TypeNet: Towards Camera Enabled Touch Typing on Flat Surfaces through Self-Refinement

Ben Maman
Tel Aviv University

Amit Bermano
Tel Aviv University

Abstract

Text entry for mobile devices nowadays is an equally crucial and time-consuming task, with no practical solution available for natural typing speeds without extra hardware. In this paper, we introduce a real-time method that is a significant step towards enabling touch typing on arbitrary flat surfaces (e.g., tables). The method employs only a simple video camera, placed in front of the user on the flat surface — at an angle practical for mobile usage. To achieve this, we adopt a classification framework, based on the observation that, in touch typing, similar hand configurations imply the same typed character across users. Importantly, this approach allows the convenience of un-calibrated typing, where the hand positions, with respect to the camera and each other, are not dictated.
To improve accuracy, we propose a Language Processing scheme, which corrects the typed text and is specifically designed for real-time performance and integration with the vision-based signal. To enable feasible data collection and training, we propose a self-refinement approach that allows training on unlabeled flat-surface-typing footage; A network trained on (labeled) keyboard footage labels flat-surface videos using dynamic time warping, and is trained on them, in an Expectation Maximization (EM) manner.
Using these techniques, we introduce the TypingHands26 Dataset, comprising videos of 26 different users typing on a keyboard, and 10 users typing on a flat surface, labeled at the frame level. We validate our approach and present a single camera-based system with character-level accuracy of 93.5% on average for known users, and 85.7% for unknown ones, outperforming pose-estimation-based methods by a large margin, despite performing at natural typing speeds of up to 80 Words Per Minute. Our method is the first to rely on a simple camera alone, and runs in interactive speeds, while still maintaining accuracy comparable to systems employing non-commodity equipment.

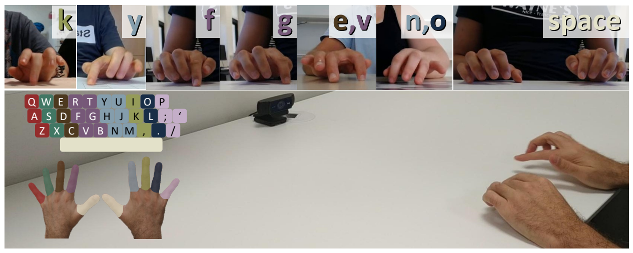


Figure 1. Our camera-based touch typing system. The user types on a flat surface w/o a keyboard in front of a camera (bottom right), according to the touch typing standards. For each character, these standards dictate using a specific finger, inducing a distinct hand pose and motion (bottom left). Our system accordingly classifies said hand configurations and motion to predict the typed text.

1. Introduction

Text entry is one of the most commonly used modes of interaction with computers. As small mobile devices, such as tablets or phones, become more popular, the need for fast and accurate text entry mechanisms, to replace the physical keyboard, grows crucial [24, 10, 4]. Currently, the most popular mechanism for such devices is a soft keyboard, displayed on the device's screen. Due to screen size, this method is difficult and slow, especially for long texts [6].

Approaches that attempt using the device's surrounding, instead of its screen, typically require additional hardware, such as marked gloves, a projector, or a wearable gyroscope, rendering them unfit for ubiquitous usage [13, 12, 11]. Hence, a Computer Vision based method, which employs nothing but the device's built-in camera, is naturally preferable when considering an application that should be widespread. Current vision-based methods, however, are also fundamentally lacking, since they either require a depth sensor [29], placing the camera in non-feasible positions (*e.g.*, above the hands or on the wrist [19]), or are not performance driven, and hence do not allow typing at natural speeds (which can reach up to 80 Words Per Minute or more for a skilled typist). For example, Yi *et al.* [29] do not ex-

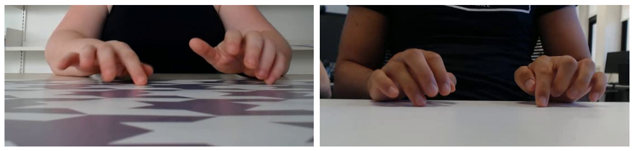


Figure 2. Example frames from a video of free typing on a surface. Left: the keys 'a' and 'y' are active. Right: keys 'o' and 'r' are active. Both frames were labelled automatically using the method explained in section 3.

ceed 29.2 Words Per Minute, and Murase *et al.* [22] disregard computational performance all together.

In this paper, we present typeNet, a novel vision-based method that relies on a single front facing camera, at an angle practical for mobile devices, allowing accurate typing on flat surfaces at natural speeds, for typists trained in touch typing. To achieve such a task, the most natural approach would be to estimate and track fingertip positions, in an attempt to form a virtual keyboard. This approach, however, is challenging in terms of computational performance, and is prohibitive to the user since the hands must be placed accurately on an imaginary keyboard surface without visual or tactile feedback. In touch typing, on the other hand, every key is assigned to a specific finger (see Figure 1, bottom left). This basic principle, that every typist is trained for, is at the core of our method, since this induces similar hand configuration for the same typed character across different users, regardless of gender, ethnicity, or age. For example, pressing the key 'e' is always performed by moving the left middle finger forward. Hence, typists can easily recreate the typical motion they would have used with a keyboard on any flat surface, using muscle memory. Indeed, all our users required no training or explanation to transition from keyboard to surface.

