Gesture Spotter: A Rapid Prototyping Tool for Key Gesture Spotting in Virtual and Augmented Reality Applications

Junxiao Shen, John Dudley, George Mo, and Per Ola Kristensson

Abstract—In this paper we examine the task of key gesture spotting: accurate and timely online recognition of hand gestures. We specifically seek to address two key challenges faced by developers when integrating key gesture spotting functionality into their applications. These are: i) achieving high accuracy and zero or negative activation lag with single-time activation; and ii) avoiding the requirement for deep domain expertise in machine learning. We address the first challenge by proposing a key gesture spotting architecture consisting of a novel gesture classifier model and a novel single-time activation algorithm. This key gesture spotting architecture was evaluated on four separate hand skeleton gesture datasets, and achieved high recognition accuracy with early detection. We address the second challenge by encapsulating different data processing and augmentation strategies, as well as the proposed key gesture spotting architecture, into a graphical user interface and an application programming interface. Two user studies demonstrate that developers are able to efficiently construct custom recognizers using both the graphical user interface and the application programming interface.

Index Terms—Gesture interaction, prototyping tools, machine learning, augmented reality, virtual reality

1 Introduction

A critical component for developing compelling interactive Virtual Reality (VR) and Augmented Reality (AR) applications not reliant on controllers is an online system for real-time recognition of hand gestures. In this paper, we explore a sub-class of online gesture recognition which we refer to as *key gesture spotting* (KGS). KGS is online recognition with the additional requirement that only a single-time activation should be triggered upon recognition of a new gesture. In VR and AR, KGS involves detecting key hand gestures within a continuous flow of data that also contains many incidental gesticulations (e.g., pointing, swiping, grasping, etc.) that occur as part of normal interface interaction tasks. Robustly and rapidly recognizing intended gestures while reliably ignoring these incidental gesticulations is what makes KGS particularly challenging.

Our goal in this work is to streamline the process of prototyping KGS functionality for deployment into VR and AR applications. Developers of these types of applications may want to design their own custom gestures and implement a gesture recognizer for these novel gestures during the early development phase. For example, a VR game developer may wish to design several gestures to activate various skills, but at this early stage in development wants to quickly explore which gestures will provide the right user experience and support reliable recognition. To do this currently, the developer would require a certain degree of expertise and need to spend significant time building, training, and implementing an effective and efficient deep learning-based gesture recognizer. This current state motivates our goal of providing a simple to use tool that developers and researchers can use to create a robust real-time gesture recognizer. We share the same philosophy espoused in the $-family [3,46,48], which is to "take what are typically complex, arcane technologies understood only by specialists and make them easy to convey, implement, and deploy on any platform for non-specialists whose objective is quickly enhancing interactivity" [1].

Different modalities of hand gesture data can be used as input for recognition, including RGB, optical flow, depth, IR, IR-disparity and 2D/3D skeletons [33]. Modern AR and VR head mounted devices (HMDs) track the user's hands and provide real-time access to 3D hand skeleton data. In this paper, we focus on 3D hand skeleton data as the input for KGS given that it is readily available in modern HMDs as well as the fact that it is largely sensor/device agnostic. Furthermore, skeleton-based recognition algorithms tend to have fewer parameters, resulting in faster real-time recognition performance which is especially pertinent for mobile computing device applications.

Delivering on this vision of efficient and simple to deploy KGS functionality, however, requires careful navigation of the various requirements briefly identified above and summarized below:

1. **Ignore incidental gestures:** The incidental gesticulations that occur naturally as part of interacting in VR and AR must not produce false activations of key gestures.

2. **Single-time activation:** Only a single activation should be triggered when a key gesture is performed (i.e. during the whole period of the key gesture).

3. **Early detection:** A key gesture must be spotted as soon as possible. Ideally the gesture is recognized before it is completed to minimize latency in triggering gesture-driven events. Such a recognition with zero or negative lag is called *early detection*.

4. **Deployability:** The gesture recognizer should be practically sized and efficient in terms of memory and power, and the data input should not require onerous online preprocessing.

5. **Low data overhead:** The KGS implementation needs to be data efficient as hand gesture data in new applications is usually limited in volume. This is particularly true when developers wish to evaluate a set of completely novel key gestures and can only collect data by themselves.

6. **Facilitate feature selection:** Data preprocessing is crucial in training a stable gesture recognizer as hand skeleton data may include in excess of 20 joints per hand to choose from. Each joint has both a position and an orientation which could be included in the model input. A tool for building KGS functionality must therefore facilitate the process of experimenting with different model input alternatives.

7. **Support rapid evaluation:** Although some gestures may be visually distinct from others, the latent representations may be very similar. This can result in a trained deep learning model confusing

- *Junxiao Shen is with University of Cambridge. E-mail: js2283@cam.ac.uk.*
- *John Dudley is with University of Cambridge. E-mail: jjd50@cam.ac.uk.*
- *Geroge Mo is with University of Cambridge. E-mail: gm621@cam.ac.uk.*
- *Per Ola Kristensson is with University of Cambridge. E-mail: pok21@cam.ac.uk.*

Manuscript received 11 March 2022; revised 11 June 2022; accepted 2 July 2022. Date of publication 01 September 2022; date of current version 03 October 2022. Digital Object Identifier no. 10.1109/TVCG.2022.3203004

different gestures. Therefore, a rapid prototyping tool for KGS must provide a way to perform fast training and evaluation of a model to help developers better design distinct gestures.

Most prior research in hand gesture recognition has focused on improving the offline recognition accuracy of gestures with sufficient training data, and many of the real-world integration and deployment challenges described above are not fully addressed [21]. In addressing this gap, we approach the KGS problem by proposing a novel architecture that includes a light-weight classifier model and a single-time activation algorithm. This enables rapid prototyping of online gesture recognition functionality with small amounts of data. Figure 1 demonstrates that the proposed KGS architecture can deliver reliable single-time activations and early detections when applied on the Online DHG dataset [10]. To ensure this functionality is easily adapted and deployed, we encapsulate the proposed KGS architecture into a Graphical User Interface (GUI) and a simple Application Programming Interface (API) for exercising key supporting functionality, including a variety of data preprocessing and augmentation techniques. These tools enable rapid prototyping of KGS functionality on custom gesture sets. Developers can then evaluate different configurations, explore alternative parameter settings, and confirm that the key gestures are distinguishable both from a visual perspective as well as from a model learning perspective in the latent space. This helps bridge the gap between the domain-specific knowledge required to implement such deep learning-based recognizers and the usability required for developers that want to simply create, use, and deploy these recognizers for their desired applications.

To demonstrate the effectiveness of our KGS architecture we present evaluations on four datasets: i) the SHREC'17 Track dataset [10], which is a public dataset that contains a 14-gestures setting and a 28-gestures setting, ii) the Online DHG dataset, which is a pre-segmented version of the SHREC'17 Track dataset [10], is used to simulate an online recognition scenario; iii) an AR-specific dataset from Mo et al. [32], which is a first-person-view dataset collected with a Microsoft HoloLens 2, and which we subsequently refer to as the Gesture Knitter dataset; and iv) the SHREC'2021 Track dataset [6] in which gestures were captured during generic user interaction and so executed gestures occur within long sequences of hand gesticulation.

To demonstrate the usability of the GUI and API we present the results of two user studies, which demonstrate that participants were able to efficiently construct custom recognizers using both the GUI and the API.

Overall, in this paper, we make the following contributions:

1. We propose a key gesture spotting architecture consisting of a novel hand gesture classifier, which we call Attention-Enhanced Residual LSTM (A-ResLSTM), and a novel single-time activation algorithm. We demonstrate that A-ResLSTM delivers robust and data-efficient key gesture spotting while the single-time activation algorithm achieves an effective balance between online recognition accuracy and early detection.

2. We present *Gesture Spotter*: a tool implemented as a GUI and API that enables rapid prototyping of KGS functionality. This tool integrates different data preprocessing and augmentation techniques and facilitates model training and evaluation.

The remainder of the paper is organized as follows. Section 2 discusses the related work of offline gesture recognition, online gesture recognition, and developer tools for buidling such recognition systems. Section 3 presents the structure of the novel classifier, A-ResLSTM, and the single-time activation algorithm. Section 4 describes the details of our implementation and evaluation methods and metrics. Section 5 reports both offline and online recognition performance for the SHREC'17 Track dataset, Online DHG dataset, Gesture Knitter dataset and SHREC'2021 Track dataset. Section 6 presents a user evaluation of the graphical interface for rapid prototyping of a KGS system. Section 7 presents a user evaluation of the Python API for developing a bespoke KGS system. Section 8 discusses limitations and future work and Section 9 provides concluding remarks.

