StegoType: Surface Typing from Egocentric Cameras

Mark Richardson
Meta Reality Labs
Seattle, WA, USA
echo@meta.com

Fadi Botros
Meta Reality Labs
Redmond, WA, USA
fadibotros@meta.com

Yangyang Shi
Meta Reality Labs
Redmond, WA, USA
yyshi@meta.com

Bradford Snow
Meta Reality Labs
Redmond, WA, USA
brsnow@meta.com

Pinhao Guo
Meta Reality Labs
Redmond, WA, USA
pinhaoguo1@meta.com

Linguang Zhang
Meta Reality Labs
Redmond, WA, USA
linguang@meta.com

Jingming Dong
Meta Reality Labs
Redmond, WA, USA
djingming@meta.com

Keith Vertanen*
Michigan Technological University
Houghton, MI, USA
keith@keithv.com

Shugao Ma
Meta Reality Labs
Redmond, WA, USA
shugao@meta.com

Robert Wang
Meta Reality Labs
Redmond, WA, USA
rywang@meta.com

ABSTRACT

Text input is a critical component of any general purpose computing system, yet efficient and natural text input remains a challenge in AR and VR. Headset based hand-tracking has recently become pervasive among consumer VR devices and affords the opportunity to enable touch typing on virtual keyboards. We present an approach for decoding touch typing on uninstrumented flat surfaces using only egocentric camera-based hand-tracking as input. While egocentric hand-tracking accuracy is limited by issues like self occlusion and image fidelity, we show that a sufficiently diverse training set of hand motions paired with typed text can enable a deep learning model to extract signal from this noisy input. Furthermore, by carefully designing a closed-loop data collection process, we can train an end-to-end text decoder that accounts for natural sloppy typing on virtual keyboards. We evaluate our work with a user study (n=18) showing a mean online throughput of 42.4 WPM with an uncorrected error rate (UER) of 7% with our method compared to a physical keyboard baseline of 74.5 WPM at 0.8% UER, showing progress towards unlocking productivity and high throughput use cases in AR/VR.

CCS CONCEPTS

• **Human-centered computing** → **Text input**; Virtual reality; • **Computing methodologies** → Natural language generation.

*Keith Vertanen contributed the statistical decoder experiments in Section 6.1: Study 5.

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0628-8/24/10
https://doi.org/10.1145/3654777.3676343

KEYWORDS

text input; hand-tracking; mixed reality, augmented reality; virtual reality

ACM Reference Format:
Mark Richardson, Fadi Botros, Yangyang Shi, Bradford Snow, Pinhao Guo, Linguang Zhang, Jingming Dong, Keith Vertanen, Shugao Ma, and Robert Wang. 2024. StegoType: Surface Typing from Egocentric Cameras. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3654777.3676343

Figure 1: A person in mixed reality performs a transcription task on a virtual keyboard superimposed on a flat surface, with StegoType mapping hand motion into typed keys.

1 INTRODUCTION

Hand-tracking has gained popularity as a low-friction mode of interaction in AR and VR, alleviating the need for controllers to enable human computer interaction (HCI) that mirrors how we interact in the physical world. Despite this evolution, natural text input remains a challenge. While egocentric headset based hand-tracking is sufficient for coarse heuristic interactions and socially acceptable self-presence in VR, its accuracy is limited due to issues like self occlusion and image fidelity. Existing VR solutions for hand-based text entry including raycast keyboards or poke-based mid-air keyboards primarily utilize thumb and index fingers. This is likely due to the relative ease for which these digits can be tracked from an egocentric perspective. However, the throughput afforded by thumb or index finger based methods is limited [9, 11].

Other technologies for text entry exist but suffer their own drawbacks. Speech recognition enables high throughput input but may not be socially acceptable in public spaces and lacks effective mechanisms for correction, thus limiting applicable use cases. Gesture typing on virtual keyboards can increase throughput with only thumb or index-finger tracking but is limited to dictionary word decoding, and because mistakes occur at a word granularity, corrections are costly [11, 22].

Multi-finger typists can achieve higher words per minute (WPM) on physical keyboards than two finger typists [7]. Enabling multi-finger surface typing in AR/VR could unlock higher throughput text entry [9], but in practice this is challenging because of limited hand-tracking accuracy.

This work demonstrates that multi-finger touch typing on flat surfaces can be achieved on a consumer VR headset, relying only on egocentric hand-tracking as input. Following the direction of Richardson et al. [34], we also train an end-to-end motion model which maps hand motion onto typed text. However, we replace the OptiTrack motion capture based hand-tracking used by Richardson et al. with egocentric camera based bare hand-tracking. We show that with a sufficiently diverse training dataset of hand motions paired with typed text, we can train a deep learning model that extracts signal from noisy egocentric hand-tracking and decodes touch-typing on a virtual keyboard.

The contributions of this work are as follows:

(1) A real-time system integrating imperfect (egocentric and markerless) hand-tracking with a downstream text decoding model on a compute constrained device.
(2) A scalable closed-loop data collection setup capturing natural typing behaviors including corrections and mistakes as they naturally occur on virtual keyboards
(3) A modern expressive machine learning model for decoding surface typing with generalization across typists. Notably, we tailor a speech-inspired architecture to our domain by modeling left and right hand input and explicitly addressing key emission latency.

2 RELATED WORK

2.0.1 Text decoding on virtual keyboards. Soft keyboards are the dominant form of text input for mobile phones and tablets. Goodman et al. [14] introduced a text decoding method that combines key touch distributions (modeled as bi-variate Gaussians) and language models (modeled with n-grams). Yan et al. [28] extend the modeling of the key touch distribution with rotational dual Gaussians. Vertanen et al. extended soft keyboard text decoding to use sentence-level decoding [43] and watch-sized surfaces [42].

Grady et al. [16] show that a single camera can be used to detect touch down events and locations on a surface. Grady et al. [15] extend this work to diverse surfaces and create a soft keyboard prototype that achieves 26 WPM. In our work, we do not model individual touch events as an intermediate representation, but rather decode the motion of the fingers directly.

2.0.2 AR/VR text entry. Mid-air typing has been a popular approach for spatial computing in AR/VR and imposes the minimal constraints on the user. ATK [46] uses a LeapMotion sensor to decode 10 finger tapping in mid-air. Dudley and colleagues have explored both tapping in mid-air with the VISAR keyboard [10] and gesture typing [11]. Yi et al. [45] apply probabilistic touch detection to improve the accuracy of mid-air tapping. Shen et al. [36] apply 3D trajectory decoding to mid-air gesture typing. While mid-air text input offers ultimate convenience for spatial computing, it can be more fatiguing and slower than typing on a surface [6].

Recent work that leverages the comfort of a surface require additional instrumentation, e.g., from touchpads or inertial sensors on the fingers or the wrist. TapGazer [20] combines touchpad-based tap detection with gaze. TypeAnywhere [47] uses accelerometers on each finger to accurately decode finger taps into text at average speeds of 71 WPM. TapID [31] combine two wristbands (with inertial sensors) and egocentric hand-tracking to facilitate tap event identification, tap finger identification and ultimately, typing. TapType [40] extends TapID by using a more advanced Bayesian neural network to decode text, discharging the reliance on egocentric hand-tracking and achieving average speeds of 19 WPM. Both TypeAnywhere and TapType rely on language models to decode the ambiguous input. Our work pushes the frontier of using egocentric hand-tracking alone to facilitate accurate touch typing, notably even without a language model.

We are inspired by the work of Richardson et al. [34] that decoded touch typing on a surface from vision-based signals. However, Richardson et al. relied on near-perfect hand-tracking from a multi-camera motion capture system and marker gloves. Our work instead aims to use noisier egocentric hand-tracking without the use of special gloves.

2.0.3 Image/video based action recognition. Our approach of recognizing key presses from cameras is an instance of video action recognition, which has been typically addressed with 3D-CNNs [5, 12] and transformers [2, 4]. Pose-based action recognition extends this work to explicit consideration of skeletal full-body [8, 26] or hand pose [35]. Our work ultimately combines both pose information (i.e., finger coordinates) and image encodings (latent embeddings) as input for supervision.

2.0.4 Automatic speech recognition. We follow recent work on end-to-end automatic speech recognition (ASR) [23] by adopting the Connectionist Temporal Classification (CTC) loss [18, 32] and the Emformer architecture [37], a memory efficient transformer. Similar to streaming ASR systems, our streaming text decoder can be

extended with a language model using the popular RNN-transducer framework [17, 24].

3 APPROACH

We aim to enable touch typing in VR to capitalize on a user's existing physical typing skills, eschewing the need for other peripherals and using only egocentric hand-tracking as input. We construct an environment where users don a VR headset and sit in front of a virtual keyboard superimposed upon a real-world physical surface. The headset contains a stereo pair of downward-facing cameras, calibrated to enable the hand-tracking methodology of UmeTrack [19]. Cameras frames are sampled at 30 Hz and UmeTrack then yields a stream of hand pose features at the same frequency. The regressed pose features are used to skin hand meshes which the user then sees as their own hands, thus enabling placing their hands on the virtual keyboard to type (Figure 1).

