), swipes (drags, pinches, flicks, etc.), and long-press, offer users an intuitive and convenient means of interacting with their devices. However, as technology evolves and user demands grow, the limitations of these traditional touch gestures have become increasingly apparent. Firstly, their capability in executing complex tasks and handling multitasking is limited, failing to meet users' higher expectations for efficiency and flexibility. Moreover, as screen sizes increase, one-handed operation becomes more challenging, and users may experience fatigue during prolonged use.

Smartphone shortcuts typically have fixed functions and constraints, unable to fully adapt to users' personalized needs [14]. The long-press gesture typically relies on a time threshold for recognition [7], which can lead to delays in response. Swiping gestures, especially those requiring extensive movement like direct dragging [27], can be highly inefficient and tiring for users. Tap gestures are prone to inadvertent touches in small touch areas, often leading to errors. While multi-touch gestures are designed to augment the limited functionality of single-touch controls, they also come with their own set of challenges, such as a lack of intuitiveness in operation. For instance, users might wish for quick access to specific applications or features, but due to the fixed number and functionality of shortcuts, such personalized configurations are often difficult to achieve. This leads to a scenario where users must perform multiple steps to complete desired tasks in their daily use, thereby reducing efficiency and user satisfaction.

Devices with hardware keyboard and mice inputs, offering over 100 distinct keys, facilitate precise text input and the definition of shortcuts, thus enhancing user interaction efficiency and versatility in computing environments [12, 25]. This setup facilitates numerous classic shortcuts, such as copy, paste, search, and quick launch, effectively minimizing the need for extensive graphical interface navigation and subsequently enhancing operational efficiency. In contrast, touchscreen smartphones, lacking any external input devices, require the user's fingers to perform both the roles traditionally reserved for mouse clicks and keyboard typing. This limitation often results in slower input rates and reduced accuracy. Thus, the integration of innovative and non-conflicting touch-based gestures could substantially enhance the input methodology of touchscreen phones, offering users an elevated level of efficiency and an improved overall experience.

Existing touch gestures primarily focus on the logic of finger movements on a two-dimensional plane, with little exploration into the realm of three-dimensional finger motion. We have identified that capacitive touchscreens can capture more nuanced features such as the shape, area, and biological structure of the finger touch surface, offering richer information than mere touch coordinates. Consequently, we have decided to analyze finger movements in three-dimensional space using a series of capacitive images captured by the touchscreen. This approach allows us to define corresponding gestures, breaking free from the traditional constraint of limiting gestures to a two-dimensional focus.

The movement of fingers in three-dimensional space can be defined by three angles: yaw, pitch, and roll. Numerous studies have proposed a variety of algorithms [9, 11, 20–22, 24, 28, 35, 45, 47, 49] for estimating finger angles. Among these, algorithms [21, 28, 47] based on capacitive touchscreens have gained significant attention due to their ease of use and the widespread availability of such screens in the market. Despite the existence of many methods, these algorithms still exhibit considerable errors in angle estimation and are typically limited to predicting only two angles: yaw and pitch. Such significant errors in using continuous capacitive images for determining the primary rotation of the finger introduce serious uncertainties in gesture classification. This makes it challenging to set a reasonable threshold for rotational movements for classification purposes. Relying on highly noticeable user rotation actions is impractical, as individual rotation habits can vary greatly. Moreover, our experiments have revealed that conflicts between some three-dimensional rotational gestures, or between these and traditional touch gestures, lead to ambiguity in classification.

In this work, we introduce SwivelTouch, a 3D gesture system designed for touchscreen operation with either one-handed or two-handed modes, as illustrated in Figure 1. We employed a combination of convolutional neural network (CNN) and Long Short-Term Memory Network (LSTM) to process a set of capacitive touchscreen images from commodity smartphones. Additionally, we proposed an algorithm to determine the first frame of an action, which significantly enhanced the accuracy of gesture recognition.

We observed that in smartphones, there are numerous text-heavy activities [39], with text editing being a prime example, where many operations are executed inefficiently. By implementing our newly defined gestures as shortcuts for these operations, we conducted a user study that demonstrated significant improvements in time efficiency and user comfort compared to traditional smartphone operations.

2 RELATED WORK

2.1 Touchscreen Interaction

Touch interaction is inherently direct and immediate, enabling users to respond swiftly to spatial and temporal differences [26]. In contrast, indirect input methods involve using hands to control external devices. Direct touch input is particularly beneficial for complex interactive tasks [13]. Modern touch-screen interactions employ direct manipulation in a spatial dimension, utilizing gestures such as tapping, dragging, swiping, and multi-finger pinching on a two-dimensional plane to trigger screen controls and perform tasks. Temporally, touch control can incorporate long-press gestures, where users maintain contact on the control object for a set threshold time (typically 500-1000ms [7]), allowing system response based on the duration of user action.

Most touch-based interaction methods predominantly utilize a single 2D location of the finger, severely limiting the input dimension and overlooking the rich information fingers can provide. Numerous techniques have been explored to enhance the input capabilities of fingers on touchscreen devices, including inputs based on finger pressure and shear force [3, 18, 31, 33, 45], characteristics of the finger contact area [4, 30], finger orientation and angle [43, 44, 47], as well as recognition of different finger parts [19]. Among these technologies, estimating a finger's 3D motion based on capacitive touchscreens, as opposed to mere finger movements on a two-dimensional plane, liberates the finger from the constraints of 2D space. This significantly enhances the dimensions and richness of the input. Researching finger angle estimation on commercial capacitive sensing [16] without auxiliary devices represents a key advancement in user interface technology. This field blends the accessibility of smartphones with sophisticated interaction techniques, offering new possibilities in touch-based user interfaces. Wang et al. [44] estimated finger yaw angles based on the shape of the response area in capacitive images. Xiao et al. [47] extracted 42 features, including ellipse orientation and response, and employed a Gaussian Regression model to estimate pitch and yaw angles. Building on this, Mayer et al. [28] proposed the use of a fully supervised convolutional neural network approach for estimating pitch and yaw angles, achieving enhanced accuracy. Ullerich et al. [40] employed a CNN model for predicting the thumb's pitch angle during one-handed phone usage. He et al. [21], recognizing the instability in estimating finger angles, introduced a deep learning model utilizing self-attention mechanisms for time series modeling, further stabilizing the estimation of finger angle sequences. These studies share a commonality: they estimate only one or two aspects of finger motion on touch screens and still encounter certain estimation errors. In recent developments, the burgeoning interest in utilizing capacitive images for hand pose estimation reflects a significant shift towards more nuanced and sophisticated interaction technologies in mobile computing. This trend is exemplified by the development of TouchPose by Ahuja et al. [1], which employs a multi-task network architecture for depth image estimation, 3D hand pose estimation, and touch classification. Simultaneously, the hand pose estimation approach of Choi et al. [8] comparing input capacitive images against a template library further underscores the innovative strides being made in this field.

Closer to our work, Roudaut et al. [37] attempted to differentiate between the roll of the thumb and swipe gestures, ingeniously utilizing the kinematic trajectory of the finger for action determination. However, this gesture recognition method has to wait until the finger is lifted from the screen, leading to a delayed response. This is incompatible with current interaction rules of touch devices, where swiping and drag gestures need rapid response. Additionally, it does not consider swipe and drag gestures with varying tilt angles due to confusion with roll gestures, also incompatible with current touchscreen gestures. Lastly, this method focuses solely on the gestures of the thumb holding the device, and is thus not directly applicable to fingers in double-handed situations. In contrast, our work employs touch images to accurately and promptly distinguish between the rolls and movements of two fingers (thumb and index) with low latency. In addition, both single-handed and double-handed cases are considered. Zaliva et al. [49] estimates finger angles and distinguish finger gesture by extracting features such as image intensity, centroid, and asymmetry of finger shape from the capacitive image on the touch surface. These estimated finger angles are then used for gesture recognition in three-dimensional space. However, the resolution of the capacitive touch surface they used is much higher than that of mainstream touch devices. The empirically designed angle estimation method in [49] cannot handle fingers captured by low resolution touch sensors. Moreover, they did not report prediction errors of individual angles or the accuracy of gesture classification.

2.2 Text Editing based on Touchscreens

Using fingers as pointing devices in text editing on mobile devices often leads to accuracy and occlusion issues (the "fat finger problem" [2, 23, 42]), making text editing less common. Bragdon et al. [5] found that gesture commands, especially in attention-divided contexts, outperform touch buttons like QWERTY keyboards in performance, and skilled gesture memory significantly reduces attention to screen widgets, thus decreasing focus on tapping accuracy. Fuccella et al. [14, 15] enabled text editing with single and double finger gestures, later implementing bimanual text input with similar gestures. Their experiments showed gesture techniques were faster than editing widgets in most Android-based mobile devices. Zhang et al. [50] introduced novel manipulation gestures for cursor control and mode switching, proving these gestures facilitated operations like copy, paste, and cut in one-handed mode, with faster and easier learning. De Rosa et al. [10] proposed a text editing technique based on arrow soft keys, using directionally arrowed keys for cursor movement and dedicated buttons for editing. Rakhmetulla et al. [32] utilized gesture shortcuts inspired by keyboard hotkeys for editing and formatting, demonstrating operation speeds surpassing traditional Google keyboards. Many researchers have explored screen edge interaction techniques to address occlusion issues in text editing [29, 34, 36, 46]. This approach allows users to initiate gestures in specific edge areas to perform corresponding operations. However, these gestures are limited to 2D planes, unfamiliar and unintuitive compared to traditional touch gestures, often causing conflicts and confusion. Schweigert et al. [38] proposed knuckle gestures for text editing tasks. Although they reported high accuracy in knuckle gesture recognition, no practical application experiments were conducted to validate the performance of these gestures. Gutwin et al. [17] used three buttons on smartphone cases, performing chords for text editing. Their study indicated rapid learning of multiple mappings, but no user experiments were conducted, and it involved additional hardware. Buschek et al. [6] investigated the patterns of targeting behavior using fingertips and styluses on mobile touchscreens, demonstrating a higher accuracy of stylus-based positioning. However, users are still required to purchase additional stylus devices. Le et al. [25] developed several back-of-device gestures for shortcuts in smartphone text editing, requiring the integration of additional capacitive sensing hardware. There are also various researchs that combine tactile interactions with human sensory perception for text editing. Zhao et al. [51] proposed a multimodal text editing and correction method for smartphones, using a combination of voice and touch inputs to facilitate text editing. EyeSayCorrect [52], on the other hand, employs eye-tracking and voice input as a means of text correction.

3 DATA COLLECTION

In our survey of existing literature, we discovered an absence of publicly available datasets capturing 3D finger motion gestures using smartphone capacitive touchscreens. Two relevant studies were identified: one by Mayer et al. [28], which did not construct the dataset in a sequential format. After reconstructing the sequences using timestamps, it was observed that the frames in many capacitive image sequences were closely spaced, with average yaw and pitch angle variations of only 6 and 4 degrees [21], indicating a lack of significant 3D gesture movement during collection. The other study by He et al. [21] compiled a large-scale temporal series of capacitive touchscreen data but lacked labeled motion gestures and specifically omitted the collection of finger roll sequences. Moreover, this dataset has not been publicly released. Consequently, our research endeavors to compile a comprehensive dataset of complete 3D finger motion gestures using smartphone capacitive touchscreens, and we will further delve into the specific details of our gesture collection methodology.

3.1 SwivelTouch Gesture Definition

Our 3D motion gestures are primarily defined by the rotation around the yaw, pitch, and roll axes, originating from the fingertip contact points along the fingers. Figure 2 illustrates the definition of finger angles, as well as examples of 3D gesture motions. We designate the contact position of a user's fingertip pressing on the smartphone screen as the interaction area. Subsequently, users are required to swivel around the fingertip contact area along the distinct axes (yaw, pitch, roll), thereby crafting the respective 3D gesture actions. Each action possesses two distinct directions of movement, thereby enabling the manipulation of six types of 3D gestures. Taking into account the realistic scenario of a user holding a smartphone, Figure 1 presents examples of 3D finger movements in both two-handed operation and one-handed holding modes. Considering the definitions of finger angles from Figure 2, in the context of using the device with both hands, the fingertip area is used for interaction. Figure 1a exemplifies a rotation of the finger along the pitch axis, Figure 1b along the yaw axis; meanwhile, Figure 1c illustrates a rotation along the roll axis in one-handed mode.