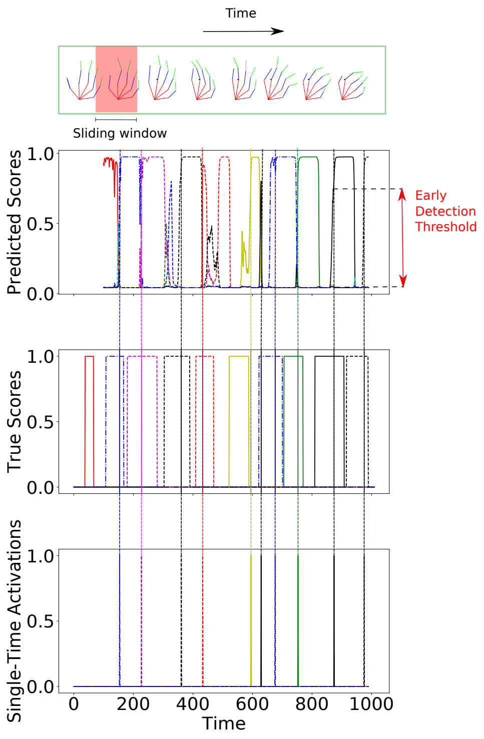


Fig. 1: The KGS architecture recognizing and acitvating on distinct gestures in the DHG dataset [10]. The *Predicted Scores* show the normalized scores of the predictions from the key gesture spotting model. The *True Scores* show the true labels of each gesture performed along the time axis. Each square wave represents a class of gesture and its duration. The *Single-Time Activations* show the activations fired by our model when the predicted score passes the threshold. The light vertical line shows the corresponding activations and the early detections. In this example, the detection threshold is set to 0.8.

2 Related Work

Hand and gesture-based interactions can provide a natural and immersive interaction experience in AR and VR. There are, however, significant technical and usability challenges to delivering compelling hand-based interactions that continue to attract research attention [14, 51]. Predating the hand tracking capabilities of modern AR and VR HMDs, WeARHand [14] allowed users to manipulate virtual 3D objects with their bare hand in AR. Yoon et al. [51] investigated the effect of hand model fidelity on user experience and social presence during hand-based 3D remote collaboration.

A key requirement for delivering smooth gesture interactions in these types of AR and VR user interfaces is robust and rapid gesture recognition. In this section, we briefly discuss the related work on offline and online gesture recognition before providing a brief overview of gesture-focused developer tools.

2.1 Offline Gesture Recognition

Much prior work has focused on gesture classification performed offline, i.e., where the gesture is recognized from within a deliberately segmented portion of data as opposed to a continuous stream. Gesture recognition algorithms employed for this purpose can be categorized into five main methods: 1) hand-crafted methods; 2) Convolutional Neural Network (CNN)-based methods; 3) graph-based methods; 4) manifold-learning-based methods; and 5) Recurrent Neural Network (RNN)-based methods. Hand-crafted methods, such as HON4D [36], use a histogram capturing the 4D representation distribution of the surface normal orientation (time, depth and spatial coordinates) to recognize gestures from depth sequences. Deep learning methods using CNNs and RNNs have become more popular with the success of deep learning. Further, graph and manifold learning has also had success in gesture recognition recently, such as DG-STA [8] and ST-TS-HGR-NET [34]. With regards to applications in AR and VR, there has been an interest in gesture recognition in terms of 3D hand pose estimation with an infrared camera [37] and fast recognition of foot gestures for virtual locomotion [42].

However, most of these models are not suitable for a real-time gesture recognition system in AR/VR for three key reasons. First, the models are large and cannot run efficiently on local devices, such as the Microsoft HoloLens 2 or Oculus Quest 2. Second, some of these models require heavy preprocessing, such as converting depth images into point data [31], which makes real-time performance difficult to achieve. Third, very few of these methods are designed to perform classification on hand skeleton data.

2.2 Online Gesture Recognition

There is limited prior literature on modern online hand gesture recognition suitable for AR/VR headsets. Hegde et al. [16] proposed recognition of simple hand gestures (up/down swipes etc.) using feature points. Chalasani et al. [7] generated augmented data by using a green screen to capture hand gestures and then overlaid these on different backgrounds. Xie et al. [49] performed simultaneous detection of hands and tracking of keypoints to support gesture recognition while Wang et al. [47] examined online gesture recognition for interaction with a vehicle HUD. However, all these prior studies use image data. In contrast, in this work we rely on hand skeleton data. Gunduz et al. [21] separated the online gesture recognition problem into two stages: 1) detection; and 2) classification. They first explicitly trained a light-weight CNN-based detector to detect whether a gesture is present in a continuous sliding window. This acts as a switch to activate another CNN-based classifier to classify the *gesture* classes while ignoring the *no gesture* class representing when the hand is out-of-view. Notably, the detector is light-weight, allowing it to run in real-time. This approach means that the second classifier has less stringent constraints on size or complexity since it only operates when a *gesture* is detected by the first classifier. However, such a method requires the training and implementation of two different models, and this can potentially complicate the system. Molchanov et al. [33] addressed online gesture recognition by applying connectionist temporal classification (CTC), which is a training technique for recognizing unsegmented handwriting [20]. The recognition model is a recurrent 3D CNN that performs detection and classification simultaneously. This approach introduces another issue: it does not implement a single-time activation algorithm to prevent multiple activations for one key gesture, which may lead to a high false-positive rate and be impractical for real-world applications.

Both Gunduz et al. [21] and Molchanov et al. [33] evaluated their proposed methods on the NVIDIA Dynamic Hand Gesture Dataset which was collected under a very different setting to one typically experienced in an AR/VR use case. Specifically, this is a dataset that contains data of different modalities such as RGB, optical flow, depth, IR, and IR-disparity but it does not contain any hand skeleton data. Further, the dataset was collected in an automotive interaction scenario, and provides a clear indication of when there is no gesture performed, as the hand is outside the tracking area. Hence, these systems are not optimized for scenarios in AR/VR where the hand is mostly present in the tracking area and the two-stage “first detect and then classfy” approach may not be suitable for key gesture spotting. We are interested in spotting key gestures from a continuous stream of different gestures with the hand constantly present in the sensor’s field-of-view, and where sporadic key gestures are performed interspersed between much more frequent incidental gesticulations, such as pointing, grasping or swiping. This is a much more difficult task but we believe this is critical for successful gesture spotting in AR/VR applications. Caputo et al. [6] introduced a hand skeleton gesture dataset collected to represent a head-mounted AR/VR context. However, the dataset contains key gestures interleaved between periods of time during which the hand is predominantly stationary. This lack of incidental gesticulations in between key gestures also fails to accurately reflect typical interaction scenarios in an AR/VR setting. In practice, users are likely to continually interact with both the physical and virtual environment while only sporadically performing key gestures. Detecting key gestures when the hand is otherwise not moving is a much simpler problem than detecting key gestures while the hand is actively engaged in other interaction tasks.

2.3 Developer Tools

With respect to systems that allow users to construct gesture recognizers, there have been many systems proposed to make it easier for users to create recognizers with low data overhead for rapid prototyping. The $-family of gesture recognizers [3, 46, 48] addresses the need for quick integration of recognizers for simple unistroke and multistroke 2D gestures. At the conceptual level, there are similarities between our work and the objectives of the $-family of gesture recognizers in terms of prototyping ease. To aid in prototyping, many systems propose methods to synthetically generate gesture samples, such as employing the kinetic theory of rapid movements to create synthetic human-like 2D stroke gestures in Gestures à Go Go [23] and re-sampling and perturbation methods to rapidly create synthetic stroke gesture samples [45]. Other tools that allow developers to automatically create recognizers by demonstration include Gesture Coder for multi-touch gestures [29] and Gesture Script for 2D stroke gestures [28]. Recently, in the domain of hand gesture recognition for immersive headset applications, Gesture Knitter was proposed to allow the creation of novel hand gesture recognizers with low data overhead and novel synthetic sample generation methods using a visual declarative script [32]. However, the online recognition accuracy was relatively poor in Gesture Knitter at 72.5%. This lack of suitable tools motivates our focus on tackling the problem of online hand gesture recognition for AR/VR HMDs by providing functionality to developers allowing them to easily create neural network-based recognizers with good performance accuracy.