Equipped with this key observation, we propose using a classification framework for the task at hand, applied on raw video frames, rather than relying on estimating finger tip locations or on tracking. Such an approach allows for faster response times, and does not suffer from drift or recovery effects following lost tracking. This approach also naturally allows for un-calibrated typing, i.e. for free positioning of the hands, not restricting the typist to a carefully considered spatial location and angle, nor even requiring both hands to be in alignment.

Developing such a system is challenging mainly due to two reasons. The first is training data. Typing a key in isolation is typically different, in terms of hand pose and motion, compared to typing the same key in the midst of a long text entry session. This means that in order to collect usable data, typists should be filmed while typing non-trivial text on a flat surface. This type of data, however, is difficult to label, since it is unclear exactly what was the typist's intent at every frame. To overcome this challenge, we propose a self-refinement process, where data is first collected on a keyboard (where ground truth labeling is easily attainable, see Figure 3 phase I), and later on flat-surface (Figure 3, phase III). A model that is trained on the former is then used to generate the labeling for the latter (see Figure 3, phase II). This labeling is then used to re-train the network, in a manner typical to Expectation Maximization (EM) approaches, dramatically improving its accuracy on flat-surface footage (see Section 5), even though no frame-level supervision is provided for this domain. Using this scheme, we were able to generate a high-quality labeled dataset consisting of 24:50 hours of keyboard typing videos from 26 different users, and 5:50 hours of flat-surface typing footage from 10 different users. These, along with the code of our method, can be found at (`https://github.com/benadar293/typeNet`).

Another great challenge in realizing such a system is the need for real-time performance. Every aspect of our design is also aware of this constraint, from the general approach of classification, to the network's architecture. Most notably, we have developed a light-weight language layer, which resolves uncertainties the vision produces through language dependent priors, that is specifically designed for performance and to integrate well with the vision-based predictions (see Section 3.3). Additionally, we boost performance by avoiding a seemingly necessary step: to reduce the amount of required data, we separate the hands from their background during training. However, we avoid the burden of segmentation during inference time using a training scheme based on features removal and their gradual introduction [1]. Using this scheme, we guide the network to focus on the hands themselves instead the background.

We demonstrate our approach through a series of experiments, and witness a character-level accuracy of 93.5% on average for users the system has already seen, and 85.7% accuracy on average for previously unseen users, at natural typing speeds. Compared to other text entry works (Table 2), it is clear our method is the first to rely on a simple camera alone, while maintaining accuracy that is comparable to equipment-heavy methods, even without considering interactivity. We perform further ablation studies to measure the effectiveness of the various steps of the method, including a comparison to classifying frames according to estimated hand poses, rather than raw frames (see Section 5.2 and Supplementary). This validates the design of our real-time system to recognize arbitrary text with high accuracy from a single, conveniently located, camera. We strongly believe that this line of work, when further investigated and with enough training data, could revolutionize the way we interact with mobile devices in the near future.

2. Related work

Virtual keyboards have been sought after through many different directions, employing various sensing devices to replace the physical keyboard - an RGBD camera (such as the Kinect), a stereo configuration, or even a single RGB camera. Other devices are also proposed, such as motion sensors, either in a fixed position (such as Leap Motion), or placed on the typer's arms.

Virtual keyboard methods can be divided into two categories: (i) Methods that allow any virtual key to be pressed by any finger, and (ii) methods that require each key to only be pressed by a certain finger, according to the standard touch typing guidelines.

Non-standard typing. In the first group, notable earlier works are Samsung's SCURRY [13], which uses gyroscopes and motion sensors worn on the fingers, Senseboard [11], which uses bands of rubber and plastic worn on the palm, Key-glove [25], which uses a glove that detects finger motion, and VKB [12], which projects a keyboard on a surface using a laser diode, and detects keystrokes using an infra-red camera. Du *et al*. [7] use an LCD projector, that projects a keyboard image onto the typing surface, and a 3D range camera for keystroke detection. The main drawback of the latter works is the need for cumbersome equipment that is impractical for mobile devices. Also, methods in this category set absolute key locations, and therefore provide no flexibility in the placement of the hands and in key localization. Our method requires no equipment other than the camera, and allows flexibility in hand location and angle.

Touch typing. In the second category, Mujibiya *et al*. [19] use a depth camera located above the typer's hands. They detect key strokes by matching hand postures to a database of 3D images, and by detection of finger tips touching the surface. The drawbacks of this method are the camera angle, and the need for a depth camera, which are impractical for mobile applications. Similar work was done by Murase *et al*. [21, 22], who use a single mobile phone camera. They estimate depth (to decide which row is pressed) by various hand features, e.g. ratio between hand height and width. In the follow-up work, they improve depth estimation using Real Adaboost on the hands' HOG features. The setup in this work is the most similar to ours, however the system is tested on a single user typing the same sentence 10 times, while the typing speed is not even reported. These acute differences make this and our systems incomparable.

Yi *et al*. [29] use a Leap Motion sensor to detect taps. Labeling of images was done manually. A language model provides the user with suggestions for words that are consistent with the sequence of tapping fingers. The main drawback in this work is the low typing speed, and relying on a Leap Motion camera rather than a simple camera. Richardson *et al*. [23] use a hand tracking system to detect typing. However, they require marked gloves to allow for sufficient tracking accuracy, which makes the system impractical. In contrast, our method provides natural typing speed with no additional equipment besides the single RGB camera — a setting most suitable for mobile usage.

