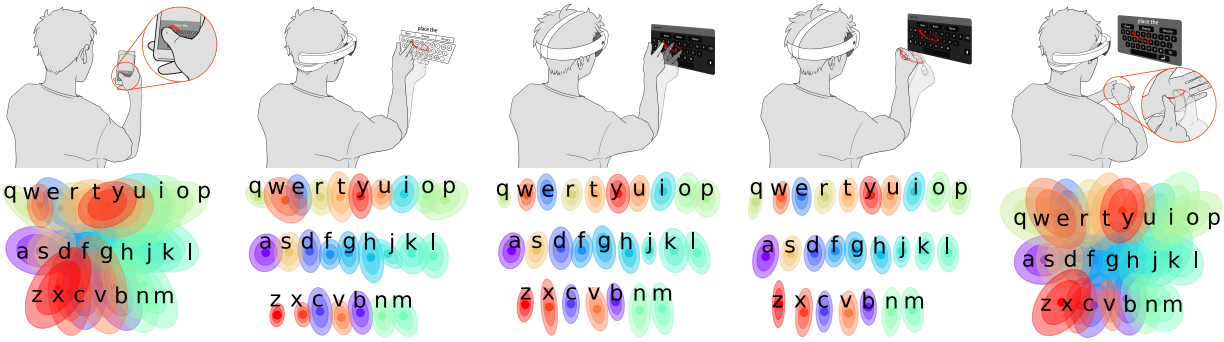
This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2024.3456198

Gesture2Text: A Generalizable Decoder for Word-Gesture Keyboards in XR Through Trajectory Coarse Discretization and Pre-training

Junxiao Shen, Khadija Khaldi, Enmin Zhou, Hemant Bhaskar Surale, and Amy Karlson



(a) Mobile Phone WGK (b) Mid-Air Poke (AR) WGK (c) Mid-Air Poke (VR) WGK (d) Mid-Air Pinch (VR) WGK (e) On-Surface WGK

Fig. 1: We pre-trained a neural-network-based word-gesture keyboard (WGK) decoder for **Gesture2Text** decoding, also noted as a neural decoder, using a novel one-hot encoded coarse discretized trajectory representation. This approach enhances generalizability across various WGK systems illustrated above. The distribution of touch points in the second row illustrates that different WGK systems exhibit unique data patterns due to variations in interaction modes, keyboard sizes, and user behaviors.

Abstract—Text entry with word-gesture keyboards (WGK) is emerging as a popular method and becoming a key interaction for Extended Reality (XR). However, the diversity of interaction modes, keyboard sizes, and visual feedback in these environments introduces divergent word-gesture trajectory data patterns, thus leading to complexity in decoding trajectories into text. Template-matching decoding methods, such as SHARK^2 [32], are commonly used for these WGK systems because they are easy to implement and configure. However, these methods are susceptible to decoding inaccuracies for noisy trajectories. While conventional neural-network-based decoders (neural decoders) trained on word-gesture trajectory data have been proposed to improve accuracy, they have their own limitations: they require extensive data for training and deep-learning expertise for implementation. To address these challenges, we propose a novel solution that combines ease of implementation with high decoding accuracy: a generalizable neural decoder enabled by pre-training on large-scale coarsely discretized word-gesture trajectories. This approach produces a ready-to-use WGK decoder that is generalizable across mid-air and on-surface WGK systems in augmented reality (AR) and virtual reality (VR), which is evident by a robust average Top-4 accuracy of 90.4% on four diverse datasets. It significantly outperforms SHARK^2 with a 37.2% enhancement and surpasses the conventional neural decoder by 7.4%. Moreover, the *Pre-trained Neural Decoder*'s size is only 4 MB after quantization, without sacrificing accuracy, and it can operate in real-time, executing in just 97 milliseconds on Quest 3.

Index Terms—Pre-trained models, text entry, word-gesture keyboard, discretization

1 Introduction

Extensive techniques have been proposed and developed to enable fast and accurate text entry in Extended Reality (XR) environments [15, 21, 33, 46, 60]. Among these, word-gesture keyboards (WGK) emerges as a promising solution, achieving text entry speeds ranging from 20 to 40 words per minute (WPM) for proficient users [16, 22, 26, 35, 39, 50, 59, 62, 66]. This method not only offers excellent learnability, being widely adopted on touchscreen devices [34], but also inherently handles noisy and ambiguous input due to the ambiguous nature of word-gesture trajectories [32]. This is analogous to word-gesture typing on small screens of smartwatches, which effectively addresses the 'fat finger' problem [19].

However, the majority of previous studies focus on the development of new interaction techniques. The word-gesture decoding process, which translates the word-gesture trajectory into text, predominantly employs the classic SHARK^2 [32] decoder. SHARK^2 [32] is a template-matching algorithm that computes the similarity of the input trajectory with the pre-defined word-gesture templates constructed from a word corpus and gives top-ranked predictions based on trajectory-template similarity. The popularity of template-matching decoders stems from their simplicity; they merely require pre-defining word-gesture templates and similarity metrics, enabling the matching algorithm to be used in a plug-and-play fashion. However, these algorithms are not without their limitations, including inability to predict out-of-vocabulary (OOV) words and lack of decoding accuracy for noisy input trajectories [47].

Alsharif et al. [7] and Shen et al. [50] propose training a neural-network-based decoder, or neural decoder, to decode word-gesture trajectories into text. Shen et al. [50] explicitly compared neural decoders with SHARK^2, suggesting that neural decoders significantly outperform SHARK^2. However, the adoption of neural decoders is limited due to a significant drawback: they require substantial amounts

- *Junxiao Shen is with Reality Labs Research, Meta and University of Bristol.*
- *Khadija Khaldi, Enmin Zhou, Hemant Bhaskar Surale and Amy Karlson are with Reality Labs Research, Meta.*

of training data. We hypothesize that different WGK systems exhibit distinct trajectory patterns, rendering a neural decoder trained on one system non-generalizable to another. This limitation is due not only to variations in interaction modes, but also to the different keyboard sizes and user behaviors, as indicated by the touch point distributions of the on-surface and mid-air WGK systems in AR and VR in Figure 1. Moreover, it requires deep expertise to develop and train a neural decoder, as these models are more complex to build and train compared to some classic deep-learning models which already have many open-sourced codebases [11, 58]. We aim for a generalizable neural decoder that combines ease of configuration and implementation with high decoding accuracy.

To achieve this, we propose a novel trajectory representation, which is one-hot encoded coarse discretized trajectories (see example illustration in Figure 6c). This novel representation tolerates high noise and differences in trajectories from different WGK interaction modes, keyboard sizes, and user behaviors. Therefore, it enables pre-training a neural decoder on datasets from one WGK system to be generalized across other WGK systems without explicit fine-tuning. This eliminates the need for collecting training data and complicated configuration.

More specifically, we pre-trained the decoder on a public word-gesture trajectory dataset collected from Mobile Phone WGK [34] combined with a dataset of synthetic word-gesture trajectories [49]. We further validated our *Pre-trained Neural Decoder* on four datasets: one publicly available dataset for mid-air WGK in AR from Shen et al. [50, 51] and three datasets collected ourselves, including mid-air WGK in VR with two different interaction modes and on-surface WGK (illustrated in Figure 1). Initially, we compared the *Pre-trained Neural Decoder* with a conventional neural decoder from Alsharif et al. [7] and SHARK^2, demonstrating a 7.4% improvement and a 37.2% improvement, respectively, in Top-4 accuracy. To this end, the *Pre-trained Neural Decoder* could be used out of the box with an average decoding Top-4 accuracy of 90.4%, which is adequate for large-scale applications [47]. We also investigated fine-tuning the pre-trained decoder, achieving a modest improvement. Additionally, we explored its individual components, including different discretization techniques, encoding methods and model structures. Lastly, we conducted a model efficiency validation test by deploying the model onto Quest 3 [42] through model quantization without sacrificing decoding accuracy [23]. The latency was approximately 97 millisecond (ms), which is adequate for commercial use.

In conclusion, our contributions are as follows:

1. We propose a novel trajectory coarse discretization approach to enable the pre-training of a word-gesture neural decoder that can be generalized to on-surface and mid-air word-gesture keyboards in AR and VR. The resulting *Pre-trained Neural Decoder* is fast and easy to deploy and does not require additional data collection for training or fine-tuning.

2. We validated the *Pre-trained Neural Decoder* on datasets from four different AR/VR word-gesture keyboard systems and showed that the *Pre-trained Neural Decoder* performs significantly better than conventional decoders, with a Top-4 word prediction accuracy of 90.4%.

3. We tested the real-time performance of the pre-trained decoder on a mobile VR device, Quest 3, by quantizing the model, suggesting low latency of 97 ms and real-world applicability of the model.