The hand poses estimated from egocentric hand-tracking have limited accuracy, often due to fundamental issues with the camera locations. In typing poses the fingertips and distal joints are often self-occluded, making the pose estimation problem ambiguous (Figure 7). This especially affects ring and pinky fingers which typically only have proximal phalanges visible. Rather than treating typing as modeling a set of touch contacts with keys, which would be directly compromised by hand-tracking errors of the distal joints, we instead opt for a modeling approach that maps hand motion onto typed text.

4 TRAINING DATA

Training and evaluating a model to map hand motion onto key-presses necessitates a training corpus of hand motion labeled with key-presses. We construct a data collection lab study that mimics our desired online VR experience while allowing us to collect ground-truth information about the key presses that correspond to given hand motion.

During data collection, participants wear a head-mounted stereo pair of downward pointing cameras that see the active typing area 2. In front of the participant is a printed paper keyboard with an industry standard 19 mm key pitch affixed to a surface (similar to what would be projected in VR). Both the head-mounted cameras and the surface keyboard are tracked using a marker based OptiTrack motion capture setup.

The paper keyboard was affixed on top of a pair of Sensel pressure sensitive touchpads mounted inside a milled aluminum board to achieve a large flat surface with a touch-sensitive region under the keys we are modeling (Figure 3).

4.1 Open-loop data collection

In front of the participant is a monitor that displays short text prompts to be transcribed on the paper keyboard. Following the direction of [34], we initially experimented with an open-loop setup where participants would type without any feedback on the paper keyboard and then self-declare whether they felt they performed the transcription correctly. We recorded data from researchers on our team yielding a dataset of hand motion paired with the text we *intended* to produce.

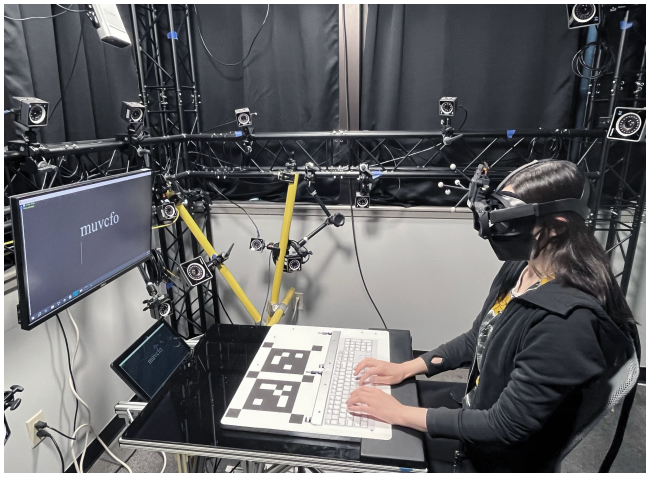


Figure 2: A data collection setup with a typist sporting a head-mounted stereo camera rig. The user is typing on a touchpad keyboard in front of a monitor that displays the transcription prompts and their typed text.

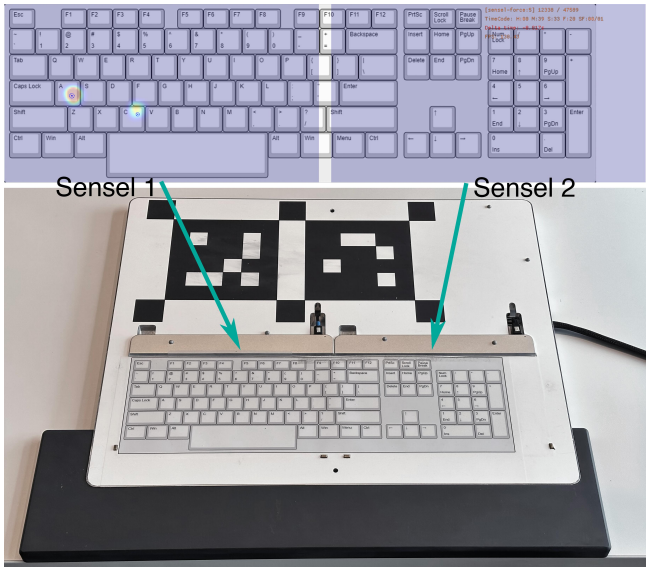


Figure 3: (Top) The output of the touchpads composed into a full-size keyboard layout which can turn contacts into key-presses. (Bottom) Two Sensel boards embedded into a milled aluminum plate creating a large flat surface with active sensing area underneath a printed keyboard layout. Aruco tags are used to verify the motion capture surface tracking alignment.

The scale of data recorded from researchers alone, while sufficient for training user-specific models achieving an average of 6.1% CER, was insufficient to train a user-generic model (K-folds cross validation measured cross-user CER at 24.6%). Presuming training data diversity to be the primary problem we sought increased data scale by recruiting data collection participants from the public, but

then found that these participants made frequent undeclared transcription errors causing mismatched hand-motion/intended text. Even as we scaled up to 40 users in our training set, cross-user performance still plateaued around 18.2% CER.

We analyzed participants' typos and categorized them into two types of errors:

(1) **Slop errors**: Slop errors result from a user over or under-reaching for intended keys, leading to a neighboring key being hit. This is especially common on soft keyboards without the haptics of key-edges to help maintain hand-keyboard alignment. In these cases a user intended to hit a different key than the one they physically hit.
(2) **Compliance errors**: Any error which is not a slop error is a compliance error. This includes not making contact when trying to hit a key, hitting a spurious key with another finger, or making a transcription mistake such as reading and then typing the wrong word. We posit that when a compliance error is made that users should have less expectation for the system to deviate from what they physically did.

Our objective is to train a model to produce the keys a typist intends to be produced. To this end, slop errors should be included in data collection labeled according to the key a user meant to hit. However, compliance errors are not necessarily aligned with this objective. For example, if a participant misread a word from a prompt they would not expect that word to be produced given their typing motion. In a post-hoc analysis of data collected in an open loop setting, we found that participants made compliance errors in over 40% of transcriptions. These had either the wrong number of contacts for the number of characters in the prompt or they had a touchpad contact more than 30 mm away from the key that should have been hit.

Another weakness of open-loop is that it does not naturally exercise the backspace key. To obtain training labels with backspace, we need to prompt the user to use backspace at seemingly random points within a prompt. Transcribing text with backspaces randomly interspliced is a much more challenging task than transcribing natural language, closer in complexity to transcribing a password or other random string which further exacerbates compliance errors. Additionally prompted backspaces often result in motion that is different from natural backspaces from reacting to an error, which introduces a domain gap between training and live testing.

4.2 Closed-loop data collection

To address both compliance errors and unnatural backspace motion we pivoted to an interactive closed-loop data collection setup similar to using a real virtual keyboard. We built an online keyboard decoder using the pair of Sensel touchpads underneath a paper keyboard layout. The display showed both the prompt and live feedback about the user's transcription. In this setup the prompt was only used to guide users, but the sequence of keys a participant physically touched on this touchpad keyboard were recorded as the ground truth labels in the resulting dataset. Participants can make and see their own mistakes and naturally correct them with backspace, yielding a dataset with labeled realistic motions of backspace key presses.

We empirically tuned two thresholds on the Sensel touchpad keyboard, retaining contacts over 15 g of force and more than 3 mm in contact width, so that nearly any finger-touch on the touchpad would produce a key-event. These threshold exists to eliminate false positive events caused by sensor noise and artifacts. For example, because we place a paper keyboard on top of the touchpad for visual reference, this affixed sheet of paper can cause spurious low-force events which we filter out in this way. Because the force threshold is sufficiently low, typists can typically type with whatever force they find comfortable and non fatiguing, but at the expense of not being able to rest their fingers during data collection.

This setup resolves the issue of compliance errors, but it also inadvertently removes slop errors. If a user over or under-reaches for a key, our touchpad keyboard would produce labels according to the keys they physically touched instead of those they intended to touch. Without representation of slop errors, our model cannot learn to read past them and will be limited to the geometric accuracy of users' behavior.

4.3 Closed-loop intent data collection

In data collection we have access to a privileged piece of information: the transcription prompt. We generally assume users are trying to type the prompt, which can then be leveraged to bias our touchpad keyboard. Similar to Dudley et al. [9], we develop a prompt-based oracle that biases the decoded keys to match the prompt. For instance, when a prompt first appears, we bias the key corresponding to the first character in this prompt, inflating the key bounding box by 13 mm on each side (Figure 4). If the user physically touches inside this bounding box (which includes part of each physically neighboring key) we record an *intent label* corresponding to the key biased by the oracle.

In our apparatus design we chose the 13 mm key bias size empirically. Our researchers underwent pilot data collection and found this bias captured all intentional key-presses without us having to consider finger placement accuracy. Evaluating all of the (non-pilot) data subsequently captured with this apparatus, 99% of key-presses are within an 8.5 mm inflated bounding box indicating that a 13 mm inflation is aggressive. The data produced by this apparatus thus reflects an extreme where users are being as sloppy as they want, and our offline analysis thus represents our ability to cope with that extreme.

Users make mistakes, so they may hit outside of this biased key bounding box resulting in a key that no longer matches the prompt. In this case we know neither the key they intended to hit nor which key should come next. The oracle is said to be "off-track" and requires the user backspace until the transcription again matches some prefix of the prompt before the oracle key biasing is re-enabled. When the oracle is off-track, the backspace key is exclusively biased (Figure 4). Because intent labels can only be generated when the oracle is on-track, and our goal is to collect more intent labels than typo labels, we played an audio buzzer sound for any press that was off-track.

