KeySense: LLM-Powered Hands-Down, Ten-Finger Typing on Commodity Touchscreens

Tony Li
Department of Computer Science
Stony Brook University
Stony Brook, New York, USA
haolili@cs.stonybrook.edu

Yan Ma
Computer Science Department
Kean University
Union, New Jersey, USA
yan.ma@kean.edu

Zhuojun Li
Department of Computer Science and Technology
Tsinghua University
Beijing, China
lizj23@mails.tsinghua.edu.cn

Chun Yu
Department of Computer Science and Technology
Tsinghua University
Beijing, China
chunyu@tsinghua.edu.cn

IV Ramakrishnan
Computer Science
Stony Brook University
Stony Brook, New York, USA
ram@cs.stonybrook.edu

Xiaojun Bi
Department of Computer Science
Stony Brook University
Stony Brook, New York, USA
xiaojun@cs.stonybrook.edu

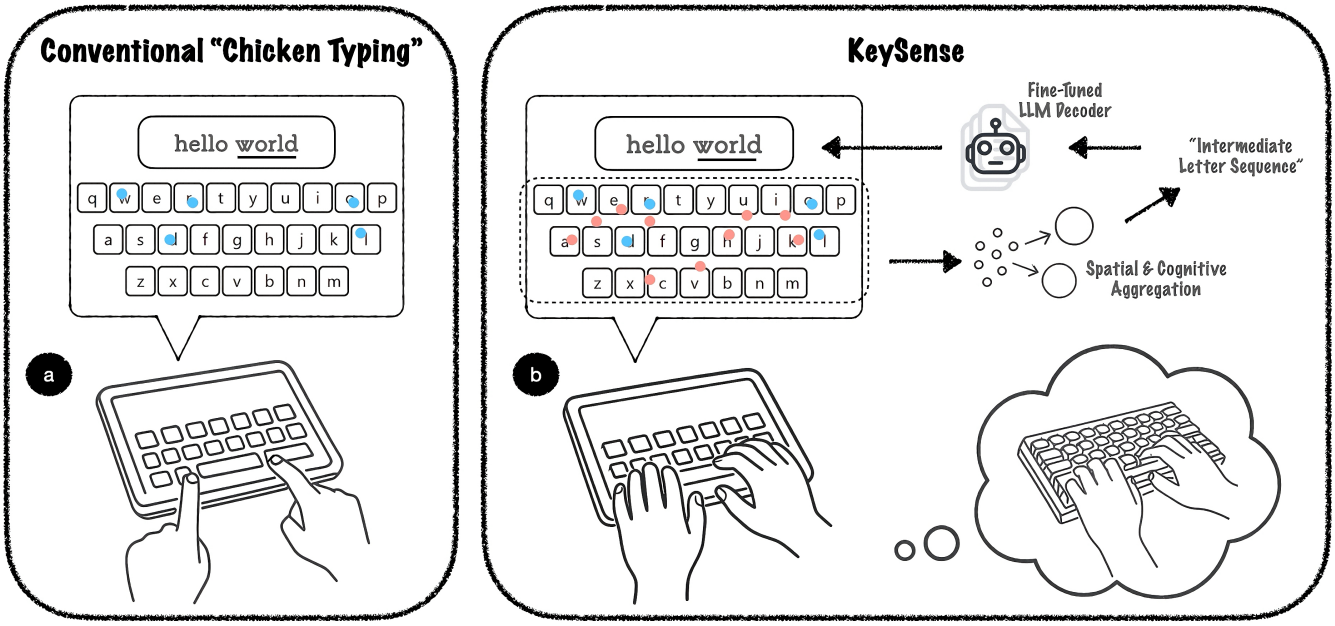

Figure 1: KeySense enables hands-down, ten-finger typing on commodity touchscreens by letting users rest their fingers while the system leverages an LLM to decode the intended word. (a) Conventional touch keyboards encourage two-finger "chicken typing" where users hover. (b) With KeySense, users keep both hands down; blue dots mark intentional reaches, red dots mark resting touches, and the underlined text shows the current word in progress.

Abstract

Existing touchscreen software keyboards prevent users from resting their hands, forcing slow and fatiguing index-finger tapping ("chicken typing") instead of familiar hands-down ten-finger typing. We present KeySense, a purely software solution that preserves physical keyboard motor skills. KeySense isolates intentional taps from resting-finger noise with cognitive–motor timing patterns, and then uses a fine-tuned LLM decoder to turn the resulting noisy letter sequence into the intended word. In controlled component tests, this decoder substantially outperforms 2 statistical baselines (top-1 accuracy 84.8% vs 75.7% and 79.3%). A 12-participant study shows clear ergonomic and performance benefits: compared with the conventional hover-style keyboard, users rated KeySense as markedly less physically demanding (NASA-TLX median 1.5 vs 4.0),

CHI '26, Barcelona, Spain

ACM ISBN 979-8-4007-2278-3/26/04
https://doi.org/10.1145/3772318.3790964

and after brief practice, typed significantly faster (WPM 28.3 vs 26.2, p <0.01). These results indicate that KeySense enables accurate, efficient and comfortable ten-finger text entry on commodity touchscreens, without any extra hardware.

CCS Concepts

• **Human-centered computing → Text input**.

Keywords

text entry, ten-finger typing, soft keyboards, language models

ACM Reference Format:
Tony Li, Yan Ma, Zhuojun Li, Chun Yu, IV Ramakrishnan, and Xiaojun Bi. 2026. KeySense: LLM-Powered Hands-Down, Ten-Finger Typing on Commodity Touchscreens. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26), April 13–17, 2026, Barcelona, Spain.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3772318.3790964

1 Introduction

Ten-finger typing on physical keyboards is fast, comfortable, and built on decades of motor learning: all fingers rest on a home row, and keystrokes are produced through small reaches. On touchscreens, however, this motor program breaks down. Because any surface contact risks triggering an unintended keystroke, users must hover their hands and peck with one or two fingers ("chicken typing"; Figure 1a), leading to fatigue and preventing transfer of motor skills.

This difficulty reflects a deeper issue articulated by Buxton's three-state model of input [7]. Whereas some pointer devices such as mice can cleanly separate out-of-range, tracking, and dragging states, touchscreens collapse tracking and dragging states: any contact with the screen is treated as an intentional input, producing nearly identical (x, y, t) signals. This phenomenon also known as the MIDAS-touch problem [27]. This loss of state separation is the fundamental reason why hands-down typing on touchscreens is inherently ambiguous.

Prior approaches have explored specialized hardware such as pressure sensors, depth cameras, or instrumented surfaces [23, 33, 60] to separate accidental contacts from intentional touch input, but such components are absent from commodity tablets. Software-only methods have instead focused on sentence-level decoding or simplified heuristics [29, 30, 51], which do not resolve the ambiguity introduced by resting fingers in hands-down input. The central challenge remains: with only raw (x, y, t) touch events, can intentional keystrokes and incidental contact be distinguished?

We present *KeySense*, a software-only system that enables comfortable, hands-down ten-finger typing on commodity, pressure-insensitive touchscreens. The system consists of two stages. A lightweight front end reconstructs touch *threads*, groups near-synchronous contacts within a short cognitive window, and selects one representative per cluster using a reach-aware heuristic that suppresses resting touches. The resulting noisy letter sequence is then corrected by a compact LLM (FLAN–T5-small) fine-tuned on a synthetic corpus that models human surface-typing errors.

Our evaluations show that this combination of structured preprocessing and LLM-based decoding yields strong accuracy. On a synthetic test set, the fine-tuned FLAN–T5-small achieves **84.8%** Top-1 accuracy, substantially outperforming two statistical baselines (**75.7%** and **79.3%**) and zero-shot LLM variants.

A 12-participant study further demonstrates the practical impact of this approach. Participants rated KeySense as far less physically demanding than the hover-typing baseline (NASA-TLX median 1.5 vs. 4.0), and after a brief learning period they typed significantly faster with KeySense in the final session (28.3 vs. 26.2 WPM; $F(1, 118.71) = 8.58, p = 0.0041$). These results show that by decoding noisy multi-touch input rather than suppressing it, commodity touchscreens can support a comfortable ten-finger typing style that ultimately surpasses conventional interaction.

Taken together, this work shows that a purely software-based approach can restore the motor and ergonomic benefits of ten-finger typing on commodity touchscreens. Beyond demonstrating a practical system, the paper also provides the first detailed characterization of hands-down typing behavior on passive surfaces, highlighting how resting and reaching patterns emerge as users adapt to this interaction style.

2 Related Work

As an essential component of Human–Computer Interaction, text entry has been extensively studied in the past few decades. This section discusses two key areas on which this work is built: ten-finger typing and tap-typing decoders.