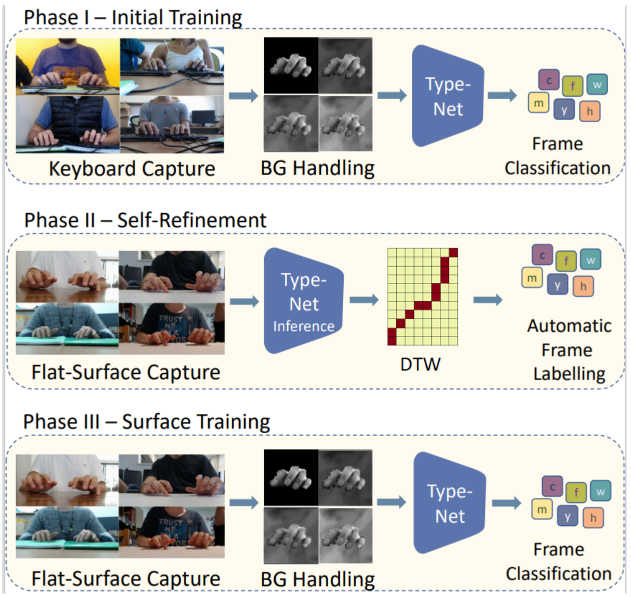


Figure 3. Pipeline of our method. We first train typeNet on fully labelled keyboard-based frames (Phase I). We then use the network to label surface-based footage with pre-defined texts, by optimizing alignment of predicted probabilities with the pre-defined text, using dynamic time warping (Phase II, Section 3.2) — a process we call self-refinement. Lastly, we train the network again on both the surface-based and keyboard-based sequences (Phase III). During inference, we use our custom language layer (Section 3.3), to produce a more likely and accurate result.

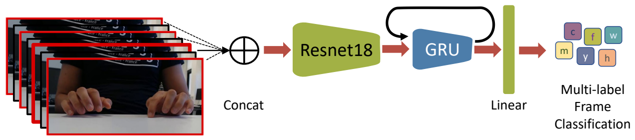


Figure 4. typeNet architecture. For every frame of an input video, a short temporal context sequence is extracted and fed into a lightweight convolutional network (Resnet18, He *et al*. [9]). The latent features are encoded Using a GRU, and are finally converted to keypress probabilities through a fully connected layer.

3. Our method

The core idea behind our method is the classification of short sequences, acquired by a single camera (*e.g*., mobile or webcam). Our problem setup is such that the sequences contain the two palms of a user while typing, from the front (see Figure 1, bottom right). Note that while other angles are better in terms of accuracy (*e.g.*, from above the palms), they are less practical for mobile use, where having the screen visible during typing is critical. Additionally, we assume touch typing standards (see Figure 1, bottom left).

The process from a bird's eye view is depicted in Figure 3. Our training process is divided into three phases, which we describe below. During inference, we estimate which of the keys are active (*i.e.*, currently pressed), for each frame in a multi-label manner. Usually, no more than a single key is pressed during a frame, but in some cases, such as an overlap between rapid key presses, or while pressing the *shift* modifier, we could see up to three keys being pressed at the same frame. Inference is run sequentially over the video frames, where for each frame we predict the probabilities for its active keys. Using these probabilities, we search for a sequence of predictions that maximizes these estimations, along with a language model score, using Beam Search, as explained in section 3.3. For this purpose, we have collected training data of two kinds:

- **Keyboard typing:** We collected videos of typists on a keyboard, from the aforementioned angle. The keyboard is used to mark the timing of key events, thus allowing automatic labeling of all frames in the video.
- **Surface typing:** In this case, the typists were recorded while typing on an arbitrary flat surface, in a similar angle. Some used the desk, while others preferred some padding, like notebooks. This data is used to fine-tune the network during the third phase for free surface typing.

Note that the network trained on keyboard data can readily be used on another surface. However, as we demonstrate, the style of free typing on a surface differs from that of a keyboard, which impacts accuracy, which is why training on surface data is necessary.

3.1. Phase I - initial training

During the first phase, we directly train typeNet to classify per-frame key presses on a keyboard. To create a similar environment as possible to the next phases, we have use a split and flat keyboard (see Figure 3, *Keyboard Capture*).

The architecture of typeNet can be seen in Figure 4. As can be seen, we employ temporal context, which intuitively helps disambiguate between keys of the same finger by detecting the finger movement rather then just the hand configuration. We incorporate temporal context in two ways. First, during feature extraction, we stack neighboring frames on the one in question as additional channels (see Figure 4, left). Secondly, we apply an RNN over the sequence of extracted features. For computational efficiency, we use the slim Resnet18 backbone for feature extraction, and a 2-layer GRU followed by a fully connected layer for classification (see Supplementary for details). In addition, in order to reduce the amount of training data required, we encourage the network to ignore the image backgrounds.

Ignoring the background serves two purposes. The first is to prevent the network from considering spurious details in the background, thus requiring less training data. The second is better generalization. This aids the following phases, since teaching the network to ignore the background also teaches it to ignore the keyboard being used during typing. Our solution starts with an off-the-shelf hand segmentation solution, which removes the background (including the keyboard) before training, (see Figure 3, *BG Handling*, top left). Since segmentation networks are computationally intensive, we seek to avoid running them during inference. To do this, we employ traditional augmentations, i.e. we move the hand positions in the image, and replace the background with a random patch from the COCO dataset. In addition, we gradually increase the background's intensity from $\alpha = 0$, to $\alpha = 1$, thus teaching the network to implicitly ignore the background (Figure 3, *BG Handling*), as proposed by Arar *et al.* [1] (See Supplementary for details).