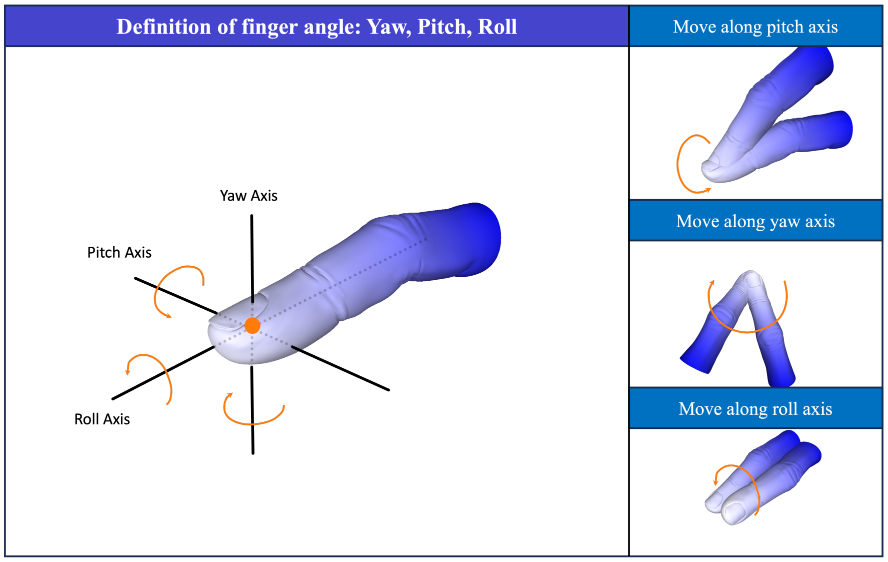


Fig. 2. Illustration of SwivelTouch Gesture. The left side illustrates the definitions of finger angles (yaw, pitch, roll), while the right side demonstrates examples of movements along these axes.

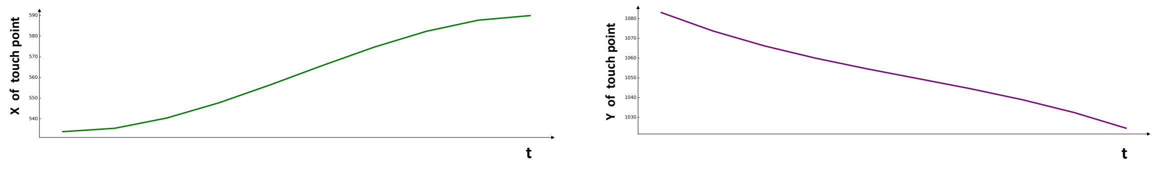
3.2 Touch Gesture Ambiguity and Categories

While our aim is to design a system capable of distinguishing finger movements in three-dimensional space, it is crucial to differentiate these movements from fundamental gestures to avoid confusion with traditional touch interactions. Traditional touch gestures can be broadly categorized into three types: finger movements on the screen (i.e. swiping, sweeping, dragging), stationary finger touches (i.e. tapping, long press), and multi-finger touches (i.e. pinching, flicking).

From the standpoint of of capacitive touch images, simple finger touches symmetrically expand around the centroid and stabilize without further changes in the contact image [31]. Long press gestures, typically triggered between 500-1000ms [7], can be identified by their duration. Multi-finger touches manifest as multiple distinct bright spots on the capacitive image, indicating multiple finger contacts. Our proposed single-finger 3D rotational gesture can be distinguished from these traditional gestures, with the exception of finger movement gestures, which can be confused with pitch and roll movements. This is because pitching or rolling the finger causes the contact point with the screen to move across the 2D space, as shown in Figure 3. It is imperative to swiftly determine whether the displacement of the contact point on the screen is caused by a three-dimensional motion gesture or merely a simple movement gesture. If not promptly discerned, these two types of gestures may interfere with each other, leading to ambiguous interactions. Therefore, in our collected gesture dataset, we include sequences of finger 3D motion gestures as well as sequences of finger movements in various directions (such as sliding, dragging, swiping, etc.) to comprehensively cover the range of potential gestures. All gestures our dataset includes are listed in Table 1.

3.3 Apparatus

Our data collection system primarily utilizes the Realme C11 2021 smartphone. This device features dimensions of 165.2mm in length, 76.4mm in width, and 8.9mm in height, and weighs approximately 190g. It boasts a 6.5-inch main screen with a resolution of 1600 × 720, operating on the Android OS 11 system. The touch sensor's



(a) (b)

Fig. 3. Illustration of gesture actions related to Roll and Picth angles inducing movement of 2D plane contact points coordinates. (a) displays the rightward movement of the x-coordinate point on the screen when the thumb performs a roll right gesture. (b) shows the upward movement of the y-coordinate point on the screen when the index finger performs a pitch up gesture. Note that the origin of the screen coordinate points is at the top-left corner of the screen.

Table 1. Illustration of gesture sequence types and their corresponding action modes in our dataset.

Category	Yaw		Pitch		Roll		Move			
Direction	Left	Right	Up	Down	Left	Right	Left	Right	Up	Down

coverage area is 160 × 75 mm^2. The controller, ICNT8962 from Chipone Technology[1] is capable of outputting capacitive images sized 32 × 18 at a refresh rate of 50FPS. Notably, the phone's Android kernel has been modified to grant developers access to real-time raw capacitive values from the touchscreen.

3.4 Participants

For our data collection phase, we engaged 10 volunteers (8 males, 2 females) aged between 19-31 years (M=23.7, SD=3.83). The entire data collection session lasted about 30 minutes per individual, and each participant received a stipend upon completion. As all volunteers were right-handed, we primarily collected data from the right index finger and thumb of each volunteer, noting the distinct shapes, sizes, and various characteristics of different fingers.

3.5 Procedure

Each participant was required to perform the ten gestures listed in Table 1 during the collection phase, executing each gesture at least ten times. This included operations with the thumb in a single-handed mode and with the index/thumb finger in a two-handed mode (one hand holding the phone and the other manipulating). On average, each user spent around 5 minutes familiarizing themselves with the 3D rotational gestures. The gestures were performed according to each user's comfortable habits without any restriction on the speed of the gestures. While we encouraged volunteers to start the gestures from the screen's frontal direction, we did not strictly correct their movements to maintain consistency, as ensuring comfort in gestures is vital for interaction and adds diverse characteristic samples to our dataset. We have developed a specialized gesture collection application that prompts users to perform designated hand gestures. Each gesture sequence, upon completion, is captured in its entirety as a series of capacitive images. Simultaneously, the application automatically annotates these sequences with the corresponding gesture labels.

[1]http://www.chiponeic.com/en/TT/229

Following these guidelines, we collected a total of 1,214 gesture sequences, comprising 76,542 capacitive images, and the gesture count is almost consistent across each category.

4 SWIVELTOUCH MODEL

In this section, we meticulously detail the process of data preprocessing applied to the raw capacitive image sequences gathered. Furthermore, we introduce a novel methodology based on time-series smoothing to identify the initial frame of a gesture action. Subsequently, we propose a hybrid deep neural network that integrates Convolutional Neural Network (CNN) with Long Short-Term Memory (LSTM) network, designed to simultaneously identify the initial frame of a gesture action through temporal smoothing and classify gestures within our collected dataset, leveraging the spatial recognition capabilities of CNN and the temporal data processing strengths of LSTM.

4.1 Dataset Preprocessing

Our preprocessing rules for the original capacitive touchscreen images are outlined as follows:

- **Noise Elimination.** Considering factors that may affect data accuracy during collection, such as temperature drift, instability in current and voltage, and sensor linear errors, we note slight capacitive deviations in non-touch areas. We set a threshold μ to address this, assigning a zero value to any point where the absolute capacitive value is below μ. After balancing factors, including low edge capacitance values near finger press areas, we chose $\mu = 50$. This effectively zeroes the capacitive array when there is no finger touch, allowing us to exclude non-touch frames from the collection.
- **Missing Value Padding.** Our CNN-LSTM framework requires a fixed length of image sequences. For gestures with rapid movements, the sequence may be shorter than our preset length $\hat{l}$. We adopt a padding strategy using the last frame to extend the original sequence to $\hat{l}$. The selection of $\hat{l}$ will be further discussed in subsequent sections.
- **Central Alignment.** Regardless of where the fingers initially contact the screen to initiate a gesture, the model should yield consistent predictive outcomes. Achieving such result is feasible through dataset augmentation. however, considering the memory and time constraints in model training, augmentation solely at the data level, while undoubtedly increases the computational burden. We consider that gesture actions represent a relative movement process. Consequently, we have employed a method of central alignment within gesture sequences. For a given gesture sequence $I = [I_1, I_2, \cdots, I_l]$, we compute the centroid C_x, C_y of the touch area in the first frame of the capacitive image. This centroid is then subtracted from the center of the original image, denoted as $\frac{H}{2}, \frac{W}{2}$, to obtain the offset values Δ_x, Δ_y. Each frame in the sequence is subsequently shifted according to these offset values, as demonstrated in Eq 1. Through this methodology, we ensure that the first frame of all gesture sequences is centrally positioned in the image, and the relative motion across sequences is preserved.

$$I_i(x, y) = I_i(x - \Delta_x, y - \Delta_y) \tag{1}$$

where $I_i(x, y)$ represents the pixel value at position (x, y) in the ith frame of the capacitive image sequences.

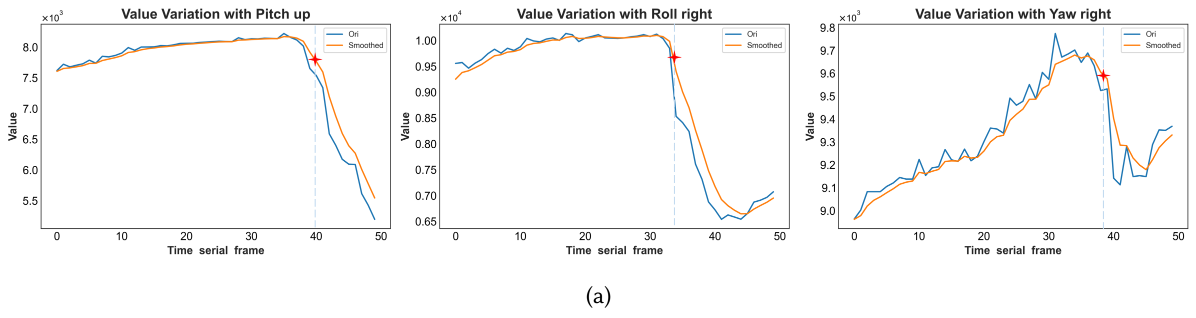
4.2 Gesture Initial Frame Identification

One evident issue we noticed was the presence of irrelevant temporal frames (e.g. incomplete touch or stationary finger pressure) between the user's finger contacting the screen and the commencement of the actual gesture. It is crucial to identify the first frame of the actual gesture to eliminate redundant input, allowing our model to focus solely on the gesture sequence. A natural approach is to detect when users disrupt the previously stable pressing pattern, causing dramatic changes in pixel values in the capacitive sensor contact area. By analyzing the data in our dataset, we discovered that the total capacitive value, calculated by summing every pixel capacitive value, also exhibits notable changes during gesture execution, as illustrated in Figure 4a. The noticeable change in the gesture sequence can be accurately captured by the first derivative of the total capacitive value time series. However, due to the inherent measurement errors of capacitive sensors, slight fluctuations in the total capacitive value may occur even when finger pressure and posture remain unchanged. To address this, we employed an exponential moving average algorithm to smooth the total capacitive value series. Let's assume the unsmoothed total capacitive value series is denoted as C_{ori}, with the capacitive value at frame t represented as $C_{\text{ori}}(t)$. Then, for the smoothed total capacitive value series C_{smo}, the relationship is defined as Eq 2.

$$C_{\text{smo}}(t) = \alpha C_{\text{ori}}(t) + (1 - \alpha) C_{\text{smo}}(t - 1) \tag{2}$$

where α represents the weighting coefficient, which signifies the impact of original data on future values. A higher α results in the smoothed data closely resembling the original series, while a lower α significantly smoothens the original series' inflection points. After reprtitious experimentation and observation of the smoothing effects, we have chosen $\alpha = 0.4$ for our study.

Upon obtaining the smoothed series of the total capacitive values, we performed a first-order difference calculation on this series. Figure 4b illustrates the resultant patterns corresponding to different gestures. In our analysis, we defined the starting frame of a gesture as the time frame in which the absolute value of the first-order differential series exceeds 100, and this elevation persists for three consecutive frames.



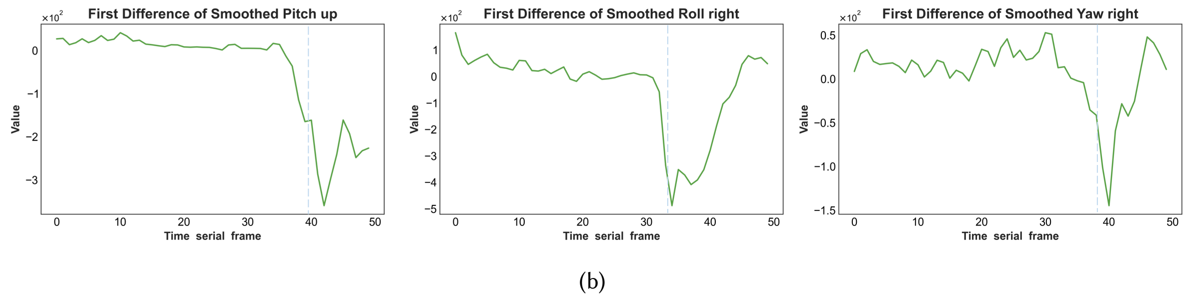


Fig. 4. (a) The variation over time of the total capacitance value of the touch screen and its corresponding smoothed values. (b) The first-order difference of the smoothed sequence. Each graph, from left to right, represents the actions of Pitch up, Roll right, and Yaw right, respectively, with the top and bottom images corresponding to the same sequence of actions. The red markers indicate the positions determined by the algorithm as the start of the action.

