CRTypist: Simulating Touchscreen Typing Behavior via Computational Rationality (Supplementary Material)

This supplementary material is organized as follows:

- Section A shows the ablation study of working memory.
- Section B describes the processing of large-scale human typing data in the benchmark.
- Section C introduces the details of the screenshots collection in the benchmark.
- Section D presents the graphical user interface for simulation on screenshots.
- Section E describes the model's integration with the Android emulator.

A ABLATION STUDY

We ran an ablation study to assess the contribution of working memory on model's overall performance. Specifically, we validate the effectiveness of the working memory design with its two key components: correctness and certainty. To compare the performance of CRTypist, we tested three model designs:

- CRTypist-NoCorrectness: the model excluded correctness in the working memory,
- CRTypist-NoCertainty: the model excluded uncertainty in the working memory
- CRTypist-NoWM: the model excluded working memory altogether.

We trained all models with the same hyperparameters and set identical model parameters to compare performance in terms of the typing metrics. Table A1 shows the comparison results. Only CRTypist was able to simulate a similar typing performance with the human baseline. The other models had various flaws: CRTypist-NoCorrectness was able to proofread, but it shifted its gaze much more frequently than humans; CRTypist-NoCertainty was very confident with its typing results, leading to no longer looking at the text display. However, to increase the correctness, it slowed down its finger movement, resulting in a significant increase in IKI; CRTypist-NoWM was the worst. It had the slowest movement and frequently used backspaces to pursue a good performance.

	WPM	**IKI**	**BS**	**Error%**	**GS**	**kbd%**
Human Baseline	27.2 (3.6)	380 (51)	2.6 (1.8)	0.6 (0.9)	3.9 (1.5)	70 (14)
CRTypist	+1.7	-14	-0.2	-0.5	+1.6	+1
CRTypist-NoCorrectness	+0.7	+52	-2.5	+5.3	+6.2	-19
CRTypist-NoCertainty	-8.3	+241	-2.6	-3.9	-3.9	+30
CRTypist-NoWM	-24.3	+455	+22.2	+2.0	-3.9	+29

Table A1. CRTypist performs more realistic than the model without working memory or part of it in the ablation study results in terms of Words per minute (WPM), Internal-key interval (IKI), Number of Backspaces (BS), Error rate (ER%), Number of gaze shifts (GS), and Gaze-on-keyboard time ratio (kbd%). The color encodes the level of deviation from the standard: dark green (1 std), light green (2 std), light red (3 std), and dark red (more than 3 std).

B MOBILE TEXT-ENTRY DATA FROM HUMANS

To understand individual differences in typing, we gathered a large amount of mobile text-entry data from a web-based transcription task [1]. For single-finger vs. two-thumb typing, we divided the data into two groups (see Figure **??**). For guaranteed similarity to routine typing behavior, only entries with a final character-error rate below 5% are included. If the charactor error rate is over 5%, the word error rate can be very high (over 22.6%), which could be intended fast but sloppy typing. Compared to the average typing speed, the median gives a better sense of the "center" of the data without being affected by extreme values. Because the median typing speeds are slightly below the average, the dataset shows a typically right-skewed distribution (some individuals type significantly faster than the others).

C COLLECTION OF REAL-WORLD MOBILE KEYBOARD SCREENSHOTS

To build a pixel-based training environment, we collected screenshots with diverse keyboard applications from Google Play. Three criteria informed the collection process: 1) for consistency, the keyboard had to have a QWERTY layout; 2) we set a popularity threshold of one million downloads for validity; and 3) the keyboard design had to support both one-finger and two-thumb typing. The screenshot collection includes 38 highly ranked keyboard applications from Google Play (see Table C2), covering more than 95% of the Google Play keyboard market's downloads. For each keyboard, we enumerated all design variants, thereby arriving at 1,028 unique screenshots. To speed up the training on screenshots, we label the position of letter keys on each screenshot, along with special keys, including Enter, Backspace, and Space bar. We release the screenshots and labeling data with the benchmark.