3 Key Gesture Spotting Recognizer

There are no existing design tools involving neural network architectures available to developers for key gesture spotting (KGS) on skeleton data in AR/VR. Therefore, we propose, to the best of our knowledge, the first design tool specifically tailored for KGS on skeleton data in AR/VR. Rapid prototyping of hand gesture recognizers for AR/VR is more suitable to be performed on hand skeleton data instead of raw RGB video or depth signals. Video-based real-time hand gesture recognition models do exist but they are significantly more complicated to integrate into modern AR/VR HMDs, which provide streamlined access to hand tracking data.

The key gesture spotting model must provide early detection and single-time activation for each gesture performed. We did not separate the detector and the classifier for several key reasons: i) detection and then classification leads to longer response times due to a two-stage system; ii) the window of data which activates the detector may not necessarily contain the whole gesture performed; and iii) there is no clear *no gesture* class in a continuous stream of hand gestures performed with the hand continuously present in the field-of-view.

To aid in assessing the performance of our KGS system, we first outline the main components of a dynamic hand gesture. Pavlovic et al. [38] describe how dynamic gestures have a preparation phase, nucleus phase, and retraction phase, among which the nucleus phase is the most discriminate. Each of these three phases is now briefly defined below. **Preparation phase**: This is the hand movement when the user starts to move the hand from the rest position to a ready position. Most gestures will have a similar preparation phase. **Nucleus phase**: This is the most critical part of a hand gesture since it is the most distinctive phase. It is the phase where the most distinguishable spatial and temporal movement is performed. Recognition early in the nucleus phase enables early detection, leading to negative or zero lag. **Retraction phase**: This is the post-execution phase containing the motion corresponding to the user returning their hand to a rest position.

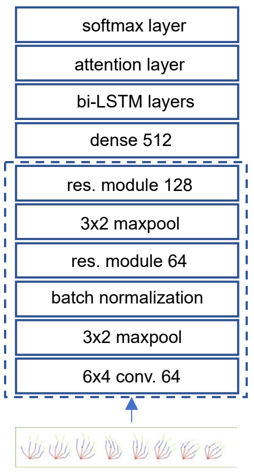


Fig. 2: Figure showing the neural network architecture of A-ResLSTM. *3×2 maxpool* is a max pooling layer which is added to downsample the input along its spatial dimensions by taking the maximum value over an input window of size 3×2. The *batch normalization* was introduced to standardize the inputs to the layer. The *dense 512* layer is a regular densely-connected neural network layer of 512 neurons. The *bi-LSTM layer* is composed of two bi-directional LSTM layers with 256 neurons on each layer. The *attention layers* implement a dot-product attention layer, a.k.a. Luong-style attention [27]. The *softmax layer* is used to normalize the output from the attention layer into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers, where K is the number of gesture classes in the classifier.

3.1 Gesture Classifier

Deep neural networks are especially well-suited for our domain as the network depth increases the expressiveness of the recognizer. A gesture can be decoupled into hand posture variations and hand movements. The former reflects the spatial layout changes of hand joints, and the latter captures the global motion trajectories [25]. Intuitively, the difference between the two representations motivates us to develop a model that can inspect both the spatial and temporal features simultaneously. Convolutional Neural Networks (CNNs) have had groundbreaking results in image processing [9, 24]. CNNs use local receptive fields to efficiently extract spatial information and share weights to significantly reduce the number of parameters. Recurrent neural networks (RNNs) have gained success with sequential data of variable lengths, with applications including language modeling [30], video captioning [13], RGB-based and skeleton-based activity recognition [26]. However, when the gap between the relevant input data is large, conventional RNNs can no longer connect the relevant information. LSTMs are then proposed to handle such "long-term dependency" [18, 31, 39]. Therefore, we utilize a mixture of CNN and LSTM components in our model since skeleton gesture data is a data representation that consists of both spatial and temporal features, with the posture variants featuring more spatial information and the hand movements reflecting more temporal information.

Figure 2 shows the detailed structure of the A-ResLSTM model we propose. This architecture design is inspired by that introduced by He et al. [15], where the residual module blocks with the skip connections are employed to combat the vanishing and exploding gradient problem. The original domain application of He et al.'s [15] architecture was image recognition, and this problem has some similar qualities to our domain of skeleton-based gesture recognition, such as the need for invariance to translation. Below we detail features of the A-ResLSTM model that are bespoke to our application.

First, key gesture spotting is used in a continuous flow of data which contains incidental gesticulations or what we subsequently refer to as *null gestures*. When a sliding window approach is used, a window may only partially contain a key gesture, and the rest in the window consists of *null gestures*. Therefore, enabling the gesture recognizer to pay different 'attention' to the preparation, nucleus and retraction phases is critical in early detection. We use an attention mechanism to empower the recognizer with such an ability to learn weight coefficients that capture the relationship between time points in input data samples after the CNN and LSTM layers as shown in Figure 2.

Many researchers currently solving real-world problems using deep neural networks have discovered that problems tend to arise when a large number of hidden layers are used. During training, updates in weights of a previous layer change the distribution of input values for the current layer. In this process the network learns more complex features. However, deep neural networks do have limitations as He et al. [15] empirically demonstrated in the context of traditional CNNs and the existence of practical constraints on the maximum of number layers. The failure of very deep neural networks is a result of the network initialization and the vanishing or exploding gradient problem [15]. To solve this issue, training deep networks has been alleviated with the introduction of a new neural network layer—the residual block. The residual block provides skip connections between layers and adds the outputs from previous layers to the outputs of stacked layers. The skip connection allows the gradient of the higher layer to be directly passed to the lower layers during backpropagation, mitigating the vanishing gradient or exploding gradient problem [15]. We thus added residual blocks into the CNN layer.

3.2 Single-Time Activation Algorithm

The single-time activation algorithm for key gesture spotting outputs the predicted gesture class and the associated probability. We first set the window length l to feed to the classifier model, as well as the slide step size s of the sliding window. When a particular window is passed to the classifier model, it returns the predicted gesture class and its corresponding detection probability.

Two thresholds determine if a gesture is spotted: the detection probability threshold and the recurrence threshold. For a particular gesture class to be deemed to be spotted the following must be true. First, the gesture's detection probability must be greater than the detection probability threshold for gesture detection. Second, the number of times the same gesture has been detected in a row (its recurrence) must be above the recurrence threshold.

Both thresholds are important in determining the trade-off between recognition accuracy and early detection performance as setting low threshold values could lead to better early detection performance but lower accuracy, whereas higher threshold values could lead to the reverse. The choice of an appropriate recurrence threshold is influenced by the slide step size s and the frame rate of the device. Developers can set the recurrence threshold by determining the minimum time required to complete a gesture and the refresh rate of the tracking sensor.

4 Experiments

4.1 Implementation Details

We used TensorFlow 2 in the implementation of the classifier. For the classifier, we used Adam as the optimizer, and the learning rate was set to 0.0001. The learning rate scheduler was set to be reduced on plateau and we set the patience to 3. The learning rate is chosen by performing grid search to optimize for the model's performance. We used early stopping and we set the patience to 5. Patience is the number of epochs to wait before reducing the learning rate, or an early stop when no progress is made on the validation set. The patience value for early stopping is set using an established rule-of-thumb. Varying the patience does not play an important role in determining the model performance. We used sparse categorical accuracy to evaluate the performance of the model. The training batch size was 64, which is the maximum number that does not exceed the capacity of the GPU. We added *null gesture* classes to the training data to reflect the fact that sporadic key gestures are interspersed between more frequent incidental gesticulations.

4.2 Evaluation Datasets

We performed evaluations on the four datasets enumerated below. The first dataset mainly serves as a comparison of our system with state-of-the-art approaches in offline recognition. The second dataset is the unsegmented version of the first dataset. The third dataset, which is the Gesture Knitter dataset [32], serves to illustrate performance in a real-world setting using data collected from a Microsoft HoloLens 2. The fourth dataset, which is the SHREC'2021 dataset [6], provides an evaluation dataset for analysis of the choice of data dimensions.