2.1 Ten-Finger Typing

Ten-finger typing preserves bimanual parallelism and transfers decades of QWERTY muscle memory to touch surfaces, enabling higher sustained throughput with lower visual demand than single- or two-finger methods, especially on tablet-sized displays where hands can rest naturally. Against this backdrop, work on ten-finger text entry without physical keycaps has developed along three complementary fronts: characterizing how experts type on glass, modeling and adapting the keyboard to users, and improving separability between intentional taps and incidental contacts.

Foundational empirical studies showed that expert touch patterns on flat glass are systematic and thus learnable despite the absence of tactile landmarks [18]. Building on this, personalized spatial models that adapt key-target distributions to each individual demonstrably reduce errors and established personalization [17].

A major challenge for ten-finger posture is the resting contacts that confound naive tap detection. *Kim et al.* proposed *TapBoard*, utilizing a distinction between brief, low-motion keystrokes and longer, stable rests [30]. *Kim and Lee* later introduced *TapBoard 2*, streamlining the interaction so users could fluidly switch between typing and pointer-like manipulation on the same surface [29]. Complementary explorations augmented QWERTY touch keyboards with multi-touch gestures for non-alphanumeric input [16] and compressed the full layout into a single horizontal line while preserving familiar spatial ordering [32]. Together, these systems showed that simple temporal and kinematic cues and layout/gesture design, can suppress noise but also impose behavioral constraints on how users must "tap."

A second line of work adds sensing. Pressure-sensitive keyboards such as *TypeBoard* identify unintentional touches [23], while *T-Force*

extends this idea to a three-state virtual keyboard that leverages typing force to distinguish touch states [14]. Building on *TypeBoard*, *ResType* further explored adaptive and invisible layouts on pressure-sensitive hardware [33]. Wearable-augmented systems attach lightweight sensors to fingers and decode QWERTY input on arbitrary surfaces [46, 60], and vision-based approaches infer keystrokes from hand tracking, showing that surface typing can be decoded without capacitive touch [42, 43]. These sensing-rich approaches push performance and flexibility but raise hardware and deployment issues that limit adoption on commodity tablets.

On the decoding side, probabilistic inference has made "invisible keyboards" feasible even on ordinary flat surfaces. *Shi et al.* introduced *TOAST*, which learns user-anchored latent layouts and combines spatial–temporal evidence with lexical priors to recover word sequences with competitive accuracy and speed in eyes-free scenarios [45]. At the extreme, mid-air methods reconstruct ten-finger sequences from 3D hand tracking, revealing the capacity of expert motor plans once contact ambiguity is removed [58]. These results collectively argue that expert ten-finger behavior is modelable, that adaptation pays off, and that principled inference can overcome much of the ambiguity inherent in on-glass typing.

Within this broader landscape, KeySense targets the most practical and widespread scenario: enabling an ergonomic, *hands-down, ten-finger posture* directly on an *unmodified, pressure-insensitive touchscreen*. This posture, where users can comfortably rest their fingers as they would on a physical keyboard, is key to reducing fatigue and leveraging existing muscle memory. Within this context, Table 1 systematically compares our approach to the key prior systems that have also successfully enabled a hands-down posture: *TypeBoard* [23] and *ResType* [33]. KeySense shares the same interaction goal but makes the crucial leap to commodity, pressure-insensitive hardware.

2.2 Tap-Typing Decoders

Tap typing has long relied on probabilistic decoders that fuse a spatial touch model with a language model to resolve noisy taps. A representative family of systems models per-key touch likelihoods (often Gaussian or learned from user data) and searches for the most probable string under a character or word language model. *Vertanen et al.* proposed this paradigm at the *sentence* level with *VelociTap*, showing that deferring commitment and decoding whole phrases boost speed and robustness [51]. On small displays, *WatchWriter* adapts statistical decoding to smartwatch and corrects "fat-finger" taps [22]. Prior work demonstrated that strong language priors can render motor-memory–driven typing practical even when key borders are not visually shown [62]. More broadly, the decoder itself can be framed as Bayesian inference over touch uncertainty, exploring learned touch models and user-controlled uncertainty (e.g., via pressure) [54], while follow-up studies quantified how expanding context from single words to multiple words and sentences further improves decoder performance [49, 50].

Beyond direct finger taps, several tap-like modalities alter the sensing or disambiguation channel. *GlanceWriter* uses gaze dwell as a "tap" over letter regions and decodes text, enabling writing by rapidly glancing across letters [11]. *TapGazer* couples ambiguous multi-finger tapping with gaze-directed word selection and offloads final disambiguation to the user's eyes [25]. In mixed reality, *TouchInsight* leverages egocentric vision to estimate tap intent with uncertainty-aware inference, allowing rapid touches that are resolved with linguistic context [47]. Outside capacitive sensing, *I-Keyboard* demonstrates a fully imaginary on-surface keyboard with a deep neural decoder that maps touch patterns to characters without rendering keys [31], and *Nomon* combines a probabilistic selection model with high-order language models to achieve competitive efficiency in extreme access scenarios [6].

Neural and LLM-based decoders have begun to directly rank or generate intended text from noisy taps under strong priors over words and phrases. It is demonstrated that compact seq2seq models can jointly correct and complete noisy mobile keyboard input [20]. *Type, Then Correct* learns to correct mobile keyboard input errors using post-hoc neural inference rather than real-time decoding [59]. *Ma et al.* proposed an LLM-powered smartphone decoder that unifies tap and flexible typing modes and integrates spatial evidence with powerful contextual predictions, and further showed that the same decoder can be applied to virtual reality text entry [35, 36]. Transformer-based language models can dramatically reduce keystrokes by expanding minimal tap input (e.g., initials) into full phrases [8]. *SkipWriter* also expands abbreviated handwriting into full phrases on tablets [57]. Intelligent post-hoc correction systems such as *JustCorrect* apply semantic cues to repair earlier tap errors with minimal user effort [12].

Taken together, this literature establishes that combining spatial uncertainty with linguistic priors, from word to sentence and now to neural and LLM decoders, substantially improves tap typing [36, 49–51, 54]. Our contribution addresses ten-finger typing on *unmodified* touchscreens by focusing on temporal disambiguation of near-synchronous taps and software-only decoding, complementing prior spatial–linguistic decoders.

3 Pre-Decoding Data Processing Pipeline

Figure 2 summarizes our pipeline and how the components fit together. The upper track converts noisy, low-level touch events into a clean, temporally ordered sequence of key hypotheses; the lower track shows how we train a compact LLM to map that sequence to the intended word.

From events to a sequence (top row). Starting with raw points $(x, y, type, t)$, we (1) aggregate nearby down/move/up samples into *touch threads*, (2) group threads into *clusters* if their onsets fall within a short cognitive window in Section 3.2), and (3) select one representative per cluster using a reach-sensitive *travel score* (Section 3.4). Mapping each selected thread to the nearest key center yields a provisional letter sequence that preserves typing order while suppressing incidental contacts.

From sequence to word (bottom row). We collect typing logs (Section 4.1), synthesize a length-balanced corpus of misspelled–gold pairs (Section 4.2), and fine-tune *FLAN-T5-small* to correct a single word from its noisy letter sequence (Section 4.3). At inference, the fine-tuned model decodes the letter sequence into the final word.

3.1 Cognitive Hypothesis on Typing Interval

We consider the setting where one finger executes an *intentional* press while other resting fingers may produce nearly synchronous,

System	Hardware Requirement	Primary Innovation	Decoder Type
KeySense (Our work)	**Commodity Touchscreen**	Heuristic Filtering + LLM Decoding	Fine-tuned LLM
TypeBoard [23]	Pressure-sensitive Surface	Pressure-based Touch Prevention	Statistical
ResType [33]	Pressure-sensitive Surface	Adaptive Layout	Statistical

Table 1: Design space of systems enabling a *hands-down*, ten-finger typing posture. While TypeBoard and ResType validated this interaction paradigm on pressure-sensitive hardware, KeySense is the first to achieve it on commodity touchscreens through a purely software-based, LLM-powered decoding approach.