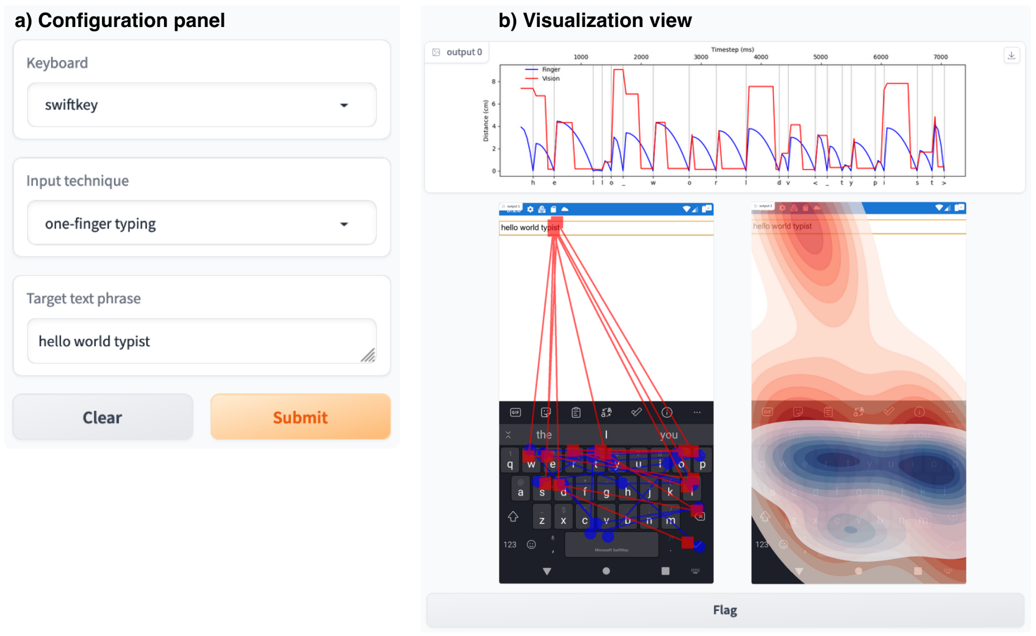
Table C2. Keyboard applications included in the benchmark

ID	Keyboard	DL (M)	Rating	ID	Keyboard	DL (M)	Rating
0	Gboard	5000	4.5	19	Deco Keyboard	10–50	4.0
1	Microsoft Swiftkey	1000–5000	4.2	20	2023 Keyboard	10–50	4.4
2	Emoji keyboard	100–1000	4.5	21	Classic Big Keyboard	10–50	4.2
3	Fonts Keyboard	100–1000	4.3	22	iKeyboard GIF Keyboard	10–50	4.5
4	GoKeyboard	100–1000	4.4	23	Stylish Fonts & Keyboard	10–50	4.0
5	Facemoji Keyboard	100–1000	4.6	24	My Photo Keyboard	10–50	4.3
6	Bobble Keyboard	50–100	4.5	25	Laban Key	10–50	4.5
7	LED Keyboard	50–100	4.6	26	Malayalam Keyboard	10–50	4.6
8	Neon LED Keyboard	50–100	4.6	27	Hacker's Keyboard	10–50	4.0
9	Giphy Keyboard	50–100	4.1	28	iMore Keyboard	10–50	4.2
10	Cute Emoji Keyboard	50–100	4.3	29	Decoration Keyboard	10–50	4.5
11	Ridmik Keyboard	50–100	4.3	30	All Arabic Keyboard	10–50	4.6
12	Grammarly Keyboard	10–50	4.4	31	Fast Typing Keyboard	10–50	4.4
13	Yandex Keyboard	10–50	4.7	32	Ginger Keyboard	5–10	4.3
14	Design keyboard	10–50	4.5	33	Red Keyboard	5–10	4.8
15	Kika Keyboard	10–50	4.3	34	Frozen Keyboard	5–10	4.6
16	Fonts Art: Keyboard	10–50	4.1	35	Flames Keyboard	5–10	4.4
17	Stylish Text Keyboard	10–50	4.4	36	Playkeyboard	1–5	4.4
18	Go Keyboard Pro	10–50	4.3	37	AnySoftKeyboard	1–5	3.8