1. **Smedt et al. [10]**: The SHREC'17 Track dataset [10] is an offline dataset that has 2,800 sequences containing 14 gestures performed by 28 individuals in two ways—using a single finger and the whole hand. The 14 gestures include *Grab*, *Expand*, *Pinch*, *Rotation Clockwise (CW)*, *Rotation Counter Clockwise (CCW)*, *Tap*, *Swipe Right*, *Swipe Left*, *Swipe Up*, *Swipe Down*, *Swipe X*, *Swipe X*, *Swipe* $+$, *Shake*, among which *Grab*, *Expand*, *Pinch*, *Rotation CW*, *Rotation CCW* are fine gestures. Fine gestures always involve hand posture changes, while coarse gestures involve hand movements. The dataset provides depth images and skeleton joint data. The split between training and testing for offline performance is defined and well-established in the literature [10]: 1,960 sequences for training and 840 sequences for testing.

2. **Smedt et al. [10]**: Online DHG [10] is the online (unsegmented) version of the SHREC'17 Track, which provides 280 sequences of 10 unsegmented gestures occurring sequentially. The online performance is evaluated under a leave-one-subject-out protocol. The 3D coordinates of 22 joints are provided for each hand skeleton.

3. **Mo et al. [32]**: The Gesture Knitter dataset taken from Mo et al. [32] contains an offline dataset with 10 gestures. The online dataset contains 40 samples, and each has three gestures performed in a 20-second sequence. Each subject performed each of the five predefined sequences of three different gestures. This dataset was collected using the Microsoft HoloLens 2. It provides skeletal data of the hand and fingers corresponding to six joints. We used a leave-one-subject-out experimental protocol for testing the online recognition performance. For offline recognition performance evaluation, we trained the model using a different set of subjects than the test data to perform cross-validation.

4. **Caputo et al. [6]**: The SHREC'2021 Track dataset [6] is a dataset made for practical application scenarios where gesture recognizers need to work in real-time and to be able to detect and correctly label gestures "in the wild" within a continuous sequence of hand movements. The dataset includes 18 gesture classes belonging to different types. A subset of 7 classes are static, 5 classes are coarse dynamic gestures, and 6 classes are fine dynamic gestures. Static gestures are characterized by keeping a fixed hand pose for a minimum amount of time, while dynamic gestures are characterized by a single trajectory with an unchanged hand pose or with finger articulation over time. The hand skeleton data is collected using a Leap Motion device mounted on a headband worn by the participants to simulate a sensor mounted on an HMD.

4.3 Evaluation Metrics

As key gesture spotting is still in its infancy, the literature on evaluation metrics is limited. To separate the measures of recognition accuracy and early detection ability, we evaluate the model using Levenshtein distance for accuracy and Normalized Time to Detect (NTtD) for early detection performance.

1. **Early Detection**: We use Normalized Time to Detect (NTtD) [17] to evaluate early detection performance. Given a correct key gesture spotting, define the time for the start of the gesture as idx_{start} and the time for the end of the gesture as idx_{end}. Further, define the time the key gesture spotting model fires as $idx_{activation}$.

   For a successful gesture recognition without any lag, $idx_{start} \leq idx_{activation} \leq idx_{end}$. NTtD is the fraction of the gesture that has occurred before the system fires a detection:

$$\frac{idx_{activation} - idx_{start} + 1}{idx_{end} - idx_{start} + 1} \tag{1}$$

   NTtD is 0 for a false detection ($idx_{activation} < idx_{start}$) and ∞ for a false rejection ($idx_{activation} > idx_{end}$). We report NTtD only for gestures for which there is early detection. Therefore, we stress that the only valid range of values for NTtD is NTtD $\in [0, 1]$.

2. **Recognition Accuracy**: Recognition accuracy for online recognition is different compared to offline settings. Offline recognition only considers the class accuracies. However, an online recognition system needs to consider other scenarios: 1) **False activation**: the key gesture spotting system falsely fires when there is actually no key gesture performed; and 2) **Multiple activations**: the key gesture spotting system fires multiple times when the gesture is only performed once. These scenarios make it complicated to calculate the actual performance of the real-time key gesture spotting system, considering we do not know the actual start and end times of the actual gesture. To address this we use the Levenshtein distance as the evaluation metric for online recognition performance [21]. The Levenshtein distance is used to capture the recognition accuracy irrespective of whether the recognition is an early or late detection. Levenshtein distance (sometimes referred to as minimum edit distance) is a metric defined as the minimum number of single-character insertions, deletions and substitutions required to transform one string into another. The accuracy for online performance is defined as:

$$accuracy = 1 - \frac{\text{levenshtein}(y_{predict}, y_{true})}{\text{length}(y_{true})} \tag{2}$$

   where $y_{predict}$ and y_{true} are the predicted and true list of labels of the gestures respectively.

Caputo et al. [6] used the Jaccard Index to measure the average relative overlap between the ground truth and the predicted label sequences for the gesture sequences recording. They also used detection rate and false-positive rate to measure the performance of the models.

Category	Method	Modality	Accuracy (%)	
			14G	**28G**
Hand-crafted	Boulahia et al. [5]	skeleton	90.50	80.50
CNN	HPEV+HMM +FRPV [25]	skeleton	94.88	92.26
Graph	DG-STA [8]	skeleton	94.40	90.70
Manifold learning	ST-TS-HGR -NET [34]	skeleton	94.29	89.40
	Ours	skeleton	**95.27**	**91.57**

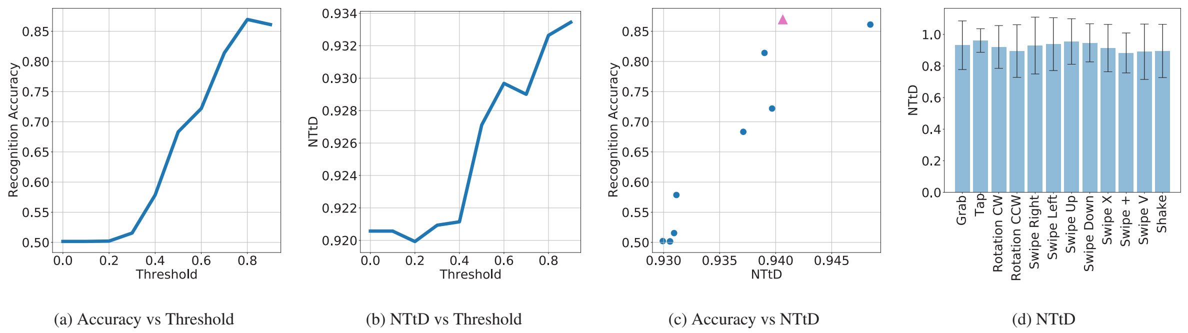
Table 1: Offline recognition accuracy and comparison with the state-of-the-art approaches on the SHREC'17 Track dataset. 14G and 28G represent a 14 and a 28 gesture configuration respectively.

5 Results

We split the results section into offline and online gesture recognition performance.

5.1 Offline Gesture Recognition

We first tested our classifier model's offline recognition performance. The state-of-the-art methods are shown for comparison with our model on the SHREC'17 Track dataset in Table 1. The results of the compared methods are collected from Liu et al. [25]. Among the alternative models evaluated, our approach ranked top on the 14 gestures setting and ranked second on the 28 gestures setting for the skeleton joint



(a) Accuracy vs Threshold (b) NTtD vs Threshold (c) Accuracy vs NTtD (d) NTtD

Fig. 3: Detect probability threshold values ranging from 0.1 to 0.9 with step 0.1 to determine the trade-off between recognition accuracy (a) and NTtD (b) for the Online DHG dataset [10]. Larger threshold values lead to higher accuracy but also higher NTtd. We arrived at a balance between accuracy and NTtD at a threshold equal to 0.8, as shown by the pink triangle marker in (c). (d) shows NTtD values averaged over all testing sequences for the 14 gestures in the Online DHG dataset [10] using key gesture spotting with the threshold set to 0.8. The bars show NTtD standard error. Gestures which are either falsely rejected or falsely detected are not presented in this plot.

data. This result demonstrates that our model achieves state-of-the-art accuracy performance on the simpler problem of offline recognition. However, a model that performs well in offline gesture classification may not necessarily perform well in key gesture spotting as data from real-time gesture recognition is more noisy and does not have a clear indication/boundary of the start and end of a gesture.

Finally, our model is lightweight. A suitable metric for effectively measuring the computational efficiency of a model is floating-point operations per second (FLOPs). The second-best performing model in Table 1, HPEV+HMM+FRPV [25], performs 1.46 GFLOPs while our model performs 59 MFLOPs. In other words, our mo22del requires roughly 25× fewer operations. A modern HMD can easily run dozens of GFLOPs.