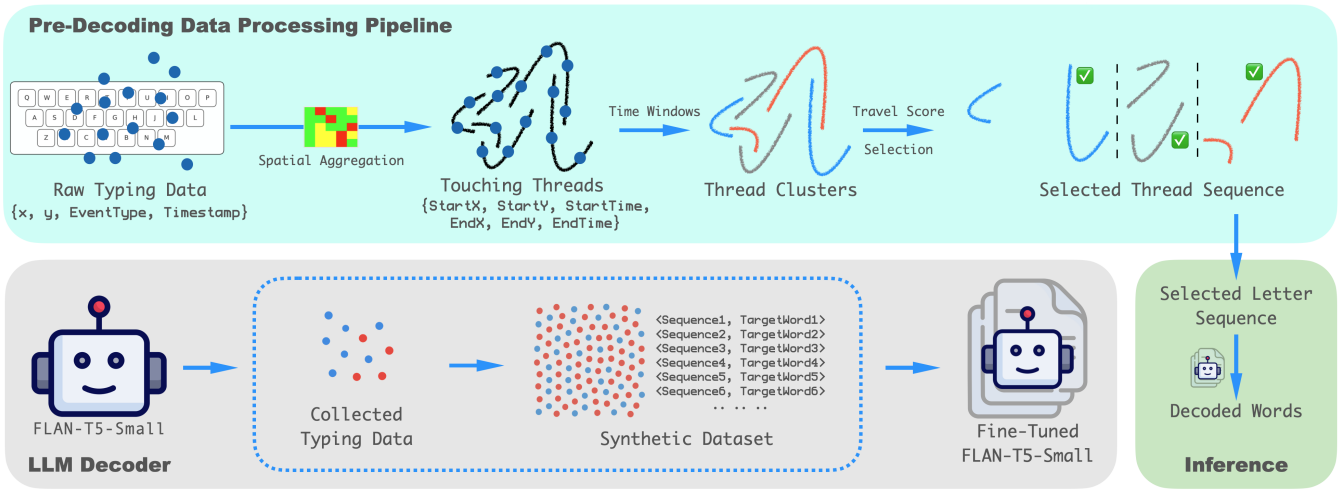

Figure 2: End-to-end overview. Top (pre-decoding): raw touch events are aggregated into *touch threads*, grouped into *time clusters*, and resolved by a reach-sensitive *travel score* to yield one representative per cluster. Each representative is mapped to a key, producing a letter sequence. Bottom (LLM decoder): we parameterize a synthetic corpus from the collected typing logs, fine-tune *FLAN-T5-small* on these pairs, and at inference decode the letter sequence into a word.

unintentional contacts. Our central hypothesis is that human processing limits impose a minimum temporal separation between two *independent* intentional key activations: within a short window τ_c, co-occurring contacts are overwhelmingly likely to reflect a single planned action accompanied by incidental co-contacts. Prior work has similarly addressed grouping near-simultaneous touches: *Evans et al.* distinguish multi-user touch identities on tabletops by statistically modeling touch pairs occurring within short temporal windows [13].

This view is supported by classic human performance models that place cognitive and motor processor cycles on the order of 50–100 ms [9], and by dual-task research on the *Psychological Refractory Period* (PRP), which shows a central bottleneck that serializes response selection when two responses are scheduled in rapid succession [1]. Large-scale typing studies report typical inter-key intervals near 200–250 ms with only a thin lower tail around 60–80 ms, and sustaining rates above 10 keys/s (< 100 ms per keystroke) is difficult without marked errors [15]. From a perception standpoint, a window of roughly 100 ms is a canonical bound for events to appear near-simultaneous in interaction [40].

These strands suggest a compact interval of roughly 100 ms where launching two independent intentional keystrokes is highly implausible. We therefore adopt a cognitive window $\tau_c = 100$ ms: touches whose onsets fall within this gap are treated as near-synchronous, while touches separated by more than τ_c are treated as sequential keystrokes.

3.2 Study on Intentional Touch Interval

Motivated by this cognitive picture and existing practice, we adopt a 100 ms threshold and ask: on real ten-finger typing data, how often do adjacent *intentional* keystrokes fall within this window? Our goal is to verify that treating ≤ 100 ms as near-synchronous merges only a small fraction of genuine successive keystrokes.

We re-analyze the user dataset from *ResType* [33], focusing on Study 2. In that study, participants (8 users) typed MacKenzie phrases [37] on a fixed QWERTY layout rendered on a Sensel Morph touchpad [28], while the same unintentional-touch prevention as *TypeBoard* [23] filtered resting contacts. The log provides per-event timestamps, contact states, and intentionality labels at a fixed sampling rate of 50 fps. We use these labels and timestamps only, as this condition best matches our scenario of ten-finger typing on a standard soft keyboard. Figure 3 shows the spatial distributions: intentional touches cluster over keys, whereas unintentional contacts form a smooth resting arc.

For each participant, we extract the onset time of every intentional contact and compute the adjacent inter-onset gaps $\{\Delta t_m\}$.

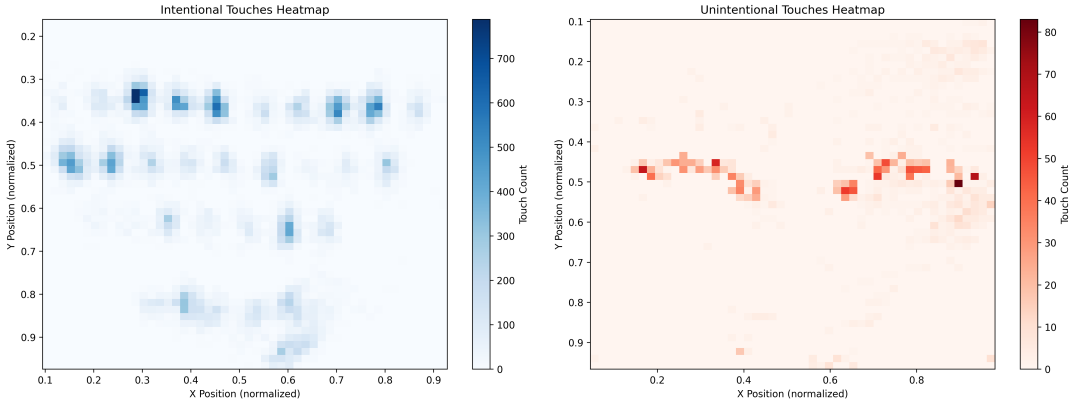

Figure 3: Touch distributions from the ResType dataset [33]. Intentional taps concentrate over key centers (left), whereas unintentional contacts follow a resting-hand arc (right).

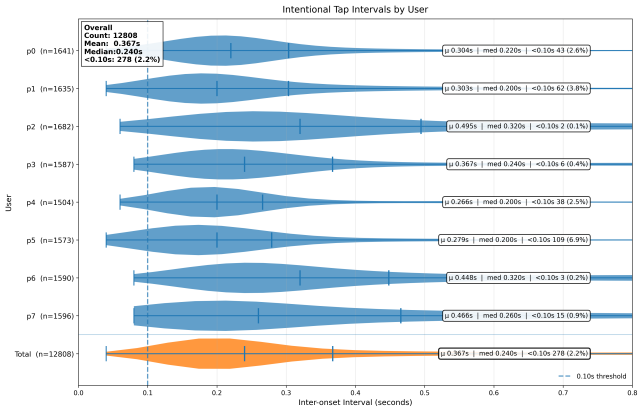

Figure 4: Inter-onset intervals by user (violins) with a 100 ms reference line. The rightmost violin aggregates all users. Labels show counts and percentages of gaps ≤ 100 ms. Means (green) and medians (red) are overlaid.

Our primary statistic is the fraction of gaps at or below 100 ms, i.e., the proportion of successive taps that would be merged if ≤ 100 ms were treated as synchronous. Across $U = 8$ participants and $M = 12{,}808$ adjacent intentional gaps, only **2.17**% of intervals are ≤ 100 ms. A user-stratified bootstrap yields a 95% confidence interval of [**0.87**%, **3.75**%], indicating that this mass is consistently small across users. Figure 4 visualizes the per-user gap distributions and highlights the small share below 100 ms; the pooled median inter-key interval is about **240** ms, with most gaps falling well above this threshold. A sensitivity check that jitters the 100 ms cutoff by ± 10 ms produces nearly identical percentages, suggesting that 20 ms frame quantization has negligible effect on this conclusion.

For system design, these numbers mean that adopting $\tau_c = 100$ ms as a merging window will fuse only about 2% of adjacent intentional keystroke pairs. We therefore use $\tau_c = 100$ ms as a principled, user-agnostic default in our pipeline.

3.3 Thread Formation and Time Clustering

This subsection introduces the first two stages: (i) reconstruct coherent *touch threads* from the raw event stream, then (ii) group near-synchronous threads into short-lived time clusters.

Thread formation from the event stream. The logger yields a sequence of low-level events $e_k = (\texttt{type}_k, t_k, \mathbf{z}_k)$, where $\texttt{type} \in \{\texttt{down}, \texttt{move}, \texttt{up}\}$, t_k is the timestamp, and $\mathbf{z}_k = (x_k, y_k)$ is a normalized location. We maintain a set of open threads, each storing its first/last timestamps and endpoints and a polyline of intermediate moves. A down event starts a new thread; subsequent move and up events are associated to the nearest open thread whose endpoint is within a small spatial and temporal neighborhood, and close the thread when an up is seen.