[1]https://userinterfaces.aalto.fi/typing37k/

D GRAPHICAL USER INTERFACE FOR VISUALIZING SIMULATION DATA ON SCREENSHOTS

We developed a web interface to visualize the simulation data on touchscreens. The purpose is to visually illustrate the typing behavior of the trained model given a keyboard screenshot and a target text phrase. The web interface, as shown in Figure D1, consists of two major components: configuration panel and visualization view. In the configuration panel (see Figure D1-a), users can select a keyboard as the design choice, enter a target text phrase as the goal, and pick an input technique (single-finger or two-thumb typing). After clicking the "Submit" button, the trained model will be loaded and simulate the typing behaviors based on the given inputs. The simulated behavior data is visualized in the visualization view through three visualizations (see Figure D1-b): a time series chart, a trajectory view, and a heatmap. The time series chart presents the key-by-key distances from the positions of gaze and finger to the next key to tap over time. The movement from one position to another is quadratically interpolated. The trajectory view and the heatmap are visualized on the given screenshot to show the positions of gaze and finger. The trajectory view reveals the movement patterns during typing, and the heatmap highlights the regions traversed by finger(s) and gaze. This web interface is built on Gradio [2], a Python library that can generate a web interface for machine learning models. It is integrated into the source code and is locally hosted.


a) Configuration panel

Keyboard: swiftkey

Input technique: one-finger typing

Target text phrase: hello world typist

b) Visualization view

Fig. D1. The graphical user interface for visualizing simulation data on screenshots

[2]https://www.gradio.app

E INTEGRATION WITH ANDROID EMULATOR

To support training, evaluating, and demonstrating models on real mobile keyboards, we integrate our model with the Android emulator via AndroidEnv [1], a Python library that exposes the Android system as an interactive environment. A trained model can be fine-tuned via AndroidEnv interaction: the model takes pixels from the emulator as input and feeds click actions to the emulator. With the integration of AndroidEnv, the model can be used for simulation with real mobile keyboards, greatly improving its practical utility. Users can easily set up the model to run on an Android emulator by following the steps:

1. Follow the AndroidEnv guide [3] to set up the Android environment.
2. Launch the target AVD on the emulator; Install the target keyboard via Google Play Store or drag the local APK file onto the emulator; Enable the target keyboard as the default in the setting.
3. Run the scripts to train the model or fine-tune a model that is pre-trained on the screenshot. This step is optional, as users can directly use a pre-trained model for evaluation.
4. Put the trained models under the model's folder "crtypist/android_typing_env/outputs"; Run the scripts to evaluate and demonstrate the trained model.

The AndroidEnv will show the real-time typing simulation on an Android emulator during evaluation and demonstration. To visualize the finger and gaze movements, we render a dynamic view (left in Figure E2) to show gaze and finger on the RGB values corresponding to the displayed pixels on the Android emulator (right in Figure E2).

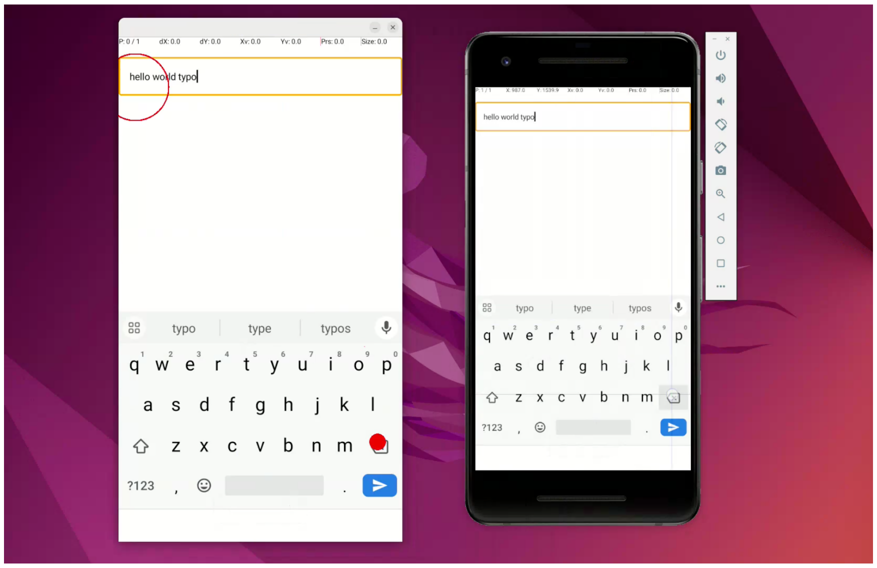


Fig. E2. The model simulates typing behavior on the Android emulator.

REFERENCES

[1] Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. 2021. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231* (2021).

[3]https://github.com/deepmind/android_env/blob/main/docs/emulator_guide.md