4.3 Model Design

Gesture recognition demands a high rate of accuracy and the capacity for real-time response. The complexity inherent in this task stems predominantly from the dual requirements of extracting spatial features and analyzing temporal dynamics. Traditional methodologies, which often employ manual feature extraction techniques such as analyzing image intensity and finger shapes [47, 49], typically are insufficient in capturing both the complex and inherent spatial structures, as well as the temporal sequencing of frames within gestures. Benefiting from the robust spatial feature extraction capabilities of Convolutional Neural Network (CNN) and the adept recognition of complex temporal input patterns by Long Short-Term Memory (LSTM) network, we adopt a CNN-LSTM network architecture. This integrated approach allows for the interpretation of gesture sequences as cohesive entities, rather than as a series of disconnected spatial features.

The SwivelTouch model, as depicted in Figure 5, is structured into two primary components: a Convolutional Neural Network (CNN) serving as the backbone for feature extraction, and a temporal processing module grounded in Long Short-Term Memory (LSTM) units. The network inputs a sequence of gestures denoted as $I = [I_1, I_2, \cdots, I_l]$, where I_1 represents the initial frame of the gesture sequence, determined using the algorithm described in Section 4.2. This frame serves as the baseline for the sequence, with l continuous image frames subsequently recorded. The CNN architecture begins with 64 convolutional kernels of size 3×3, followed by a Batch Normalization (BN) layer. This is succeeded by four residual convolutional layers, each employing 3×3 sized kernels. The number of kernels in these layers incrementally increases, with respective counts of 64, 128, 256, and 512. Following the flattening of the feature map, the architecture employs a fully connected network, which serves to extract feature vectors of a 512-dimensional space. Each frame I_i in the sequence I is processed through this shared-weight CNN, sequentially yielding features $F = [F_1, F_2, \cdots, F_l]$. These features are then channeled into the temporal processing module. The temporal processing module comprises a dual-hidden-layer LSTM network, each with a feature dimensionality of 100. Each feature F_i inputted into this module produces an output O_i. The hidden parameters of this output, in conjunction with the subsequent feature F_{i+1}, are fed back into the LSTM, facilitating the prediction of O_{i+1}. Finally, the output O_l is processed through a fully connected layer and a softmax structure, culminating in the gesture prediction results.

4.4 Model Training

The recognition of gesture actions should be characterized by real-time responsiveness, negating the need for a waiting period until the fingers are lifted from the touchscreen. Instead, the recognition process should initiate as soon as the user's intent of motion is fully expressed, enabling the model to discern the category of the gesture. Concurrently, it is essential to consider that three-dimensional gesture movements can cause deviations in the touch coordinates on the screen. Hence, the quicker the model distinguishes between 3D gestures and traditional gestures, the more effectively it can mitigate the impacts of coordinate shifts, such as minor screen sliding or inadvertent touches. This consideration necessitates an examination of the optimal sequence length, l, for the input network. Specifically, the inquiry focuses on determining the number of sequential image frames required to achieve a gesture recognition model with satisfactory accuracy. To this end, we experimented with sequence lengths ranging from 2 to 10, resulting in the development of nine distinct models. During the training phase of all models, we employed a ten-fold leave-one-out cross-validation approach. This method involves cyclically selecting data from nine individuals as the training set and data from one individual as the test set, culminating in the training of $9 \times 10 = 90$ models in total.

We utilized the Adam optimizer combined with the ReduceLROnPlateau strategy for adjusting the learning rate, setting the initial learning value at 0.001. The training was conducted over 100 epochs and the loss funtion

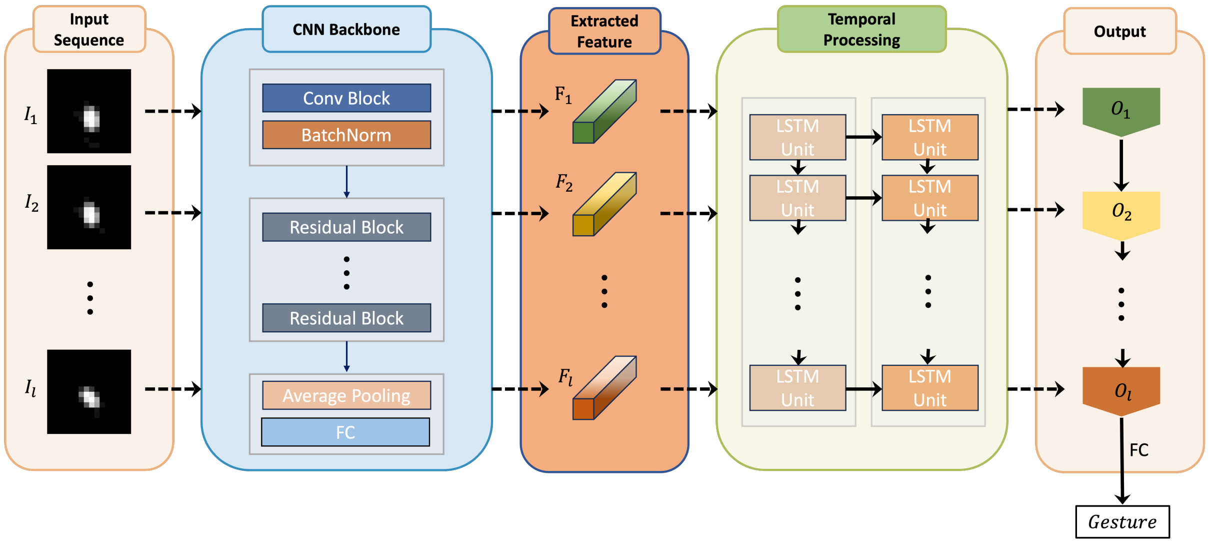


Fig. 5. Illustration of the proposed model structure. The input time-series capacitance images are first processed by a CNN backbone to extract features, which are then fed into a dual-layer LSTM network architecture.

was set as Eq 3.

$$\mathcal{L}(p,q) = -\sum_{i=1}^{C} p(x_i)\log(q(x_i)) \tag{3}$$

where $p(x_i)$ denotes the true probability of the i-th class in the distribution, and $q(x_i)$ is the predicted probability of the i-th class. Additionally, the symbol C represents the total number of classes.

5 EXPERIMENTS

In this section, we initially validate the performance of the model presented in Section 4, covering aspects such as dataset spliting methods, evaluation metric, and ablation study. Inspired by the methods [21, 28, 47] for estimating finger angles, we estimated the yaw and pitch for both the first and last frames of a sequence. We attempted to classify gestures based on the relative angular movement between these two frames and compared this straightforward approach with the performance of our model.

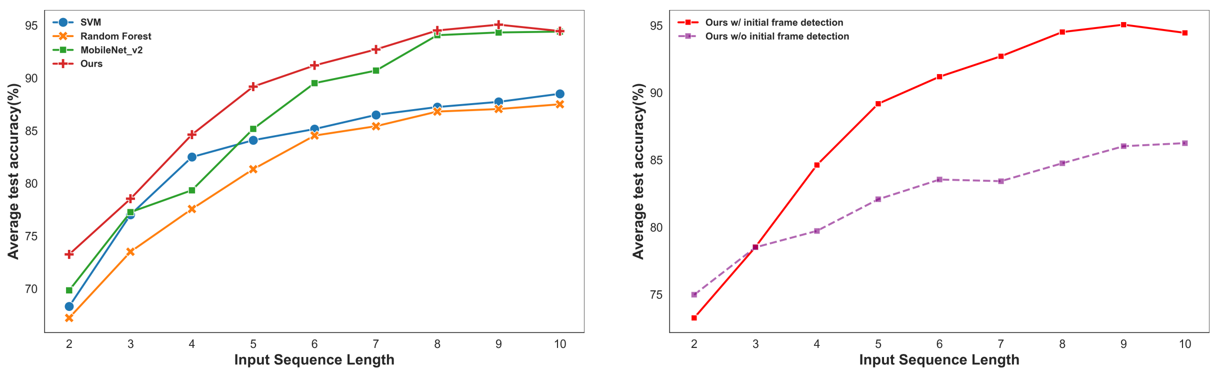
5.1 Study I: Model Evaluation

5.1.1 Evaluation Metric. As described in Section 4.4, we employed a 10-fold ‘leave-one-out’ cross-validation method to evaluate our model. This involved spliting the dataset into ten folds $[P_1, P_2, \cdots, P_{10}]$, where in each fold, a different set of nine people were used for training and the remaining one for testing. To determine the optimal gesture sequence length l for model input, we varied l from 2 to 10 and performed a ten-fold cross-validation at each length. Assuming the dataset is divided into ten folds $[P_{l;1}, P_{l;2}, \cdots, P_{l;10}]$ at sequence length l, with the test accuracy in each fold being $[E_{l;1}, E_{l;2}, \cdots, E_{l;10}]$, the average accuracy for this sequence length is defined by Eq 4.

$$E_l = \frac{1}{10}\sum_{j=1}^{10} E_{l;j} \quad l = 2, 3, \cdots, 10 \tag{4}$$

5.1.2 Ablation Study. To demonstrate the effectiveness of the methods used in our study, we conducted two ablation experiments using the evaluation approach described in Section 5.1.1.

- **Classifier Model.** To ensure the robust performance of our deep learning architecture, we compared various classification methods, including lightweight network structures and traditional classifiers: a) substituting the CNN backbone with MobileNet_v2; b) employing an SVM; and c) using a Random Forest. Both SVM and Random Forest classifiers were implemented using the scikit-learn library[2]. The experimental performance, as shown in Figure 6a, indicate that our model achieves higher average prediction accuracy across all sequence lengths. It showcased the residual structure's capability in capturing deep spatial features of the network. Compared to traditional classification methods like SVM and Random Forest, our model's average accuracy improves more significantly as the length of the input sequences increases. This demonstrates that LSTM is particularly effective at recognizing complex features in longer sequences.
- **Initial Frame Detection.** Without detecting the initial frame in the gesture sequence, we regarded the moment of finger contact as the gesture action to start. The performance was shown in Figure 6b, which illustrates higher average prediction accuracy across various sequence lengths when initial frame detection is employed. This proved the effectiveness of the initial frame detection algorithm.



(a) (b)

Fig. 6. (a) The variation in gesture recognition performance of different models with changes in the length of the input sequence. (b) The impact of using the initial frame detection algorithm on model performance.

5.1.3 Bootstrapping Validation. Although ten-fold cross-validation serves as a method for verifying model performance, its effectiveness may be compromised in datasets with a limited number of samples, where it might not fully mitigate the risk of overfitting. This issue stems from the models' tendency to memorize specific details within the training data under conditions of limited samples, rather than acquiring a generalized understanding of data features and distributions. Therefore, we decided to employ an additional validation technique, the bootstrapping method, to demonstrate the robustness of our model. The bootstrapping validation process is described as follows:

- **Step 1:** Set the number of resampling iterations $n = 10$. Denote the original dataset as D with a sample size of $m = |D|$. Initialize the training set $D_{\text{train}} = \{\}$, the test set $D_{\text{test}} = \{\}$, and the set of test accuracy results $E = \{\}$.

[2]https://scikit-learn.org

- **Step 2:** For each iteration, randomly select a sample from D (with a sample size of m) and add it to D_{train}. Then, return the sample back to D. Repeat this process m times to obtain a training dataset D_{train} consisting of m samples. Subsequently, construct D_{test} with the remaining unselected samples in D, i.e., $D_{\text{test}} = D \setminus D_{\text{train}}$.
- **Step 3:** Utilize the data in D_{train} for model training, then test the model using the D_{test} dataset to obtain the test result E_i, which is then added to E.
- **Step 4:** Check if the resampling iterations are completed. If so, terminate the process and compile the test accuracy results as $E = \{E_1, E_2, \cdots, E_n\}$. Otherwise, reset $D_{\text{train}} = \{\}$ and $D_{\text{test}} = \{\}$, and back to Step 2.