We perform data association via a light-weight spatial hash. The keyboard plane is partitioned into a uniform grid of cell size h and index the current endpoints of all open threads. Given an event at $\mathbf{z}_k$, we probe the occupied entries in the cell containing $\mathbf{z}_k$ and its 8 neighbors, and select the nearest open thread whose last endpoint within a spatial radius r_g and a short temporal gap T_{gap}. If found, we append $\mathbf{z}_k$ to that thread and update its last endpoint; otherwise, we treat the event as a stray contact (ignored) or the start of a new thread. On a matched up, we *close* the thread and emit $\mathbf{t}_i = (t_i^{\text{start}}, t_i^{\text{end}}, \mathbf{x}_i^{\text{start}}, \mathbf{x}_i^{\text{end}}, \text{id}_i)$, where $\mathbf{x}_i^{\text{end}}$ is the last endpoint and id_i denotes our *internal* thread identifier.

Although iPadOS exposes hardware touch identifiers, in practice they are often unreliable. We therefore treat all events as *anonymous* touches and perform aggregation via the spatial hash, which yields $O(1)$ expected-time association per event and produces coherent threads in real time.

Time clustering. Given the set of threads for a word, we sort threads by onset time and grow clusters greedily. Intuitively, a thread joins the current cluster if it starts within the cognitive window τ_c of the cluster's anchor. Large, near-simultaneous multi-finger contacts likely correspond to calibration-like gestures (e.g. place the fingers on the home-row before typing); clusters whose size exceeds a cap κ are discarded.

3.4 In-Cluster Selection Using Travel Score

Within each cluster, we select the representative touch of user's intent. Our heuristic is informed by motor control principles: we favor deliberate *reaches*, spatially significant departures over incidental, low-motion contacts.

We model the hand's recent state as a *hand-state cloud* $\mathcal{H}(t)$, a decaying memory of recent finger endpoints. Each point $\mathbf{p}_m$ in the cloud stores its location and time stamp. Points older than a horizon $T_{\max}$ are pruned. The influence of each prior point decays geometrically over time with factor ρ and step Δ, so more recent contacts carry higher weight.

For each candidate touch i in a cluster, with endpoint $\mathbf{x}_i$ and onset time t_i, we compute a *travel score* measuring its spatial departure from the cloud, down-weighted by the age of the nearest prior contact:

$$S_i = \min_{(\mathbf{p}_m, \tau_m) \in \mathcal{H}(t_i)} \frac{\|\mathbf{x}_i - \mathbf{p}_m\|_2}{\max\{w_m(t_i), \epsilon\}}, \tag{1}$$

where $w_m(t_i)$ is the decayed weight of point m at time t_i and ϵ is a small constant to avoid division by zero. Larger scores correspond to more distinct and more recent reaches. Conceptually, this score represents a time-discounted distance: large, fresh movements away from the recent pattern are scored highest. We then choose the intended activation as:

$$i^* = \arg\max_{i \in C} S_i. \tag{2}$$

To handle cases where travel scores are nearly tied (e.g., fast in-place taps), we select the most recent touch in the cluster. This rule leverages a characteristic pattern of ten-finger posture: when the user reaches toward a target key, nearby resting fingers are often slightly *dragged* in the same direction and tend to land earlier with shorter travel, producing premature incidental contacts, whereas the true intentional press occurs last. The same logic also governs stationary taps on home-row keys, which exhibit minimal travel but still appear as the final contact in their cluster.

Example. Figure 5 illustrates the process for the word *eligible*. Eight clusters are formed, and their selected representatives yield the intermediate sequence `ekigible`. The substitution (`k` for `l`) arises from a small lateral drift, which is one type of structured noise addressed by our downstream decoder component.

Implementation. The parameters (e.g. cognitive window τ_c, hand-state cloud's decay parameters ($\rho, \Delta, T_{\max}$) were optimized via a simple grid search. The resulting pre-decoding stage produces a letter sequence with structured, local noise typical of this interaction style: adjacent-key substitutions and merge/split artifacts from temporal clustering.

4 Synthetic Noise Modeling and LLM Fine-Tuning

With front-end producing letter sequences, we now address mapping the noisy letter sequence to a word. We first collect touch logs of hands-down typing behavior, then fit a simple noise model that generates synthetic word pairs, and finally fine-tune a LLM (FLAN–T5-small) for correction.

4.1 Data Collection for Behavior Modeling

We collected touch logs as participants typed prompted words on a tablet. The goal was to capture event-level traces (touch start/move/end timestamps and locations) under realistic ten-finger posture so we could characterize where near-synchronous and reach-induced incidental contacts occur.

Apparatus and Participants. We implemented a web-based typing interface on an 11-inch iPad Pro [2] using a standard QWERTY layout (Figure 6a). The logger records all touch lifecycle events with millisecond timestamps and normalized coordinates. Eight volunteers (7 male, 1 female), aged 21–26 years (M = 23.38, SD = 1.60), participated with informed consent. They were instructed to rest all ten fingers on the surface while typing (Figure 6b).

Procedure. Each session comprised blocks of *word* prompts sampled from google-10000-english [19]. For each prompt, participants tapped Start, typed on the keyboard, and tapped End. Avoiding mid-word corrections allowed the logger to capture incidental and near-synchronous contacts that the decoder would later resolve.

Obfuscated in-progress feedback. To prevent participants from reacting to noisy intermediate sequences, the interface presented only password-like progress dots (Figure 6a). This offered minimal awareness without revealing errors caused by overlapping touches or reach dynamics, preserves natural behavior [4]. The obfuscated feedback was used only during data collection; all later user studies used visible intermediate feedback (Section 6.2).

Dataset. We collected **1,752** prompted words. Word lengths were well distributed: ≤ 3 letters 8.7%, 4–6 letters 41.4%, 7–9 letters 37.2%, and ≥ 10 letters 12.8%. For every contact we logged the event type, timestamp, normalized coordinates, and session/word identifiers. This dataset covers both short bursts (where near-synchronous touches are more common) and longer words with richer hand travel, and serves as the basis for parameterizing our synthetic error generator.

4.2 Synthetic Training Data Generation

Real-world error data from hands-down typing are sparse and unevenly distributed. We therefore augment them with synthetic noisy–clean pairs [44, 53, 56] that mimic the physical structure of our interaction while spanning a controlled range of difficulties. Each pair consists of a target English word w (sampled from `google-10000-english` [19]) and a noisy letter sequence u that simulates what our pre-decoder would output for that word. We write $\mathrm{ED}(u, w)$ for their Levenshtein edit distance.

From human pairs to a simple error model. We start from human noisy–clean pairs collected in our study and align them with Levenshtein distance to extract atomic edits (substitutions, insertions, deletions, and local swaps). For each position we attach simple geometry features from the QWERTY layout (row, hand, finger assignment, local travel distance). Positions that require long reaches or unstable postures thus receive higher edit propensity.

Two dominant error channels. Empirically, hands-down typing errors are dominated by two mechanisms: near-key slips of the intended finger and co-activation of nearby resting fingers.

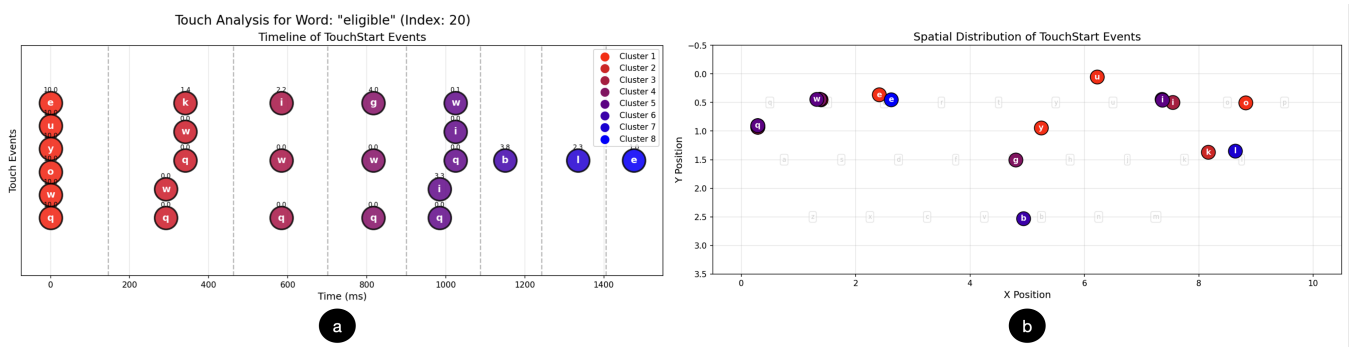

Figure 5: Cluster-level selection for a representative word instance (*eligible*). (a) timeline with cluster assignments; (b) spatial distribution of cluster start positions with nearest keys faintly shown. Our in-cluster rule yields `ekigible`.