Overall, our closed-loop intent data collection effectively leverages compliance errors, i.e., mistakes, to collect natural backspace key presses, while mapping slop errors to their intended key press for providing ground truth labels.

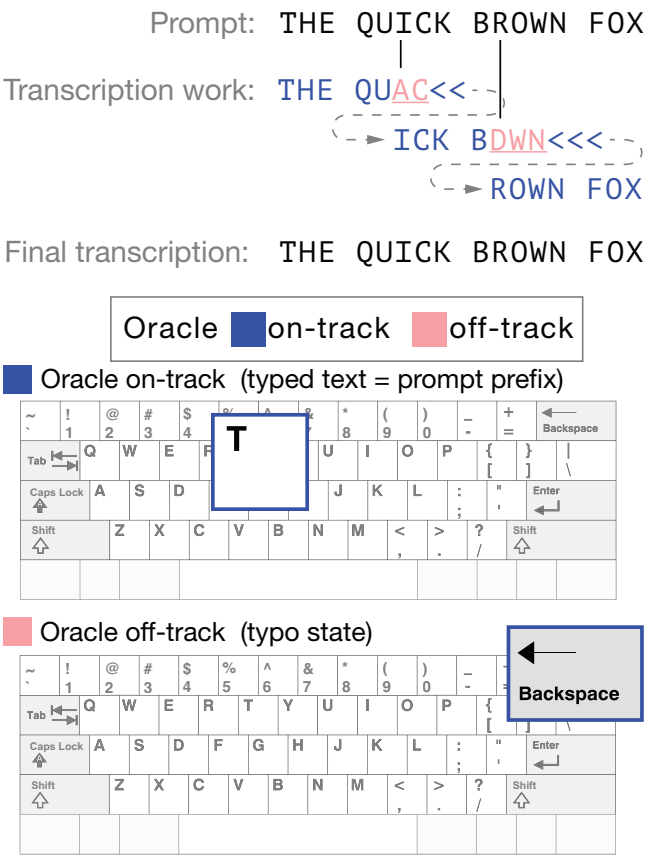


Figure 4: (Top) An oracle with knowledge of a transcription prompt biases the key corresponding to the expected next character to be typed. (Bottom) When a typo occurs, the backspace key is biased until the transcription is deleted back to an error-free prefix of the prompt.

4.4 Participant instruction

We recruited typists to participate in data collection where each participant performed transcription over a 90 minute session. The session was broken up into 5 minute blocks separated by 1 minute breaks, with additional breaks as needed. Each block consisted of prompts drawn from a specific corpus, sometimes with additional verbal guidance, with the participant completing as many prompts as time allowed. The blocks were structured as follows:

(1) Pangrams (1 block): Phrases of English words strung together so each prompt contains every letter of the alphabet, e.g. "the quick brown fox jumps over the lazy dog"
(2) Random repeats (2 blocks): Prompts containing either one or two random keys separated by a space. Each random key was repeated a random number of times, e.g. "LLL ;;"
(3) DailyDialog (6 blocks): Natural language phrases sampled from the DailyDialog corpus [25]. In 30% of these blocks participants are instructed to only type with index fingers and thumbs

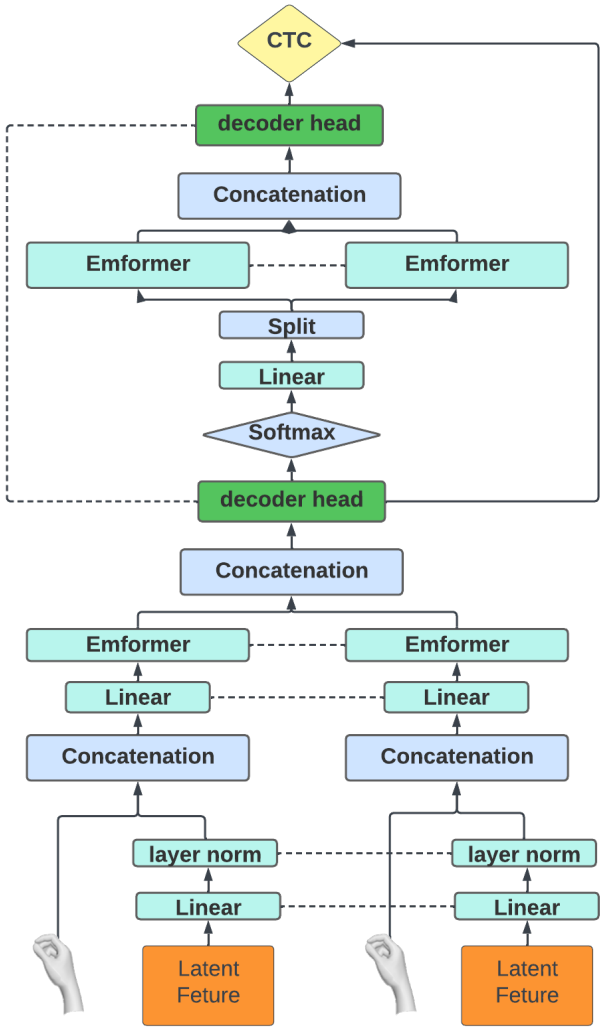


Figure 5: Motion model architecture. The boxes connected by dotted lines share the same weight. The blue boxes are operations which don't have weights.

Because this work focuses on touch typing, we screened for participants with touch typing ability. Candidates were given a 60 second typing test on a physical keyboard and had to sustain 45 WPM to be included in our data collection.

5 MOTION MODEL

Figure 5 illustrates the comprehensive architecture of the motion model, which is an adaptation of the Emformer architecture used in ASR to the task of bi-manual typing from hand-tracking inputs. We describe the input feature, the backbone model, the loss function and its regularization methods in the following subsections.

5.1 Input features

The pose features produced by UmeTrack consist of a single hand scale factor, 20 skeletal joint angles, a wrist transform and position,

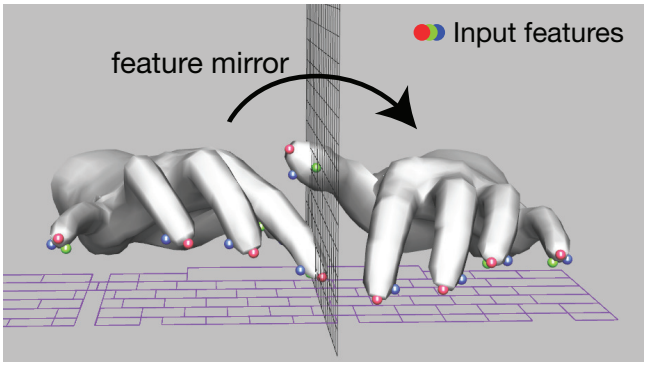


Figure 6: The pose input features to the motion model consist of three vertices per fingertip measured relative to the keyboard coordinate frame. For the Siamese model architecture we mirror the right hand features over the keyboard midline to capitalize on hand motion symmetry.

and a hand confidence score for each hand. Together these can be used to scale and rig a template hand mesh from which we can select specific vertices of interest. The input features for our model consist of the two scalar hand-confidence scores along with pose features composed of three vertices per fingertip (Figure 6). These three finger vertices sufficiently represent the 6-degree of freedom coordinate transform of each fingertip.

The fingertips from the hand are measured relative to a coordinate frame centered on the virtual keyboard. To reduce overfitting and false positives, we clamp features in a bounding box around the active keyboard such that finger vertices outside of this box are replaced with a placeholder value of $(0, 0, 0)$.

Along with these pose features we also incorporate a latent feature vector extracted between the multiview fusion module and the temporal model inside the UmeTrack model. The temporal model is trained using an acceleration loss to reduce high frequency jitter, which results in a smooth hand motion for easier interactions in VR. However, fast finger movements in surface typing often resemble high frequency jitter and are inadvertently smoothed by the temporal model, leading to a loss of information. Moreover, hand pose alone would not capture valuable image information such as hand-surface shadows from contact. We therefore rely on a latent feature vector to compensate for pose. The original latent feature vector has dimensionality of 960, which is infeasibly large for training long sequences. We therefore train a fully connected layer to project the feature to 128. The fully connected layer is trained using a loss function that enforces the pairwise feature distances to be the same before and after the dimension reduction:

$$\mathcal{L}_{reduction} = \frac{1}{N^2} \sum_{i=0}^{N} \sum_{j=0}^{N} (d(F_i, F_j) - d(f_i, f_j))^2 \quad (1)$$

where N is the batch size, F is the 960-D latent feature, f is the reduced 128-D latent feature, and $d(.)$ denotes the L_2 distance. We also tried using an autoencoder to perform dimension reduction but found its performance to be slightly worse. The reduced latent feature undergoes a linear projection layer followed by layer normalization, as depicted in Figure 5. The output is then concatenated with the pose features, serving as the input for the backbone model. The roles of the linear projection layer and layer normalization are crucial in maintaining model stability during quantization.