3.2. Phase II - Self-Refinement

One of the central challenges of our work lies with the difference between typing styles on surfaces and on a keyboard. The typical user does not accurately imitate the hand movements performed on a keyboard when typing on a surface, due to the lack of haptic feedback that guides the fingers to specific positions [8]. Therefore, the system must be trained on free typing performed on surfaces as well. Labeling such data on a frame-by-frame level is challenging since the accurate timing of a key's press and release is required. Hence we introduce a self-refinement procedure: After training the network on keyboard-based data, we record users typing known text on a flat surface. In order to obtain per-frame labeling for such videos, we find the optimal alignment between the video and the text using dynamic time warping. This paradigm of weakly supervised learning, where weakly supervised data is assigned maximum likelihood labels for training, is known as Expectation Maximization (EM) [5].

Dynamic time warping (DTW) is a dynamic programming algorithm, designed for finding the optimal alignment between two sequences which may vary in speed [20]. It is common in areas such as speech recognition, where the alignment between an audio sequence and a text sequence is required. It operates in quadratic time and space - $O(MN)$, where M, N are the lengths of the two sequences, and requires a local distance function, between tokens of the two sequences.

We use our trained network as a local distance measure. We infer the active key probabilities vector for each frame, and minimize the alignment between the video and the pre-defined text, padded between characters with the *none* class, using DTW. As a distance function, we use the Euclidean L_2 distance between the network's predicted probabilities vector, and the one-hot representation of the character.

Typically, a key press lasts 3-5 frames at 30FPS. Frames that are mapped to more than one character (singular points), are typically due to type errors of the user (*e.g.*, user missed a character from the target text), and are not assigned a label. We perform further post-processing to extend this labeling to be multi-label, see Supplementary.

The result is a frame-by-frame labeling of typing performed on surfaces, with some false negative detection (i.e., a keypress was performed but not detected), but very few false positives. The missing frames usually stem from a typing error made by the user (which we disregard during the labeling process), or by the system marking a single press incompletely (*e.g.*, a key is active for 4 frames but is labeled for only 1 or 2 of them). To avoid these mislabelings, we do not label frames with no predicted active key. To detect the *none* class (where no key is active), we rely on the keyboard sessions. This configuration is preferable since it does not inject noisy labeling into the training process, while not hurting detection since specifically for the *none* case, it does not matter whether the underlying typing mechanism is a keyboard or a surface - the user is not interacting with it anyway.

We note that this self-refinement process differs from Pseudo-labelling [27, 14, 28]. The latter relies on the existing network's predictions alone, while self-refinement uses a stronger prior of the pre-defined text for labeling, which allows labeling with higher accuracy (see Table 4 for comparison).

After obtaining the per-frame labeling, we can perform the **final training** phase of the network, which employs the data acquired from both the keyboard and flat surface (Figure 3, Surface Training). Example frames, as seen through the camera, are depicted in Figure 2.

3.3. Language model

For many text entry systems (*e.g.*, [23, 29]), a language model over the predicted text is employed to improve accuracy. A language model, by definition, receives a string, of any length, and returns its likelihood. In our method, we also make use of such a model, but in a way that considers the original probabilities predicted by the vision model rather than just the predicted text, making it stronger than standard automatic correction. We combine the vision and language scores using Beam Search, as explained below.

Since we add a single character at a time, we use a character-level language model, rather than a word-level

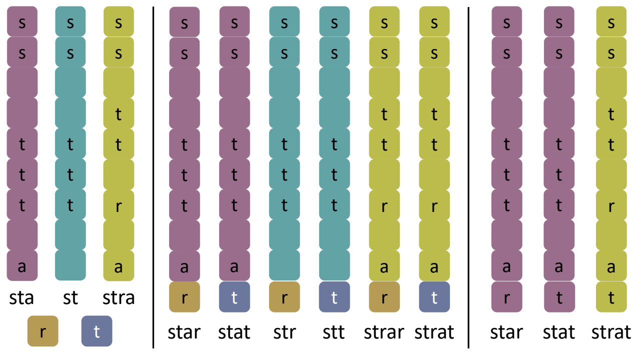


Figure 5. Beam Search illustration: Given N candidate sequences (3 in this example), and candidate labels of a new frame (left), each of them is concatenated to each of the candidate sequences, yielding 6 new ones (middle). The N sequences with highest scores are selected, and the rest are pruned (right).

one. We normalize likelihood by string length in order to avoid the natural bias (undesired in our case) towards shorter strings: the probability of a string is considered to be the geometric mean of the probabilities of each of its characters given the ones preceding it.

Real-time performance. The current SOTA in character level language models is achieved by Transformers. Indeed, using a Transformer (such as shown by Sukhbaatar *et al.* [26]) results in very accurate text predictions. However, it is not feasible in real-time. To perform the Beam Search in real-time, we use an LSTM based character-level language model [18, 17]. See Supplementary for further details on optimization of the Beam Search performance.