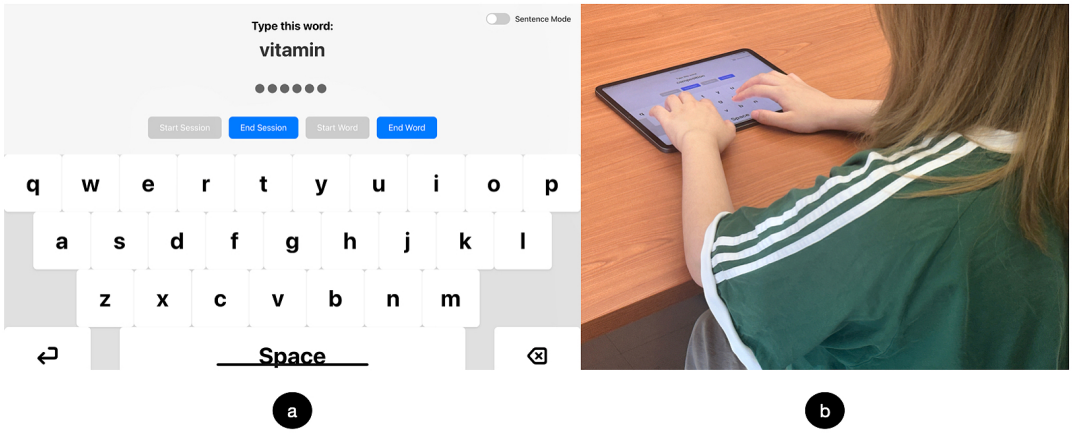

Figure 6: Word collection on an iPad Pro. (a) A target word appears at the top; instrumentation gates precise logging. During typing, the interface shows only a password-like progress bar (gray dots) instead of intermediate letters, encouraging natural input without mid-word corrections. (b) A participant types with all ten fingers resting on the surface.

Near-key slips (Near). When the user aims at a key for letter a, the active finger may land slightly off its center. For each letter a, we fit a small 2D Gaussian mixture model (GMM) over fingertip landing offsets $\boldsymbol{\delta}$, obtained from human touch data. Let $\mathbf{p}(a)$ denote the center of key a in keyboard coordinates, and let the perturbed contact location be

$$\tilde{\mathbf{p}} = \mathbf{p}(a) + \boldsymbol{\delta}. \tag{3}$$

We then map $\tilde{\mathbf{p}}$ back to letters with a soft nearest-key kernel

$$q(c \mid \tilde{\mathbf{p}}) \propto \exp\left(-\alpha \|\tilde{\mathbf{p}} - \mathbf{p}(c)\|^2\right), \tag{4}$$

where $\alpha > 0$ controls how sharply probability decays with spatial distance. Taking the most probable letter under q yields plausible **substitutions**, sampling from q yields occasional **extra taps** around the target key, and a special "no-key" outcome models **deletions** when the slip falls between keys. In practice, a small number of mixture components per letter suffices to capture typical offset patterns.

Co-activation of resting fingers (CoAct). When one finger moves, other fingers resting nearby may briefly touch the surface. To capture this, we examine clusters of concurrent touches in the human data and, for each target letter a and coarse geometry bucket (e.g., row×hand), count how often other touches within the same cluster fire as letter c. From these counts we fit a smoothed categorical distribution $P_{\text{CoAct}}(c \mid a, \text{bucket})$, which describes, for a given intended letter and posture, which other letters are likely to appear as co-activated taps. During synthesis, sampling from this distribution inserts extra letters (and can replace the aimed letter with small probability), and we allow occasional local swaps between adjacent positions, with probabilities that decrease as cross-row or cross-hand distance grows. This channel introduces realistic multi-letter bursts and small re-orderings that pure geometry cannot explain.

Synthesis. Given a target word y of length L, we generate a synthetic noisy sequence x in three steps.

(1) Choose how many edits to apply. We draw an intended edit count from length-aware prior and cap difficulty by

$$E_{\max}(L) = \begin{cases} 2, & L \le 6, \\ 3, & 7 \le L \le 9, \\ 4, & L \ge 10, \end{cases} \tag{5}$$

so that very short words are not overwhelmed by edits, while longer words can exhibit richer corruptions.

(2) Select positions and instantiate errors. Positions in y are sampled according to their learned edit propensities. For each selected position we choose an error channel (Near or CoAct) based on coarse geometry/context, assign an operation type (substitution, insertion, deletion, or swap), and instantiate its content using the corresponding mechanism: Near-key slips generate letters via the soft kernel in (4), and co-activation events draw from the empirical P_{CoAct}. Operations are applied in a fixed, simple order (swaps, then deletions, then insertions/substitutions from left to right) to obtain a candidate noisy string x. Because operations can cancel or interact, the realized edit distance $\text{ED}(x, y)$ may differ slightly from the initial intended count.

(3) Balance difficulty across edit distances. To avoid overproducing trivial near-copies and to expose the model to a range of difficulties, we target an approximately uniform distribution over edit distances up to $E_{\max}(L)$:

$$\widehat{\Pr}(\text{ED}(x, y) = e \mid |y| = L) \approx \frac{1}{E_{\max}(L) + 1}, \tag{6}$$

and accept only samples with $\text{ED}(x, y) \le E_{\max}(L)$. We resample until the empirical edit-distance histogram for each length regime is close to the target. Mild randomness in the GMM offsets, co-activation counts, and sampling steps injects diversity while keeping the overall statistics aligned with human data.

This process yields **146,331** synthetic pairs over **7,700** unique target words. By explicitly modeling both geometry-driven slips and concurrent touches, and by balancing edit distance across length regimes, the generator presents the LLM with a realistic and difficulty-controlled corruption manifold.

4.3 Model Fine-Tuning

We cast single-word correction as conditional sequence generation: given a corrupted character sequence x from our synthetic generator (Sec. 4.2), an fine-tuned LLM p_θ (FLAN–T5-small [10, 52], 77M parameters) predicts the intended word y. Although the noise-producing model operates at the character level, FLAN–T5 internally processes *subword tokens* produced by its SentencePiece tokenizer. We lowercase each noisy string and tokenize it into a short subword sequence (e.g., "eligible" → ["eli", "gi", "ble"]), and train the decoder to map these tokenized inputs back to a single word. Throughout this paper we use "characters" for elements of x and "tokens" for FLAN subwords.

Model choice. We adopt FLAN–T5-small because its encoder–decoder Seq2Seq design matches our "noisy sequence to one word" objective: the encoder produces a fixed representation of the corrupted input, and the decoder attends over it to generate the corrected word, reducing copy bias and instruction leakage. The small 77M-parameter model trains quickly on a single GPU and supports real-time inference for interactive decoding.

Training objective. Each training pair maps one noisy sequence to one lowercase English target word. We fine-tune all layers end-to-end under teacher forcing and minimize the negative log-likelihood

$$\mathcal{L}_{\text{NLL}}(\theta) = -\sum_{i=1}^{N} \log p_\theta(y_i \,|\, p(x_i)), \tag{7}$$

where $p(x_i)$ denotes the tokenized input. To gently bias the model toward orthographically close outputs, we add a small surrogate penalty on the edit distance between the teacher-forced prediction $\widehat{y}_i$ and the target:

$$\mathcal{L}_{\text{ED}}(\theta) = \lambda_{\text{ED}} \cdot \frac{1}{N} \sum_{i=1}^{N} \text{ED}(\widehat{y}_i, y_i), \tag{8}$$

and optimize $\mathcal{L} = \mathcal{L}_{\text{NLL}} + \mathcal{L}_{\text{ED}}$. In practice, most gains come from the synthetic curriculum so λ_{ED} is kept small. We prepend a short spell-correction prompt during training to preserve the instruction-following context of FLAN models, but performance is stable across different prefixes, indicating that the model learns primarily from the synthetic error distribution rather than from the exact wording of the instruction.

Optimization and implementation. We fine-tune using AdamW [34] with a linear learning-rate schedule and 5% warmup: learning rate $5{\times}10^{-4}$, weight decay 0.01, batch size 32 for training and 64 for validation. Training runs for 16 epochs with a 95/5 train/validation split. The pipeline is implemented in PyTorch and Hugging Face Transformers [41, 55] and trained on a single 48 GB NVIDIA RTX A6000 GPU.

Validation metrics. For validation we report *exact match* (EM),

$$\text{EM} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{1}\{\widehat{y}_i = y_i\}, \tag{9}$$

and additionally monitor the single-word validity rate to detect multi-token or non-alphabetic outputs. Because character-level typos can induce large changes in the model's subword sequence, the LLM effectively learns to repair tokenization-distorted inputs rather than apply simple character edits. This also clarifies our following comparison with the character-based Bayesian baselines.

5 Understanding Decoding Performance on Synthetic Data

We evaluate the LLM decoding component on a synthetic corpus of misspelled–gold pairs (Sec. 4.1). We disable cross-word context, so the comparison isolates phrase-level hints, matching prior statistical decoders [51, 54].