2 Related Work

2.1 Word-Gesture Keyboards

The word-gesture keyboard (WGK), originally developed for text entry on personal digital assistants (PDAs), Tablet PCs, and mobile phones using a stylus [32, 64], has become one of the dominant text entry methods on modern touchscreen devices. Its success is attributed to the ease of learning [31, 64, 65] and the high entry rates achievable [31, 32, 47].

Leiva et al. [34] conducted a study to collect word-gesture typing data from 909 users. As this study focused solely on data collection, no statistical decoding process was implemented in the keyboard. Expert users achieved a typing speed of 50 words per minute (WPM), while those unfamiliar with gesture keyboards reached 40 WPM. In contrast, Reyal et al. [47] conducted a separate study with 12 participants using Google Keyboard (Gboard) to assess text entry speeds. In this study, the entry rate for participants increased from 33.6 WPM in the initial block to 39.1 WPM by the ninth block. Despite the difference in participant numbers, with the latter study involving only 12 participants, a clear gap in text entry rate (50 WPM vs 40 WPM) is evident between the actual word-gesture keyboard text entry rates and the ideal rates assuming perfect decoding. This discrepancy underscores the critical need for the development of a more accurate decoder to enhance performance.

Given the benefits of word-gesture keyboards on mobile devices, researchers have explored adapting this input method for use in AR and VR environments [12, 22, 39, 49, 57, 59, 61, 62]. In these studies, the entry rates for the most proficient participants or expert users typically span from 20 to 40 WPM. A commonality among these methods is the utilization of the SHARK^2 decoder, chosen for its ease of configuration, as these studies assert their novelty lies in the unique interaction modalities they introduce. However, the potential for enhanced performance through the adoption of a more advanced decoder is acknowledged, albeit with the caveat of requiring substantial implementation efforts. Therefore, there is a need for an advanced decoder ensuring both high decoding accuracy and plug-and-play capability in academic and research applications.

2.2 Word-Gesture Decoders

In this section, we first formally define the word-gesture decoding problem. Then, we introduce two commonly used approaches for word-gesture decoding: template-matching decoders and neural decoders. Lastly, we discuss the advantages and disadvantages of the two types of decoder, highlighting the need for an efficient joint approach.

2.2.1 Problem Formulation

When a cursor moves on a virtual keyboard, the movement is captured as a word-gesture trajectory, denoted by g. The initial step involves encoding this trajectory into a structured representation, $E(g)$. One commonly used encoding approach involves using the Cartesian coordinate positions of the trajectory at sequential time intervals (illustrated in Figure 6a), expressed as $E(g) = [x_1, x_2, ..., x_T]$, where T is the length of the resampled cursor trajectory, and each $x_t \in \mathbb{R}^2$ is the location of the cursor at time t. Both the current template-matching decoder and neural decoder primarily use this encoding approach.

Following this, word-gesture decoding is mathematically articulated as finding $w^* = \arg\max_w P(w|E(g))$, which essentially means determining the word w that, given the encoded trajectory $E(g)$, has the highest probability of being the intended input.

2.2.2 Template-Matching Decoder

Template-matching algorithm is a process that involves comparing an input trajectory g with a set of pre-defined word-gesture templates P to identify the best match. $P = \bigcup_{i=1}^{n} P_i$ is constructed from a lexicon $C = \{w_1, w_2, ..., w_n\}$ where n denotes the number of unique words in the lexicon. Each word w_i in the lexicon is mapped to a word-gesture template P_i through a specific process (ConstructTemplate). A naive (ConstructTemplate) process is to sequentially connect the central points of each letter within a word w_i on a keyboard to generate a simple template P_i. More specifically, for a given encoded input trajectory $E(g)$, the algorithm computes a similarity score $S(E(g), P_i)$ (eg. Euclidean distance) for each pre-defined template P_i. The algorithm then selects the template P_j with the highest similarity score as the best match for the encoded trajectory $E(g)$: $P_j = \underset{i \in \{1,2,...,n\}}{\arg\max} \; S(E(g), P_i)$.

SHARK^2 [32] is one of the most frequently used template-matching decoders. The SHARK^2 algorithm employs a sophisticated template-matching technique that leverages a multi-channel architecture to enhance recognition accuracy for word-gestures [32]. Through these mechanisms, SHARK^2 efficiently narrows down the vast space of potential matches by quickly discarding non-viable candidates and focusing on those most likely to be correct. This multi-channel approach,

combining shape and location information with a smart pruning strategy, allows SHARK^2 to offer high accuracy and speed in word-gesture decoding, even with a large vocabulary of possible inputs.

2.2.3 Neural Decoder

A neural decoder [25, 50] is represented by a function D_θ, which maps an input y to a probability distribution over the alphabet for each time step. This function is parameterized by weights and biases θ, which are learned during training. The model is trained using the Connectionist Temporal Classification (CTC) loss [20], which aligns the output sequence π with the target label sequence z by maximizing the probability of z given π across all possible alignments, considering insertions of the CTC blank label ϕ where necessary.

For each time step t, the model outputs a probability distribution π_t over the extended character alphabet L'_{char}. The extended alphabet includes all the characters in the model's alphabet plus a special token for the CTC blank label, denoted as ϕ. If the original alphabet $L_{\text{char}} = \{a,\ldots,z\}$ represents 26 lowercase English letters, then the extended alphabet $L'_{\text{char}} = L_{\text{char}} \cup \{\phi\}$ includes these letters plus the blank token. The output π_t for each time step t is a vector in the simplex $\Delta^{|L'_{\text{char}}|}$, meaning that it represents a probability distribution across the extended alphabet. The dimension of π_t is $|L'_{\text{char}}|$, where $|L'_{\text{char}}|$ is the size of the extended alphabet. If we consider just the lowercase letters plus the blank token, $|L'_{\text{char}}| = 27$.

The complete output of the model π for the entire swipe gesture is a sequence of these probability distributions across all time steps, so $\pi = [\pi_1, \pi_2, \ldots, \pi_T]$. Each $\pi_t \in \Delta^{|L'_{\text{char}}|}$, making the dimension of π to be $T \times |L'_{\text{char}}|$. Figures 6b and 6d both visualize the probability distributions π using heatmaps.

Formally, the neural decoder model can be mathematically represented as follows:

$$D_\theta : \mathbb{R}^{m \times T} \to (\Delta^{|L'_{\text{char}}|})^T$$

where $D_\theta(y) = \pi$ and each $\pi_t \in \Delta^{|L'_{\text{char}}|}$ for $t = 1,\ldots,T$.

2.2.4 Comparison Between SHARK^2 and Neural Decoder

- **SHARK^2 is Easier to Configure and Implement.**
  SHARK^2 supports customizable (ConstructTemplate) process, thus it enables the adaptability to different technologies. In contrast, a neural decoder requires a large dataset for effective training, especially for novel WGK systems in AR and VR. One could theoretically utilize existing word-gesture data from Mobile Phone WGK [34] to adapt the data to different keyboard sizes through various transformation functions. However, as illustrated in Figure 1, the intrinsic properties of the data may significantly differ, resulting in suboptimal performance. Additionally, creating a neural decoder demands deep learning expertise, and costly resources on complex training and hyperparameter tuning processes.
- **Neural Decoder has Significantly Better Decoding Accuracy.**
  Despite SHARK^2's ease of configuration, SHARK^2 suffers from persistent performance gaps compared to neural decoders. Shen et al. [50] demonstrated a substantial improvement in accuracy when employing a neural decoder, achieving a low error rate of 5.41%, as opposed to the higher Character Error Rate (CER) of 35.34% found with SHARK^2. Character Error Rate is the percentage of characters that were incorrectly predicted compared to the total number of predicted characters. Naturally, for efficient text entry, high prediction accuracy from word-gesture decoders is essential.
- **SHARK^2 is a Word-Level Model and Neural Decoder is Character-Level Model.**
  SHARK^2 uses word-gesture templates from a lexicon, enabling predictions only for words within this set. Thus, SHARK^2 is a word-level model. Conversely, a neural decoder, embodying a character-level model, predicts characters sequentially without relying on a set of pre-defined templates. This character-level model can progressively predict with each character word-gesture, offering enhanced flexibility and immediacy in text entry.

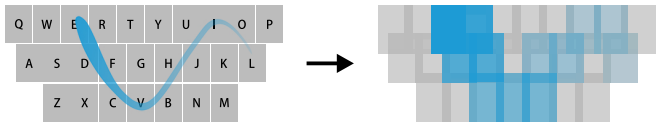


Fig. 2: The discretization process converts a continuous word-gesture trajectory into discrete 'pixels' via a mapping function. The 'pixels' are larger than key tabs to compensate for ambiguous and noisy trajectories.