Note that the probability of any given sample in D not being selected during m sampling iterations is $P = (1-\frac{1}{m})^m$. By taking the limit, we obtain

$$P = \lim_{m\to+\infty} (1 - \frac{1}{m})^m = \frac{1}{e} = 0.368$$

Thus, after each sampling iteration, approximately 36.8% of the samples in the original dataset D will serve as the test set. We conducted bootstrapping validations for input sequence lengths l ranging from 2 to 10, with each input length l undergoing $n = 10$ iterations. This process yielded validation results $[E_{l;1}, E_{l;2}, \cdots, E_{l;10}]$ for each length. The average test accuracy E_l was calculated using the definition provided in Eq 4. Figure 7 displays the box plots of bootstrapping validation accuracies for different input sequence lengths, with the pink curve representing the trend of average test accuracies E_l. Additionally, Table 2 presents the average test accuracies obtained by both validation methods across various input sequence lengths l. We observed that the accuracy trends of the bootstrapping validation method closely align with those of ten-fold cross-validation, yet bootstrapping consistently demonstrates higher accuracy. This suggests that it has learned a more generalized feature distribution across the entire dataset. Additionally, this finding indicates that there is a certain degree of variability in the gesture distributions among different participants, which contributes to the slightly lower accuracy of cross-validation, especially at shorter sequence lengths ($l = 2, 3, 4$). However, as the input sequence length increases, the results of the two validation methods begin to converge, which sufficiently demonstrates the robustness of the model.

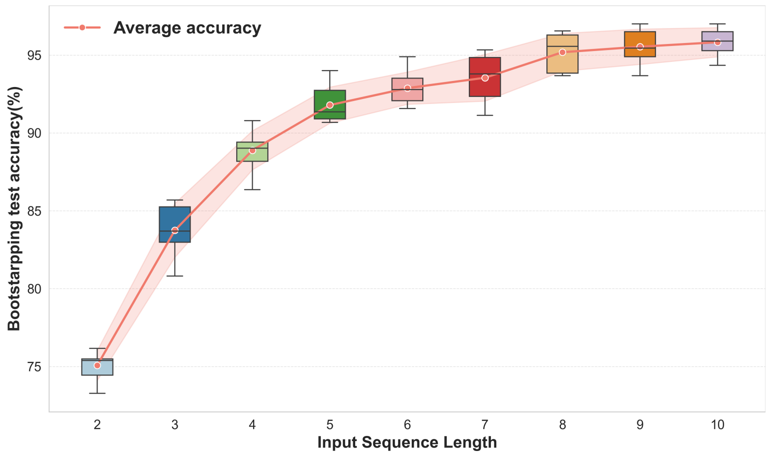


Fig. 7. The boxplot of test accuracy for different input sequence lengths, validated through bootstrapping. The pink ribbon-like curve indicates the average test accuracy and standard deviation.

Table 2. Average accuracy (%) of two validation methods.

	Method	
L_{in}	**10-fold CV**	**Bootstrap**
2	73.25	75.07
3	78.53	83.74
4	84.62	88.89
5	89.18	91.79
6	91.19	92.88
7	92.71	93.55
8	94.51	95.19
9	95.06	95.54
10	94.45	95.83

5.2 Study II: Method Comparsion

Inspired by algorithms for estimating finger angles, a natural approach is to estimate the finger angles of the initial and final frame, predicting gestures by calculating the angular displacement. We attempted to classify gestures based on the relative angular movement between these frames, comparing the performance with our model. However, since the finger angle estimation algorithm based on capacitive touchscreen [21, 28, 47] is only capable of predicting two angles (yaw, pitch) and not the roll angle, our comparison was limited to the classification performance of gestures based on yaw and pitch. To ensure fairness, a new dataset was collected for model performance comparison.

5.2.1 Dataset Development. We invited four additional volunteers not involved in the original data collection (2 males, 2 females), aged 18 to 24 years (M=20.25, SD=2.63), to participate in data collection. The apparatus and procedures were consistent with those described in Section 3. Then we collected 437 gesture sequences with a nearly equal number of each gesture.

5.2.2 Classification Based on SwivelTouch Model. Based on the model validation results presented in Section 5.1.2, we observed an improvement in model performance with increasing lengths of input sequences. However, gesture recognition requires timely responses to accurately capture user intent. Thus, balancing the model's test accuracy and the length of input sequences, we selected the model that performed best at a sequence length of $l = 6$ for subsequent comparisons.

5.2.3 Classification Based on Finger Angles. We replicated the method with the smallest estimation error from current research [21], utilizing this model to estimate the yaw and pitch of the first and last frames of a sequence. From the original dataset collected in Section 3, we extracted the first and sixth frame images from each gesture sequence, estimating the corresponding finger angles $(yaw_1, pitch_1)$ and $(yaw_2, pitch_2)$. By subtracting the estimated angles of the first frame from those of the last frame, we obtained the relative angular offsets $(\Delta yaw, \Delta pitch)$. We then illustrated the distribution of angular offsets for four gesture sequences from the original dataset: Yaw Pitch, Yaw Left, Pitch Up, and Pitch Down, as shown in Figure 8. Consequently, a straightforward approach we considered was to employ the K-Nearest Neighbors (KNN) algorithm for gesture prediction on a new dataset. We utilized the sklearn toolkits[3] to implement the KNN algorithm. Through experimental testing, we found that the algorithm performs optimally when the number of nearest neighbors is set to 3. We constructed the sample space using the original dataset. Then, for a new gesture sequence, we calculate its relative angular offsets $(\hat{\Delta} yaw, \hat{\Delta} pitch)$ from the first to the sixth frame and determine which gesture category in the sample space it most closely resembles.

5.2.4 Results and Discussion. We compared the performance of the two methods using the new dataset collected in section 5.2.1. The results are shown in Table 3. Additionally, the confusion matrix of the model testing results is presented in Figure 9. In terms of the four gestures: yaw right, yaw left, pitch up, and pitch down, our approach has resulted in a relative reduction in error rates by 53%, 84%, 86%, and 94%, respectively, compared to methods that estimate gestures based on angle offsets. Overall, this has led to a 83% decrease in the classification error rate. The relative reduction of error rates is calculated as

$$\text{Error Relative Reduction} = \frac{error_1 - error_2}{error_1} \times 100\%. \tag{5}$$

Thus, we indicate that recognizing gestures through finger angle offsets is unstable, potentially due to the following reasons:

- **Finger Angle Estimation Errors.** The finger angle prediction algorithm is prone to errors and uneven error distribution. According to the current state-of-the-art method for estimating finger angles on capacitive

[3]https://scikit-learn.org

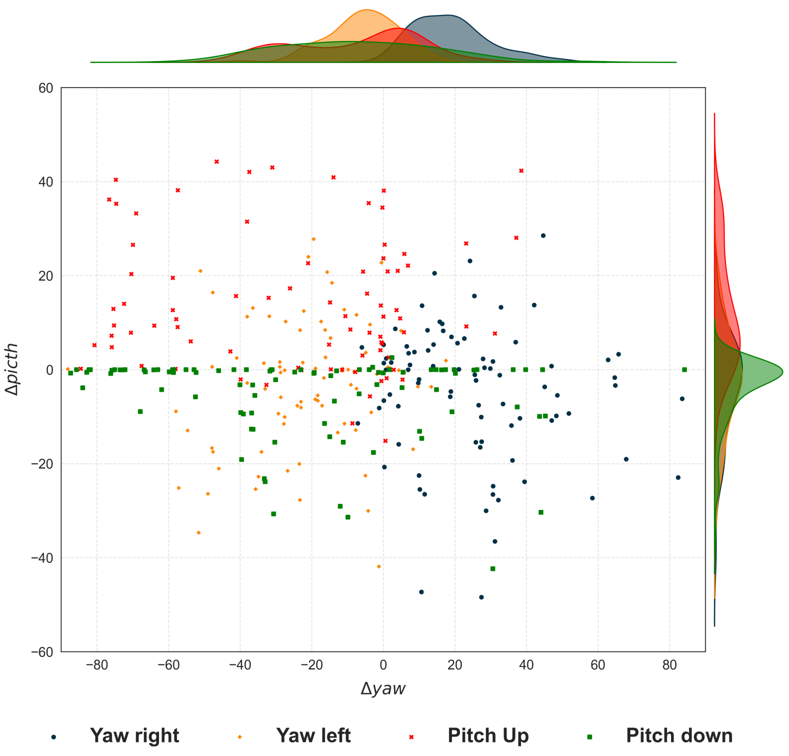


Fig. 8. Distribution of angle offsets for different types of gestures in the original dataset. Here, the angle offset refers to the difference in finger angle estimations between the first and sixth frames of a sequence, as calculated by the finger angle estimation algorithm [21].

touchscreens, the average prediction error remains at 9.1 degrees [21], with significant variance. Moreover, calculating the relative movement of finger angles requires the computation of the difference between the initial and final finger angles, resulting in a doubling of numerical errors. As illustrated in Figure 8, there is considerable overlap between the distributions of different categories, largely attributable to errors in angle estimation. Additionally, due to the uneven distribution of angle estimation errors, extreme angle predictions tend to have larger errors, and it is often inevitable for gestures to involve movements to these extreme finger angles.

- **Variations in User Behavior.** Users have different understandings and habits about gestures, typically performing them in ways they find comfortable or habitual. Our observations from the dataset identified two typical scenarios that complicate classification: one involves users gradually lifting their fingers during yaw movements, as shown in Figure 10a. This complicated classification as the pitch angle changes even when performing a gesture in yaw direction. The other scenario involves minimal finger movement, where the error margin exceeds the actual range of finger motion, as depicted in Figure 10b.

These findings further validate the efficacy of our algorithm. Our model, by examining the relationships between successive frames in a gesture sequence, captures the intent behind human gestures, thereby achieving superior gesture classification performance.

Table 3. The chart presents the average accuracies (%) for different categories of methods as well as the overall mean accuracy. Our SwivelTouch model utilizes a CNN-LSTM architecture. The technique for classifying finger angles is predicated on the size of the relative angle difference between the initial and final frames in a sequence of finger movements (see Section 5.2.3).

	Yaw		Pitch		Roll		Move				
Method	**Right**	**Left**	**Up**	**Down**	**Right**	**Left**	**Up**	**Down**	**Left**	**Right**	**Overall**
Classified by finger angle difference	57.50	27.91	66.67	60.47	-	-	-	-	-	-	52.98
Ours	80.00	88.37	95.24	97.67	83.33	80.00	97.62	100	100	100	92.04

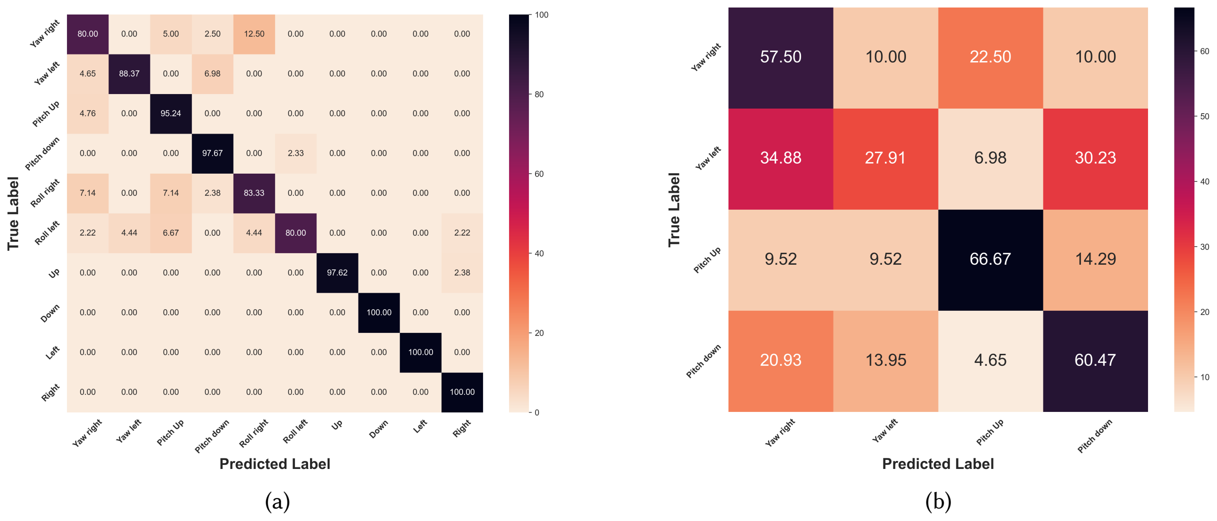


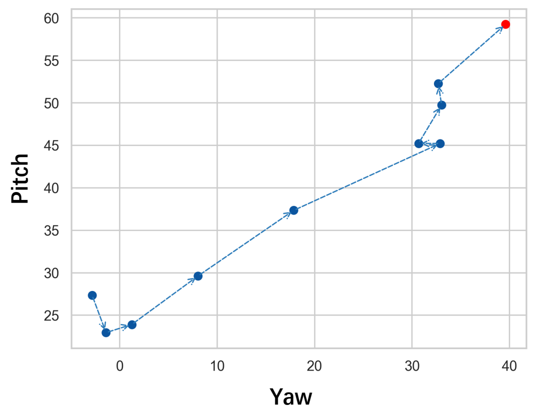
Fig. 9. Confusion Matrix for Gesture Recognition Classification. (a) represents the results of an all-category test using our model, with an average accuracy of 92.04%. (b) shows the results of testing four gestures classified by the difference in finger angles between the first and last frames, achieving an average accuracy of 52.98%.