5.1 Two Baseline Decoders

We compare our LLM decoder against two baselines. The primary baseline is a touch-location informed Bayesian decoder, the standard text entry decoder that leverages full touch geometry [21, 62]. To match the same letter-only input consumed by LLM, we also include a letter-only n-gram decoder that operates solely on the nearest-key sequence.

Touch-location Informed Bayesian Decoder. We implemented this decoder following the commonly used Bayesian decoding principle (a.k.a, statistical decoding principle) [21, 62]. More specifically, given a sequence of touch points $S = \{s_1, s_2, s_3, ..., s_n\}$, this decoder is to find word w^* in lexicon L that satisfies:

$$w^* = \arg\max_{w \in L} P(w|S). \tag{10}$$

Using the Bayesian rule and assuming $P(S)$ is an invariant across words, this equation becomes:

$$w^* = \arg\max_{w \in L} \frac{P(S|w)P(w)}{P(S)} = \arg\max_{w \in L} P(S|w)P(w), \tag{11}$$

where $P(w)$ is from a language model (LM) and $P(S|w)$ is from a spatial model (SM). Similar to common practice (e.g., [62]), We adopted a bivariate Gaussian touch model as our spatial model, and used a character n-gram model ($n = 5$) with standard add-k smoothing as our language model. Here, the LM serves as a lightweight lexical prior over candidate words, while the dominant contribution comes from spatial likelihoods and alignment constraints; larger n yields diminishing returns under this edit-bounded setting.

Letter-only N-gram Decoder. When spatial coordinates S are not available, this decoder selects word w using the nearest-key sequence u from candidate set $C(u) = \{ w \in D : \text{ED}(u, w) \leq 4 \}$. $\text{ED}(\cdot, \cdot)$ denotes the editing distance. Following equations 10 and 11 and treating $P(u)$ as constant, the optimal word is:

$$w^* = \arg\max_{w \in C(u)} P(u \mid w)P(w) = \arg\max_{w \in C(u)} [\log P(u \mid w) + \log P(w)]. \tag{12}$$

In the absence of spatial likelihoods, $P(u \mid w)$ rely solely on how well the nearest-key sequence matches a candidate word. We approximate $\log P(u \mid w)$ using the negative editing distance between u and w:

$$w^* = \arg\max_{w \in C(u)} [\log P(w) - \alpha \cdot \text{ED}(u, w)], \tag{13}$$

where α is a control parameter that balances language prior and matching likelihood, which has been tuned.

5.2 Decoding Accuracy Comparison

Evaluation setup. We evaluate on the synthetic misspelling–gold pairs (Sec. 4.2). The touch-location informed Bayesian Decoder and the letter-only n-gram Decoder (Sec. 5.1) are chosen as baselines. For LLMs, candidate scoring is obtained via decoding with beam width k; beam search exposes the highest-probability k hypotheses. This multi-hypothesis retrieval requires no additional fine-tuning.

Overall accuracy. Figure 7 a summarizes aggregate performance across all decoders. The fine-tuned LLM decoder attains **84.8%** Top-1 and **97.2/98.7/99.4%** at Top-2/3/5. The letter-only n-gram baseline reaches **75.7%** Top-1 and **92.1/94.8/96.8%**. Adding touch-location improves the Bayesian decoder to **79.3%** Top-1 and **95.6/97.0/97.4%**. Yet the LLM, which receives only the letter sequence, still ranks candidates more accurately, showing that its advantage stems from stronger sequence-level priors rather than spatial cues.

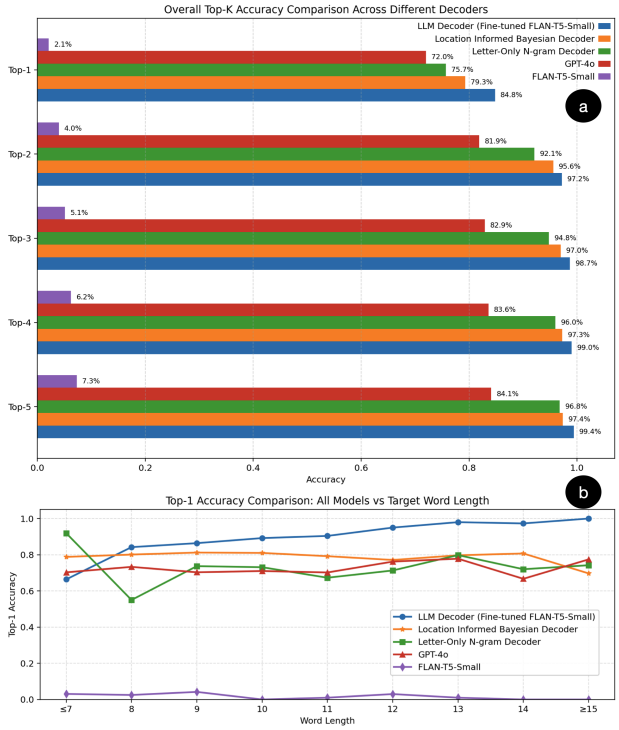

Figure 7: (a) Overall Top-k accuracy comparison across five decoders. The Spatial Bayesian variant narrows the gap between the standard statistical model and the fine-tuned LLM decoder by 3–4 points at Top-1. The LLM still achieves the best ranking and lowest residual error. (b) Top-1 accuracy versus word length shows that the spatial model helps most for short words, while the LLM retains superiority for medium and long words.

Effects of word length. Figure 8 details the dependence on target length; for reporting we aggregate to three buckets (≤ 7, $8-14$, ≥ 15; cf. Sec. 5). *Short ($\leq$ 7 chars).* Local regularities dominate and are well captured by a character n-gram prior; the n-gram baseline is strongest (Top-1 $\approx$ **91.9%**), while our LLM trails (Top-1 $\approx$ **66.4%**). The Bayesian decoder reaches **80.0%** Top-1. Short words are already well separated by symbolic patterns alone, so injecting noisy coordinate likelihoods offers limited benefit.

Medium (8 – 14 chars). Ambiguity grows as the candidate set expands with length L; windowed n-grams underfit longer-range orthography and morphology. Here the LLM reaches **91.0%** Top-1 vs. **69.9%** for the n-gram baseline, and the Top-1–Top-5 gap narrows sharply. Spatial cues offer marginal help but cannot compensate for the absence of global linguistic priors, leaving the Spatial Bayesian decoder far below the LLM at these lengths.

Long ($\geq$ 15 chars). With more global cues (e.g., *-tion*, *-ment*), the LLM's sequence prior excels, achieving **100%** Top-1 while the n-gram baseline yields **74.2%**. The Bayesian decoder drops to **68.6%** Top-1, reflecting the difficulty of aligning long, dispersed touches. Overall, spatial likelihoods help moderate-length words but fade

on very short or very long ones, while the LLM maintains robust performance across all lengths.

Why the fine-tuned LLM wins. **(i) Sequence-level priors.** Fine-tuning gives the model global orthographic and morphological priors. Statistical decoders rely on local symbolic or geometric cues, leading to *n*-gram underfitting. LLM errors, in contrast, tend to be morphology-preserving near misses. **(ii) Task-shaped output space.** Fine-tuning aligns the model's SentencePiece segmentation with our one-word objective, suppressing multi-token drift and producing stable outputs. **(iii) Learned conservatism.** Exposure to balanced edit distances (Sec. 4.2) encourages small, plausible corrections; even when wrong, outputs remain close (AvgED 0.318).

Specialization vs. scale. GPT-4o reaches **72.0%** Top-1 and **84.1%** Top-5, while an un-fine-tuned FLAN–T5-small performs poorly (Top-1: **2.1%**, Top-5: **7.3%**). GPT-4o's free-form generation tends to hedge with plausible but lexicon-mismatched guesses, and the small untuned model exhibits copy bias and instruction bleed-through. Fine-tuning FLAN–T5-small aligns its subword segmentation and teaches robust edit patterns, yielding a compact, task-specialized decoder that decisively outperforms both a much larger general model and its own untuned initialization.

Takeaways. Across all settings, the fine-tuned FLAN–T5-small achieves the best accuracy, especially on medium and long words, even though it sees only the letter sequence. Spatial likelihoods remain insufficient to close the gap, and a large general model (GPT-4o) underperforms a small task-specialized model on this task. These patterns motivate our choice to deploy the compact fine-tuned LLM decoder component.

6 User Study

We conducted a controlled within-subjects lab study to evaluate KeySense in realistic transcription tasks and to characterize how people adapt to hands-down, ten-finger surface typing. Our study examined: (1) typing speed, (2) errors and corrections, (3) user behavior and subjective rating, and (4) decoding latency.