5.2 Online Gesture Recognition

As previously suggested, the detect probability threshold determines the trade-off between recognition accuracy and the normalized time to detection (NTtD). Increasing this threshold helps to prevent false activations and classifications but setting the threshold too high may delay activation on valid gestures. It is therefore necessary to find a detect probability threshold value that maximizes recognition accuracy while minimizing detection time. This reasoning is illustrated by Figure 3 which plots the trade-off between accuracy and NTtD values averaged over all testing sequences for the 14 gestures in the Online DHG dataset [10]. Both Figure 3a and Figure 3b indicate that a high threshold will likely lead to a high accuracy and NTtD, which is expected as a high threshold for detection results in fewer false positives but also later detections. We can inspect Figure 3c to determine an appropriate threshold by finding the highest accuracy while still maintaining a low NTtD. By setting the threshold to 0.8 we achieved an online recognition accuracy of 86.70% and an NTtD of 0.94. The recognition accuracy is calculated using the Levenshtein distance which captures accuracy independent of early or late detection. The NTtD is reported by computing the average of the NTtD values of the gestures that are early detected. Falsely rejected gestures, despite being correctly recognized, are not included in the average. These cases typically occur when the detection happens outside the gesture's start/end window. Figure 3d shows the NTtD values for the gestures in the Online DHG dataset when the detect threshold is set to 0.8. In particular, we note that the NTtDs for *Expand* and *Pinch* are not included as they are either falsely detected or falsely rejected, resulting in invalid values for the NTtD as per the definition in subsection 4.3. The static hand gestures require a larger portion of the nucleus to be seen before a decision with high confidence is given, while coarse gestures exhibit a larger standard error since they involve hand movements that can be determined before the whole movement is seen.

The online recognition performance on the Gesture Knitter dataset was tested using a leave-one-subject-out experimental protocol. Using

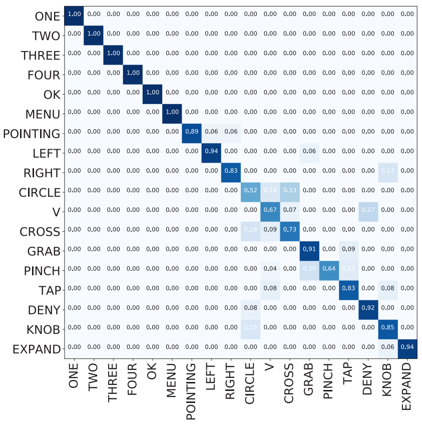


Fig. 4: Confusion matrix for the SHREC'2021 Track dataset. The vertically listed labels represent the true class whereas the horizontally listed labels represent the predicted class.

a threshold value of 0.8 we achieved an online recognition accuracy of 89.74% and a NTtD of 0.94.

Caputo et al. [6] introduced four different models to be tested on the SHREC'2021 track dataset using the aforementioned Jaccard Index, detection rate, and the false-positive ratio. We also used these three metrics to evaluate our model on this dataset to enable a comparison with the four models introduced by Caputo et al. [6]. Our system achieved a detection rate of 0.9032, a false positive rate of 0.053, and a Jaccard index of 0.8785. We outperformed the best model in the literature, which has a detection rate of 0.8993, a false positive rate of 0.066, and a Jaccard index of 0.8526. The comparable performance achieved by the best model in Caputo et al. [6] is in part down to the fact that their model has been hand-crafted and optimized for this specific dataset. As a result, the Caputo et al. [6] model is unlikely to offer good generalizability to different forms of gestures, as demanded by a rapid prototyping tool.

The confusion matrix in Figure 4 shows the recognition accuracy for the 18 gestures in the SHREC'2021 track dataset, and illustrates that our model has effectively learned both spatial and temporal relationships. We see that dynamic gestures are much harder to recognize than static ones. *Circle*, *V* and *Cross* are frequently confused with each other, which is expected given that they are similar in terms of both posture variants and hand movements.

	Config A1	Config A2	Config A3	Config B1	Config B2	Config B3
Position	✓		✓	✓		✓
Orientation		✓	✓		✓	✓
Jaccard Index	0.8896	0.8600	0.8785	0.6267	0.8007	0.8674

Table 2: Caputo et al. [6] included 20 keypoints of the hand's skeleton in the data. We introduce two configurations of keypoints. Config A: all the hand keypoints; Config B: palm and all fingertips. The sub-configurations are represented by 1,2,3 and contain position, orientation, or both respectively.

5.2.1 Selection of Model Input

We further analyzed the impact of altering the model inputs (including the choice of hand skeleton keypoints and their positional and orientation information) on the KGS recognizer performance in order to highlight the importance of model input selection on the hand skeleton data. Our Gesture Spotter tool provides a rapid prototyping functionality to quickly test what model inputs can deliver optimal performance in online recognition. Specifically, we evaluated how many keypoints of the hand skeleton data to include and the choice between using position, orientation or both. We assessed two different configurations based on the number of keypoints included. Config A includes all the keypoints while Config B includes just the palm and the fingertips (ThumbTip, IndexTip, MiddleTip, RingTip, PinkyTip using the notation of the finger keypoints from [2]). Intuitively, Config A contains both the posture variants from the fingertips and global hand trajectory from the palm movement. Config B includes the global hand movement information but lacks information from posture variants.

The inclusion of position and/or orientation informs the model from different perspectives. A gesture involving extensive rotation of the hand may need the orientation to be more distinctive to the model, while a gesture involving a complicated global trajectory may need the position data to be more informative. We define three sub-configurations per configuration reflecting the the choice of including position, orientation or both (termed as sub-configurations 1, 2, 3 respectively). Table 2 shows the results from the nine different configurations evaluated. Configuration A1 outperforms other configurations in terms of Jaccard Index. However, this configuration is only the best on average for the 18 gestures and there does not exist an absolute best configuration for all gestures. It is important to note that the comparisons of the Jaccard Index from the different configurations given in Table 2 do not reflect a universal rule on the importance of the selection of data dimensions. For example, inclusion of orientation is only superior to inclusion of position data for Config B but not for Config A.

Our results suggest that the choice of configuration must also consider the qualities of the key gestures to be recognized. A static gesture may only require the fingertips. Dynamic fine gestures that only involve the variants of the thumb may only require the palm and the full set of thumb keypoints. Including all keypoints and both position and orientation as the model inputs should be avoided because this will lead to a large input dimension that may be detrimental to model performance.

6 Graphical User Interface

To facilitate rapid prototyping, we implemented a GUI to enable developers to build and evaluate a key gesture spotting model out-of-the-box. The GUI has five different tabs to support the critical tasks of preprocessing, augmentation, training, evaluation and visualization.

1. **Preprocessing:** The preprocessing tab offers different types of preprocessing strategies, such as decoupling the gesture data into hand movement data and posture variant data, as well as making the trajectories relative to the wrist position. It also allows for selection of the model input, for example, choosing which of the 23 different hand keypoints (adopted from the skeleton hand bone guideline from [2]) to include and whether to use position and/or orientation data.

2. **Augmentation:** The augmentation tab supports different augmentation strategies, such as generating synthetic data from a Generative Adversarial Network (GAN). GANs have been used to generate realistic samples in image [12, 43, 50], speech [19], and motion trajectory [41] synthesis. We adopted the *Imaginative GAN* from Shen et al. [40] as one of our data augmentation strategies, which can approximate the true distribution of the input data and sample new data from the approximated distribution. The goal of the Imaginative GAN is to learn the latent attributes, such as behavioral attributes (for example, the speed of performing the actions/gestures) and physical attributes (for example, hand sizes). Thereafter, these learned latent attributes are applied to other data from different gesture classes. The GUI also supports other classical data augmentation strategies, such as adding noise, scaling and shifting which are adopted from Nunez et al. [35].

3. **Training:** The training tab allows the user to tune the hyperparameters of a machine learning model, such as the input/output dimension, the learning rate, the number of epochs and the batch size. Different models, in addition to the proposed A-ResLSTM model, are also provided in the GUI for the user to select from such as a conventional CNN by Nunez et al. [35] and an LSTM model by Lai et al. [22]. These models can then be evaluated and compared as different gestures may result in distinct performance with different models. For example, a CNN may be more suitable for gestures with more spatial variation while an LSTM model may be more suitable for gestures with more temporal variation.

4. **Evaluation:** The evaluation tab enables tuning of the parameters in the single-time activation algorithm, which are the detect probability threshold, recurrence threshold, and the slide step used in the sliding window strategy. We also support the generation of a plot similar to Figure 1 so that users can use this plot to inspect the predicted scores at every prediction step and utilize the understanding obtained to tune the parameters.