6 IMPLEMENTATION

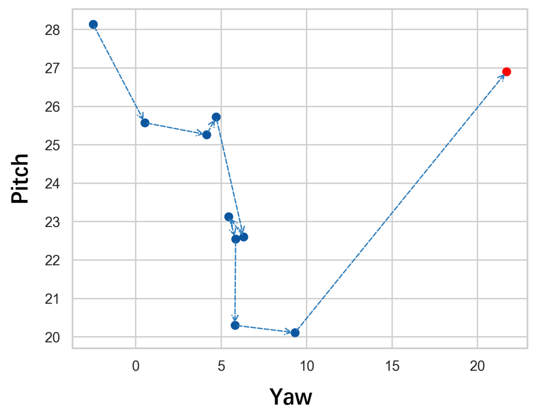
In this section, we discuss the details of deploying our model, along with strategies for simplifying it to facilitate deployment on devices with limited computational resources. Then we discuss how to integrate our gesture recognition seamlessly with standard mobile phone operations during regular usage.

6.1 Model Deployment

We will provide a detailed explanation of how our SwivelTouch model (with an input sequence of $l = 6$) is deployed on a Macbook M2Pro, a Realme C11 smartphone used for data collection (as introduced in Section 3.3, with 2GB RAM, armeabi-v7a architecture, and an 8-core CPU), and an Honor 100 smartphone released in November 2023 (with 16GB RAM, arm64-v8a architecture, and an 8-core CPU). Considering devices with limited resources, we have simplified the original CNN-LSTM network architecture and retrained a smaller model. Firstly, we reduced the number of convolutional kernels in the CNN layers to 64, 128, 128, and 64, respectively. Secondly, in the LSTM design, we shifted from a dual-layer LSTM to a single-layer structure while reducing the



(a)



(b)

Fig. 10. Examples of finger angle changes in two gesture sequences. (a) indicates an increase in pitch angle during the yaw gesture; (b) shows that in small-scale finger movements, the changes in yaw and pitch angles are not significant.

feature dimension from 100 to 10. Utilizing tools such as memory_profiler[4] and psutil[5] in Python, we recorded model memory usage and CPU usage on a MacBook Pro equipped with 16GB RAM and an M2 Pro chip. We also compared both models' sizes and FLOPs, which significantly influence computation speed. Subsequently, the accuracy of the two models was measured using 10-fold cross-validation. The related results are presented in Table 4, where we observe that although the model's accuracy slightly decreased after simplification, its size was reduced by 87%, and FLOPs decreased by 41%, facilitating shorter computation times for the model. Furthermore, to enhance computational efficiency, we converted the precision of the simplified model from the original fp32 to fp16 and compared the inference speeds of several models across different devices. For mobile devices, we employed ncnn[6] computing framework, one of the most efficient open-source frameworks on mobile phone CPUs. For the MacBook, we utilized Pytorch for model inference. The inference times are detailed in Table 5, and each value represents the mean±std time of 100 tests conducted in the same environment. The result demonstrates the feasibility of deploying our model on current commercial mobile devices.

Table 4. Comparison of Performance Between the Original and Simplified SwivelTouch Models.

	SwivelTouch	**SwivelTouch-Sim**
Size(MB)	43.85	5.64
FLOPs	3.33×10^{10}	1.97×10^{10}
Memory(MiB)	64.8	15.5
CPU usage(%)	25.32	21.45
10-fold CV Acc(%)	91.19	90.13

Table 5. Inference Time (ms) of Original and Simplified Models (FP32, FP16 Precision) Across Devices.

Device	**SwivelTouch**	**SwivelTouch-Sim**	
		FP32	**FP16**
MacBook M2Pro	22.16 ± 0.49	17.33 ± 0.42	10.18 ± 1.60
Realme C11	84.68 ± 3.09	59.65 ± 1.78	42.57 ± 1.45
Honor 100	18.87 ± 1.77	14.29 ± 2.86	10.35 ± 3.68

Although we confirmed the model's performance on smartphones, it was observed that the inference speed on the Realme phone was slower compared to that on MacBook. Additionally, the model deployed on mobile

[4]https://github.com/pythonprofilers/memory_profiler
[5]https://github.com/giampaolo/psutil
[6]https://github.com/Tencent/ncnn

phone, compressed via ncnn, exhibited a slight decrease in accuracy. Meanwhile, despite the Honor phone's faster inference speed, it lacked permission to access capacitive touch images. Therefore, in subsequent user experiments, we opted to perform inference on computers and then transfer the results back to the mobile device using Android Debug Bridge[7] (adb). This deployment strategy ensures the model's accuracy during testing. The entire process flow is illustrated in Figure 11: collecting 6 capacitive frames requires 120ms, transferring the collected data from the phone's memory to the computer program takes about 30ms, and finally, sending the results back via adb causes a delay of approximately 30ms.

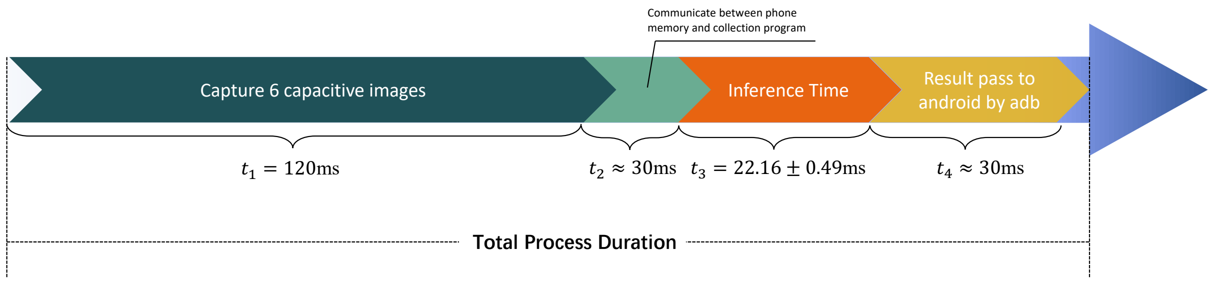

Communicate between phone memory and collection program
Capture 6 capacitive images
Inference Time
Result pass to android by adb
$t_1 = 120$ms
$t_2 \approx 30$ms
$t_3 = 22.16 \pm 0.49$ms
$t_4 \approx 30$ms
Total Process Duration

Fig. 11. Latency time of the entire SwivelTouch gesture detection process.

6.2 System Integration

The existing touchscreen interactions have become widely accepted and habitual for users. Consequently, it is essential to integrate our SwivelTouch gesture classifier smoothly with current touch input to minimize possible false classification and delay. We propose potential methods across three levels, which can be combined, for smooth integration into current touchscreen devices:

- **Level 1:** Operating System Level. Developers consider whether to support the SwivelTouch gesture interface of an app. If an active app does not support the SwivelTouch gesture, the system will not process the new gestures at all, only reporting traditional touch events. For apps that support SwivelTouch gestures, the enabled SwivelTouch API will continuously listen in the background. This needs to accommodate the delay caused by the SwivelTouch gesture recognition model.
- **Level 2:** App Level. Developers consider whether to activate the SwivelTouch gesture interface for different Activities within an app. An app typically comprises various Activities, each responsible for different interactive tasks. New gestures will only be recognized in Activities that have the SwivelTouch gesture interface activated. This approach does not introduce any delay or false recognition to other activities within the app.
- **Level 3:** Activity[8] Level within an App. Developers consider designating specific View areas within an Activity's GUI, where 3D SwivelTouch gestures are recognized exclusively within those designated View areas. Gestures made outside these specified areas will not be detected or triggered. Although this approach sacrifices a small portion of screen space, it effectively reduces the false recognition rate during normal use. Moreover, areas not designated for gesture detection do not experience delays caused by the SwivelTouch detecting algorithm. This approach is particularly suitable for apps whose certain GUI area requires fast response to traditional touch inputs.

[7]https://developer.android.com/tools/adb
[8]In Android development, it's called "Activity"; in iOS development, it's called "ViewController".

For example, if developers are interested in integrating our gestures for actions like copy and paste within text boxes, they can follow the integration design illustrated in Figure 12. Colored modules indicate they support SwivelTouch gestures, while gray modules represent no support. The SwivelTouch gesture interface should not be activated for apps within the operating system that do not require text editing (such as Clock, Camera, etc.). For activities within an app that necessitate text editing, the SwivelTouch gesture interface should be enabled. A fixed View should be established in the pop-up text box interface to facilitate text editing operations.

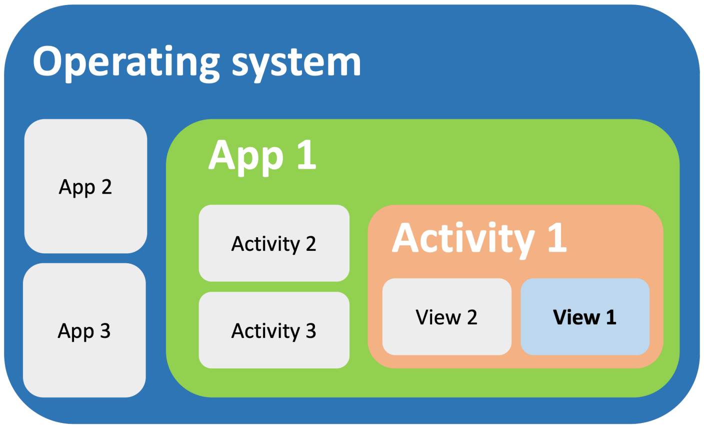


Fig. 12. Schematic diagram of system integration levels for SwivelTouch gestures.

7 USER STUDY

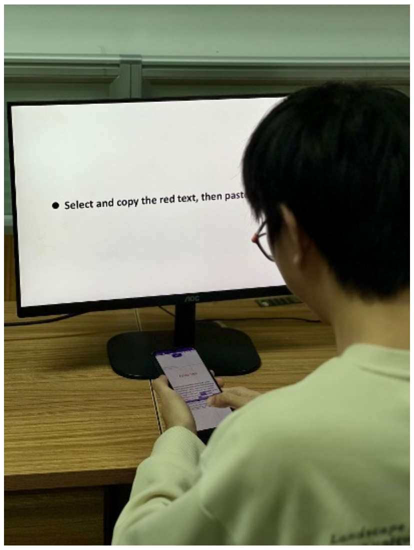
Compared to the rapid and efficient editing operations available with a mouse and keyboard, existing methods for text editing on smartphones are particularly inefficient. A significant factor contributing to this difficulty is the absence of shortcut mappings for certain text editing tasks, such as copy, paste, and select all, where users often need to double-tap or press and hold to access the text editing menu. In an attempt to enhance the efficiency of users' text editing, We conducted a user study focused on basic text editing tasks, employing gestures predefined in Table 1 as shortcuts for text editing, and compared this approach with the traditional text editing methods based on Android's default settings.

7.1 Participants

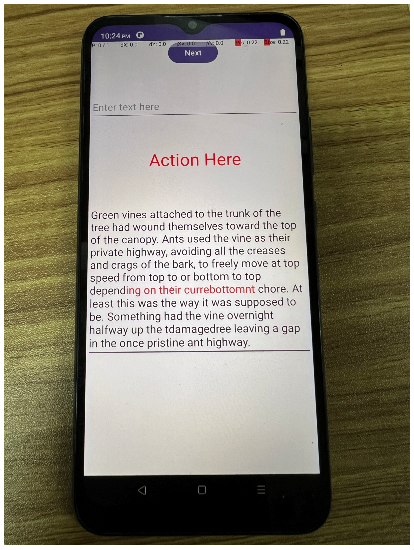
We recruited 12 participants through social media to take part in our user study, with ages ranging from 18 to 33 years old (M = 24.08, SD = 4.56), consisted of 9 males and 3 females, all of whom were experienced smartphone users (Mean years of usage = 8.58, SD = 1.93). Among these participants, three were users of the iOS operating system and exclusively utilized the default iOS keyboard. The remaining nine participants were Android users and all employed the default Android keyboard that comes with the system. All participants were right-handed.

7.2 Apparatus

In our user study, we utilized a realme C11 smartphone (165.2 × 76.4 × 8.9 mm, weight: 190g) with a display resolution of 1600× 720, running on the Android 11.0 platform. The default keyboard on this device was the Android Keyboard (AOSP). We designed a specialized application using Android Studio Giraffe, SDK 34, to display the tasks for our user study. This application also integrated the system's built-in stopwatch to record the task completion times. The layout of the application shown in Figure 13a was designed in a relative manner, consisted of a text box for pasting the cut or copied text, a display area for gesture recognition results, and a text area for manipulation tasks. Given that our gesture recognition algorithm was powerd by PyTorch, we deployed the model on a MacBook Pro integrated with 16GB RAM and an Apple M2 Pro chip. Details on the deployment can be found in Section 6.1. To avoid conflicts between gestures and Android's default text selection, we adopted the second integration strategy from Section 6.2, reserving specific areas for new gesture detection. In our user study, gesture detection was enabled everywhere except within text boxes.