6.1 Participants and Apparatus

Participants. 12 volunteers (10 male, 2 female; age 21–28, *Mean*=24.73, *SD*=2.45) participated, with prior experience in both physical and virtual keyboards (physical: *Mean*=14.25 years, *SD*=3.05; on-screen: *Mean*=6.75 years, *SD*=4.90). All provided informed consent.

Apparatus. The study used a web-based prototype on an 11-inch iPad Pro [2] via Safari. The interface presented a standard QWERTY layout and logged full touch lifecycles (down/move/up with timestamps and normalized coordinates). LLM decoding ran on an NVIDIA RTX A6000 GPU. No additional sensors were used.

6.2 Experiment Design and Procedure

Task and sessions. We used a within-subjects design with two conditions: (i) a conventional soft-keyboard baseline (hover typing; one contact per key) using a n-gram decoder, and (ii) a ten-finger condition decoded by KeySense. In each condition, participants performed a transcription task with phrases from the MacKenzie and Soukoreff's dataset [37]. Each part contained one warm-up block (10 phrases) followed by five formal sessions (10 phrases each). All participants typed the same phrases; only the order was randomized per participant and condition (Figure 9a).

Interface and feedback. The interface displayed the target phrase above the user's input (Figure 9a). We showed the underlined intermediate letter sequence produced by our pre-decoding pipeline so participants could monitor how their touches were interpreted. Decoding was triggered only when the user pressed Space: the underlined intermediate sequence was replaced by the Top-1 decoded word, and ranks 2–5 appeared as tap-selectable suggestions above the keyboard (generated via beam search with width 5). This setup receives decoding only after word completion, reducing inference cost and preserving continuous typing flow.

Editing controls. Enter submitted the current phrase and loaded the next one. Backspace acted in two modes: it cleared the current letter sequence if mid-word, or removed the previously committed word otherwise. Participants could rest non-reaching fingers on the surface when controlling as well: only the reaching finger moved to Space, Enter, Backspace, or suggestions.

Procedure. After a brief demonstration, participants first completed one of the two conditions (order counterbalanced), then the other. In the conventional part they hovered and tapped keys individually; in the ten-finger part they were encouraged to rest non-reaching fingers (Figure 9b). The editing controls were identical across conditions. Participants were instructed to transcribe as *quickly and accurately* as possible. Each condition started with a 10-phrase warm-up and then five formal sessions (50 phrases). After both conditions, participants completed a usability questionnaire.

6.3 Results

6.3.1 Typing speed. We report speed in Words-Per-Minute (WPM) following standard practice [5]:

$$\text{WPM} = \frac{|S_{transcribed}|}{5 \times T_{minutes}}, \tag{14}$$

where $|S_{transcribed}|$ is the length of the final transcription in characters and T is the total time in minutes. Timing begins at the start signal, so the character count is used directly.

Figure 10 shows session-wise WPM. With Ten-finger, participants exhibited a clear learning effect: a linear mixed-effects model (LMM) on sentence-level WPM estimates a mean gain of 5.12 WPM from Session 1 to Session 5 ($F(1, 223.06) = 57.90, p < 10^{-12}$). In contrast, baseline speeds peaked around Session 3 and then declined by 1.94 WPM by Session 5 ($F(1, 227) = 5.71, p = 0.018$), consistent with fatigue on the physically demanding tablet keyboard. By Session 5, Ten-finger was significantly faster than the baseline (28.3 vs. 26.2 WPM; $F(1, 118.71) = 8.58, p = 0.0041$), while overall mean speeds across all sessions remained similar (Ten-finger: 25.63 ± 4.23 WPM; Baseline: 24.73 ± 4.67 WPM) due to initial learning cost. Hands-down ten-finger typing yields a clear speed benefit once users adapt.

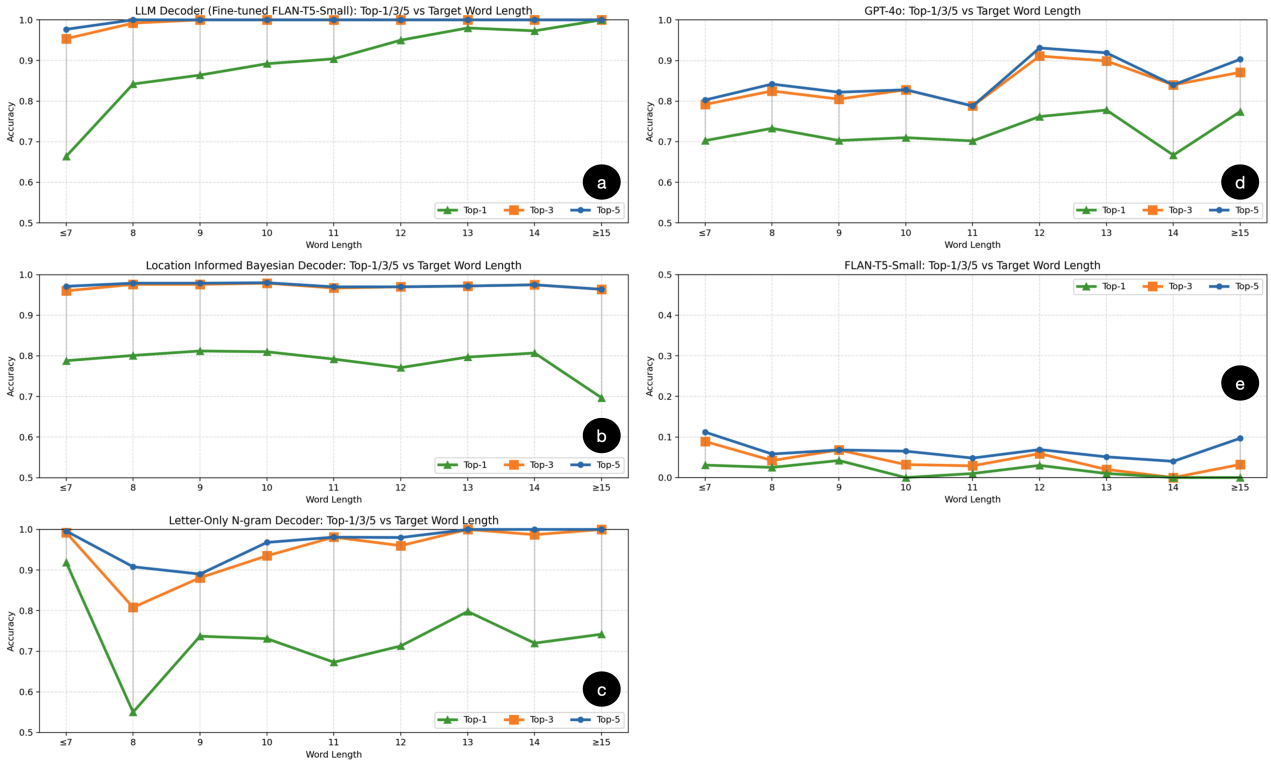

Figure 8: Top-1/3/5 accuracy versus target length for each decoder (a–e). (a) Fine-tuned FLAN–T5 (LLM decoder) quickly closes the Top-1 and Top-5 gap as length increases. (b) Bayesian decoder benefits short words via coordinate-based likelihoods but plateaus earlier on longer words. (c) N-gram decoder relies solely on symbolic letter sequences and shows larger length sensitivity. (d) GPT-4o remains steady but below Bayesian variants without lexicon constraints. (e) Un-fine-tuned FLAN–T5 fails to capture the task semantics.

6.3.2 Errors and corrections. We measure accuracy using Word Error Rate (WER), the word-level minimum string distance normalized by the target length:

$$\text{WER} = \frac{\text{MSD}(S_{transcribed}, S_{presented})}{N}, \tag{15}$$

where MSD counts word insertions, deletions, and substitutions, and N is the number of words in the presented sentence. Error rates were low for both conditions: ten-finger averages 1.20% WER (SD=1.75%) and the baseline 0.85% (SD=1.37%); a paired t-test shows the gap is not statistically significant ($t(11) = 1.38, p = 0.20$).

To probe correction effort, we use Corrected Error Rate (CER, proportion of words that were initially mistyped but later corrected with Backspace) and correction frequency (corrections per minute). Baseline typing yielded a CER of 3.23% (SD = 2.47) and KeySense 4.60% (SD = 3.06), again with no significant difference ($t(11) = 1.56, p = 0.148$). Correction frequency shows a similar pattern (Baseline: 1.77 ± 1.19 per minute; KeySense: 2.03 ± 0.95; $t(11) = 0.94$, $p = 0.369$). Ten-finger delivers higher speed without imposing a correction burden.

6.3.3 Ten-finger behavior and user experience. To understand how people actually type with hands-down contact, we combine two views: behavioral signals from touch logs and subjective ratings from post-study questionnaires.