Beam Search is an optimized search method designed to enable retroactive error correction on sequential domains (which a greedy search would fail to provide) [16]. During inference, each frame of the video is assigned a multi-label probability vector by the vision model, rectified to account for same finger keys correlation. From all possible prediction sequences, we search for those that maximize the sum of the vision model probabilities, and the language model scores assigned to the corresponding string. To reduce complexity, during inference we select one key per frame at most, even though this is a multi-label problem. We have seen that this does not affect text accuracy. We search for the optimal sequences as in standard Beam Search (see illustration in Figure 5): for each frame, we hold the N best candidate sequences so far, where N is a hyper parameter (beam width), empirically chosen to be 10 for all our experiments ($N = 3$ in the figure). Denote by C the candidate classes in the current frame (including the *none* class). Each of the classes is concatenated to each of the N current sequences, yielding $N \cdot |C|$ new candidate sequences. From these, only the best N are chosen, according to the target score. In order to maintain effectiveness on long sequences, the average scores are computed over a fixed-length suffix of each sequence. Keeping the N best sequences after each frame, rather than only the first, allows to correct errors retroactively over a short temporal window.

The adaptation of standard Beam Search to our setting includes the following (see Supplementary for full details): (i) A low threshold of vision score, for candidate classes (ii) From all candidate sequences yielding the same string, keeping only the highest scoring sequence (iii) Maximizing the sum of vision and language scores.

4. Evaluation

4.1. Dataset

We introduce the TypingHands26 dataset. Our dataset comprises videos of standard touch typing of 26 users. Total typing time for each user varies between 30 minutes and 90 minutes on the keyboard, summing up to 24:50 hours. The dataset also contains flat-surface typing data (where the users receive no live visual feedback) of 10 of the users: 20-50 minutes for 8 users, and 7-8 minutes for 2 additional ones. Detailed information on the dataset is provided in the Supplementary material. Overall, there are approximately 3M labeled frames, of which $\sim$39% contain an active key. All keyboard sessions are fully labeled at the frame level. From the surface data, $\sim$41% of the frames are labeled. The unlabeled frames mostly have no active key, as explained in Section 3.2.

The texts are mainly excerpts from classic English literature, to cover a wide vocabulary and combinations of letters. Texts also include New York Times articles and scientific articles, and some word lists containing less frequent letters, such as 'j', 'q', 'x' and 'z'.

Each of the videos is segmented frame by frame to extract the palms and remove the background, as mentioned in section 3. After segmentation, we extract bounding boxes around each of the hands. During training, we place the hands in random image locations, and on random backgrounds from the COCO dataset (in varying intensity, as proposed by Arar *et al.* [1]), on the fly.

4.2. Metrics

Although our model's performance can be evaluated using standard classification metrics on frames: precision, recall, etc., we are more concerned with the text accuracy. We use the normalized Levenshtein distance [15], which is a standard metric for evaluating text entry systems [3]. It measures the minimum amount of edits required to change a given string into a target string, relative to the target string's length. We apply it both on the character-level and word-level over the produced text. The target length (used for normalization) is the number of target characters, or words, for the character- and word-level metrics, respectively. We define the text accuracy as:

$$\text{Acc}(pred, target) = 100 \cdot (1 - d_N(pred, target)) \quad (1)$$

where d_N is the normalized metric, on either character or word level. Thus we can speak of text accuracy percentage.

5. Experiments

10 of the 26 users in the dataset have recorded surface data, therefore we focus our analysis on them, leaving one surface session for each user for testing. The lengths of the test sessions are between 2:30-5:00 minutes, except for the users Sh. and Jo. (see Supplementary material for further information about our dataset and in particular time and text length of the test sessions). In total, the test sessions sum up to $\sim$31 minutes and $\sim$13K entered characters.

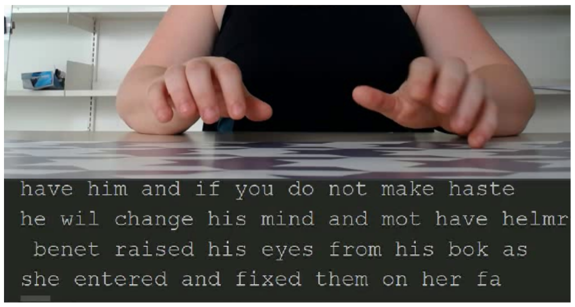
Training details. We train the network with a batch size of 24 and sequence length of 48 frames, using a sticky hidden state. We use an Adam optimizer with learning rate 10^{-4} and weight decay 10^{-6} for all our experiments. Additionally, we reduce the learning rate by a factor of 0.5 upon 5 epochs with no improvement in the training loss. We perform random temporal subsampling during training for robustness to variance in typing speeds.

As already mentioned, for the surface data we assign labels only to frames with at least one active key. However, the RNN requires full video sequences. Therefore, during training we feed it with the full sequence, but backpropagate the loss only for frames with assigned labeling.

We note that in all surface sessions recorded using this prototype, the users received no visual feedback — an undoubtedly disadvantageous scenario. We leave investigating the effect of live visual feedback on users' surface accuracy to future work.

The results are shown in Table 1. We report character level, word level, and frame level accuracy, with 4 different language-based correction policies. We also report frame-level precision, recall, and F1 score on the test sessions' labeled frames. As can be seen, the system handles various typing speeds in real-time (30 FPS) at a promising accuracy. Mean character level accuracy with our real-time language layer is 93.5%. We note that the average character-level accuracy for proficient touch typists on a physical QWERTY keyboard is 98.2% [2], which is our upper bound.