(a) Experimental environment



Green vines attached to the trunk of the tree had wound themselves toward the top of the canopy. Ants used the vine as their private highway, avoiding all the creases and crags of the bark, to freely move at top speed from top to or bottom to top depending on their currebottomnt chore. At least this was the way it was supposed to be. Something had the vine overnight halfway up the tdamagedree leaving a gap in the once pristine ant highway.

(b) Application interface

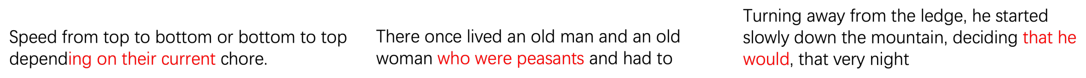
Fig. 13. The images illustrate: (a) the setting for a user study where a participant, holding a smartphone, edits text based on instructions shown on the screen in front of him; (b) the interface of an Android application designed for text editing. Please see video demo for more details.

7.3 Study Design

Our study adopted a within-groups design approach, where we utilized SwivelTouch gestures as shortcut keys for text editing, defining this as the experimental group. In contrast, the default Android input method served as the control group. We specifically examined two tasks: 1) text selection followed by copy/cut and paste actions; 2) text editing with multiple actions. The details of these tasks are outlined as follows:

$$12 \text{ participants} \times 2 \text{ models(default, SwivelTouch)} \times \begin{cases} 40 \text{ select-copy(cut)-paste tasks} = 960 \text{ tasks, in total} \\ \quad 8 \quad \text{multi action tasks} = 192 \text{ tasks, in total} \end{cases}$$

7.3.1 Gesture Shortcuts Mapping. We imitated the logic of text selection and copy-pasting with a mouse and keyboard. This process first positioning the cursor, then extending the selection either left or right, and finally employing keyboard shortcuts to execute commands like copy and paste. Consequently, we mapped the model's gestures to shortcut operations, with the specific mappings detailed in Table 6. The yaw gesture is responsible for cursor movement when not in text selection mode. The pitch-up gesture, using the left or right halves of



Speed from top to bottom or bottom to top depending on their current chore.

There once lived an old man and an old woman who were peasants and had to

Turning away from the ledge, he started slowly down the mountain, deciding that he would, that very night

(a) Middle selection (b) Complete selection (c) Cross-line selection

Fig. 14. Examples of Different Selection Modes in Task 1.

the screen, controls copy or cut operations.The pitch-down gesture is designated for pasting content from the clipboard. The roll gesture triggers the text selection mode, activating two markers: one stays at the initial cursor location, and the roll gesture guides the movement of the second marker. It continues to move until it reaches the subsequent space, thereby enabling the selection of single words by stopping there.

Table 6. The Mapping of SwivelTouch Gestures to Text Editing Function Keys.

Yaw		Pitch		Roll	
Right	Left	Up	Down	Right	Left
Move Cursor Right	Move Cursor Left	Copy/Cut	Paste	Select to Right End	Select to Left End

7.3.2 Task I: Text Selection Followed by Copy/Cut and Paste Actions. We seriously selected our experimental tasks to minimize the influence of external factors, ensuring the reliability of our results. We observed that the "fat finger" issue could obscure the target text during selection, making it challenging for users to precisely select adjacent words or characters. Hence, we aimed to explore whether using SwivelTouch gestures as shortcuts for text editing could effectively reduce the difficulty of selecting text at the word level. Our focus was on selecting 1-3 consecutive words, covering situations such as selection starting from the middle within a word, complete word selection, and cross-line word selection. Furthermore, we employed a random paragraph generator[9] to obtain five different paragraphs, extracting text that met the criteria for selection. This approach was designed to eliminate the impact of text content on the user's ability to select text, preventing familiarity with similar textual expressions. Figure 14 presents examples of the possible scenarios in text selection. Following text selection, users were required to perform copy/cut operations, and then paste the contents from the clipboard into another text box.

7.3.3 Task II: Text Editing with Multiple Actions. We observed that operations such as text selection and copy-pasting often need a long press or double tap to activate the context menu, followed by carefully searching and selecting the target option. This process will cause the user to wait for the context menu to be triggered and spend time in navigating the options. Additionally, the small size of the context menu on the screen can lead to accidental selections of incorrect options, making users feel challenging to execute a series of actions during text editing. Therefore, we sought to explore whether using gestures as shortcuts could enhance the efficiency of performing sequential operations during text editing. Our experimental design involved executing three actions (excluding text selection), which could either be three identical actions or a mix of different ones. The former scenario could consist of executing three consecutive paste operations, whereas the latter may involve a sequence of a copy-paste action followed by a cutting operation.

[9]https://randomword.com/paragraph

7.3.4 *Evaluation Metrics.*

The evaluation methods for the experiment mainly consider the following two aspects:

- **Task Completion Time (seconds).** This represents the average time taken by users to complete a single task. For Task 1, we can specifically divide the time into text selection time and action time to represent the average time spent completing these two steps respectively.
- **Error Rate.** As we require users to continue working on the current task until it is completed correctly before starting the next one, our standard for task completion accuracy is 100%. However, during the execution of tasks, we can record the number of mistakes or instances of incorrect gesture recognition as a measure of the error rate.

7.4 Procedure

We conducted individual user study for all participants in a quiet meeting room at the scheduled time, as shown in Figure 13a. Seated at a table with a display screen in front, participants were first introduced to our Swivel-Touch gesture set for about five minutes. This introduction included an explanation of the Euler angles (yaw, pitch, roll) in a spatial coordinate system, followed by a demonstration video. Subsequently, participants used a smartphone to perform gesture operations, with the predicted results of their gestures displayed in real-time on the screen for feedback. They held the phone in their preferred and comfortable manner, with some opting for one-handed use and others for a two-handed approach. We required participants to stick to their chosen method without changing it throughout the experiment. After familiarizing themselves with the various gestures, shortcut functionalities corresponding to each gesture appeared on the screen. A test text box was opened for them to practice the gesture shortcut based on on-screen instructions, with tasks presented according to the prompts on the display. Each text box shown in Figure 13b, contained text highlighted in red, indicating the words to be selected. Once they were fully accustomed to the gesture operations, we proceeded with Task 1. They were asked to carefully read the instructions and complete each task as quickly and accurately as possible. Our custom Android application displayed one piece of text at a time for the task, and after completing each task, participants clicked the 'Next' button to move to the next one. After completing all the tasks using gestures, they practiced text editing using the default Android method in a test text box to ensure a fair comparison. Once they were familiar with the default text editing method, they were asked to redo the same number of experiments in Task 1 using this method.

After completing all experiments in Task 1, participants were allowed a short break, during which we introduced the requirements for Task 2. Similarly, participants had the opportunity to familiarize themselves with both operation methods and the process in a test text box. Like Task 1, each participant completed each task according to the instructions on the display and proceeded to the next task by clicking the 'Next' button. It's important to note that all tasks used for practice were not included in the final results.

Throughout the study, we observed that participants almost invariably adhered to the established protocols. Rest periods for users were maintained between a minimum of 2 minutes and a maximum of 5 minutes to balance the need for fatigue prevention with the need to keep the operational procedures fresh in their minds. Upon completion of the entire experiment, each participant filled out a questionnaire assessing the usability and preference of the gestures on a seven-point Likert scale. In addition, they completed a NASA-TLX questionnaire for each of the two tasks, which evaluated the perceived workload under each method for the respective tasks. The entire experiment took approximately 40 minutes, and each participant received a reward of $15.

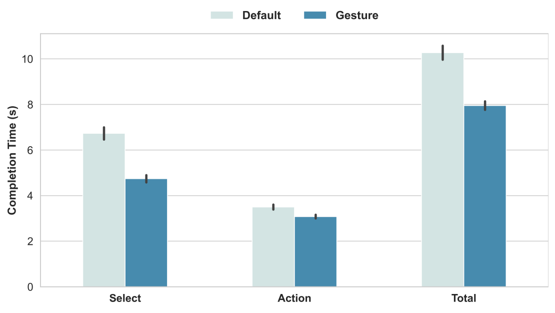
7.5 Results

We will present the results of each part of our user study, including task completion time, error rate, gesture usability and preference, as well as the results of the NASA-TLX questionnaire. Since the experiment was conducted with two different methods, we chose to use the One-way repeated measure ANOVA method to quantify intra-subject variations. We conducted a Shapiro-Wilk test to check for normality within each group, with results shown in Table 7. All p-values were greater than 0.05, indicating a normal distribution for each dataset. Additionally, we performed Mauchly's test to check the variance of paired differences, with all p-values exceeding 0.05. Therefore, in this case, we could directly analyze the results without adjustments. Lastly, we used the Wilcoxon Signed-Rank Test to analyze the questionnaire results.

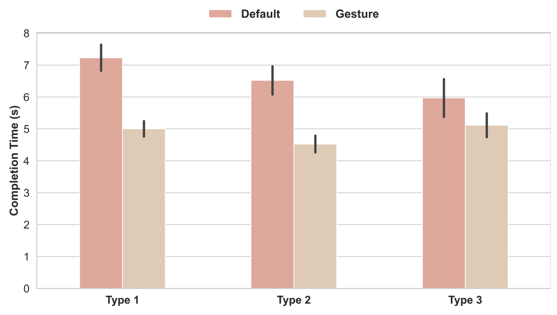
7.5.1 Task Completion Time. We assessed the completion time for both Task 1 and Task 2. For Task 1, we used an ANOVA to verify the differences in task completion time between the default method and SwivelTouch gesture. Significant differences were observed in total completion time ($F_{1,11} = 41.46, p < .0001$), action time ($F_{1,11} = 6.88, p < .05$), and selection time ($F_{1,11} = 45.64, p < .0001$). The average completion times for the default method and our model were 11.56s (SE=0.31) and 8.60s (SE=0.19), respectively, with our model showing a 25.6% reduction in time, as shown in Figure 15a. Additionally, we separately analyzed the selection times for the three types of text selections in Task 1, and Figure 15b shows that gesture operations had an advantage in text selection for each task. For Task 2, the total completion time as determined by ANOVA was ($F_{1,11} = 136.03, p < .0001$). The average completion times for the default method and SwivelTouch gesture were 7.69s (SE=0.23) and 4.21s (SE=0.12), respectively, with SwivelTouch gesture showing a 45.3% reduction in time, as depicted in Figure 16a. We also analyzed the times for the two types of tasks in Task 2, and Figure 16b illustrates that gestures had an advantage in performing consecutive shortcut operations.

Table 7. The p-values of Two Methods Under Shapiro-Wilk Test Across Different Groups.

Type	Selection		Action		Total		Errors	
	Default	Gesture	Default	Gesture	Default	Gesture	Default	Gesture
Task 1	.699	.860	.730	.640	.931	.090	.113	.449
Task 2	-	-	-	-	.447	.785	.227	.121



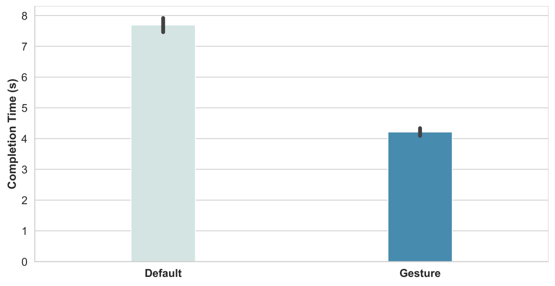
(a) Task 1 Completion Time



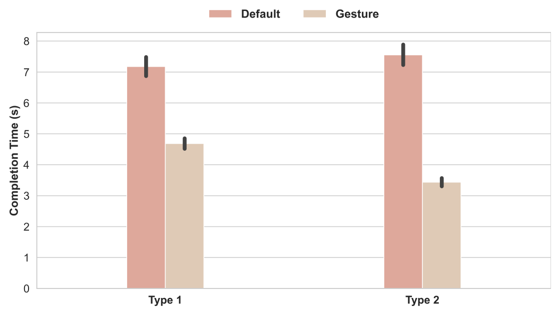
(b) Task 1 Selection Time for three types

Fig. 15. The left graph presents the average text selection time, action time, and total time for the two methods in Task 1; the right graph shows the average text selection time corresponding to three different types during text selection. Error bars represent ±1 standard error (SE).

7.5.2 Error Rate. For both Task 1 and Task 2, the ANOVA results did not show significant evidence of a difference in error rates between the methods ($F_{1,11} = 1.47, p = .251$ and $F_{1,11} = 2.02, p = .183$, respectively). Figure 17 displays the average error rates for the two tasks.