5. **Visualization:** A visualization tab is also provided to visualize the gesture skeleton data with a slide bar controlling the index of the logged sequence. This tool is useful when determining the index of the start and end of a gesture to evaluate the NTtD.

6.1 User Study

6.1.1 Participants

We recruited 10 participants (3 female, 7 male, average age: 23.5, standard deviation: 2.64, min: 20, max: 29) using opportunity sampling from within our institution. Participants represent a range of backgrounds: nine participants had prior experience with programming; six participants had prior experience with VR/AR; four participants had prior experience with software development in VR/AR; and one participant had previous experience in programming gesture recognition systems. We also gauged participants' level of pre-existing familiarity with machine learning (ML). Participants were categorized into one of two groups depending on their level of familiarity: 1) having experience in programming with ML ('experience'); and 2) having some passing knowledge of ML terminology ('exposure'), such as training epochs, learning rate, batch size, etc. Out of the 10 participants in this study, two had 'experience' with ML and six had 'exposure'.

6.1.2 Apparatus

We used the Oculus Quest 2 to collect gesture data. The training of the deep learning models are performed on a PC with GPU Nvidia GeForce RTX 2080 Ti, and CPU Intel Core i7-9800X. We replicated on-device recognition with the Oculus Quest by sending the gesture data in a continuous live stream to the PC. We currently make use of a client/server architecture for recognition as this enables rapid development and convenient separation of function. This aligns with our focus on rapidly producing hand gesture recognizers for pre-deployment evaluation, thereby allowing developers to quickly prototype and test different forms of gesture interaction. We anticipate the system will be able to run on-device in the near future.

6.1.3 Procedure

Participants were asked to design six novel hand gestures corresponding to six hypothetical commands relevant to AR/VR. The six commands were: *answer call*; *bring up the main menu*; *exit from the application*; *confirm*; *enable pass-through mode (VR specific)*; and *shutdown the device*. We also introduced null gestures, which include the coarse dynamic gestures from [6]. The null gestures were treated as one class in the model training and prediction. Ten scenarios were created, composed of the six key gestures interspersed with fifteen randomly chosen null gestures. These scenarios serve to provide a realistic evaluation setting given that key gestures are not performed as frequently as incidental gesticulations.

Participants were instructed to use a Unity application running on the Quest 2 over USB (using Oculus Link) to collect hand skeleton gesture data for the null gestures and the six custom key gestures. The null gestures were collected using a 200 second clip where participants were instructed to perform the null gestures repeatedly and randomly. Each of the six custom key gestures was collected with a 100 second clip where the gesture was performed repeatedly with deliberately random rest positions, that is, the positions before the preparation phase and after the retraction phase. The training and testing sets were generated by sliding windows, with window length l and step size s configurable in the GUI, over the recorded clips so that each window contained part or all of a gesture's nucleus. The purpose of this data collection scheme was to provide the model with realistic data, rather than data containing clear gesture segments, so that the model can achieve better robustness in key gesture spotting. Participants were then instructed to perform the corresponding gestures in three scenarios to provide the evaluation dataset used in the GUI.

Once all gesture data was recorded, participants started the GUI session. Participants were instructed to perform the following steps: 1) load the data and perform preprocessing; 2) perform data augmentation; 3) select and train a model; 4) visualize the data and annotate the start and end index of the key gestures; and 5) evaluate the recognizer in terms of accuracy and NTtD. Participants could either use the default setting, or freely use the GUI and test different configurations of the data structure, the augmentation strategies, the model parameters, the single-time activation strategy parameters, and so on. Participants were encouraged to find an optimal configuration for the KGS recognizer by testing different configurations. Model training took around two minutes and the whole process of finalizing a particular configuration of KGS recognizer took approximately 5–10 minutes. When participants were satisfied with the performance of their KGS recognizer, as evaluated in the GUI, they were asked to deploy the newly trained model for direct use with the Quest 2. Participants then performed the remaining seven interaction scenarios as part of a real-time evaluation of the constructed recognizer.

Throughout all stages of the study, participants were asked to describe what they were thinking and doing using a think-aloud protocol. After completing all tasks, participants completed the System Usability Scale (SUS) to evaluate the usability of the GUI. Participants also completed a post-experiment questionnaire to capture additional feedback on the performance and usability of the GUI. The entire duration of the study was approximately two hours and participants received a $25 voucher in appreciation for their involvement.

6.2 Results

6.2.1 Quantitative

The GUI received an average System Usability Scale score of 77.5. This score is generally classified as "Good" usability according to Bangor et al. [4]. Each user managed to arrive at a final recognizer that achieved, on average, an accuracy of 87.95%, and a NTtD of 0.96. Note that these performance results were achieved by only trialing 2–3 different design configurations, that is, experimenting with different choices of model input, data augmentation techniques, model choices and hyperparameters. We suggest that although these performance measures are good, they could be further improved if participants were allowed more time to optimize the configurations.

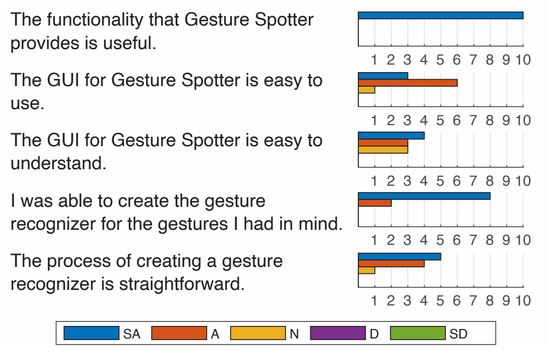


Fig. 5: The five-point Likert responses to the GUI post-study questionnaire. Likert responses are color coded and the scale shows the number of participants with the same rating. SA is strongly agree, A is agree, N is neutral, D is disagree, and SD is strongly disagree.

Figure 5 summarizes participants' responses to the post-study questionnaire. All participants strongly agreed that Gesture Spotter provided useful functionality. The GUI was generally considered easy to use and understand. Participants also responded positively to statements regarding the ability to create a recognizer with custom gestures and the straightforward process for doing this.

6.2.2 Qualitative

In the post-study questionnaire, participants were also asked to provide written responses to questions asking what they liked/disliked about the Gesture Spotter GUI, as well as what aspects of the GUI could be improved. A range of responses were captured regarding what participants liked about the GUI. *P4* commented that, "I can clearly identify the functions of the GUI, and the parameters are easy to adjust. The interface is user-friendly and gives users flexibility and freedom to adjust the parameters." Similarly, *P9* commented that, "The GUI made a task that seems very complicated and hard to do, very straightforward and surprisingly robust." Reflecting on the alternative to using the Gesture Spotter GUI, *P7* observed that, "I like the fact that it provides an alternative for fast model training without going through code line by line." We grouped the participants' responses regarding what they liked about the GUI by identifying related key words referring to attributes of the system: i) accurate, accuracy, fast, robust; ii) easy, simplify, straightforward, convenient, user-friendly; iii) flexibility, flexible, freedom; and iv) personalize, new. The common feedback that emerged from participants' comments about what they liked is summarized below:

1. The GUI allows for creation of an accurate gesture recognizer by just using the default settings. (*P1, P3, P5, P7, P9, P10*)
2. The GUI is easy to use and the process of designing a gesture recognizer is straightforward. (*P4, P10*)
3. The GUI offers good flexibility in that it supports experimentation with different parameters and configurations in the preprocessing, augmentation, and model training stage. (*P1, P5, P6, P8*)
4. The GUI allows users to quickly design new and personalized gestures. (*P2, P6*)

Examining participants' responses to the question regarding what they disliked about the GUI allowed us to identify opportunities for further improvements. Several participants commented on the poor aesthetics of the GUI ("vintage", "old school"). This suggests that the GUI would benefit from a more modern appearance. *P7*, who was experienced in gesture recognition programming, commented that, "I dislike the GUI being slightly over-simplified. I want to ask for more flexibility, such as the number of neurons and number of layers of the model. " This comment highlights a difficult design trade-off between simplicity and complexity. Our goal in this work focuses on supporting developers with limited existing expertise as this is the group likely to benefit the most from using Gesture Spotter.