In addition, we tested our system's performance on a new user - for each user, we trained a network on all other ones. Validation and early stopping were done on surface sessions of the seen users. Results can be seen in Table 3. As can be expected, the accuracy is lower for users that the network has never seen, however this does demonstrate encouraging



```
have him and if you do not make haste
he wil change his mind and mot have helmr
 benet raised his eyes from his bok as
she entered and fixed them on her fa
```

Figure 6. Example of our system's output text in test time, after applying beam search with an LSTM language model, for the user Ni. More examples can be seen in the Supplementary material, and the video.

generalization capabilities, suggesting that given a more diverse dataset, the system would deal well with unseen users.

5.1. Ablation studies

In order to examine the effect of our training scheme and self-refinement, we followed five different training schemes, reported in Table 4: training only on keyboard data (row 1), training first on keyboard data, and fine-tuning on surface data labeled using self-refinement (row 2), training on all data from scratch (row 3), and finally our scheme - training first on keyboard data, and fine-tuning on surface data (labeled using self-refinement) combined with keyboard data (row 4). In addition, we performed the second training phase using Pseudo-labels [14], rather than labels obtained by self-refinement (row 5). All five schemes were run for the same amount of time, where the used point was selected according to the character-level Levensthein distance, measured on validation surface sessions different than the test surface sessions.

To evaluate the effect of the language layer on the test accuracy, we tested our system with no language model, with an LSTM-based one, and with a Transformer based one (see Table 1, columns 2-9). Both models increase the accuracy significantly, on both character- and the word-level. An excerpt from a test output text can be seen in Figure 6. The full test outputs can be found in the Supplementary. We also applied Google's auto-correct on the output of the LSTM-based language layer, which gave further improvement.

Details on our experiments, as well as further experiments and ablation studies, are provided in the Supplementary material - robustness to changes in angle, changes in typing speed, the effect of temporal context, and more.

5.2. Comparisons

Alternative approach - hand pose estimation. We compare our method to classification based on estimated hand pose from the Google Media Pipe Hand Hand Landmark Model [31] and show superior performance of 93.5% compared to 54.8%, as can be seen in the Supplementary.

Existing virtual keyboards. We summarise existing virtual keyboard methods in Table 2. As can be seen, ours is the only method using only a simple camera, not requiring any additional equipment or advanced sensing devices. Beside having a more simple and applicable setting, our method is the only one allowing natural typing speeds, with the exception of Richardson *et al*. (row 7), who use marked gloves and an HMD, and a camera located above the hands.

The only existing work operating with a simple camera alone, to our knowledge, is that of Murase *et al*. [22], who also use the challenging yet favorable camera angle as we do, and report almost identical accuracy. However, they disregard typing speed altogether, and test their system on a single user, typing a single sentence of 69 characters repeated 10 times. In contrast, our test set comprises 10 users, with overall time of $\sim$ 31 minutes, and $\sim$ 13K entered characters. Due to this very limited usage scenario, we did not include this work in the Table.

6. Discussion

In this paper, we have presented typeNet, the first system, as far as we know, that lends itself to rapid text entry for mobile devices. The system employs nothing but a single camera, runs in interactive rates, and operates from an angle that is challenging, but convenient for mobile use. We note that for reaching a product level, higher accuracy is required — we believe the accuracy should be at around 97% (i.e. 99% excluding human typing errors) for the system to be productizable. Regardless, this paper has demonstrated that the presented approach, of leveraging touch typing standards to drastically improve computational performance and accuracy, holds great merit, even though it does not address untrained typists. Relying on relatively scarce data of only 26 users, the system still presents high levels of accuracy for users it is already familiar with, at natural typing speeds of up to 80 Words Per Minute, and shows promising preliminary results for unknown users. We note that an extensive user study is required in order to provide a user-friendly system that can be deployed on end-devices.

Except for collecting significantly more data and incorporating more elaborate feedback for the user (*e.g*., through a suggestions mechanism), we believe the most merit for the system can stem from its better generalization to new users. This includes creating larger datasets, but also implies an interesting direction for future work, that takes user specific typing styles into account.

Another direction for future work lies with the language layer. Strengthening the language layer will most likely improve performance. This includes examining the layer's architecture, the data used for training, and perhaps applying another language pass on the output text, regardless of the vision signal.