5.2 Backbone model

Capitalizing on motion symmetry of the left and right hands, we apply a "Siamese" style architecture across the hands. Specifically, we apply two separated models with shared weights to the left and mirrored-right hand features as shown in Figure 6, then concatenate the output embeddings to be fed into a final linear projection and softmax head to produce our final distribution. In Figure 5, the boxes connected by the dotted lines are the identical components with the same learnable weights.

The temporal nature of hand motion lends itself to modeling by sequence neural networks. We apply a transformer based motion model to a 30 Hz stream of hand pose features, where for every frame of hand pose we predict a corresponding categorical distribution over the set of possible keys, with an additional label for the common case of no key being pressed.

A keyboard should produce keys as they are hit, which necessitates the motion model to operate in a streaming fashion. In the backbone model, the Emformer [37, 38] layer, a variant of transformer which enables efficient incremental self attention as new frames of input data become available. Our backbone model used 8 Emformer layers. Each Emformer layer uses a macaron structure [27] with a masked attention window of left context size 40, a convolutional kernel size of 7, and 256 channels per layer. This model has a total of 12 million parameters with an effective temporal receptive field of 12.3 seconds.

5.3 Model Training

Our training dataset consists of 268 K transcriptions recorded from 606 typists. We train models for 48 epochs using a batch size of 64 where each item in the batch consists of a variable (depending on how long the transcription took the participant) number of frames of hand features, along with a training label containing the sequence of keystrokes produced by the touchpad keyboard during data collection. The touchpad keyboard produces keypresses with known timestamps which could enable supervising with a frame-wise cross entropy loss, but we opt to instead use the Connectionist Temporal Classification (CTC) loss function [18] which allows the model to learn the optimal alignment from the data. For slow or subtle keypresses the model benefits from seeing more follow through motion, and using learned alignments with CTC allows the model to vary its latency as needed. By default, however, we find that CTC trained models tend to suffer from high emission latency in general.

People can perceive contact latency as small as 20ms [21] and many consumer USB keyboards operate around this latency range [44]. In order to minimize the latency of our decoding strategy while also retaining the advantages of variable latency we introduce a regularization term $\mathcal{L}_{latency}$ in training that minimizes the KL divergence between the model output distribution $\mathbf{P}_t$ and a target distribution set to the model output one frame later, $\mathbf{P}_{t+1}$ (with

a temperature hyperparameter τ applied to both the source and target softmax distributions):

$$\mathcal{L}_{latency}(\mathbf{P_t}) = \sum_{x \in X} P_t(x) \log\left(\frac{P_t(x)}{P_{t+1}(x)}\right) \tag{2}$$

$$P_t(x) = \frac{\exp(y_t(x)/\tau)}{\sum_{x \in X} \exp(y_t(x)/\tau)} \tag{3}$$

$$\mathcal{L} = \mathcal{L}_{CTC} + w_{L-reg} * \mathcal{L}_{latency} \tag{4}$$

If the model originally predicted a key-press at time t_0 then this regularizer encourages the model to instead predict at $t_0 - 1$, effectively pushing predictions to occur earlier in time. We use a weighted combination of this regularizer with the original CTC loss where the loss weights are determined empirically.

5.4 Label dependence

The CTC loss function can suffer from the label independence assumption, so we adopt the approach of [33] in which softmax heads are attached to select intermediate layers in the model and are supervised with the same loss function that supervises the final model output. Intermediate layers in the model then predict labels which are linearly projected back to the dimensionality of the model trunk and fed into the subsequent layer (Figure 5), effectively enabling later layers in the network to leverage label predictions from previous time steps as context.

The total loss of the entire model is calculated as a weighted combination of the intermediate CTC losses and the final output CTC loss, as expressed in the following equation:

$$\mathcal{L}_{CTC} = \mathcal{L}_o + \sum_i w_i * \mathcal{L}_i$$

Here, $\mathcal{L}_o$ represents the loss derived from the final Emformer layer output, while $\mathcal{L}_i$ denotes the loss from the intermediate output. In this study, the intermediate outputs from the 3rd and 6th Emformer layers were utilized. The weight w_i of the intermediate loss was 0.3.

6 EVALUATION

We segment our evaluations into offline and online. Since our method predicts individual keys as they are hit, we focus our evaluation on character level error. In offline evaluation we use character error rate, and in online evaluation we report uncorrected character error rate and WPM.

6.1 Offline evaluation

We stand up two offline evaluation datasets collected using the same rig as our training data. The first dataset includes 1448 transcriptions from 11 participants who are not represented in our training dataset. Prompts are drawn from the MacKenzie and Soukoreff phrase corpus [30], giving coverage of natural language typing. The second dataset includes 223 transcriptions from 5 participants with prompts of random n-grams, exercising non-language typing.

As described in Section 4, ground truth key-press labels can either correspond to typo or non-typo (intent) events. For typo labels we cannot infer which key the user meant to hit (if any), therefore measuring our model's prediction accuracy on these events is not related to our objective of predicting intended key-presses. In addition to standard character error rate (CER) we also calculate a metric, intent character error rate (I-CER) which captures the error rate specific to the intent subset of events. A detailed derivation of I-CER is provided in the Appendix.

Study 1: Model architecture Richardson et al. [34] applied a temporal convolutional network (TCN) based on the architecture [3] to map marker based hand-tracking onto typed text. However, in addition to requiring near perfect hand-tracking, Richardson trained trained a different model for each typist in the evaluation set–cross-user generalization remained unaddressed. We compare this TCN architecture to the Emformer based architecture presented in this work. For fairness we construct a TCN with a similar parameter count and receptive field to our Emformer architecture models. Our TCN implementation has 8 layers with a convolution kernel size of 48, a dilation factor of 1, and 768 channels per layer. All Emformer variants tested use a trunk with 8 layers, a self-attention window of 40, a convolutional kernel size of 7, and 256 channels per layer. The results of these comparisons are summarized in table 1 which shows that the Siamese Emformer outperforms all other architectures on accuracy. The Emformer's state-keeping allows it to save compute when running incrementally so that, even with a similar parameter count, the number of operations required per input frame is an order of magnitude lower than the TCN baseline, allowing Emformer models to run at 30 Hz on a compute constrained Quest 3 headset.

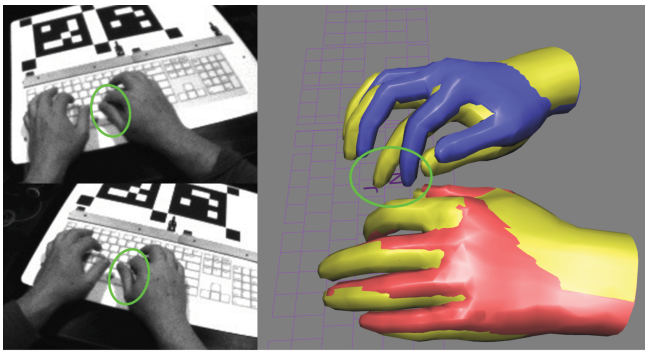
Study 2: Latency regularization The impact of latency regularization as described in Section 5 is measured by comparing the timesteps at which the model emits predictions to the ground truth times for the corresponding labels as determined by the touchpad during data collection. Corresponding model predictions and labels is a nontrivial task since the CTC-trained model is allowed variable emission latency so there is an inconsistent time offset between the two streams. We take an approach inspired by forced alignment which establishes correspondences taken from the alignments that yield the minimum edit distance. A full derivation of this approach is presented in Appendix B.

This metric is used to evaluate a set of models where the latency loss weight hyperparameter w_{L-reg} is varied, with results summarized in Table 1. We see an inverse correlation between accuracy and latency, and for our user study we select the label-dependent Siamese Emformer model with a latency regularization weight of 0.5 with reasonable accuracy and an average latency of 27 ms (0.81 frames at 30 Hz), in the same regime as physical keyboards.

Study 3: Latent features By training two identical architecture models, one which includes latent feature inputs and one which only operates on pose-based features, we find that latent features reduce overall intent error rate by a relative 6% while also decreasing latency by 18ms on the MacKenzie and Soukoreff evaluation set (Table 2). By extracting samples where the latent-feature model significantly outperforms the pose-only model we can gain insight into the types of cases where latent features offer benefit. A canonical example is shown in Figure 7 where a typist's index finger is partially occluded by their thumb as they strike a key. The pose estimated by UmeTrack curls the occluded fingertip more than it should which places the estimated fingertip one key lower on the keyboard. The authors of UmeTrack [19] suggest the temporal

Model	Params	Incremental ops	Receptive field (frames)	L-reg weight	Latency (frames)	CER	I-CER
TCN	10.1 M	348 M	377	0.5	0.82	12.01%	10.27%
Emformer	12.3 M	37 M	369	0.5	0.97	8.28%	6.32%
Siamese Emformer	12.3 M	37 M	369	0.5	0.79	7.85%	5.88%
Label-Dependent Siamese Emformer	12.4 M	38 M	369	1.0	0.37	7.12%	5.10%
Label-Dependent Siamese Emformer	**12.4 M**	**38 M**	**369**	**0.5**	**0.81**	**7.02%**	**5.03%**
Label-Dependent Siamese Emformer	12.4 M	38 M	369	0.1	3.24	6.44%	4.40%
Label-Dependent Siamese Emformer	12.4 M	38 M	369	0.0	6.28	6.31%	4.20%

Table 1: A comparison of model architectures and supervision strategies. The label-dependent Siamese Emformer outperforms all other models, while varying latency regularization weighting trades off latency for accuracy. We use a latency regularization weight of 0.5 (in bold) for our online studies. Despite similar parameter counts, the Emformer architecture requires much less compute than the TCN baseline when operating incrementally.