3 Pre-Training a Neural Decoder into a Generalizable Model

As outlined in the comparison between SHARK^2 and the conventional neural decoder discussed in Section 2.2.4, there is a critical need for a model to achieve a balance between ease of configuration and high accuracy in word-gesture decoding. To this end, we introduce a model that combines the simplicity of setup with robust performance, thereby mitigating the weaknesses of each of the former approaches. Drawing inspiration from large language models known for their zero-shot learning capabilities, which are achieved through pre-training on vast corpora [10, 30], we aim to pre-train a neural decoder with a substantial volume of data to achieve generalizability for different WGK systems. However, direct training using the Cartesian coordinate sequence x utilized by both SHARK^2 [32] and the neural decoder [7] is impractical, as these sequences are fine-grained and vary significantly across different WGK systems in AR and VR.

We propose a novel encoding method that encodes the continuous trajectory sequence into a coarse discretized representation, as illustrated by Figure 2. This approach addresses the challenge of dataset variability and enhances our model's understanding of trajectory patterns. In the following sections, we will elaborate on our discretization methodology, provide details of our training data, and describe the specialized architecture of our model designed to optimize pre-training effectiveness.

3.1 Word-Gesture Trajectory Discretization

Encoding is the process of transforming unstructured data into structured elements that a computer can process. In previous attempts to encode word-gesture trajectories [7, 32, 50], the strategy involved utilizing Cartesian coordinate positions. From an information theory perspective, using coordinate positions directly for word-gesture decoding has limitations due to their continuous nature, whereas the desired output (characters) is discrete. Continuous data can vary infinitely within a given range, leading to a high degree of uncertainty and requiring more information to specify each point precisely. In contrast, discrete outputs, such as characters, have a finite set of possibilities. This mismatch means that directly mapping continuous input to discrete output can introduce inefficiencies and inaccuracies, necessitating algorithms to effectively bridge this gap by discretizing the continuous input or employing strategies to reduce the information loss during this conversion.

Our method discretizes a word-gesture trajectory into coarse discretized 'pixel' regions on the keyboard, a process we refer to as word-gesture trajectory discretization. Figure 2 demonstrate this discretization process and Figure 6c visualize the discretized trajectory in one-hot-encoding for the word 'quickly' using a heatmap. Instead of tracking continuous movement, we assign each segment of a word-gesture trajectory to the corresponding 'pixel' region it traverses according to a mapping function. This approach simplifies complex word-gesture trajectories into a sequence of discrete 'pixels.' This discretization enhances the accuracy of recognizing word-gesture patterns by minimizing the impact of noise in position tracking, variations in user behavior and keyboard sizes, thereby making the neural decoder more efficient at predicting user input. This method effectively bridges the gap between the continuous nature of word-gesture trajectory and the discrete structure of text input, enabling more accurate and efficient decoding of word-gesture trajectories. Formally:

1. Discretizing the keyboard into discretized regions based on the positions of the keys, where each region is defined by a square with height and width proportional to the key tab's dimensions, adjusted by a ratio factor.

2. Define a mapping function $C(x)$ that maps a given trajectory point (x_i) to a region based on the discretized regions it falls within. This function embodies the discretization process, assigning each point on the keyboard to a specific region.

3. The discretized trajectory T can be expressed as $T = \{C(x) \mid (x_i) \in S\}$, where each element $C(x_i)$ is the region corresponding to the segment of the trajectory passing through that region on the keyboard. The discretized trajectory is then one-hot encoded and used as the input for the neural decoder.

This mathematical formulation encapsulates the process of transforming continuous word-gesture trajectory into discrete sequences of 'pixels'.

3.2 Training Dataset

We pre-trained the pixellated neural decoder using two datasets:

1. *Mobile Phone WGK Dataset*: Our research utilizes a public dataset, 'how we swipe' [34], gathered through a web-based custom virtual keyboard on mobile devices. This dataset includes 8,831,733 touch points corresponding to 11,318 unique English words gestured by 1,338 users, with 11,295 unique words correctly gestured and 3,767 words gestured inaccurately.

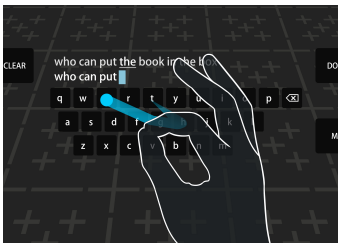
2. *Synthetic Dataset*: Our study integrates the GAN (Generative Adversarial Network)-Imitation model proposed by Shen et al. [48], which they subsequently applied this model to synthesize word-gesture trajectories [49]. They then performed extensive evaluations and comparisons of the GAN-Imitation model with other techniques for generating synthetic word-gesture trajectory data to train neural decoders [50]. For our research, we adopted the synthetic strategy detailed in [50], training the synthetic model using the *Mobile Phone WGK Dataset*. In this paper, we do not analyze synthetic data generation methods, instead concentrating our efforts on examining the discretization and pre-training approaches.

We have constructed a large-scale training dataset comprising 95,649 trajectory samples from the *Mobile Phone WGK Dataset*, alongside 100,000 trajectory samples from the *Synthetic Dataset*. This dataset consists of 32,347 unique words.

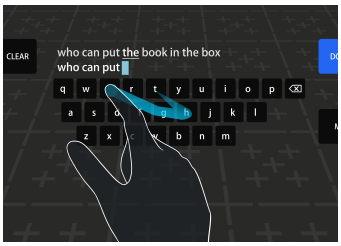
3.3 Implementation Details

Here is a detailed description of our model and the associated training specifics. We conducted a comprehensive hyperparameter optimization process to determine the values of the hyperparameters [9].

- **Model Configuration**: We use PyText [41] to implement our model. The core of our model comprises a Bi-directional Long Short-Term Memory (BiLSTM) [25, 53] layer, which is crucial for understanding the temporal dependencies within the input sequences. This representation layer consists of two stacked LSTM layers with a hidden dimension of 222, allowing the model to capture both forward and backward context effectively. To prevent overfitting, a dropout rate of 0.3 is applied within this layer, providing regularization by randomly omitting a subset of features during training. Following the creation of the representation layer, a dense fully connected layer serves as an intermediary, facilitating the transition from the LSTM output to the decoder. This layer employs a Rectified Linear Unit (ReLU) [6] activation function, layer normalization [8], and an additional dropout rate of 0.3 to maintain regularization. The final component of our model is the decoder layer, which utilizes a Connectionist Temporal Classification (CTC) [20] beam decoder. This decoder implements a beam search algorithm [18] to efficiently explore the most probable word candidates by evaluating combinations of character probabilities at each timestep. Beam search enhances the model's ability to predict sequences accurately by considering multiple hypotheses concurrently.



(a) VR mid-air pinch WGK.



(b) VR mid-air poke WGK.

Fig. 3: Data collection of 200 participants for mid-air word-gesture keyboards (WGK) in VR.

- **Training Details**: For training our model, we employed the Adam [28] optimizer with a learning rate of 0.01, complemented by a minimal weight decay of 0.00001 to prevent overfitting further. The training process was conducted over 600 epochs, with an early stopping mechanism disabled to allow the model to fully converge. The batch size for training, evaluation, and testing was uniformly set to 128 to balance computational efficiency and training stability. The model was initialized with random weights for training. To accommodate the computational demands of training and ensure numerical stability, we adopted mixed-precision training using the 'FP16OptimizerFairseq' from Fairseq [44], starting with an initial loss scale of 128. This approach not only accelerates training but also reduces the memory footprint, allowing for the use of larger batch sizes or models. Additionally, our training regimen included reporting metrics to TensorBoard [5] for real-time monitoring and analysis of the model's performance. The best model configuration, as determined by Top-4 word prediction accuracy, was automatically saved and loaded post-training to ensure that our results were based on the peak performance of the model. This configuration was achieved through a hyperparameter sweep, experimenting with over 100 configurations, including the number of LSTM layers, the dimension of the LSTM layer, the dropout rate, and the learning rate, to find the optimal model that achieves the highest accuracy while ensuring the model size remains below 5 MB.

4 Validation Datasets

We posit two hypotheses: firstly, that distinct WGK systems exhibit unique data patterns; and secondly, that our *Pre-trained Neural Decoder* serves as a universally applicable solution across various WGK systems. To explore these hypotheses, we have collected an array of datasets from different WGK systems, encompassing a range of interactions including on-surface touch, mid-air poking (pointing), and mid-air pinching (raycasting) [43] across both AR and VR platforms featuring keyboards of different sizes. Table 1 gives an overview of the four datasets.