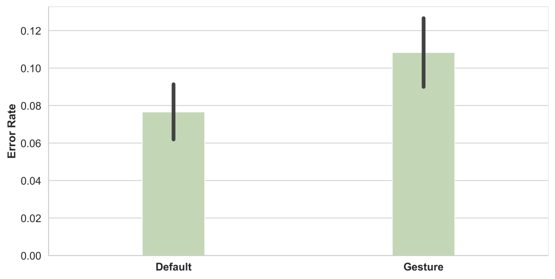


(a) Task 2 Completion Time

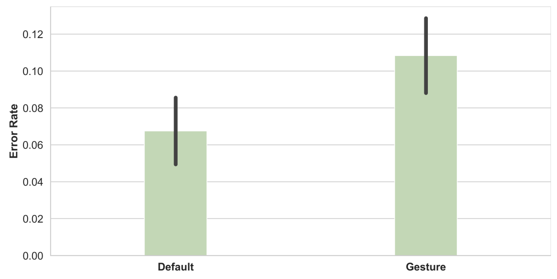


(b) Task 2 Completion Time for two types

Fig. 16. The left graph displays the average completion time for the two methods in Task 2; the right graph shows the average completion time for the two different task types within Task 2. Error bars represent ±1 standard error (SE).



(a) Task 1 Error Rate



(b) Task 2 Error Rate

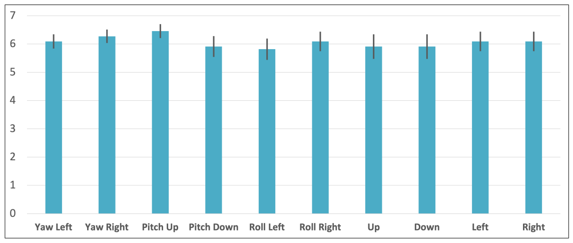
Fig. 17. The image illustrates the error rates for: (a) Task 1 and (b) Task 2. Error bars represent ±1 standard error (SE).

7.5.3 Gesture Usability and Preference. For each gesture category, we asked users to rate both its ease of use (i.e., how easy the gesture is to perform) and preference level (i.e., how much they liked using the gesture) on a seven-point Likert scale. The results are displayed in Figure 18.

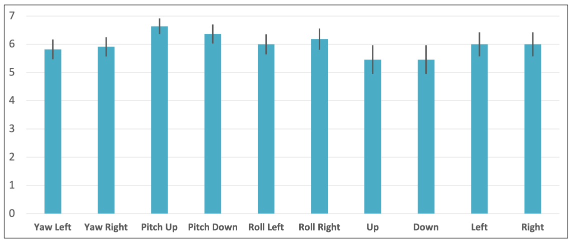
7.5.4 NASA-TLX Questionnaire. We employed the Wilcoxon Signed-Rank Test to individually examine the variables in the NASA-TLX questionnaires corresponding to Task 1 and 2. For Task 1, significant effects were observed in temporal demand ($p < .05$), performance ($p < .01$), and frustration ($p < .05$); however, there were no significant effects in mental demand ($p = .397$), physical demand ($p = .138$), or effort ($p = .278$). For Task 2, significant effects were found in physical demand ($p < .05$), temporal demand ($p < .01$), effort ($p < .05$), and frustration ($p < .05$); yet, the effects were not as pronounced in mental demand ($p = .832$) and performance ($p = .065$). Figure 19 displays the NASA-TLX questionnaire results for both tasks.

7.6 Discussion

In the text editing tasks we selected, users were significantly faster when using SwivelTouch gestures compared to the default Android input method. However, due to the limitations in the model's prediction accuracy, users inevitably made errors during operation, leading to a higher frequency of mistakes in task completion. Nonetheless, our ANOVA analysis indicated that the error rate was not significantly different from that of traditional

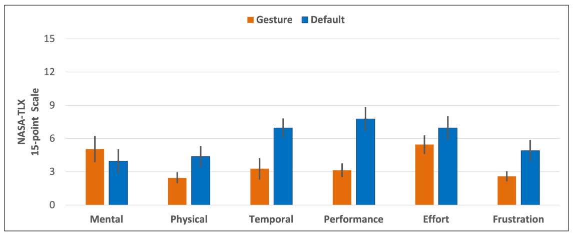


(a) Usability rating("how easy the gesture is to perform")

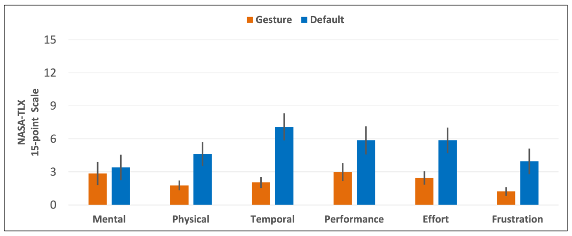


(b) Preference rating("how much they liked the gesture")

Fig. 18. Paticipants' ratings on a seven-point Likert scale for the (a) usability and (b) preference of each gesture category.



(a) Task 1 NASA-TLX



(b) Task 2 NASA-TLX

Fig. 19. NASA-TLX questionnaire results for (a) Task 1 and (b) Task 2.

input methods. The findings indicate that SwivelTouch gesture significantly enhances the efficiency of executing advanced editing tasks, surpassing the capabilities of the default Android input method.

Based on user feedback regarding the usability and preference of gestures, operations involving the Roll gesture were found to be the most challenging in SwivelTouch gestures. Additionally, traditional cursor navigation methods (up, down) were also considered less user-friendly. Users clearly preferred using our gestures for text editing tasks. We conducted a follow-up survey with participants. One participant (female, 27 years old) stated, "*After getting accustomed to the new gestures, the efficiency of copy-pasting has greatly improved*", attributing this to the traditional method requiring too many tap operations. Another participant (male, 24 years old) mentioned, "*Cross-line selection tasks can easily cause obstruction due to the movement of the cursor up and down lines*", pointing out difficulties in text selection due to the 'fat finger' problem. Most participants found the roll gesture challenging, especially in one-handed mode, where performing the gesture caused the entire thumb to rotate, potentially leading to difficulties in securely holding the phone and requiring more effort.

According to the results of the NASA-TLX questionnaire, it was again evident that using gestures led to better experimental outcomes, with users experiencing reduced time, physical demand, and effort. A participant (male, 20 years old) mentioned, "*During text selection in Task 1, I wasn't quite proficient with the shortcuts, which required more thinking time; however, the scenario in Task 2 was more straightforward, making gesture operations much simpler.*" This suggests that single-gesture shortcuts are more favored by the participants. Another participant (male, 18 years old) observed, "*Even though the gesture operation involves rotating the finger, it still feels more effortless in text selection tasks*", attributing this to the traditional method of text selection, which demands too many clicks and swipes and is more energy-consuming. Similarly, everyone agreed that the new gestures are more efficient, finding them easier to use than selecting options from a menu once they became familiar with the gestures.

Based on the observations from our user study, we recommend that users should only undertake complex tasks with SwivelTouch gesture controls after they have become proficient with the gestures and have a solid memory of the shortcut key mappings. Otherwise, unfamiliarity or forgetfulness may lead to errors. Additionally, when holding the phone with one hand, users should try to balance the phone's weight on their fingers to allow free movement of the thumb, thereby reducing the effort required for gesture operations.

8 LIMITATION AND FUTURE WORK

Our study still has several limitations. Firstly, in pursuit of optimal gesture classification performance, our network structure was designed to be complex, resulting in the model occupying a significant amount of memory when deployed on mobile devices. We found that the main time consumption was in the LSTM segment, and future work could involve compressing the model through techniques like distillation and pruning. Another issue is the delay caused by the length of the input sequence. Since our method requires the analysis of multiple frames of images (using 6 consecutive frames in the user study) to recognize gestures, a delay is inevitable, although it is much smaller compared to other gesture classification methods. In the future, leveraging touchscreens with higher frame rates, higher resolution, or fingerprint sensors could capture more gesture information, thus potentially reducing the delay in gesture recognition. Secondly, in our user data collection, we did not gather enough data from elderly users. Our analysis focused on a younger demographic and did not include a survey of middle-aged and elderly groups who might be less adept at using smartphones. Future data collection should encompass a broader range of these individuals' finger samples and verify the model's performance when used by them. Moreover, the error rate of our model in predicting gestures is still higher than the accidental touch rate during user operation, indicating a preference for stable and accurate input methods. Furthermore, our gesture system currently represents only a collection of possible finger rotation trajectories in three-dimensional space, leaving room for many potential new gestures that users might create themselves. To integrate existing gestures with user-created new ones, we plan to proceed in two ways:

1) Introduce negative categories and a significant number of negative examples. We will expand our dataset by asking users to perform actions unrelated to the established gesture sets, creating incorrect samples. Incorporating these incorrect samples into our training dataset allows the model to become robust against false positives. If the model identifies an action as a negative category, it will not respond, thereby reducing the rate of false positives.
2) Allow user customization of gesture sets through fine-tuning. We aim to construct a large-scale, diverse dataset for gestures using transfer learning. This method has been successfully utilized by Xu et al. [48] in the application of smartwatches for gesture recognition. This dataset will serve as the foundation for training a large pre-trained gesture recognition model. Once we have built a pre-trained model with good performance, we will create new branches within the network architecture. This operation is primarily aimed at creating customized gesture models for different users. Meanwhile, the whole process will require users to register by providing samples of their new created gestures for model fine-tuning and training.

In our future work, we will consider using the methods described above to to support user-created new gestures and enhance recognition accuracy. Additionally, we plan to explore other applications based on 3D SwivelTouch gesture control, such as manipulating 3D objects (see attached video demo), interacting with 3D maps, and mapping system application shortcuts.

9 CONCLUSION

Existing smartphone touch gestures are limited, resulting in fewer definable shortcuts compared to the combinations available with keyboards and mice. This makes performing complex interaction tasks, such as editing long documents on a touchscreen, more time-consuming and laborious for users. In our research, we focused on introducing a touchscreen based 3D finger rotation gesture recognition algorithm that is low-latency, highly accurate, and backward compatible with existing 2D gestures. Firstly, we proposed a method for preprocessing capacitive touchscreen gesture sequences, emphasizing the importance of detecting the first frame of a gesture for improved recognition performance. Secondly, we introduced SwivelTouch model using a CNN-LSTM structure for gesture recognition, capturing the spatiotemporal relationships between consecutive input capacitance images. Subsequently, our experiments validated that the average accuracy of gesture recognition could reach 92.04%, a 83% decrease in the classification error rate over methods based on estimating the relative finger angles of the start and end frames of a gesture sequence. Finally, we applied SwivelTouch gestures to text editing tasks, where user study showed that SwivelTouch gestures could reduce text selection time by 25.6% and save up to 45.3% in time for complex tasks like consecutive copying. User surveys also confirmed a preference for using our gestures in complex text editing tasks.

10 ACKNOWLEDGMENTS

The authors would like to thank Mr. Peng Jiang of ChipOne for the help with touch sensor. This research was supported in part by the National Natural Science Foundation of China under Grant 62376132.

REFERENCES

[1] Karan Ahuja, Paul Streli, and Christian Holz. 2021. TouchPose: Hand Pose Prediction, Depth Estimation, and Touch Classification from Capacitive Images. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 997–1009. https://doi.org/10.1145/3472749.3474801

[2] Xiaojun Bi, Yang Li, and Shumin Zhai. 2013. FFitts law: modeling finger touch with fitts' law. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 1363–1372.

[3] Tobias Boceck, Sascha Sprott, Huy Viet Le, and Sven Mayer. 2019. Force Touch Detection on Capacitive Sensors using Deep Neural Networks. In *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services* (Taipei, Taiwan) *(MobileHCI '19)*. Association for Computing Machinery, New York, NY, USA, Article 42, 6 pages. https://doi.org/10.1145/3338286.3344389

[4] Sebastian Boring, David Ledo, Xiang 'Anthony' Chen, Nicolai Marquardt, Anthony Tang, and Saul Greenberg. 2012. The fat thumb: using the thumb's contact size for single-handed mobile interaction. In *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services* (San Francisco, California, USA) *(MobileHCI '12)*. Association for Computing Machinery, New York, NY, USA, 39–48. https://doi.org/10.1145/2371574.2371582

[5] Andrew Bragdon, Eugene Nelson, Yang Li, and Ken Hinckley. 2011. Experimental analysis of touch-screen gesture designs in mobile environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 403–412. https://doi.org/10.1145/1978942.1979000

[6] Daniel Buschek, Julia Kinshofer, and Florian Alt. 2018. A Comparative Evaluation of Spatial Targeting Behaviour Patterns for Finger and Stylus Tapping on Mobile Touchscreen Devices. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 4, Article 126 (jan 2018), 21 pages. https://doi.org/10.1145/3161160

[7] Chen Chen, Simon T. Perrault, Shengdong Zhao, and Wei Tsang Ooi. 2014. BezelCopy: an efficient cross-application copy-paste technique for touchscreen smartphones. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces* (Como, Italy) *(AVI '14)*. Association for Computing Machinery, New York, NY, USA, 185–192. https://doi.org/10.1145/2598153.2598162

[8] Frederick Choi, Sven Mayer, and Chris Harrison. 2021. 3D Hand Pose Estimation on Conventional Capacitive Touchscreens. In *Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction* (Toulouse & Virtual, France) *(MobileHCI '21)*. Association for Computing Machinery, New York, NY, USA, Article 3, 13 pages. https://doi.org/10.1145/3447526.3472045

[9] Chi Tai Dang and Elisabeth André. 2011. Usage and recognition of finger orientation for multi-touch tabletop interaction. In *Human-Computer Interaction–INTERACT 2011: 13th IFIP TC 13 International Conference, Lisbon, Portugal, September 5-9, 2011, Proceedings, Part III 13*. Springer, 409–426.