Latent features		CER
no	NOIN US N THE PATIOL	4.5%
yes	JOIN US ONO THE PATIO	13.6%

Figure 7: A typist hits 'J' with their right index finger which is partially occluded by their right thumb. Under occlusion, UmeTrack (blue) reasons that the digit is more curled than the ground truth pose (yellow) indicates. Without latent features the model predicts that 'N' was struck, but with latent features the motion model performs its own occlusion reasoning and correctly predicts 'J'.

Latent features	CER	I-CER	latency (frames)
no	7.36%	5.35%	1.36
yes	7.02%	5.03%	0.81

Table 2: Two label-dependent Siamese Emformer models are trained, one with UmeTrack latent features and one without. Latent features decrease both error rate and emission latency on an evaluation set.

Training labels	Evaluation labels	
	Physical (CER)	Intent (CER)
Physical	18.41%	21.77%
Intent	25.06%	7.02%

Table 3: The character error rate (CER) of label-dependent Siamese Emformer models trained and evaluated on either physical or intent-labeled hand motion data.

module is responsible for occlusion reasoning, but the occlusion reasoning that works for free-space hand motion does not necessarily work when other objects (e.g. the table) can resist hand motion. By integrating UmeTrack's latent features our model is effectively able to learn its own task-specific occlusion reasoning policy.

Study 4: Oracle-based intent labels To measure the impact of our oracle-based data collection with slop errors labeled with user intent, we construct an evaluation that compares our oracle-based intent labels and physically-touched key labels. We transform our original dataset by replaying the 2D touchpad contacts through an offline decoder which simply assigns the label of the physically touched key for each contact. We train and evaluate models on our original and physical touch datasets to measure the impact of intent-training and intent-evaluation (See Table 3). The results show that models struggle to decode the keys that users physically touch, even if the model is trained on physical touch data in the first place.

Study 5: Touchpad baselines

In our final study, we directly compare our motion model against touch-based text decoding (e.g. an iPad). This is possible because our data collection simultaneously recorded both 2D touch contacts and 3D finger motion. While our motion model does not use an explicit language model, we explored how a language model could influence typing accuracy (both for natural language and for non-language text).

For the purposes of this analysis, we focus on a subset of the data in our evaluation datasets. Similar to our other evaluations we ignore non-intent/typo labels where the oracle was off track, but we additionally ignore intent-labels that a user later deleted. We also removed key-presses for non-character generating keys (e.g. backspace and enter). This was necessary as our touchpad decoder's language model was trained on sentences without any

key-presses associated with performing corrections or other actions while creating the sentences. The retained key-presses then all correspond to a key present in the final text. We removed any sentence where the participant's final text did not match the target text. We note that this subset of data is artificially clean typing data because, by design, we selected data without spurious or missing contact events. However, the data still contains spatial noise introduced by the contact slop licensed by the oracle in our data collector.

Our natural language evaluation test set was drawn from the MacKenzie and Soukoreff phrase set [30] (1,445 sentences with 43 K key-presses). Our non-language text evaluation test set consisted of random letter sequences of A–Z (239 sentences with 1,676 key-presses). As shown in Table 4 (row 1), approximately 18% of key-presses in both sets constituted slop errors (i.e. the key closest to the touchpad contact was not the target key).

The touchpad decoder used a 2D Gaussian per key with the means and diagonal covariances fit to the same training corpus used for the motion model. Even without a language model, this Gaussian model corrected some of the slop errors in both evaluation sets. The errors in the natural language text was reduced to 14.2% from 17.8%, while the non-language text was reduced to 16.9% from 18.4% (Table 4, row 2).

We combined the likelihood from the key-specific Gaussians with a character language model prior. Our 12-gram language model was trained using Witten-Bell smoothing on 21 B characters of text from various web sources. The language model had 408 M parameters.

Our greedy decoder simulated a system that produced a key-press as soon as a contact was made (analogous to our streaming motion model). This substantially reduced the error rate for the natural language text from 14.2% to 5.6% (Table 4, row 3). However, as might be expected, the language model made recognition of the non-language text worse, increasing it from 16.9% to 21.3%.

Rather than a greedy character-at-a-time decoder, we could instead perform a beam search and auto-correct the input after multiple key-presses (e.g. after each word as is common on a touchscreen phone keyboard). Past work shows accuracy improves if recognition is postponed until after several words or even the entire sentence [41, 43]. Sentence decoding corrected nearly all slop errors in the natural language evaluation set with an uncorrected final error rate of 0.6% (Table 4, row 4). However, this further increased the error rate for non-language text to 28.5%.

We evaluated a motion model on the same subsets of data (Table 4, row 5). This model has an average emission latency of 27 ms, functionally comparable to character-at-a-time touchpad decoders. The motion model achieves an error rate of 5.2% on natural language text compared to 5.6% from the character language model biased greedy touchpad decoder. On the other hand, the motion model yields a 16.4% error rate on non-language text compared to 16.9% from the unbiased 2D Gaussian touchpad decoder. While in both cases the motion model is close in performance to touchpad decoding accuracy, the motion model does not need a priori knowledge of whether the text being transcribed is natural language or non-language. Beam search decoding with auto-correction yielded significant improvements on natural text touchpad decoding and further exploration is warranted to see if similar improvements are possible for motion model decoding.

Model	Natural language I-CER	Non-language I-CER
Nearest	17.75%	18.39%
Gaussian	14.23%	16.89%
Greedy	5.61%	21.32%
Beam	0.62%	28.54%
Motion model	5.18%	16.36%

Table 4: The intent character error rate (I-CER) of different touchpad based models versus our motion model. Results on natural English phrases and random non-language letter sequences.

6.2 Online evaluation

We conducted a user study with 18 participants to compare the performance of StegoType in VR with that of a traditional wired physical keyboard using a PC. As explained in Section 4.1, we could not compare against the method of Richardson et al. [34] since it did not achieve cross-user generalization, required to be usable for novel users in this study.

6.2.1 Participants. We recruited 18 office professionals (11 male, 7 female) for our study. The participants had diverse levels of typing experience and preferences. The majority of participants self-identified as touch typists (15), one was a hunt-and-peck typist, and two used a combination of both techniques. Two participants were left-handed while the rest were right-handed. Two participants had no prior experience using a VR headset while the rest had used VR at least once a year.

6.2.2 Apparatus. The StegoType system was integrated with a standalone Meta Quest 3 headset. For the physical keyboard condition, we used a Lenovo KBBH21 wired keyboard connected to a PC. We built a typing test app that is compatible with both Quest 3 and PC. Within the Quest 3 application, users were shown a visualization of a full-sized keyboard, their virtual hands, and a window where the typing test was displayed (Figure 8). Each StegoType key-press was accompanied by both visual highlighting and auditory feedback. Neither condition used a language model or auto-correction.

6.2.3 Procedure. Before starting the study, participants completed a survey about their typing preferences and previous experience in VR. Participants sat at a desk with a white tabletop and we instructed them to adjust the seat to a comfortable height. We asked participants to remove any rings or bracelets to minimize interference with hand-tracking.

This was a within-subject experiment with two counterbalanced conditions: StegoType and Physical Keyboard. In the StegoType condition, participants initially completed two calibration steps. First, they held their hands in front of the headset for hand size calibration, which is used by the motion model. Second, they placed both hands on the table in order to align the virtual keyboard with the surface of the physical table. Using their virtual hands, participants then adjusted the virtual keyboard to a comfortable typing position by repositioning somewhere on the plane of the table.

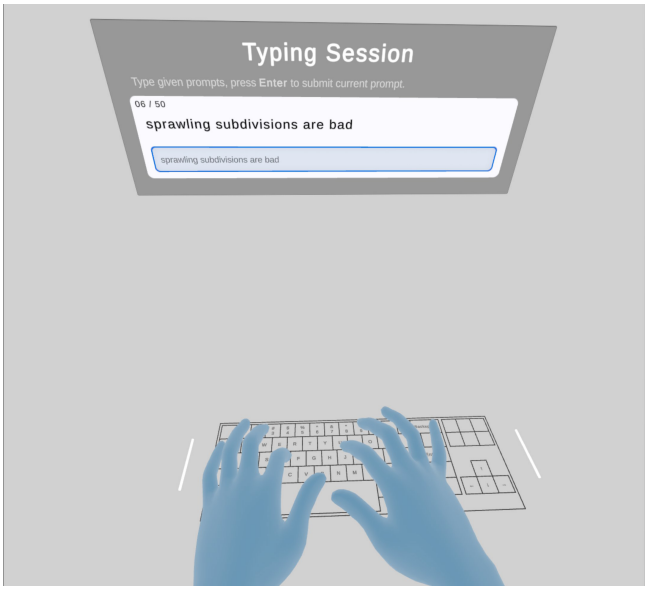


Figure 8: Our VR user study condition in which participants see their virtual hands with a virtual keyboard and are tasked with transcribing short phrases in a typing test.