4.1 Mid-Air WGK in VR

We begin our exploration by examining WGKs within virtual reality (VR), a domain that has seen extensive investigation [14]. Notably, platforms such as Quest Headsets [42] have integrated mid-air WGK functionalities in their v56 updates [40]. Our primary focus lies on two interaction modes: mid-air poke WGK and mid-air pinch WGK.

	Mobile Phone WGK	Mid-Air Poke (AR)	Mid-Air Poke (VR)	Mid-Air Pinch (VR)	On-Surface WGK
Number of Users	1,338	16	200	200	100
Device Platform	*Mobile Phone*	*HoloLens 2*	*Quest 2*	*Quest 2*	*PC*
Number of Unique Words	11,295	1,700	3,000	3,000	2,123
Number of Samples	95,649	25,513	51,413	51,433	33,594
Text Entry Rates (WPM)	31.11 (SD=12.49)	21.39 (SD=13.17)	26.9 (SD=10.19)	27.4 (SD=9.88)	24.7 (SD=11.14)
Touch Points Statistics	0.15 (0.27)	0.042 (0.063)	0.030 (0.049)	0.028 (0.045)	0.12 (0.23)

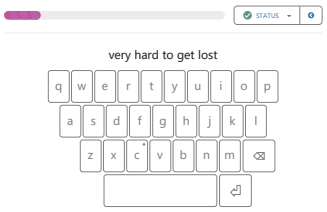
Table 1: We gathered datasets from five unique WGK systems. We report text entry rates with the mean and standard deviation (SD). We report two touch point statistics: Average Distance to Key Center (ADKC) and Average Major Axis Length (AMAL) in the parentheses. The computation of these two statistics is detailed in Section 5.1.

Despite extensive studies exploring these two interactions [15,52], there are no public datasets available on VR mid-air WGK. To address this gap, we undertook our own data collection, involving a substantial scale of 200 users. The details of our study are as follows:

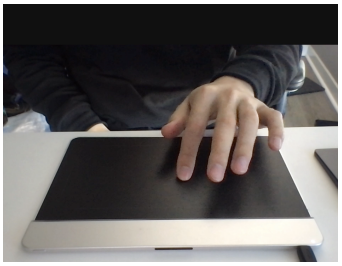
1. **Participants**: We recruited 200 volunteers as participants through an internal mailing list, with an age distribution as follows: 46 in their 20s, 74 in their 30s, 51 in their 40s, and 29 in their 50s. The group comprised 100 males and 100 females. We collected responses from participants about the frequency of their word-gesture typing usage on mobile phones: 20% had rarely or never used it, 32% sometimes used it (at least once a month), 26% often used this feature (at least once a week), and 22% always used the feature (at least once a day).

2. **Procedure**: Data collection was conducted in the participants' homes using the retail Quest 2, with the Android Application Package (APK) delivered via the experiment app on the Oculus Store. Participants were randomly assigned to two groups, each experiencing two conditions: pinch first and poke first. Before these conditions, users were introduced to the study through a step-by-step tutorial explaining the experiment. Figure 3a and Figure 3b demonstrate the word-gesture typing technique with pinch and poke interaction modes used in the study. The experiment application uploaded the study logs via a Dropbox API upon completion. We employed a 'Wizard of Oz' decoding strategy, akin to that described in Shen et al. [51], to guarantee the integrity of the data collected. Each condition for one participant takes around 30 minutes to complete.

3. **Phrase Set**: The phrase dataset was curated from the GLUE [56], MacKenzie [37], and Enron [29] datasets, resulting in a collection of 2,700 unique phrases comprising 3,000 unique words. No other information except text entry data was collected during the study, and the text entry data was anonymized.



(a) Study page showing device connectivity, study progression, and keyboard with the cursor.



(b) We use a haptic touchpad with a capacitive grid [1] to collect ground truth trajectory data.

Fig. 4: Data collection of 100 participants for on-surface word-gesture keyboards (WGK).

4.2 Mid-Air WGK in AR

The distinction between AR and VR in the context of mid-air WGK lies in the interaction feedback when directly poking the virtual keyboard: AR allows for 'real' hand interaction with a virtual keyboard, whereas VR facilitates interaction through a virtual hand with a virtual keyboard. Consequently, VR interaction is more precise because users can accurately see the virtual hand's position relative to the virtual keyboard, creating a closed-loop feedback system from the user's physical hand to the virtual hand control. Conversely, in AR, the absence of a virtual hand and the inaccuracies in hand tracking lead to an open-loop nature in hand interaction and control, potentially resulting in noisier input data [27]. In contrast, AR and VR mid-air pinch word-gesture typing operate similarly, offering a closed-loop system of projected remote cursor control on the virtual keyboard without significant differences. Therefore, in AR, the focus is on poke-based mid-air WGK.

We utilized two public datasets regarding AR mid-air pinch word-gesture typing, which were taken from the studies conducted by Shen et al. [50,51]. In one study, Shen et al. [51] introduced AdaptiKeyboard, a personalizable mid-air WGK for AR specifically designed for HoloLens 2. This keyboard employs multi-objective Bayesian optimization to dynamically adjust the keyboard size, aiming to optimize both speed and accuracy concurrently. The data collected for this study involved word-gesture typing on various sizes gathered from 12 participants. In a separate study, Shen et al. [50] presented a novel mid-air WGK design that removed visual feedback and relaxed the delimitation threshold. Throughout their user studies, they collected word-gesture typing data using different interaction designs from 34 participants.

4.3 On-Surface WGK

Apart from mid-air WGK, we also investigate on-surface WGK, which can be utilized in both AR and VR environments. On-surface WGK requires clear delimitation, similar to how WGK is performed on touchscreens; that is, placing a finger on the surface signals the beginning of a trajectory, and lifting it off the surface signals the end of the trajectory. To collect on-surface WGK data, we opt for alternatives to the built-in hand tracking from AR/VR headsets, as the current computer vision-based hand tracking cannot accurately detect the on-surface delimitation. Therefore, we use Sensel's haptic touchpad with capacitive touch and force field sensors [1] to collect ground truth trajectory data (Figure 4b). Moreover, instead of using a headset for display, we employ a monitor screen connected to a personal computer (PC), showing a virtual keyboard. This setup fully simulates a virtual touchpad, which can be on the palm [36], or any other surface in an AR environment [4, 43]. This data collection resulted in a total of 33,594 trajectories, encompassing 2,123 unique words from 100 participants.

The following provides further details on the data collection process:

1. **Participants**: We recruited 100 volunteers as participants through an internal mailing list, with an age distribution as follows: 19 in their 20s, 44 in their 30s, 24 in their 40s, and 13 in their 50s. The group comprised 64 males, 33 females, and 3 participants who preferred not to disclose their gender. We collected responses from participants about the frequency of their word-gesture typing usage on mobile phones: 12% participants had rarely or never used it, 33% sometimes used it (at least once a month), 43% often used it (at least once a week), and 12% always used the feature (at least once a day).

2. **Procedure**: Participants were seated in front of computer screens that displayed a keyboard interface, complete with a cursor and its trace, as illustrated in Figure 4a. They interacted with the interface using Sensel's haptic touchpad (Figure 4b) to facilitate word-gesture typing of prompted words on the screen. Each participant was tasked with word-gesture typing 160 words, randomly selected from an extensive phrase set, to ensure broad data collection. To mitigate fatigue, participants could take breaks of up to two minutes after completing sets of 10 words. We developed a front-end interface using Lab.js [24], which visualized the cursor and its trace and automatically generated random phrase prompts from the phrase set. Similarly to the method employed in Section 4.1, we utilized a 'wizard of oz' decoder for data collection. Each session for one participant takes around 1 hour to complete.

3. **Phrase Set**: The phrase dataset was created from two distinct sources: the Enron Mobile Corpus [29] and the MacKenzie phrase set [37], resulting in a collection of 2,320 unique phrases comprising 2,123 unique words.

5 Word-Gesture Data Analysis

We investigate the word-gesture typing datasets through two analytical perspectives: the assessment of touch point distributions and the evaluation of a geometric feature: curvature.

5.1 Touch Point Distribution Analysis

We utilized the visualization methods from Chen et al. [35] to compute the touch point distributions (visualized by plotting 95% confidence ellipses for each key) across various validation datasets. Additionally, we compute the statistics of the touch point distributions for each dataset:

- **Average Distance to Key Center (ADKC)**: The average distance from the center of each ellipse to the centers of the corresponding keys, again averaged over all keys. This metric assesses the bias in touch point locations relative to the intended target. The equation for computing this statistic is:

$$\text{ADKC} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(x_{c,i}-x_{k,i})^2+(y_{c,i}-y_{k,i})^2}$$

  where $(x_{c,i}, y_{c,i})$ denotes the center of the ellipse for key i, and $(x_{k,i}, y_{k,i})$ represents the center of key i.