User	Raw C	W	LSTM C	W	AC C	W	Transf. C	W	P	R	F1	W/M
Gi.	90.6	70.2	93.7	80.8	93.7	84.6	93.5	82.7	90.1	82.3	86.1	$\sim$ 85
Am.	89.6	56.9	95.4	85.0	95.0	88.9	93.7	87.6	93.7	85.7	89.5	$\sim$ 74
Ni.	92.3	68.8	93.4	76.1	95.1	84.2	94.9	81.8	82.6	80.7	81.7	$\sim$ 68
Joh.	94.8	73.0	96.5	83.0	96.7	87.3	96.1	84.6	89.0	87.4	88.2	$\sim$ 62
Sh.	82.3	39.2	88.3	67.6	84.1	73.0	86.8	66.2	87.2	80.0	83.4	$\sim$ 55
Be.	96.5	84.9	97.9	90.2	98.2	93.3	98.4	93.7	96.8	87.1	91.7	$\sim$ 46
Ey.	82.1	28.6	81.5	26.8	84.1	48.2	83.3	35.7	92.5	87.7	90.0	$\sim$ 45
Jor.	92.2	78.2	94.0	85.1	95.5	88.5	94.0	85.1	93.9	94.3	94.1	$\sim$ 42
Ro.	89.0	71.8	89.0	65.9	90.9	75.3	89.0	72.9	92.5	91.4	92.0	$\sim$ 36
Jo.	82.6	48.9	85.1	59.6	85.5	57.4	83.0	51.1	89.3	78.6	83.6	$\sim$ 33
Mean	**91.4**	**68.7**	**93.5**	**78.7**	**94.1**	**84.0**	**93.6**	**81.5**	**91.0**	**85.7**	**88.3**	-

Table 1. Reported accuracy for test surface sessions. From left: Test character- (C) and word- (W) level text accuracy, for no language layer (raw, columns 2-3), after Beam Search using a real-time LSTM language model (columns 4-5), after applying Google's auto-correct on the latter (AC, columns 6-7), after Beam Search using a Transformer (columns 8-9), and raw frame level metrics - precision, recall and F1 score, (rows 10-12). Last column is the user's typing speed, measured by words per minute.

Method	Free	Equipment	Angle	Feedback	Acc. C (%)	Max W/M
Canesta [24]	No	Sensor, projector + light,	Front	Yes	96.3	61
Du *et al.* [7]	No	3d range camera, projector	Above	Yes	88.6	30
CamK [30]	No	Simple camera + paper	Front	Yes	93	30
Scurry [13]	Yes	Wearble gyroscope	None	Yes	86.5	-
ATK [29]	Yes	Leap Motion	Beneath	Yes	86 / 99.7 (UER)	29.2
Richardson *et al.* [23]	Yes	HMD + Marked gloves	Above	Yes	97.35 (UER)	73
Ours	Yes	**Simple camera**	Front	**No**	93.5	**80+**

Table 2. Existing virtual keyboards. For each method we report whether it relies on fixed virtual key locations or allows free typing (column 2), the required equipment (column 3), the camera angle relative to the hands (column 4), whether or not the user receives visual feedback (column 5), the measured average character-level text accuracy (column 6) and maximum typing speed in words/minute (column 7). Kim *et al.* (Scurry, row 4) do not report typing speed. UER - uncorrected error, i.e., remaining errors after correction by user.

User	C	W	LSTM C	W	AC C	W
Be.	87.9	55.1	91.7	67.7	93.5	78.2
Ni.	76.2	28.7	81.2	47.8	82.4	59.5
Gi.	82.0	42.8	85.6	56.0	86.5	68.4
Joh.	79.0	31.7	85.8	60.2	86.0	66.4
Jor.	79.9	36.8	85.2	62.1	85.8	69.0
Am.	83.1	46.4	83.7	45.8	86.4	64.1
Me.	**81.6**	**41.3**	**85.7**	**56.8**	**86.9**	**68.1**

Table 3. Test accuracy for users unseen by the system.

Training	Raw C	W	LSTM C	W
Only KB	82.0	40.4	82.1	50.3
KB, Surface	84.8	45.8	81.0	39.2
KB+Surface	88.5	58.1	91.6	72.8
KB,KB+Surface	**91.4**	**68.7**	**93.5**	**78.7**
Pseudo-Labels	86.2	49.8	91.2	72.0

Table 4. Effect of Self-Refinement on accuracy: training only on keyboard data (row 1), training on keyboard data and fine-tuning on surface data (row 2), training from scratch on both keyboard and surface data (row 3), training on keyboard data, then fine-tuning on both keyboard and surface data with self-refinement (row 4), and training on keyboard data, then fine-tuning on both keyboard and surface data with Pseudo-labels (row 5).

More future work lies in investigating the application of our proposed self-refinement framework to other fields, where a sequential detector requires adaptation to a new domain, such as in speech recognition (ASR), automatic lip reading, and automatic music transcription (AMT).

Finally, we believe this system highlights the common problems with current text entry methods for mobile devices, and suggests a promising direction to its solution, using modern Machine Learning techniques. We hope this method will lead to significant impact on the way we interact with mobile devices, by allowing natural typing speeds even in a mobile scenario.

7. Acknowledgements

The project was supported in part by Len Blavatnik and the Blavatnik family foundation. This research was also partially sponsored by Toga Networks Ltd. We Thank Professor Joel Lanir for his insights on user experience, and Dr. Leo Poretski for help in data gathering. We thank Shahaf Goren for his help with the hand segmentation module.

References

[1] Moab Arar, Noa Fish, Dani Daniel, Evgeny Tenetov, Ariel Shamir, and Amit Bermano. Focus-and-expand: Training guidance through gradual manipulation of input features. *CoRR*, abs/2007.07723, 2020.

[2] Ahmed Arif and Wolfgang Stuerzlinger. Analysis of text entry performance metrics. pages 100 – 105, 10 2009.

[3] Ahmed Sabbir Arif and Wolfgang Stuerzlinger. Analysis of text entry performance metrics. In *2009 IEEE Toronto International Conference Science and Technology for Humanity (TIC-STH)*, pages 100–105, 2009.