In each condition, participants were presented with 50 randomly selected phrases from the MacKenzie and Soukoreff phrase set [30] wherein 37% of the words are not represented in any of the corpora used to train the motion model. Each condition was broken down into 5 blocks that each consisted of 10 phrases. There was a 30 second break between blocks which participants had the option to skip. We directed participants to type as fast and accurately as possible. After completing both conditions, we conducted a short interview to collect qualitative feedback.

6.2.4 Metrics. To evaluate StegoType, we used two main dependent measures: text entry rate and error rate. For text entry rate, we used the following formula [1] to compute words per minute (wpm):

$$WPM = \frac{|R| - 1}{T} \times 60 \times \frac{1}{5} \tag{5}$$

Where |R| is the length of the transcribed string in characters and T is the elapsed time in seconds. Elapsed time corresponds to the time difference in seconds between the first key input that the user enters and when the user submits. For error rate, we measured both uncorrected error rate (UER) and corrected error rate (CER) [39]. UER corresponds to errors that remain in the transcribed string after the user submits. CER corresponds to errors that were committed but then corrected.

To assess the learning effect across blocks, we conducted a one-way repeated measures ANOVA for text entry rate as we found it was normally distributed (tested with Shapiro-Wilk). We conducted Friedman tests for UER and CER as they were not normally distributed.

6.2.5 Results. Quantitative performance. As shown in Table 5, participants entered text faster at 74.8 wpm with the physical keyboard compared to 42.3 wpm with StegoType. Participants also made more errors with StegoType with an UER of 7.0% and a CER of 9.3% compared to an UER of 0.8% and a CER of 3.4% using the physical keyboard.

Condition	Text entry rate (WPM)	UER (%)	CER (%)
Physical Keyboard	74.8 (9.4)	0.8 (0.2)	3.4 (0.1)
StegoType	42.4 (6.4)	7.0 (2.8)	9.3 (3.1)

Table 5: The means and 95% confidence intervals of text entry rate, uncorrected error rates (UER) and corrected error rates (CER) of participants during the user study

Figure 10 demonstrates entry rate change over blocks. A repeated measures ANOVA (RM-ANOVA) indicated a significant difference across blocks for both StegoType ($F_{4,68} = 9.168, p < .001$) and physical keyboard ($F_{4,68} = 8.703, p < .001$) conditions. For StegoType in particular, block 3 (46.9 wpm) was markedly faster than block 1 (34.9 wpm). This difference was significant ($F_{1,17} = 28.463, p < .001$). From block 3 to 5, there were no significant differences in entry rate. Friedman tests did not find any significant change in UER or CER in either condition across the blocks.

Figure 9 illustrates the wide range of performances that participants were able to achieve using StegoType relative to using a physical keyboard. For example, P7 was able to achieve 67% of his physical keyboard typing speed using StegoType with a UER of 1.2%, while P5 achieved 49% text entry rate ratio with a UER of 22.0%.

Qualitative feedback. Many users expressed surprise by how well StegoType worked. P4 mentioned *"This is really good, I'm honestly surprised it worked at all"* and P9, *"When it works, it feels magical"*. Users also described adapting to the system in a few ways. P9 also mentioned, *"I was initially focusing too much on my hands but then I realized that it works better when I don't look down and type as naturally as possible."* and P11, *"It took me a few phrases to get going but then it felt great"*. P16 contrasted StegoType with the current mid-air Quest keyboard by stating *"This is already way faster than the current two-finger typing experience on Quest"*.

When asked about sources of frustration, participants mentioned the lack of haptic feedback as a primary cause. P17 said *"I had to look down a lot in the beginning to align my fingers with the keys, I didn't realize how much I rely on the little bumps on the f and j keys on a physical keyboard."* Another prevalent issue was inaccurate hand-tracking. Five participants experienced challenges in accurately tapping keys with their ring and/or pinky fingers, attributing this difficulty to inaccurate ring/pinky finger pose specifically. Six participants voiced their frustration regarding correction. P11 shared *"Correction felt very slow. I would often have to backspace and hit a key multiple times before I got the correct character."*. We observed the same behavior across the majority of participants which led to the high corrected error rate shown in Table 5. Given these challenges, six participants suggested that the inclusion of an auto-correct feature would have significantly improved their typing speed and accuracy.

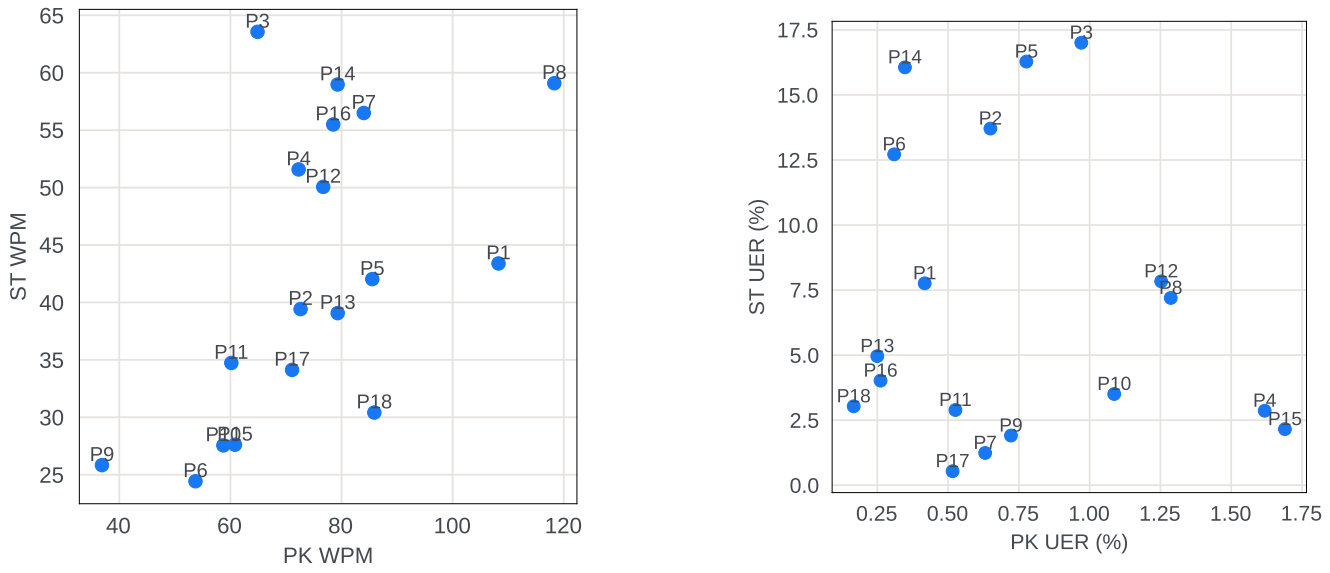


Figure 9: Scatter plots contrasting text entry rate (WPM) and accuracy (UER) of each participant using StegoType (ST) and Physical Keyboard (PK).

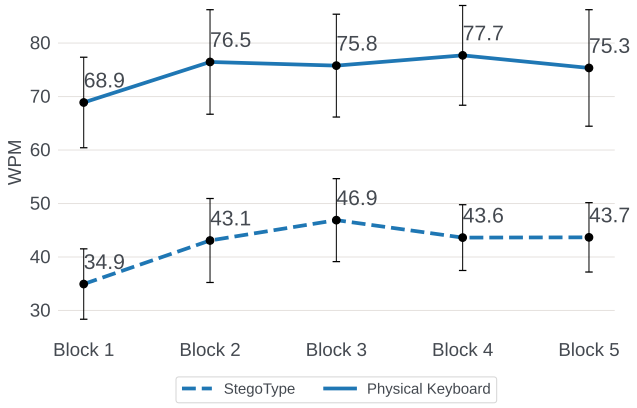


Figure 10: Participants' mean text entry speed over the 5 blocks of typing in the StegoType and physical keyboard conditions (error bars show 95% confidence interval)

7 DISCUSSION AND LIMITATIONS

Our study revealed a notable difference between offline and online system accuracy suggesting a domain gap between the two settings. Several factors could potentially contribute to this discrepancy including variation in the participant pools and scene/lighting differences that can affect hand-tracking fidelity. Of notable interest are the behavioral differences that arise when using StegoType in VR versus interacting with a touchpad keyboard in data collection. Findlater et al. [13] found that people type faster on touchpad keyboards when they assume their input is accurate compared to when interacting with a practical touchpad keyboard, which implies that people's typing motion and behavior changes depending on the feedback afforded by an interactive system. Our data collection is based on an oracle touchpad decoder, but typing behavior changes when interacting with StegoType. Because StegoType is trained to interpret hand motion, the behavior difference induced by using StegoType could lead to a domain gap.

We observed subjective behavioral differences that corroborated this behavioral domain gap. We saw participants rest fingers on the surface during our user study, but in data collection users always hold their hands above the surface to not accidentally trigger the touchpad. We also observed differences in typing cadence. In data collection participants transcribe short 3-5 word phrases leading to a *read-then-burst* typing pattern, and because they are provided audio buzzer to minimize post-typo events (Section 4.2) participants do not hesitate to check their work unless they hear this audio cue. However, in live testing no audio correctness feedback is provided which leads to a slower visual feedback loop and a different rhythm of typing.