- **Average Major Axis Length (AMAL)**: The average of the major axes lengths of the 95% confidence ellipses, calculated across all keys. This quantifies the spatial dispersion of touch points for each key, providing a measure of touch accuracy and precision. The equation for this statistic is:

$$\text{AMAL} = \frac{1}{N}\sum_{i=1}^{N} 2\sqrt{\lambda_{i,max}}$$

  where $\lambda_{i,max}$ represents the largest eigenvalue of the covariance matrix for the touch points on key i, and N is the total number of keys.

The computation of these statistics is based on normalizing trajectory data to a unit-width keyboard for a fair comparison between keyboards of different sizes. The plots for these distributions are displayed in the second row of Figure 1, and the statistics are presented in Table 1. Our observations revealed distinct word-gesture typing interactions, each showcasing a unique distribution of touch points. Specifically, AR mid-air WGK exhibits more noise compared to VR WGK, attributable not only to the aforementioned distinction between AR and VR hand interactions in Section 4.2, but also to other device-dependent factors. First, the HoloLens 2 offers lower hand tracking accuracy than the Quest 2, affecting precision. Second, in optical see-through AR, users can see a virtual keyboard without occlusion of their physical hands, complicating the task of accurately locating the virtual keyboard in relation to their hands. This challenge is compounded by the fact that the HoloLens 2 does not perform hand segmentation, allowing users to see their hands both before and beyond the virtual keyboard simultaneously. Lack of occlusion and the positioning of the virtual keyboard far from their hands introduce inaccuracies not only in tracking but also in user perception.

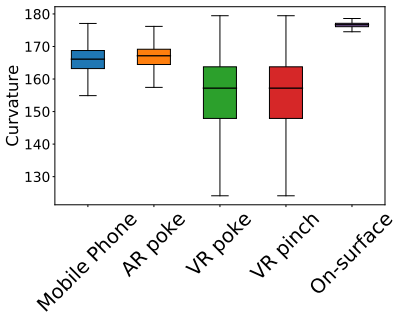


Fig. 5: Box plots depicting mean, median, and quartiles of the curvature of the trajectories from the five datasets.

The 'fat finger' problem becomes particularly noticeable with on-surface word-gestures and mobile phone word-gestures, marked by a scattered distribution of touch points and the larger AMAL and ADKC values. The term 'fat finger' effect describes the difficulty users face when their fingers, which are relatively large compared to the keys or touchpoints, mistakenly press adjacent keys or register incorrect word-gestures. This problem is a direct result of the small touch surfaces of mobile devices and touchpads. Unlike mid-air keyboards, which are larger and thus more accommodating to the natural size of human fingers, the touch surfaces on mobile phones and touchpads are considerably smaller, making it challenging to hit the intended keys accurately.

5.2 Geometric Feature Analysis

We evaluate the geometric distinctions across datasets through a geometric feature: curvature [49][1].

Curvature quantifies the local bending at any point along a curve, serving as an indicator of the degree to which it diverges from a straight line. A high curvature value signifies a pronounced bend, whereas a low curvature corresponds to a minor bend or a linear trajectory.

Figure 5 showcases the diversity in curvature present across the datasets. We note that VR-based WGK exhibits a broader and lower curvature range, suggesting a predominance of more linear trajectories. This pattern could stem from extensive body movements in a large interaction space, which tend to produce more straightforward trajectories. Conversely, AR maintains less linear trajectories due to the challenges associated with the interaction between physical hands and virtual keyboard. Among the datasets analyzed, on-surface WGK displays the most pronounced curvature, indicative of highly curved trajectories. This observation is attributed to the less direct control over the cursor compared to the control mechanisms found in Mobile Phone WGK and Mid-Air WGKs.

6 Experiments

Our research uses a series of experiments to assess the efficacy of a *Pre-trained Neural Decoder* across the four datasets. Initially, we evaluate the performance of the pre-trained decoder against benchmarks, specifically SHARK[2] and a conventional neural decoder, to establish a

[1] Details of the computation of curvature can be found in [49].

	Mid-Air Poke (AR)		Mid-Air Poke (VR)		Mid-Air Pinch (VR)		On-Surface WGK	
	Top-1	Top-4	Top-1	Top-4	Top-1	Top-4	Top-1	Top-4
SHARK^2 [32]	43.2% (±3.2%)	51.3% (±2.4%)	50.3% (±3.9%)	55.9% (±3.1%)	48.9% (±2.5%)	54.7% (±2.7%)	42.1% (±4.1%)	49.2% (±4.3%)
Conventional Neural Decoder [7]	74.6% (±1.8%)	81.0% (±1.3%)	79.8% (±2.0%)	86.3% (±1.9%)	77.9% (±2.1%)	84.8% (±2.4%)	71.5% (±2.2%)	81.6% (±1.9%)
Pre-trained Neural Decoder	82.5% (±1.4%)	89.8% (±1.1%)	85.1% (±1.9%)	91.9% (±1.4%)	82.7% (±1.5%)	90.2% (±1.0%)	83.0% (±2.0%)	89.5% (±1.8%)

Table 2: Decoding accuracy on four datasets of our proposed ***Pre-trained Neural Decoder*** compared to two baselines: SHARK^2 and conventional neural decoder. Results are reported in Top-1 and Top-4 accuracy, standard deviation is reported in parentheses.

comparative baseline. Subsequently, we discuss the potential enhancements achievable through fine-tuning the decoder on specific datasets. Furthermore, we conduct studies to examine the influence of different discretization techniques. Additionally, we explore alterations to the model's architecture to identify which structural configurations yield the optimal results for pre-training. Lastly, we assess the latency of the *Pre-trained Neural Decoder*.

6.1 Baselines Comparison

6.1.1 Baselines

In this study, we evaluate the performance of a pre-trained decoder by comparing it with two established baselines:

1. SHARK^2 Decoder: For our evaluation, we adopt the parameter settings detailed by Kristensson et al. [32], ensuring a direct comparison under standardized conditions.

2. Conventional Neural Decoder: The backbone of the conventional neural decoder is the same as our *Pre-Trained Neural Decoder*, with the only difference being the dimension of the input layer to accommodate the dimension of the Cartesian trajectory input.

Following the setup, both baseline decoders and the *Pre-trained Neural Decoder* are subjected to thorough testing on the designated test datasets. We employ the Leave-One-Subject-Out methodology for our experiments. This structured approach allows for a comprehensive assessment of each decoder's capabilities in handling real-world data.

6.1.2 Evaluation Measure

We use Top-k accuracy to evaluate the models. Top-k accuracy measures the model's ability to predict the correct word within its Top-k predictions. Formally, let y_i be the true label for the i-th instance and let $P_{i,k}$ be the set of top k predictions for the i-th instance made by the model. The Top-k accuracy over N instances is defined as: $\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}\left(y_i \in P_{i,k}\right)$, where $\mathbf{1}(\cdot)$ is the indicator function, which is 1 if the condition is true and 0 otherwise. In our analysis, we specifically report on Top-1 and Top-4 accuracy, providing insights into both the accuracy of the model's primary prediction and its ability to offer relevant alternatives within the top four suggestions.

6.1.3 Results

Table 2 reveals that our *Pre-trained Neural Decoder* significantly surpasses both the SHARK^2 decoder and the conventional neural decoder. The *Pre-trained Neural Decoder* achieves an average Top-1 accuracy of 83.3% and an average Top-4 accuracy of 90.4%. Switching from the SHARK^2 decoder to a *Pre-trained Neural Decoder* results in an average increase of 37.2% for Top-1 accuracy and 37.6% for Top-4 accuracy. Moving from a naively trained neural decoder to a *Pre-trained Neural Decoder* leads to additional improvements in average accuracy, with a 7.4% increase for Top-1 and a 6.9% increase for Top-4. The average accuracy of the *Pre-trained Neural Decoder* across various tasks is 83.3%. This Top-1 accuracy is already sufficient for research use. By equipping the *Pre-trained Neural Decoder* with a bigram language model and an auto-correction module of vocabulary size over 50,000 [50, 55], the average improvement is 11.2%, resulting in a final average Top-1 accuracy of 94.5%, which is an adequate accuracy rate for large-scale commercial use.

	Mid-Air Poke (AR)	Mid-Air Poke (VR)	Mid-Air Pinch (VR)	On-Surface WGK
Pre-trained	82.5% (±1.4%)	85.1% (±1.9%)	82.7% (±1.5%)	83.0% (±2.0%)
Fine-tuned	85.8% (±1.2%)	88.0% (±1.5%)	86.7% (±1.8%)	86.4% (±2.3%)

Table 3: Decoding accuracy after fine-tuning the *Pre-trained Neural Decoder*.