[10] Mattia De Rosa, Vittorio Fuccella, Gennaro Costagliola, Maria Giovanna Albanese, Francesco Galasso, and Lorenzo Galasso. 2023. Arrow2edit: A Technique for Editing Text on Smartphones. In *International Conference on Human-Computer Interaction*. Springer, 416–432.

[11] Yongjie Duan, Ke He, Jianjiang Feng, Jiwen Lu, and Jie Zhou. 2022. Estimating 3D Finger Pose via 2D-3D Fingerprint Matching. In *27th International Conference on Intelligent User Interfaces* (Helsinki, Finland) *(IUI '22)*. Association for Computing Machinery, New York, NY, USA, 459–469. https://doi.org/10.1145/3490099.3511123

[12] Clifton Forlines, Daniel Wigdor, Chia Shen, and Ravin Balakrishnan. 2007. Direct-touch vs. mouse input for tabletop displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '07)*. Association for Computing Machinery, New York, NY, USA, 647–656. https://doi.org/10.1145/1240624.1240726

[13] Clifton Forlines, Daniel Wigdor, Chia Shen, and Ravin Balakrishnan. 2007. Direct-touch vs. mouse input for tabletop displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '07)*. Association for Computing Machinery, New York, NY, USA, 647–656. https://doi.org/10.1145/1240624.1240726

[14] Vittorio Fuccella, Poika Isokoski, and Benoit Martin. 2013. Gestures and widgets: performance in text editing on multi-touch capable mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2785–2794.

[15] Vittorio Fuccella and Benoît Martin. 2017. TouchTap: A gestural technique to edit text on multi-touch capable mobile devices. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter* (Cagliari, Italy) *(CHItaly '17)*. Association for Computing Machinery, New York, NY, USA, Article 21, 6 pages. https://doi.org/10.1145/3125571.3125579

[16] Tobias Grosse-Puppendahl, Christian Holz, Gabe Cohn, Raphael Wimmer, Oskar Bechtold, Steve Hodges, Matthew S. Reynolds, and Joshua R. Smith. 2017. Finding Common Ground: A Survey of Capacitive Sensing in Human-Computer Interaction. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 3293–3315. https://doi.org/10.1145/3025453.3025808

[17] Carl Gutwin, Carl Hofmeister, David Ledo, and Alix Goguey. 2020. Learning Multiple Mappings: an Evaluation of Interference, Transfer, and Retention with Chorded Shortcut Buttons. In *GI 2020*.

[18] Chris Harrison and Scott Hudson. 2012. Using shear as a supplemental two-dimensional input channel for rich touchscreen interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 3149–3152. https://doi.org/10.1145/2207676.2208730

[19] Chris Harrison, Julia Schwarz, and Scott E. Hudson. 2011. TapSense: enhancing finger interaction on touch surfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. Association for Computing Machinery, New York, NY, USA, 627–636. https://doi.org/10.1145/2047196.2047279

[20] Ke He, Yongjie Duan, Jianjiang Feng, and Jie Zhou. 2022. Estimating 3D Finger Angle via Fingerprint Image. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 1, Article 14 (mar 2022), 22 pages. https://doi.org/10.1145/3517243

[21] Ke He, Chentao Li, Yongjie Duan, Jianjiang Feng, and Jie Zhou. 2024. TrackPose: Towards Stable and User Adaptive Finger Pose Estimation on Capacitive Touchscreens. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 4, Article 161 (jan 2024), 22 pages. https://doi.org/10.1145/3631459

[22] Christian Holz and Patrick Baudisch. 2010. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 581–590. https://doi.org/10.1145/1753326.1753413

[23] Christian Holz and Patrick Baudisch. 2010. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 581–590. https://doi.org/10.1145/1753326.1753413

[24] Sven Kratz, Patrick Chiu, and Maribeth Back. 2013. PointPose: finger pose estimation for touch input on mobile devices using a depth sensor. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces* (St. Andrews, Scotland, United Kingdom) *(ITS '13)*. Association for Computing Machinery, New York, NY, USA, 223–230. https://doi.org/10.1145/2512349.2512824

[25] Huy Viet Le, Sven Mayer, Maximilian Weiß, Jonas Vogelsang, Henrike Weingärtner, and Niels Henze. 2020. Shortcut gestures for mobile text editing on fully touch sensitive smartphones. *ACM Transactions on Computer-Human Interaction (TOCHI)* 27, 5 (2020), 1–38.

[26] SK Lee, William Buxton, and K. C. Smith. 1985. A multi-touch three dimensional touch-sensitive tablet. *SIGCHI Bull.* 16, 4 (apr 1985), 21–25. https://doi.org/10.1145/1165385.317461

[27] Dengyun Li, Xin Ge, Qingzhou Ma, Brinda Mehra, Jie Liu, Teng Han, and Can Liu. 2022. Evaluating Three Touch Gestures for Moving Objects across Folded Screens. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 3, Article 124 (sep 2022), 28 pages. https://doi.org/10.1145/3550309

[28] Sven Mayer, Huy Viet Le, and Niels Henze. 2017. Estimating the finger orientation on capacitive touchscreens using convolutional neural networks. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*. 220–229.

[29] Ali Neshati, Bradley Rey, Ahmed Shariff Mohommed Faleel, Sandra Bardot, Celine Latulipe, and Pourang Irani. 2021. BezelGlide: Interacting with Graphs on Smartwatches with Minimal Screen Occlusion. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 501, 13 pages. https://doi.org/10.1145/3411764.3445201

[30] Ian Oakley, Carina Lindahl, Khanh Le, DoYoung Lee, and MD. Rasel Islam. 2016. The Flat Finger: Exploring Area Touches on Smartwatches. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 4238–4249. https://doi.org/10.1145/2858036.2858179

[31] Philip Quinn, Wenxin Feng, and Shumin Zhai. 2021. Deep Touch: Sensing Press Gestures from Touch Image Sequences. *Artificial Intelligence for Human Computer Interaction: A Modern Approach* (2021), 169–192.

[32] Gulnar Rakhmetulla, Yuan Ren, and Ahmed Sabbir Arif. 2023. GeShort: One-Handed Mobile Text Editing and Formatting with Gestural Shortcuts and a Floating Clipboard. *Proceedings of the ACM on Human-Computer Interaction* 7, MHCI (2023), 1–23.

[33] Gonzalo Ramos, Matthew Boulos, and Ravin Balakrishnan. 2004. Pressure widgets. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 487–494.

[34] Radiah Rivu, Yasmeen Abdrabou, Ken Pfeuffer, Mariam Hassib, and Florian Alt. 2020. Gaze'N'Touch: Enhancing Text Selection on Mobile Devices Using Gaze. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3334480.3382802

[35] Simon Rogers, John Williamson, Craig Stewart, and Roderick Murray-Smith. 2011. AnglePose: robust, precise capacitive touch tracking via 3d orientation estimation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 2575–2584. https://doi.org/10.1145/1978942.1979318

[36] Volker Roth and Thea Turner. 2009. Bezel swipe: conflict-free scrolling and multiple selection on mobile touch screen devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1523–1526.

[37] Anne Roudaut, Eric Lecolinet, and Yves Guiard. 2009. MicroRolls: expanding touch-screen input vocabulary by distinguishing rolls vs. slides of the thumb. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 927–936. https://doi.org/10.1145/1518701.1518843

[38] Robin Schweigert, Jan Leusmann, Simon Hagenmayer, Maximilian Weiß, Huy Viet Le, Sven Mayer, and Andreas Bulling. 2019. KnuckleTouch: Enabling Knuckle Gestures on Capacitive Touchscreens using Deep Learning. In *Proceedings of Mensch Und Computer 2019* (Hamburg, Germany) *(MuC '19)*. Association for Computing Machinery, New York, NY, USA, 387–397. https://doi.org/10.1145/3340764.3340767

[39] A Smith. 2015. Chapter Three: A „week in the life analysis of smartphone users. *URL: http://www. pewinternet. org/2015/04/01/chapter-three-a-week-in-the-lifeanalysis-of-smartphone-users* (2015).

[40] Jamie Ullerich, Maximiliane Windl, Andreas Bulling, and Sven Mayer. 2023. ThumbPitch: Enriching Thumb Interaction on Mobile Touchscreens using Deep Learning. In *Proceedings of the 34th Australian Conference on Human-Computer Interaction* (Canberra, ACT, Australia) *(OzCHI '22)*. Association for Computing Machinery, New York, NY, USA, 58–66. https://doi.org/10.1145/3572921.3572925

[41] Craig Villamor, Dan Willis, and Luke Wroblewski. 2010. Touch gesture reference guide. *Touch Gesture Reference Guide* (2010).

[42] Daniel Vogel and Patrick Baudisch. 2007. Shift: a technique for operating pen-based interfaces using touch. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 657–666.

[43] Jonas Vogelsang, Francisco Kiss, and Sven Mayer. 2021. A Design Space for User Interface Elements using Finger Orientation Input. In *Proceedings of Mensch Und Computer 2021* (Ingolstadt, Germany) *(MuC '21)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3473856.3473862

[44] Feng Wang, Xiang Cao, Xiangshi Ren, and Pourang Irani. 2009. Detecting and leveraging finger orientation for interaction with direct-touch surfaces. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (Victoria, BC, Canada) *(UIST '09)*. Association for Computing Machinery, New York, NY, USA, 23–32. https://doi.org/10.1145/1622176.1622182

[45] Yoichi Watanabe, Yasutoshi Makino, Katsunari Sato, and Takashi Maeno. 2012. Contact force and finger angles estimation for touch panel by detecting transmitted light on fingernail. In *Haptics: Perception, Devices, Mobility, and Communication: International Conference, EuroHaptics 2012, Tampere, Finland, June 13-15, 2012. Proceedings, Part I*. Springer, 601–612.

[46] Pui Chung Wong, Kening Zhu, Xing-Dong Yang, and Hongbo Fu. 2020. Exploring Eyes-free Bezel-initiated Swipe on Round Smartwatches. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3313831.3376393

[47] Robert Xiao, Julia Schwarz, and Chris Harrison. 2015. Estimating 3D Finger Angle on Commodity Touchscreens. In *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces* (Madeira, Portugal) *(ITS '15)*. Association for Computing Machinery, New York, NY, USA, 47–50. https://doi.org/10.1145/2817721.2817737

[48] Xuhai Xu, Jun Gong, Carolina Brum, Lilian Liang, Bongsoo Suh, Shivam Kumar Gupta, Yash Agarwal, Laurence Lindsey, Runchang Kang, Behrooz Shahsavari, Tu Nguyen, Heriberto Nieto, Scott E Hudson, Charlie Maalouf, Jax Seyed Mousavi, and Gierad Laput. 2022. Enabling Hand Gesture Customization on Wrist-Worn Devices. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 496, 19 pages. https://doi.org/10.1145/3491102.3501904

[49] Vadim Zaliva. 2012. 3D finger posture detection and gesture recognition on touch surfaces. In *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*. 359–364. https://doi.org/10.1109/ICARCV.2012.6485185

[50] Mingrui Zhang and Jacob O Wobbrock. 2019. Gedit: Keyboard gestures for mobile text editing. In *Graphics Interface 2020*.

[51] Maozheng Zhao, Wenzhe Cui, IV Ramakrishnan, Shumin Zhai, and Xiaojun Bi. 2021. Voice and Touch Based Error-tolerant Multimodal Text Editing and Correction for Smartphones. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 162–178. https://doi.org/10.1145/3472749.3474742

[52] Maozheng Zhao, Henry Huang, Zhi Li, Rui Liu, Wenzhe Cui, Kajal Toshniwal, Ananya Goel, Andrew Wang, Xia Zhao, Sina Rashidian, Furqan Baig, Khiem Phi, Shumin Zhai, IV Ramakrishnan, Fusheng Wang, and Xiaojun Bi. 2022. EyeSayCorrect: Eye Gaze and Voice

Based Hands-free Text Correction for Mobile Devices. In *27th International Conference on Intelligent User Interfaces* (Helsinki, Finland) *(IUI '22)*. Association for Computing Machinery, New York, NY, USA, 470–482. https://doi.org/10.1145/3490099.3511103