Our oracle touchpad decoder inflated biased key bounding boxes by 13 mm on each side, a threshold we chose empirically to capture most cases of slop we found in pilot testing. In interactive data collection the size of this bias will affect how much slop typists use and thus also their general typing behavior. A future study designed to study the impact of this and other factors could offer valuable insights into the nuances of offline versus online user interactions.

Comparisons against touchpad decoding (Table 4) illustrated that motion models achieve similar performance to language-model biased touchpad decoders without an explicit language model prior. However, that the motion model did not significantly exceed touchpad performance on non-language typing suggests that the motion model may have learned its own language prior from the training data and also when to dynamically apply it. Adapting the motion

model to incorporate an explicit language model during training may allow it to leverage a stronger language prior than can be learned from the limited representation of language in our training set, which could help performance on natural language typing.

8 CONCLUSION

Text input is a cornerstone of any general purpose computing system, and this paper works towards enabling natural and efficient text input in AR/VR by letting typists leverage their physical keyboard typing skills to touch type on uninstrumented flat surfaces. We have proposed a novel and expressive motion model that adapts ideas from modern end-to-end ASR (e.g., Emformers, direct prediction of keys) with the domain-specific qualities of two-handed typing hand motions (e.g., Siamese hand architecture, low latency requirements). We show that our system can be adapted to handle even the noise and uncertainty of egocentric markerless hand-tracking by using latent features in addition to pose. We show that training such a system requires a data collection system that records interactive (closed-loop) typing and that accounts for both sloppy typing on a soft keyboard and user compliance errors.

We have shown with an 18 person user study that typists can achieve 42.4 WPM while retaining an uncorrected error rate of 7%, compared to a physical keyboard baseline of 74.5 WPM at 0.8% UER. Notably, this is without the benefit of a language model.

There remains a performance gap between offline evaluation and user study results, and we hypothesize several influential factors, but more exploration is needed to gain a full understanding. Our experiments showed decoding touchpad data benefited from a language model prior, and exploration is warranted to see if this benefit transfers to surface typing.

ACKNOWLEDGMENTS

We thank Braden Copple, Lee Zuhars, Shae Herrmann, Tamera Marshall, Stephanie Doring, Tiffany Dumlao, Elena Shchetinina, Atishi Bali, Steve Miller, Matt Maas, Kaichen Sun, Matthew Prasek, Kevin Harris, Steve Olsen.

REFERENCES

[1] Ahmed Sabbir Arif and Wolfgang Stuerzlinger. 2009. Analysis of text entry performance metrics. In *Proceedings of the IEEE Toronto International Conference Science and Technology for Humanity*. 100–105.

[2] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. 2021. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*. 6836–6846.

[3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *CoRR* abs/1803.01271 (2018). arXiv:1803.01271 http://arxiv.org/abs/1803.01271

[4] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. 2021. Is space-time attention all you need for video understanding?. In *ICML*, Vol. 2. 4.

[5] Joao Carreira and Andrew Zisserman. 2017. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6299–6308.

[6] Yi Fei Cheng, Tiffany Luong, Andreas Rene Fender, Paul Streli, and Christian Holz. 2022. ComforTable user interfaces: Surfaces reduce input error, time, and exertion for tabletop and mid-air user interfaces. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality*. 150–159.

[7] Vivek Dhakal, Anna Maria Feit, Per Ola Kristensson, and Antti Oulasvirta. 2018. Observations on Typing from 136 Million Keystrokes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, Article 646, 12 pages. https://doi.org/10.1145/3173574.3174220

[8] Haodong Duan, Yue Zhao, Kai Chen, Dahua Lin, and Bo Dai. 2022. Revisiting skeleton-based action recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2969–2978.

[9] John J. Dudley, Hrvoje Benko, Daniel Wigdor, and Per Ola Kristensson. 2019. Performance Envelopes of Virtual Keyboard Text Input Strategies in Virtual Reality. In *2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)* (Beijing, China). 289–300.

[10] John J. Dudley, Keith Vertanen, and Per Ola Kristensson. 2018. Fast and Precise Touch-Based Text Entry for Head-Mounted Augmented Reality with Variable Occlusion. *ACM Trans. Comput.-Hum. Interact.* 25, 6, Article 30 (Dec. 2018), 40 pages. https://doi.org/10.1145/3232163

[11] John J Dudley, Jingyao Zheng, Aakar Gupta, Hrvoje Benko, Matt Longest, Robert Wang, and Per Ola Kristensson. 2023. 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).

[12] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. 2019. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*. 6202–6211.

[13] Leah Findlater, Jacob O. Wobbrock, and Daniel Wigdor. 2011. Typing on flat glass: examining ten-finger expert typing patterns on touch surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (<conf-loc>, <city>Vancouver</city>, <state>BC</state>, <country>Canada</country>, </conf-loc>) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 2453–2462. https://doi.org/10.1145/1978942.1979301

[14] Joshua Goodman, Gina Venolia, Keith Steury, and Chauncey Parker. 2002. Language Modeling for Soft Keyboards. In *Proceedings of the 7th International Conference on Intelligent User Interfaces* (San Francisco, California, USA) *(IUI '02)*. Association for Computing Machinery, New York, NY, USA, 194–195. https://doi.org/10.1145/502716.502753

[15] Patrick Grady, Jeremy A Collins, Chengcheng Tang, Christopher D Twigg, Kunal Aneja, James Hays, and Charles C Kemp. 2024. PressureVision++: Estimating Fingertip Pressure from Diverse RGB Images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 8698–8708.

[16] Patrick Grady, Chengcheng Tang, Samarth Brahmbhatt, Christopher D. Twigg, Chengde Wan, James Hays, and Charles C. Kemp. 2022. PressureVision: Estimating Hand Pressure from a Single RGB Image. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part VI* (Tel Aviv, Israel). Springer-Verlag, Berlin, Heidelberg, 328–345. https://doi.org/10.1007/978-3-031-20068-7_19

[17] Alex Graves. 2012. Sequence transduction with recurrent neural networks. In *ICML Workshop on Representation Learning*.

[18] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *Proceedings of the 23rd International Conference on Machine Learning*. 369–376.

[19] Shangchen Han, Po-Chen Wu, Yubo Zhang, Beibei Liu, Linguang Zhang, Zheng Wang, Weiguang Si, Peizhao Zhang, Yujun Cai, Tomas Hodan, Randi Cabezas, Luan Tran, Muzaffer Akbay, Tsz-Ho Yu, Cem Keskin, and Robert Wang. 2022. UmeTrack: Unified multi-view end-to-end hand tracking for VR. In *SIGGRAPH Asia 2022 Conference Papers*.

[20] Zhenyi He, Christof Lutteroth, and Ken Perlin. 2022. Tapgazer: Text entry with finger tapping and gaze-directed word selection. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–16.

[21] Ricardo Jota, Albert Ng, Paul Dietz, and Daniel Wigdor. 2013. How fast is fast enough? a study of the effects of latency in direct-touch pointing tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2291–2300. https://doi.org/10.1145/2470654.2481317

[22] Luis A. Leiva, Sunjun Kim, Wenzhe Cui, Xiaojun Bi, and Antti Oulasvirta. 2021. How We Swipe: A Large-scale Shape-writing Dataset and Empirical Findings. In *Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction* (Toulouse & Virtual, France) *(MobileHCI '21)*. Association for Computing Machinery, New York, NY, USA, Article 11, 13 pages. https://doi.org/10.1145/3447526.3472059

[23] Jinyu Li et al. 2022. Recent advances in end-to-end automatic speech recognition. *APSIPA Transactions on Signal and Information Processing* 11, 1 (2022).

[24] Yang Li, Yuan Shangguan, Yuhao Wang, Liangzhen Lai, Ernie Chang, Changsheng Zhao, Yangyang Shi, and Vikas Chandra. 2024. Not All Weights Are Created Equal: Enhancing Energy Efficiency in On-Device Streaming Speech Recognition. *arXiv preprint arXiv:2402.13076* (2024).