6.2 Fine Tuning Improves Model Performance, but Marginally

We conducted a comprehensive analysis to evaluate the impact of model fine-tuning. Our hypothesis suggests that fine-tuning, particularly when tailored to specific datasets, could markedly enhance the neural decoder's performance. Each dataset typically exhibits unique features or distributions inherent to its domain. Fine-tuning enables the neural decoder to adjust its pre-existing representations to accurately capture these domain-specific attributes. To validate our hypothesis, we rigorously fine-tuned the *Pre-trained Neural Decoder* with the training set of each dataset and assessed the neural decoders using the respective test sets. Table 3 demonstrates that fine-tuning effectively leverages the comprehensive learning acquired by the *Pre-trained Neural Decoder*, channeling their extensive capabilities to address the unique characteristics presented by a new dataset. The average improvement from the *Pre-trained Neural Decoder* to the fine-tuned decoder across all datasets is 3.4%. However, the average improved performance is not as large as the difference between the *Pre-trained Neural Decoder* to the conventional neural decoder and the SHARK^2. Therefore, the *Pre-trained Neural Decoder* already achieves adequate accuracy for research purposes and becomes suitable for commercial use when equipped with a language model. Thus, fine-tuning is not necessary.

6.3 Word-Gesture Trajectory Discretization Analysis

Word-gesture trajectory discretization consists of two main components: the mapping function and the index encoding method. We explore various mapping functions and index encoding methods independently.

6.3.1 Mapping Function

To refine the concept of defining the mapping function $C(x,y)$ which are the regions for the trajectory coarse discretization in virtual keyboards, we examine the utility of square and elliptical regions.

- Square Regions: The condition for a point (x,y) to be within a square region centered at (x_c,y_c) with dimensions $2w$ and $2h$ is:$|x-x_c| \leq w$ and $|y-y_c| \leq h$, where w and h are half the width and height of the rectangle, respectively.
- Ellipse Regions: For an elliptical region around a key, centered at (x_c,y_c) with semi-major axis a and semi-minor axis b, a point (x,y)

	Mid-Air Poke (AR)	Mid-Air Poke (VR)	Mid-Air Pinch (VR)	On-Surface WGK
Square Region	82.5% (±1.4%)	85.1% (±1.9%)	82.7% (±1.5%)	83.0% (±2.0%)
Ellipse Region	81.9% (±1.5%)	84.6% (±1.9%)	81.2% (±1.9%)	83.2% (±1.7%)

Table 4: Decoding accuracy of using different mapping functions to map segments of a word-gesture trajectory to discrete 'pixel' regions.

	Mid-Air Poke (AR)	Mid-Air Poke (VR)	Mid-Air Pinch (VR)	On-Surface WGK
One-Hot Encoding	82.5% (±1.4%)	85.1% (±1.9%)	82.7% (±1.5%)	83.0% (±2.0%)
Integer Encoding	73.2% (±2.7%)	73.2% (±3.2%)	68.0% (±2.6%)	64.2% (±3.0%)

Table 5: Decoding accuracy of different index encoding approaches.

falls within this region if: $\frac{(x-x_c)^2}{a^2} + \frac{(y-y_c)^2}{b^2} \leq 1$.

By incorporating these shapes into $C(x,y)$, we adjust the function to account for the region's shape associated with each keyboard character. This involves first identifying the shape and parameters for each key's region, then applying the corresponding condition to map (x,y) to its character. For the Square Region, we define w and h as twice the key width and height, respectively. Similarly, in the Ellipse Region, we also define the semi-major axis a and semi-minor axis b as twice the key width and height. This decision is informed by the Mobile Phone WGK touch point distribution analysis. The analysis indicates that the average ratio, derived from comparing the mean values of the semi-minor and semi-major axes of the 95% Confidence Ellipses to the mean dimensions (width and height) of keys, is approximately 2. Having larger 'pixels' also makes the discretized trajectory more tolerant to spatial noise and improves the neural decoder's ability to handle ambiguity.

Table 4 presents the results of decoding accuracy from different mapping functions. We observed that Square Regions perform slightly better on average than Ellipse Regions, but not significantly. Therefore, the decision to use Square Regions is based on computational expense. Computing the condition for being within/outside Square Regions is less computationally expensive compared to Ellipse Regions. This is because, for a square region centered at (x_c,y_c) with side length $2s$, checking if a point (x,y) is inside is simple: $|x-x_c| \leq s$ and $|y-y_c| \leq s$. This involves basic arithmetic and logical operations, making it computationally light. Conversely, determining if a point lies within an elliptical region centered at (x_c,y_c) with semi-major axis a and semi-minor axis b requires a more complex formula: $\frac{(x-x_c)^2}{a^2} + \frac{(y-y_c)^2}{b^2} \leq 1$. This involves squaring differences, dividing by the axes' squares, and summing the fractions, which are more computationally demanding tasks than those for square regions.

6.3.2 Index Encoding

Word-gesture trajectory, even when discretized into pixels, still require representation through numerical values for machine learning models to effectively learn from them. This essential step is known as encoding.

In our exploration, we delve into various 'pixel' index encoding methods: 1. **Integer Encoding**: This method assigns a unique integer value to each 'pixel'. For instance, 'a' is represented by the number 1, 'B' by the number 2, and so on. This simple yet effective technique ensures that each element is distinctly identifiable by a specific numerical value. 2. **One-Hot Encoding**: One-hot encoding takes a different approach, where each 'pixel' is represented by a vector. This vector contains all zeros except for a single one at the position corresponding to the pixel. For a 26-letter alphabet, the letter 'a' would be represented by a vector starting with [1, 0, 0, ..., 0], and 'b' would follow as [0, 1, 0, ..., 0]. This method provides a clear, binary representation of each character, distinguishing each one within a high-dimensional space.

Table 5 demonstrates that one-hot encoding performs better than integer encoding. Moreover, one-hot encoding avoids implying a numerical relationship between pixels. Integer encoding could lead the model to assume an ordinal relationship where none exists, potentially skewing the learning process. One-hot encoding represents each 'pixel' as a distinct, equally distant vector, facilitating more accurate predictions.

6.4 Model Structure

We conducted tests using a transformer [54] as the representation layer to learn latent representations, contrasting it with an LSTM-based approach. We experimented with various transformer model hyperparameters, and our findings indicate that the transformer layer struggles to effectively learn the patterns, consistently achieving a Top-1 accuracy of below 60%, rendering it impractical for use. This underperformance of transformers compared to LSTMs in word-gesture typing decoding may stem from their inferior capability to manage the precise, localized context inherent in word-gesture trajectories. The inherent sequence processing ability of LSTMs better captures temporal dependencies crucial for this task. Additionally, transformers may require even larger datasets with millions of training samples to avoid overfitting and their complex architecture might not suit the latency requirements of a real-time word-gesture decoder.

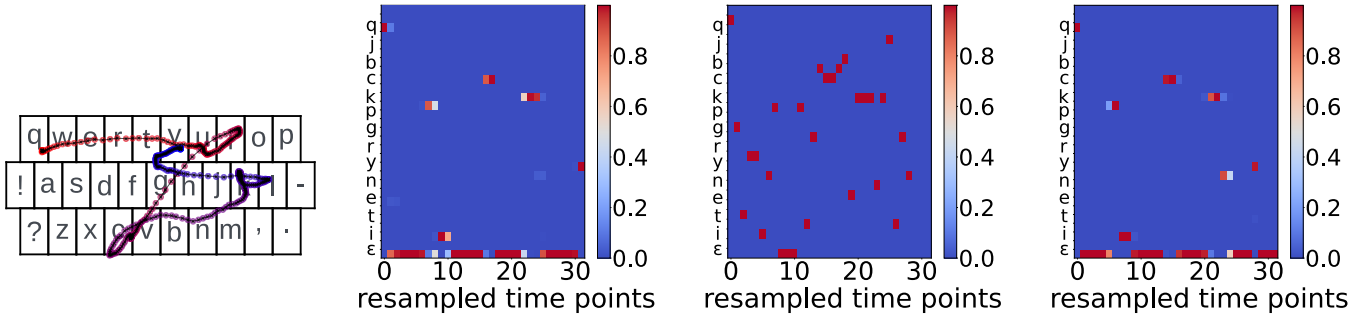
6.5 Latency Analysis

Despite their high performance, deep-learning models can sometimes be too large and incapable of running in real-time. This is critical for applications like word-gesture typing decoders, which must operate in real-time. To address the real-time performance of our model, we used TorchScript [45] to script and export the model. Additionally, we applied PyTorch's quantization tool to reduce the model size significantly without sacrificing the accuracy [45]. This is possible because we report Top-K and Top-1 word prediction accuracy, evaluating the model's predictions at the word level, even though the model predicts a sequence of characters. After quantization, the model's performance increased by an average of 0.8% in character error rate (CER), which measures the percentage of incorrectly predicted characters. However, when measured in Top-1 word prediction accuracy, which is calculated as $100 -$ word error rate (WER), the accuracy remained unchanged because evaluating at the word level has a coarser granularity than evaluating at the character level. The quantized model is only 4 MB in size and can run with a latency of 97 ms on a Quest 3. Generally, latencies below 100 ms are considered good for real-time interactions because they are perceived as nearly instantaneous by users.