[4] Hsi-Jen Chen and Chia-Ming Kuo. Investigation of the effect of letter labeling positions on consecutive typing on mobile devices. In *International Conference on Human-Computer Interaction*, pages 3–16. Springer, 2019.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[6] Gencay Deniz and Pınar Onay Durdu. A comparison of mobile form controls for different tasks. *Computer Standards & Interfaces*, 61:97–106, 2019.

[7] Huan Du, Thierry Oggier, Felix Lustenberger, and Edoardo Charbon. A virtual keyboard based on true-3d optical ranging. In William F. Clocksin, Andrew W. Fitzgibbon, and Philip H. S. Torr, editors, *Proceedings of the British Machine Vision Conference 2005, Oxford, UK, September 2005*. British Machine Vision Association, 2005.

[8] Leah Findlater, Jacob O. Wobbrock, and Daniel Wigdor. Typing on flat glass: examining ten-finger expert typing patterns on touch surfaces. In Desney S. Tan, Saleema Amershi, Bo Begole, Wendy A. Kellogg, and Manas Tungare, editors, *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, pages 2453–2462. ACM, 2011.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[10] Eve Hoggan, Stephen A Brewster, and Jody Johnston. Investigating the effectiveness of tactile feedback for mobile touchscreens. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1573–1582, 2008.

[11] Senseboard inc. *Senseboard*, 2010.

[12] Virtual Keyboard. *Virtual Keyboard*, 2007.

[13] Yoon Sang Kim, Byung Seok Soh, and Sang-Goog Lee. A new wearable input device: SCURRY. *IEEE Trans. Ind. Electron.*, 52(6):1490–1499, 2005.

[14] Dong-Hyun Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 07 2013.

[15] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10 (8): 707–710, 1966.

[16] Mark F. Medress, Franklin S. Cooper, James W. Forgie, C. C. Green, Dennis H. Klatt, Michael H. O'Malley, Edward P. Neuburg, Allen Newell, Raj Reddy, H. Barry Ritea, J. E. Shoup-Hummel, Donald E. Walker, and William A. Woods. Speech understanding systems. *Artif. Intell.*, 9(3):307–316, 1977.

[17] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An analysis of neural language modeling at multiple scales. *CoRR*, abs/1803.08240, 2018.

[18] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[19] Adiyan Mujibiya, Takashi Miyaki, and Jun Rekimoto. Anywhere touchtyping: text input on arbitrary surface using depth sensing. In Ken Perlin, Mary Czerwinski, and Rob Miller, editors, *Adjunct proceedings of the 23nd annual ACM symposium on User interface software and technology, UIST '10, New York, New York, USA, October 3-6, 2010*, pages 443–444. ACM, 2010.

[20] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[21] Taichi Murase, Atsunori Moteki, Noriaki Ozawa, Nobuyuki Hara, Takehiro Nakai, and Katsuhito Fujimoto. Gesture keyboard requiring only one camera. In Jeffrey S. Pierce, Maneesh Agrawala, and Scott R. Klemmer, editors, *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, Santa Barbara, CA, USA, October 16-19, 2011 - Adjunct Volume*, pages 9–10. ACM, 2011.

[22] Taichi Murase, Atsunori Moteki, Genta Suzuki, Takahiro Nakai, Nobuyuki Hara, and Takahiro Matsuda. Gesture keyboard with a machine learning requiring only one camera. In Jean-Marc Seigneur, Hartmut Koenitz, and Guillaume Moreau, editors, *Proceedings of the 3rd Augmented Human International Conference, AH 2012, Megève, France, March 8-9, 2012*, page 29. ACM, 2012.

[23] Mark Richardson, Matt Durasoff, and Robert Wang. Decoding surface touch typing from hand-tracking. In Shamsi T. Iqbal, Karon E. MacLean, Fanny Chevalier, and Stefanie Mueller, editors, *UIST '20: The 33rd Annual ACM Symposium on User Interface Software and Technology, Virtual Event, USA, October 20-23, 2020*, pages 686–696. ACM, 2020.

[24] Helena Roeber, John Bacus, and Carlo Tomasi. Typing in thin air: the canesta projection keyboard-a new method of interaction with electronic devices. In *CHI'03 extended abstracts on Human factors in computing systems*, pages 712–713, 2003.

[25] Jeff Rowberg. *Keyglove*, 2015.

[26] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 331–335. Association for Computational Linguistics, 2019.

[27] Qizhe Xie, Minh-Thang Luong, Eduard H. Hovy, and Quoc V. Le. Self-training with noisy student improves imagenet classification. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10684–10695. IEEE, 2020.

[28] I. Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. *CoRR*, abs/1905.00546, 2019.

[29] Xin Yi, Chun Yu, Mingrui Zhang, Sida Gao, Ke Sun, and Yuanchun Shi. ATK: enabling ten-finger freehand typing in air based on 3d hand tracking data. In Celine Latulipe, Bjoern Hartmann, and Tovi Grossman, editors, *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, UIST 2015, Charlotte, NC, USA, November 8-11, 2015*, pages 539–548. ACM, 2015.

[30] Yafeng Yin, Qun Li, Lei Xie, Shanhe Yi, Edmund Novak, and Sanglu Lu. Camk: A camera-based keyboard for small mobile devices. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, pages 1–9. IEEE, 2016.

[31] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *CoRR*, abs/2006.10214, 2020.