A user with basic ML experience will benefit from additional knowledge and flexibility when interacting with the Gesture Spotter GUI. However, the GUI does not demand advanced ML expertise. A user with little or no prior experience of ML can still use the default settings provided in the GUI and produce an effective recognizer. Indeed, the two participants in our study who had no prior exposure to ML were still able to create robust KGS systems.

7 Application Programming Interface

Gesture Spotter also provides a Python API enabling developers to build and evaluate a key gesture spotting model with only a few lines of code. We carried out a developer-focused study to assess the usability of the Gesture Spotter API.

7.1 User Study

7.1.1 Participants

We recruited four participants from within our institution (3 female, 1 male, average age: 24.5; standard deviation: 1.91; min: 23; max: 27). All participants were right handed, had at least two years experience with Python, and at least one year of experience with machine learning. Three participants had experience working with AR/VR immersive headsets and with Unity. We deliberately recruited participants to have some degree of experience in Python programming and machine learning as we envision the API will used by individuals who wish to exercise more control over the model development and deployment than afforded by the GUI.

7.1.2 Procedure

We asked each participant to declare a new gesture that they will then append to the Gesture Knitter gesture set. To achieve this, participants needed to collect samples of the new gesture, perform data augmentation, train and assess the model, and evaluate performance on device. At the beginning of the study, we deployed a model supporting online recognition of the ten one-handed gestures from the Gesture Knitter dataset. We gave the participants a tutorial on how to perform these 10 gestures when wearing an Oculus Quest headset and running a Unity application over Oculus Link. Note that the Gesture Knitter dataset was collected using the HoloLens 2 and the ability of Gesture Spotter to provide accurate recognition on a different device highlights the flexibility of the method and the benefits of working with hand skeleton data. We made sure that participants could perform all gestures by confirming that each gesture was recognized at least once. We then used a Python notebook to guide the participants through the API calls for data preprocessing, augmentation, classifier training, and evaluating the performance. In particular, we explained how to assess recognition rate using the confusion matrix as well as how to tune hyperparameters to trade-off between accuracy and the normalized time to detect. Following this introduction, participants were asked to declare their novel gesture and collect 30 samples, each of five seconds in length, for which 25 samples are used for training and five are withheld for offline validation. Participants were then asked to utilize the API calls to assess the offline performance using the visualizations provided in order to tune the hyperparameters appropriately. The computed recognition results were then relayed back to the device for the developer to observe while wearing the headset. They were finally asked to perform five online recognition trial sets to assess the online performance of the deployed model, each trial lasting 20 seconds and containing the novel gesture which was just created. The five trial sets were: (1) *Shrink, New, Execution*; (2) *Push, MadRiddles, New*; (3) *Flamingo, Shrink, New*; (4) *New, CheshireDance, PeaceOut*; (5) *Grow, New, TickTock*. *New* here is a placeholder for the novel hand gesture which each participant created. These gesture sets were selected to diversify the gestures and the order in which they were performed. Each participant was asked to perform two practice sessions for each online gesture set before we recorded the 20 second log.

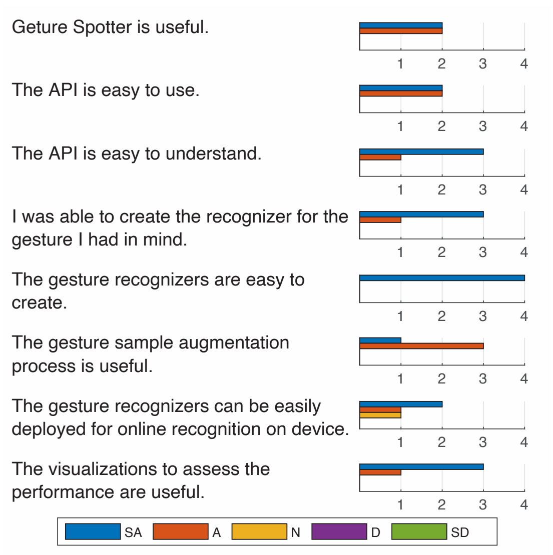


Fig. 6: The five-point Likert responses to the API post-study questionnaire. Likert responses are color coded and the scale shows the number of participants with the same rating. SA is strongly agree, A is agree, N is neutral, D is disagree, and SD is strongly disagree.

We followed a think-aloud protocol to capture the design process employed by participants during the study. At completion, participants also responded to a questionnaire examining the usefulness and usability of the different functions provided by the API. Participants were given 90 minutes for the entire study and they were provided with a workstation with a keyboard and mouse. Participants received a $15 voucher in appreciation for their involvement.

7.2 Results

All participants completed the design study by creating a novel hand gesture. Most participants were inspired by the more involved gestures in the Gesture Knitter dataset when creating their own gesture. The responses to the post-study questionnaire are summarized in Figure 6. Notably, all participants agreed that Gesture Spotter was useful, the API was easy to use, recognizers were easy to create, and that the recognizers were easily deployable for use with the device.

By examining the online recognition logs we can also assess the detection accuracy achieved for the novel gestures declared by the participants. Recognition accuracy for the online gesture sets is calculated as follows: for each set, we deem a gesture as being recognized if it is detected and then add one to the total of recognized gestures, which we then divide by the total number of distinct gestures. Aggregating all four participants, we find that the average recognition rate was 86.7%. This indicates that developers are able to easily add new gestures with little data overhead while achieving good recognition accuracy for an online application scenario.

The following observations were made as part of the think-aloud protocol used in the study. It was observed that participants were able to follow the major steps in the Gesture Spotter workflow, suggesting that the API is intuitive and easy to use. Participants were also able to analyze the visualizations provided by the API calls, thereby illustrating that they are useful in aiding the user in understanding the model and its behavior due to particular hyperparameter settings.

After the design process, some participants commented on their experience of the API and workflow. *P1* commented on how the visualizations for the trade-off between accuracy and NTtD was useful in understanding how the detection threshold affected the final performance. *P4* suggested making Gesture Spotter more accessible to non-programmers by offering a guided GUI for each step of the procedure. This point highlights the benefit of our approach to providing a tool containing both an API and GUI.

8 Limitations and Future Work

One limitation of the current GUI is that it lacks explanatory and tutorial features aimed at users with no prior ML experience. Despite this, we did observe that users with no machine learning knowledge could still create a robust decoder using the default settings. One aspect users without ML experience are likely to struggle with is in understanding the different hyperparameters in the model. This may in turn lead to the creation of a sub-optimal model in terms of recognition performance for their custom gesture set. Possible solutions include offering users more interactive assistance and access to visualization tools that can guide inexperienced users through the use of the GUI.

Different single-time activation methods may lead to different recognition results. Therefore, it may be fruitful to explore implementing more single-time activation methods. The current method is based on a deterministic approach with a predefined threshold. For future work, we suggest investigating using tools from signal processing, such as matched filters, to detect the sudden peak from the raw prediction scores. We conjecture this will further improve online recognition accuracy and early detection. Another fruitful avenue of future work is exploring transfer learning [44] for skeleton-based data to enable a pre-trained model to be fine-tuned for user-defined gestures. Transfer learning is a common technique in computer vision and natural language processing that enables a heavily pre-trained model to be used on new data to achieve better performance than merely training the model on the new data. Moreover, meta-learning approaches [11] can potentially enable few-shot learning.

9 Conclusions

We have presented a key gesture spotting architecture and encapsulated the architecture into a GUI and a series of API calls for quickly building and evaluating real-time key gesture spotting recognizers for AR and VR applications. Thanks to the different subcomponents included, such as the data preprocessing and data augmentation scheme, developers require less training data and can spend less time and effort collecting and annotating hand gesture data, which are expensive and time-consuming activities. We demonstrated the efficacy of the KGS architecture by evaluating its performance on four datasets. We then demonstrated the usability of Gesture Spotter's GUI and the API in two user studies.

In summary, this paper has presented a key gesture spotting architecture consisting of a novel hand gesture classifier and a novel single-time activation algorithm. We have demonstrated that the hand gesture classifier delivers robust and data-efficient key gesture spotting functionality while the single-time activation algorithm achieves an effective balance between online recognition accuracy and early detection. Finally, we introduced Gesture Spotter as a tool that provides developers with an easy-to-use GUI and API for rapid prototyping of KGS functionality. We hope Gesture Spotter will stimulate further research in this important research area which ultimately serves to provide users with more fluid hand gesture interaction in AR/VR.

Open Science

Complete source code for the GUI and the API, as well as data from the study can be found here: `https://github.com/CambridgeIIS/Gesture_Spotter`.