7 Discussion

In this section, we discuss the following different aspects:

- **Contribution and Novelty**: In this study, we propose a novel approach to pre-training a neural decoder for word-gesture typing that demonstrates remarkable generalizability across various systems. By employing a unique discretization method to encode word-gesture trajectories, our model effectively learns from a vast dataset of real and synthetic data. This pre-training enables the decoder to accurately predict user input across diverse word-gesture typing systems in AR and VR environments without the need for fine-tuning. The significance of our contribution lies in developing a universal solution that combines the ease of configuration with high decoding accuracy, addressing the limitations of existing approaches such as SHARK^2 and conventional neural decoders.
- **Why Our Method Works**: The success of our method can be attributed to the discretization of word-gesture trajectories. By converting continuous trajectories into discrete 'pixels', we simplify the input space and allow the model to focus on learning the essential patterns of user input, as illustrated by Figure 6. Notably, the conventional neural decoder predicts only 'quick', whereas the *Pre-Trained Neural Decoder* accurately predicts 'quickly'. This is illustrated in Figures 6b and 6d, where red 'pixels' signify high confidence in the



(a) Cartesian Trajectory for 'quickly' on QWERTY Keyboard.

(b) Heatmap Visualizing Model Output from the Conventional Neural Decoder.

(c) Heatmap Visualizing One-Hot Encoding of the Coarsely Discretized Trajectory for 'quickly'.

(d) Heatmap Visualizing Model Output from the *Pre-Trained Neural Decoder*.

Fig. 6: This illustration provides a visual comparison of both the input y and output π for the conventional neural decoder and *Pre-Trained Neural Decoder*, specifically in the context of decoding the word 'quickly'.

character on the y-axis at the time step t on the x-axis. This distinction is highlighted through the visualization of inputs: the Cartesian trajectory (Figure 6a) and the one-hot encoding of the coarsely discretized trajectory (Figure 6c). The one-hot encoding aligns with the neural decoder's output space, sharing the same dimensions in terms of the number of classes and time length, which facilitates learning. In contrast, for the conventional neural decoder, the input (i.e., the Cartesian trajectory) exists in a continuous space, differing from the output's discrete space representation, thereby complicating the learning process.

- **Adoption to Touch-Type Decoding**: While our study primarily focuses on word-gesture typing, the proposed approach can easily adapt to touch-type decoding. The discretization method can be applied to individual key presses, representing each touch point as a discrete 'pixel.' The model can learn to predict the intended characters based on the spatial distribution of touch points by training the neural decoder on a large dataset of touch-typing data. This adaptation would enable developing a robust and accurate touch-type decoder that can handle the challenges posed by the 'fat finger' problem and variations in user typing patterns.
- **Discretization as a Technique in other Applications**: In addition to its application in word-gesture typing, the discretization of continuous input has the potential to benefit various other machine learning problems. By converting continuous input into discrete representations, such as 'pixels' or bins, the complexity of the data pattern can be reduced, making it more manageable for machine learning algorithms. Potential applications that can benefit from the discretization of continuous input include eye tracking, where discretization can simplify the interpretation of eye movement data, and affective computing, where discrete emotional states can be mapped from continuous physiological signals.
- **The Potential Impact of Head-Mounted Display (HMD) Hand Tracking Accuracy on Word Prediction**: Current video see-through HMDs, such as Meta Quest [42] and Apple Vision Pro [2], offer robust hand tracking with real-time visual feedback, showing rendered illustrations of hands for mid-air interactions. For example, a rendered hand appears when poking a virtual keyboard, or a cursor shows during a mid-air pinch, creating a closed-loop interaction. This visual feedback allows users to adjust their hand movements, reducing the impact of inaccurate tracking on text entry accuracy. In contrast, optical see-through HMDs like HoloLens [3] enable open-loop interaction by relying on natural hand perception without rendered visuals. Shen et al. [50] investigated this by removing the visual feedback of a projected cursor on a mid-air gesture keyboard in HoloLens 2, finding that word prediction accuracy remained unaffected due to the robustness of conventional neural decoders. Our proposed *Pre-trained Neural Decoder*, tested on Shen et al. [50]'s dataset without visual feedback, achieved a Top-4 test accuracy of 90.1%, demonstrating its ability to handle hand tracking inaccuracies in HMDs effectively.

8 Limitations and Future Work

Our research, while comprehensive, is not without its constraints. The pre-trained decoder is optimized for QWERTY keyboards and may not perform as well with keyboards with different key arrangements. Researchers actively explored new layouts such as the Metropolis [63], Opti [38] and Dvorak [17] for potential benefits in ergonomics, typing efficiency, or language accommodation, despite that these alternative layouts introduce learnability and adoption challenges for users [13]. Our *Pre-trained Neural Decoder* may not work seamlessly with alternative layouts out of the box, as the discretization process assumes a specific key arrangement with QWERTY sequential order.

Furthermore, although we have successfully validated our model across four distinct datasets from various word-gesture typing systems in AR and VR environments, numerous other less common system designs exist, such as curved keyboards and eye-gaze-based word-gesture typing. While these were not explicitly tested, our model's functionality is rooted in its ability to process noisy and ambiguous trajectories.

Future work includes training the neural decoder on a larger, combined dataset and developing a transformer-based neural decoder. Initial tests with the transformer architecture showed poor performance, likely due to inadequate training data leading to overfitting. We aim to train the transformer-based decoder on a larger dataset and customize it to better accommodate long temporal patterns and meet latency requirements.

9 Conclusion

This paper introduces a novel *Pre-trained Neural Decoder* that demonstrates remarkable versatility and accuracy for word-gesture typing across diverse AR/VR systems. By discretizing complex gesture trajectories into coarse 'pixels' and pre-training on a large dataset, our model learns to accurately predict words from different datasets across various interaction modes and device platforms without requiring specific fine-tuning. Extensive evaluations on four challenging datasets showcase the *Pre-trained Neural Decoder*'s strong performance and generalizability, with an average Top-1 accuracy of 83.3% and Top-4 accuracy of 90.4% , which is a significant improvement over both conventional neural decoders and the popular SHARK^2 algorithm. The decoder also runs in real-time on mobile AR/VR hardware, enabling fluid gesture typing experiences. The proposed methodology illuminates a promising path towards universal **Gesture2Text** decoding that can enable efficient and expressive communication across new interactive contexts.

REFERENCES

[1] Haptic capacitive. https://sensel.com/product/#haptic-capacitve. 5

[2] Apple vision pro. https://www.apple.com/apple-vision-pro/, 2023. 9

[3] Microsoft hololens. https://www.microsoft.com/hololens, 2023. 9

[4] Use your mac with apple vision pro. https://support.apple.com/en-ca/118521, 2023. 5

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 4

[6] A. F. Agarap. Deep learning using rectified linear units (relu). *ArXiv*, abs/1803.08375, 2018. 4

[7] O. Alsharif, T. Ouyang, F. Beaufays, S. Zhai, T. Breuel, and J. Schalkwyk. Long short term memory neural network for keyboard gesture decoding. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2076–2080. IEEE, 2015. 1, 2, 3, 7

[8] J. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016. 4

[9] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, 2011. 4

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 3

[11] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 2

[12] S. Chen, J. Wang, S. Guerra, N. Mittal, and S. Prakkamakul. Exploring word-gesture text entry techniques in virtual reality. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–6, 2019. 2

[13] P. David. Clio and the economics of qwerty. *The American Economic Review*, 75:332–7, 05 1985. 9

[14] T. J. Dube and A. S. Arif. Text entry in virtual reality: A comprehensive review of the literature. In *Human-Computer Interaction. Recognition and Interaction Technologies: Thematic Area, HCI 2019, Held as Part of the 21st HCI International Conference, HCII 2019, Orlando, FL, USA, July 26–31, 2019, Proceedings, Part II 21*, pp. 419–437. Springer, 2019. 4