[25] Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. 2017. DailyDialog: A Manually Labelled Multi-turn Dialogue Dataset. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Asian Federation of Natural Language Processing, Taipei, Taiwan, 986–995. https://www.aclweb.org/anthology/I17-1099

[26] Ziyu Liu, Hongwen Zhang, Zhenghao Chen, Zhiyong Wang, and Wanli Ouyang. 2020. Disentangling and unifying graph convolutions for skeleton-based action recognition. In *Proceedings of the IEEE/CVF conference on computer vision and*

pattern recognition. 143–152.
[27] Yiping Lu, Zhuohan Li, Di He, Zhiqing Sun, Bin Dong, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Understanding and Improving Transformer From a Multi-Particle Dynamic System Point of View. *arXiv preprint arXiv:1906.02762* (2019).
[28] Yan Ma, Shumin Zhai, IV Ramakrishnan, and Xiaojun Bi. 2021. Modeling Touch Point Distribution with Rotational Dual Gaussian Model. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 1197–1209. https://doi.org/10.1145/3472749.3474816
[29] I. Scott MacKenzie and R. William Soukoreff. 2002. A character-level error analysis technique for evaluating text entry methods. In *Proceedings of the Second Nordic Conference on Human-Computer Interaction* (Aarhus, Denmark) *(NordiCHI '02)*. Association for Computing Machinery, New York, NY, USA, 243–246. https://doi.org/10.1145/572020.572056
[30] I Scott MacKenzie and R William Soukoreff. 2003. Phrase sets for evaluating text entry techniques. In *CHI'03 extended abstracts on Human factors in computing systems*. 754–755.
[31] Manuel Meier, Paul Streli, Andreas Fender, and Christian Holz. 2021. TapID: Rapid touch interaction in virtual reality using wearable sensing. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*. 519–528.
[32] Yajie Miao, Mohammad Gowayyed, and Florian Metze. 2015. EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. In *2015 IEEE workshop on automatic speech recognition and understanding (ASRU)*. IEEE, 167–174.
[33] Jumon Nozaki and Tatsuya Komatsu. 2021. Relaxing the Conditional Independence Assumption of CTC-based ASR by Conditioning on Intermediate Predictions. In *Interspeech*. https://api.semanticscholar.org/CorpusID:233168606
[34] Mark Richardson, Matt Durasoff, and Robert Wang. 2020. Decoding surface touch typing from hand-tracking. In *Proceedings of the 33rd annual ACM symposium on user interface software and technology*. 686–696.
[35] Fadime Sener, Dibyadip Chatterjee, Daniel Shelepov, Kun He, Dipika Singhania, Robert Wang, and Angela Yao. 2022. Assembly101: A large-scale multi-view video dataset for understanding procedural activities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 21096–21106.
[36] Junxiao Shen, John Dudley, and Per Ola Kristensson. 2023. Fast and Robust Mid-Air Gesture Typing for AR Headsets using 3D Trajectory Decoding. *IEEE Transactions on Visualization and Computer Graphics* (2023).
[37] Yangyang Shi, Yongqiang Wang, Chunyang Wu, Ching-Feng Yeh, Julian Chan, Frank Zhang, Duc Le, and Michael L. Seltzer. 2021. Emformer: Efficient Memory Transformer Based Acoustic Model For Low Latency Streaming Speech Recognition. In *Proceedings of The IEEE International Conference on Acoustics, Speech and Signal Processing*. 6783–6787.
[38] Yangyang Shi, Chunyang Wu, Dilin Wang, and Others. 2022. Streaming Transformer Transducer based Speech Recognition Using Non-Causal Convolution. In *The proceeding of the IEEE International Conference on Acoustics, Speech and Signal Processing*. 8277–8281.
[39] R. William Soukoreff and I. Scott MacKenzie. 2003. Metrics for Text Entry Research: An Evaluation of MSD and KSPC, and a New Unified Error Metric. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Ft. Lauderdale, Florida, USA) *(CHI '03)*. Association for Computing Machinery, New York, NY, USA, 113–120. https://doi.org/10.1145/642611.642632
[40] Paul Streli, Jiaxi Jiang, Andreas Rene Fender, Manuel Meier, Hugo Romat, and Christian Holz. 2022. TapType: Ten-finger text entry on everyday surfaces via Bayesian inference. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–16.
[41] Keith Vertanen, Crystal Fletcher, Dylan Gaines, Jacob Gould, and Per Ola Kristensson. 2018. The Impact of Word, Multiple Word, and Sentence Input on Virtual Keyboard Decoding Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. ACM, New York, NY, USA, Article 626, 12 pages. https://doi.org/10.1145/3173574.3174200
[42] Keith Vertanen, Dylan Gaines, Crystal Fletcher, Alex M. Stanage, Robbie Watling, and Per Ola Kristensson. 2019. VelociWatch: Designing and Evaluating a Virtual Keyboard for the Input of Challenging Text. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3290605.3300821
[43] Keith Vertanen, Haythem Memmi, Justin Emge, Shyam Reyal, and Per Ola Kristensson. 2015. VelociTap: Investigating Fast Mobile Text Entry Using Sentence-Based Decoding of Touchscreen Keyboard Input. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 659–668. https://doi.org/10.1145/2702123.2702135
[44] Raphael Wimmer, Andreas Schmid, and Florian Bockes. 2019. On the Latency of USB-Connected Input Devices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300650
[45] Xin Yi, Chen Liang, Haozhan Chen, Jiuxu Song, Chun Yu, Hewu Li, and Yuanchun Shi. 2023. From 2d to 3d: Facilitating single-finger mid-air typing on qwerty keyboards with probabilistic touch modeling. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2023), 1–25.
[46] Xin Yi, Chun Yu, Mingrui Zhang, Sida Gao, Ke Sun, and Yuanchun Shi. 2015. ATK: Enabling Ten-Finger Freehand Typing in Air Based on 3D Hand Tracking Data. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*. 539–548.
[47] Mingrui Ray Zhang, Shumin Zhai, and Jacob O Wobbrock. 2022. TypeAnywhere: A QWERTY-based text entry solution for ubiquitous computing. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–16.

A INTENT EDIT DISTANCE

(1) **CER (character error rate)**
The Levenshtein distance is calculated using a dynamic programming approach which populates the distance matrix using a set of edge cost rules, where the final edit distance is taken from the bottom right cell (Equation 6). This implements the minimum edit distance, i.e. the minimum number of substitutions, insertions, or deletions necessary to transform the target sequence into the predicted sequence. The character error rate is calculated as the ratio of the Levenshtein edit distance to the length of the target sequence.

(2) **I-CER (intent character error rate)**:
A subset of target tokens correspond to intent labels, and the error rate corresponding to this subset of the target sequence is congruous with our objective of predicting user intent. Because the alignment of prediction events with label events is not known *a priori* and we only know whether target tokens are intentional, not predicted tokens, we cannot simply filter the input sequences to calculate this error rate. Instead, we modify the edit distance calculation to discount substitution and deletion errors corresponding to non-intent target tokens (Figure 11). The result is in an edit distance which is always the same or smaller than the full-sequence edit distance, discarding any edits associated with non-intent labels. To calculate a meaningful character error rate this edit distance is then normalized by the number of intent tokens in the target sequence (Equation 8).

$$c_{\text{del}}(b_j) = 1, c_{\text{ins}}(a_i) = 1, c_{\text{sub}}(a_i, b_j) = 1$$
$$d_{i,j} = \min \begin{cases} d_{i,j-1} + c_{\text{del}}(b_j) \\ d_{i-1,j} + c_{\text{ins}}(a_i) \\ d_{i-1,j-1} + [a_i \neq b_j] \cdot c_{\text{sub}(a_i,b_j)} \end{cases} \tag{6}$$

$$\text{CER}(\mathbf{a}, \mathbf{b}) = \frac{d_{|\mathbf{a}|,|\mathbf{b}|}}{|\mathbf{b}|} \tag{7}$$

$$c_{\text{del}}(b_j) = [b_j \text{is intent}], c_{\text{ins}}(a_i) = 1, c_{\text{sub}}(a_i, b_j) = [b_j \text{is intent}]$$
$$\text{I-CER}(\mathbf{a}, \mathbf{b}) = \frac{d_{|\mathbf{a}|,|\mathbf{b}|}}{|\{b_i \in \mathbf{b} \mid b_i \text{ is intent}\}|} \tag{8}$$

B EMISSION LATENCY

The dynamic programming edit distance calculation can be modified to support backtracking to find the optimal set of alignments between the predicted and target token sequences [29]. With a

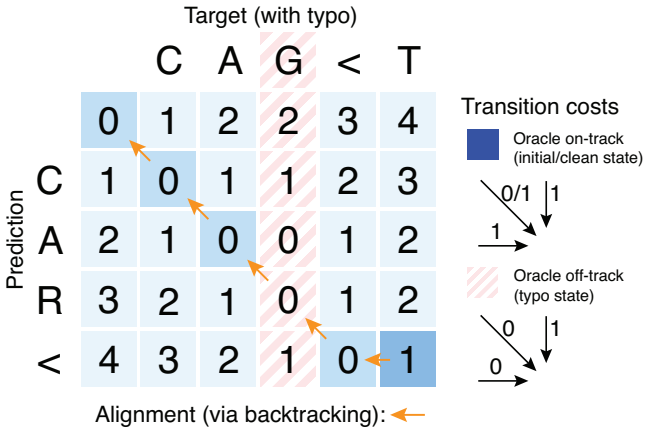


Figure 11: Levenshtein edit distance, calculated using dynamic programming, is modified to discount substitution or deletion errors corresponding to non-intent label tokens. Backtracking is used to find all prediction/label alignments corresponding to the optimal edit distance to enable derived metrics like emission latency.

reasonably accurate model there typically exist many unambiguous one-to-one correspondences between target and predicted tokens.

Since emission timestamps for both target and predicted tokens are known, we can evaluate statistics over this set of uniquely corresponded tokens to compute the emission latency and per-key accuracy of a model.

Occasionally two consecutive insertion and deletion errors which are well separated in time will be merged by the edit distance calculation into a single substitution error. While rare, these correspondences can be attributed extremely high latency and erroneously influence overall statistics. To combat this issue we introduce a supplemental filtering step where correspondences with an outlier emission latency are discarded. In our evaluation we set this latency threshold to the range $-0.17s \leq latency \leq 0.50s$ at 30Hz.