Acknowledgments

John Dudley and Per Ola Kristensson were supported by EPSRC (grant EP/S027432/1).

References

[1] Impact of $-family. `https://depts.washington.edu/acelab/proj/dollar/impact.html`.

[2] Oculus - set up hand tracking. `https://developer.oculus.com/documentation/unity/unity-handtracking/`.

[3] L. Anthony and J. O. Wobbrock. A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface 2010*, GI '10, p. 245–252. Canadian Information Processing Society, CAN, 2010.

[4] A. Bangor, P. Kortum, and J. Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *J. Usability Studies*, 4(3):114–123, may 2009.

[5] S. Y. Boulahia, E. Anquetil, F. Multon, and R. Kulpa. Dynamic hand gesture recognition based on 3d pattern assembled trajectories. In *2017 seventh international conference on image processing theory, tools and applications (IPTA)*, pp. 1–6. IEEE, 2017.

[6] A. Caputo, A. Giachetti, S. Soso, D. Pintani, A. D'Eusanio, S. Pini, G. Borghi, A. Simoni, R. Vezzani, R. Cucchiara, et al. Shrec 2021: Track on skeleton-based hand gesture recognition in the wild. *arXiv preprint arXiv:2106.10980*, 2021.

[7] T. Chalasani, J. Ondrej, and A. Smolic. Egocentric gesture recognition for head-mounted ar devices. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 109–114, 2018.

[8] Y. Chen, L. Zhao, X. Peng, J. Yuan, and D. N. Metaxas. Construct dynamic graphs for hand gesture recognition via spatial-temporal attention. *arXiv preprint arXiv:1907.08871*, 2019.

[9] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-second international joint conference on artificial intelligence*, 2011.

[10] Q. De Smedt, H. Wannous, J.-P. Vandeborre, J. Guerry, B. Le Saux, and D. Filliat. SHREC'17 Track: 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset. In I. Pratikakis, F. Dupont, and M. Ovsjanikov, eds., *3DOR - 10th Eurographics Workshop on 3D Object Retrieval*, pp. 1–6. Lyon, France, Apr. 2017.

[11] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. PMLR, 2017.

[12] M. Frid-Adar, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan. Synthetic data augmentation using gan for improved liver lesion classification. In *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*, pp. 289–293. IEEE, 2018.

[13] L. Gao, Z. Guo, H. Zhang, X. Xu, and H. T. Shen. Video captioning with attention-based lstm and semantic consistency. *IEEE Transactions on Multimedia*, 19(9):2045–2055, 2017.

[14] T. Ha, S. Feiner, and W. Woo. Wearhand: Head-worn, rgb-d camera-based, bare-hand user interface with visually enhanced depth perception. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 219–228, 2014.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[16] S. Hegde, R. Perla, R. Hebbalaguppe, and E. Hassan. Gestar: Real time gesture interaction for ar with egocentric view. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*, pp. 262–267, 2016.

[17] M. Hoai and F. De la Torre. Max-margin early event detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2863–2870, 2012.

[18] S. Hochreiter and J. Schmidhuber. Lstm can solve hard long time lag problems. *Advances in neural information processing systems*, pp. 473–479, 1997.

[19] T. Kaneko, S. Takaki, H. Kameoka, and J. Yamagishi. Generative adversarial network-based postfilter for stft spectrograms. In *Interspeech*, pp. 3389–3393, 2017.

[20] K. Kawakami. Supervised sequence labelling with recurrent neural networks. *Ph. D. thesis*, 2008.

[21] O. Köpüklü, A. Gunduz, N. Kose, and G. Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. In *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, pp. 1–8. IEEE, 2019.

[22] K. Lai and S. N. Yanushkevich. Cnn+ rnn depth and skeleton based

dynamic hand gesture recognition. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 3451–3456. IEEE, 2018.

[23] L. A. Leiva, D. Martín-Albo, and R. Plamondon. Gestures &#xe0; Go Go: Authoring Synthetic Human-Like Stroke Gestures Using the Kinematic Theory of Rapid Movements. *ACM Transactions on Intelligent Systems and Technology*, 7(2):15:1–15:29, Nov. 2015.

[24] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen. Medical image classification with convolutional neural network. In *2014 13th international conference on control automation robotics & vision (ICARCV)*, pp. 844–848. IEEE, 2014.

[25] J. Liu, Y. Liu, Y. Wang, V. Prinet, S. Xiang, and C. Pan. Decoupled representation learning for skeleton-based gesture recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5751–5760, 2020.

[26] J. Liu, A. Shahroudy, D. Xu, and G. Wang. Spatio-temporal lstm with trust gates for 3d human action recognition. In *European conference on computer vision*, pp. 816–833. Springer, 2016.

[27] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[28] H. Lü, J. A. Fogarty, and Y. Li. Gesture script: recognizing gestures and their structure using rendering scripts and interactively trained parts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pp. 1685–1694. Association for Computing Machinery, New York, NY, USA, Apr. 2014.

[29] H. Lü and Y. Li. Gesture coder: a tool for programming multi-touch gestures by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2875–2884. Association for Computing Machinery, New York, NY, USA, May 2012.

[30] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5528–5531. IEEE, 2011.

[31] Y. Min, Y. Zhang, X. Chai, and X. Chen. An efficient pointlstm for point clouds based gesture recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[32] G. B. Mo, J. J. Dudley, and P. O. Kristensson. Gesture knitter: A hand gesture design tool for head-mounted mixed reality applications. CHI '21. Association for Computing Machinery, New York, NY, USA, 2021.

[33] P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4207–4215, 2016.

[34] X. S. Nguyen, L. Brun, O. Lézoray, and S. Bougleux. A neural network based on spd manifold learning for skeleton-based hand gesture recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12036–12045, 2019.

[35] J. C. Nunez, R. Cabido, J. J. Pantrigo, A. S. Montemayor, and J. F. Velez. Convolutional neural networks and long short-term memory for skeleton-based human activity and hand gesture recognition. *Pattern Recognition*, 76:80–94, 2018.

[36] O. Oreifej and Z. Liu. Hon4d: Histogram of oriented 4d normals for activity recognition from depth sequences. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 716–723, 2013.

[37] G. Park, T.-K. Kim, and W. Woo. 3D Hand Pose Estimation with a Single Infrared Camera via Domain Transfer Learning. In *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 588–599, Nov. 2020. ISSN: 1554-7868.

[38] V. Pavlovic, R. Sharma, and T. Huang. Visual interpretation of hand gestures for human-computer interaction: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, 1997.

[39] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1010–1019, 2016.

[40] J. Shen, J. Dudley, and P. O. Kristensson. The imaginative generative adversarial network: Automatic data augmentation for dynamic skeleton-based hand gesture and human action recognition. In *2021 16th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2021)*, pp. 1–8, 2021.

[41] J. Shen, J. Dudley, and P. O. Kristensson. Simulating realistic human motion trajectories of mid-air gesture typing. In *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 393–402, 2021.

[42] X. Shi, J. Pan, Z. Hu, J. Lin, S. Guo, M. Liao, Y. Pan, and L. Liu. Accurate and Fast Classification of Foot Gestures for Virtual Locomotion. In *2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 178–189, Oct. 2019. ISSN: 1554-7868.

[43] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016.

[44] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pp. 270–279. Springer, 2018.

[45] E. M. Taranta, M. Maghoumi, C. R. Pittman, and J. J. LaViola. A Rapid Prototyping Approach to Synthetic Data Generation for Improved 2D Gesture Recognition. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pp. 873–885. Association for Computing Machinery, New York, NY, USA, Oct. 2016.

[46] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock. Gestures as point clouds: A $p recognizer for user interface prototypes. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction*, ICMI '12, p. 273–280. Association for Computing Machinery, New York, NY, USA, 2012.

[47] J. Wang, J. Chen, Y. Qiao, J. Zhou, and Y. Wang. Online gesture recognition algorithm applied to hud based smart driving system. In *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 289–294, 2019.

[48] J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, p. 159–168. Association for Computing Machinery, New York, NY, USA, 2007.

[49] H. Xie, J. Wang, B. Shao, J. Gu, and M. Li. Le-hgr: A lightweight and efficient rgb-based online gesture recognition network for embedded ar devices. *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 274–279, 2019.

[50] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1316–1324, 2018.

[51] B. Yoon, H.-i. Kim, S. Y. Oh, and W. Woo. Evaluating remote virtual hands models on social presence in hand-based 3d remote collaboration. In *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 520–532, 2020.