[15] J. J. Dudley, K. Vertanen, and P. O. Kristensson. Fast and precise touch-based text entry for head-mounted augmented reality with variable occlusion. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 25(6):1–40, 2018. 1, 5

[16] J. J. Dudley, J. Zheng, A. Gupta, H. Benko, M. Longest, R. Wang, and P. O. Kristensson. Evaluating the performance of hand-based probabilistic text input methods on a mid-air virtual qwerty keyboard. *IEEE Transactions on Visualization and Computer Graphics*, 2023. 1

[17] A. C. Dvorak, N. L. Dealey, and W. L. Merrick. Typewriter keyboard, 1936. Retrieved from Google Patents. 9

[18] M. Freitag and Y. Al-Onaizan. Beam search strategies for neural machine translation. In *NMT@ACL*, 2017. 4

[19] M. Gordon, T. Ouyang, and S. Zhai. Watchwriter: Tap and gesture typing on a smartwatch miniature keyboard with statistical decoding. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 3817–3821, 2016. 1

[20] A. Graves. Connectionist temporal classification. *Supervised sequence labelling with recurrent neural networks*, pp. 61–93, 2012. 3, 4

[21] Y. Gu, C. Yu, Z. Li, Z. Li, X. Wei, and Y. Shi. Qwertyring: Text entry on physical surfaces using a ring. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(4):1–29, 2020. 1

[22] A. Gupta, C. Ji, H.-S. Yeo, A. Quigley, and D. Vogel. Rotoswype: Word-gesture typing using a ring. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pp. 1–12, 2019. 1, 2

[23] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *arXiv: Computer Vision and Pattern Recognition*, 2015. 2

[24] F. Henninger, Y. Shevchenko, U. K. Mertens, P. J. Kieslich, and B. E. Hilbig. lab.js: A free, open, online study builder. *Behavior Research Methods*, 54(2):556–573, 2022. doi: 10.3758/s13428-019-01283-5 6

[25] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 3, 4

[26] F. Kern, F. Niebling, and M. E. Latoschik. Text input for non-stationary xr workspaces: Investigating tap and word-gesture keyboards in virtual and augmented reality. *IEEE Transactions on Visualization and Computer Graphics*, 29(5):2658–2669, 2023. 1

[27] T. Kim, A. Karlson, A. Gupta, T. Grossman, J. Wu, P. Abtahi, C. Collins, M. Glueck, and H. B. Surale. Star: Smartphone-analogous typing in augmented reality. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–13, 2023. 5

[28] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[29] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *European conference on machine learning*, pp. 217–226. Springer, 2004. 5, 6

[30] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022. 3

[31] P. O. Kristensson. *Discrete and continuous shape writing for text entry and control*. PhD thesis, Institutionen f"or datavetenskap, 2007. 2

[32] P.-O. Kristensson and S. Zhai. Shark2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, pp. 43–52, 2004. 1, 2, 3, 7

[33] L. H. Lee, K. Y. Lam, Y. P. Yau, T. Braud, and P. Hui. Hibey: Hide the keyboard in augmented reality. In *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*, pp. 1–10. IEEE, 2019. 1

[34] L. A. Leiva, S. Kim, W. Cui, X. Bi, and A. Oulasvirta. How we swipe: a large-scale shape-writing dataset and empirical findings. In *Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction*, pp. 1–13, 2021. 1, 2, 3, 4

[35] C. Liang, C. Hsia, C. Yu, Y. Yan, Y. Wang, and Y. Shi. Drg-keyboard: Enabling subtle gesture typing on the fingertip with dual imu rings. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 6(4), article no. 170, 30 pages, jan 2023. doi: 10.1145/3569463 1, 6

[36] H. Liang, J. Yuan, D. Thalmann, and N. M. Thalmann. Ar in hand: Egocentric palm pose tracking and gesture recognition for augmented reality applications. In *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 743–744, 2015. 5

[37] I. S. MacKenzie and R. W. Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI'03 extended abstracts on Human factors in computing systems*, pp. 754–755, 2003. 5, 6

[38] I. S. MacKenzie and S. X. Zhang. The design and evaluation of a high-performance soft keyboard. CHI '99, 7 pages, p. 25–31. Association for Computing Machinery, New York, NY, USA, 1999. doi: 10.1145/302979.302983 9

[39] A. Markussen, M. R. Jakobsen, and K. Hornbæk. Vulture: a mid-air word-gesture keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1073–1082, 2014. 1, 2

[40] Meta. Quest v56 software update: Hand tracking improvements, live captions, facebook livestreaming, 2023. 4

[41] Meta AI. Pytext: A natural language modeling framework based on pytorch. https://github.com/facebookresearch/pytext, 2018. Accessed: insert date here. 4

[42] Meta Platforms, Inc. Oculus Quest Series. https://www.oculus.com/quest/. Accessed: 2024-03-10. 2, 4, 9

[43] D. Mifsud, A. S. Williams, F. R. Ortega, and R. J. Teather. Augmented reality fitts' law input comparison between touchpad, pointing gesture, and raycast. *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 590–591, 2022. 4, 5

[44] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. Fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 48–53, 2019. 4

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. https://pytorch.org, 2019.

Accessed: [insert date here]. 8

[46] A. Peshock, J. Duvall, and L. E. Dunne. Argot: A wearable one-handed keyboard glove. In *Proceedings of the 2014 ACM international symposium on wearable computers: adjunct program*, pp. 87–92, 2014. 1

[47] S. Reyal, S. Zhai, and P. O. Kristensson. Performance and user experience of touchscreen and gesture keyboards in a lab setting and in the wild. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 679–688, 2015. 1, 2

[48] 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. IEEE, 2021. 4

[49] 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. IEEE, 2021. 2, 4, 6

[50] J. Shen, J. Dudley, and P. O. Kristensson. Fast and robust mid-air gesture typing for ar headsets using 3d trajectory decoding. *IEEE Transactions on Visualization and Computer Graphics*, 2023. 1, 2, 3, 4, 5, 7, 9

[51] J. Shen, J. Hu, J. J. Dudley, and P. O. Kristensson. Personalization of a mid-air gesture keyboard using multi-objective bayesian optimization. In *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 702–710. IEEE, 2022. 2, 5

[52] M. Speicher, A. M. Feit, P. Ziegler, and A. Krüger. Selection-based text entry in virtual reality. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pp. 1–13, 2018. 5

[53] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012. 4

[54] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017. 8

[55] K. Vertanen and P. O. Kristensson. Mining, analyzing, and modeling text written on mobile devices. *Natural Language Engineering*, 27:1 – 33, 2019. 7

[56] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018. 5

[57] Y. Wang, Y. Wang, J. Chen, Y. Wang, J. Yang, T. Jiang, and J. He. Investigating the performance of gesture-based input for mid-air text entry in a virtual environment: A comparison of hand-up versus hand-down postures. In *Sensors*, vol. 21, p. 1582. Multidisciplinary Digital Publishing Institute, 2021. 2

[58] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Association for Computational Linguistics, Online, Oct. 2020. doi: 10.18653/v1/2020.emnlp-demos.6 2

[59] W. Xu, H.-N. Liang, A. He, and Z. Wang. Pointing and selection methods for text entry in augmented reality head mounted displays. In *2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 279–288. IEEE, 2019. 1, 2

[60] Z. Xu, P. C. Wong, J. Gong, T.-Y. Wu, A. S. Nittala, X. Bi, J. Steimle, H. Fu, K. Zhu, and X.-D. Yang. Tiptext: Eyes-free text entry on a fingertip keyboard. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, pp. 883–899, 2019. 1

[61] N. Yanagihara, B. Shizuki, and S. Takahashi. Text entry method for immersive virtual environments using curved keyboard. In *25th ACM Symposium on Virtual Reality Software and Technology*, pp. 1–2, 2019. 2

[62] C. Yu, Y. Gu, Z. Yang, X. Yi, H. Luo, and Y. Shi. Tap, dwell or gesture? exploring head-based text entry techniques for hmds. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pp. 4479–4488. ACM, 2017. 1, 2

[63] S. Zhai, M. Hunter, and B. A. Smith. The metropolis keyboard-an exploration of quantitative techniques for virtual keyboard design. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pp. 119–128, 2000. 9

[64] S. Zhai and P.-O. Kristensson. Shorthand writing on stylus keyboard. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 97–104, 2003. 2

[65] S. Zhai and P. O. Kristensson. The word-gesture keyboard: reimagining keyboard interaction. *Communications of the ACM*, 55(9):91–101, 2012. 2

[66] M. Zhao, A. M. Pierce, R. Tan, T. Zhang, T. Wang, T. R. Jonker, H. Benko, and A. Gupta. Gaze speedup: Eye gaze assisted gesture typing in virtual reality. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pp. 595–606, 2023. 1