We first quantify how many touches our pre-decoder treats as intentional. For each word, we compute an intent ratio

$$P = \frac{1}{2}\frac{\text{intended threads}}{\text{total threads}} + \frac{1}{2}\frac{\text{intended events}}{\text{total events}}, \tag{16}$$

which captures both the number of finger threads and the raw touch events interpreted as intentional.

Across all words, the mean P is **0.367** (SD = 0.296), confirming that incidental contacts are frequent but effectively filtered. Panel 11a shows large between-user differences, reflecting variations in resting stability and hand posture. Panel 11b shows that P increases with word length: shorter words often include a brief settling phase with extra contacts, while longer words allow the hand to stabilize as only the reaching finger moves. Time allocation further reveals a consistent hands-down rhythm (Panels 11c–d). Users spent on average **82.5%** of time within words (SD = 14.4%), with most participants exceeding 80%. Across sessions, the within-word time proportion is stable or slightly rising and becomes less variable, indicating that users quickly adopt a relaxed rest-and-reach pattern.

Subjective ratings closely mirror these behavioral patterns. We adopted a subset of NASA-TLX items [24] using a simplified 5-point

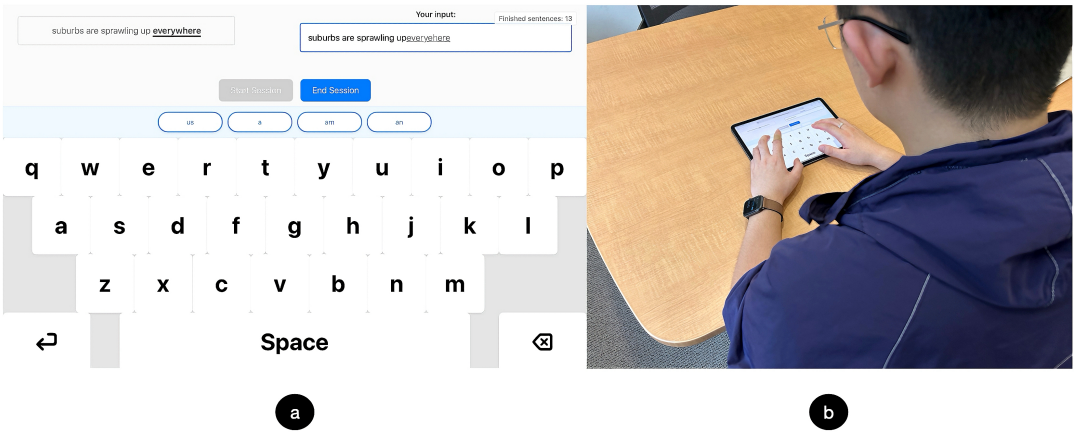

Figure 9: User study setup. (a) Transcription interface: the target phrase (left) and the user's input (right). The intermediate letter sequence is shown live underlined. In this example, the underlined word "everywhere" is still in the composing mode in which the decoding has not started yet, and the suggestion bubbles have not been updated and are deactivated. Once the user presses Space, the system sends the input sequence to the decoder, replaces the underlined segment with the finalized Top-1 word, refreshes and activates the suggestion bubbles with updated suggestions, and adds space to the output. (b) Participant posture with the tablet flat on a desk; most fingers rest on the surface while one finger reaches control keys (Space/Enter/Backspace/suggestions).

Question	Scale 1 – 5	Median Score
How do you like it?	very dislike – very like	5
I think I would use it frequently.	strongly disagree – strongly agree	4.5
It improves conventional typing.	strongly disagree – strongly agree	4
Similarity to physical typing	very different – very similar	4
Mental demand (Conventional vs. Ten Finger)	very low – very high	1 vs. 2.5
Physical demand (Conventional vs. Ten Finger)	very low – very high	4 vs. 1.5

Table 2: Summary of median subjective feedback scores.

version of the original scale (1=very low, 5=very high). Participants rated the ten-finger method highly (median liking = **5**; frequent-use intention = **4.5**). They agreed it improves conventional typing and feels similar to physical typing (both medians = **4**), suggesting successful transfer of muscle memory. For workload, ten-finger required slightly more mental attention at first (median 2.5 vs. 1.0), as participants broke the long-standing habit of avoiding incidental touches, but substantially reduced physical demand (1.5 vs. 4.0).

Open-ended comments reinforce this picture. **P3** notes, "The 10-fingers virtual keyboard performs much better for long words and feels smoother for short words since I don't need to press keys down." **P1** writes, "I'm used to keeping my fingers in the air when typing, so it's hard to switch to resting on the screen. But if virtual keyboards become common, this could really matter." **P9** summarizes, "Pros: easy to press a key, no need to push hard. Cons: no physical feedback—need a bit more attention to avoid mistakes." Collectively, these results show that users quickly settle into a stable rest-and-reach rhythm with clear ergonomic benefits.

6.3.4 Decoding latency. To check whether remote LLM decoding could be a hidden bottleneck, we also measured end-to-end latency. The latency was short and unlikely to affect the input performance. Decoding requests were sent from the iPad interface to a remote LLM server via an encrypted tunnel (Ngrok) [39]. LLM inference takes **53.70 ms** (SD=11.94ms). Outbound transmission averaged **31 ms** (95th percentile **40 ms**), and inbound transmission averaged **25 ms** (95th percentile **29 ms**). This yields an average round-trip communication delay of roughly **56 ms** per decoding request, well below typical thresholds for perceivable lag in visual feedback. The end-to-end **110 ms** delay is 55% lower than the existing cloud-based decoder [36]. No participants reported hesitation or delay in suggestion appearance or word confirmation, suggesting that latency did not measurably affect perceived responsiveness.

7 Discussion

7.1 Initial barriers and long-term potential

By the end of the study, participants achieved an average speed of 28.6 WPM on KeySense, which was similar to other ten-finger typing methods (e.g., 30 WPM on ResType [33] on the first day). Nonetheless, this performance was still slower than the typing speeds previously reported for the built-in iPad keyboard in prior literature [32]. A primary constraint was participants' unfamiliarity with the hands-down, ten-finger typing mode. While placing both

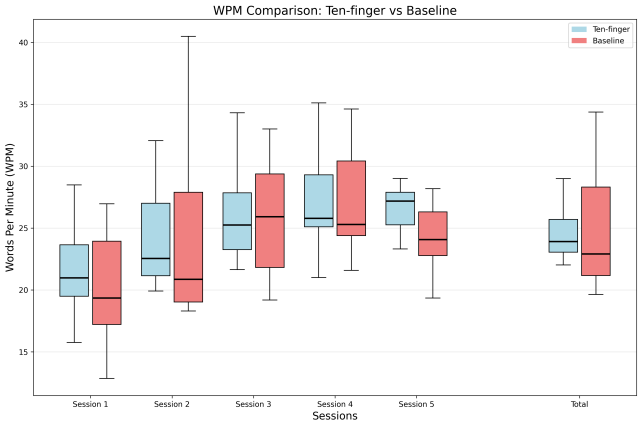

Figure 10: Session-wise WPM for *Ten-finger* (KeySense) and the *Baseline* soft keyboard. Boxes show medians and interquartile ranges; whiskers indicate the data range. Total pools all sessions. Ten-finger speeds increase steadily across sessions, whereas baseline speeds peak mid-study and decline by Session 5.

hands on the tablet surface increased physical comfort, this unfamiliar input style required users to unlearn the habit of hovering their hands above the keyboard and to develop confidence that resting both hands would not trigger false touches. This additional learning demand elevated cognitive load and initially hindered performance, as indicated by subjective ratings and user feedback. Furthermore, only 2 participants in our study identified themselves as typing experts, whereas in the prior work [32], all participants were expert typists with an average Qwerty speed of 82.6 WPM.

Despite the initial learning cost, the hands-down ten-finger typing is promising and may offer substantial long-term benefits. Typing speed improved steadily across the five sessions in our study, and the greatest improvement over the hover baseline occurred at the last session. This result also echoes earlier findings that ten-finger typing performance increases rapidly with practice [33]. As it reduces physical demands, alleviates hand fatigue, and more closely resembles the ergonomics of physical-keyboard typing, it has strong potential to become a widely adopted typing mode for tablet-scale touch surfaces.

To fully realize the potential of hands-down ten-finger typing, smoother motor transfer should be achieved. In our current design, users must frequently shift visual attention between the text area and the keyboard region, because virtual keys lack the haptic cues and home-row bumps of physical keyboards. Future work could ease this transition by adding visual anchoring cues that indicate home-row position within the main work area, which reduces gaze switching and makes users focus entirely on the text. Structured onboarding tutorials and practice curricula could also help users reframe the tablet surface as a "restable" keyboard.

7.2 LLM Architecture Affects Decoding Accuracy

Section 5 showed that a fine-tuned FLAN–T5-small can substantially outperform carefully engineered Bayesian decoders. A natural question is whether this advantage is specific to our chosen model family or would arise with any sufficiently large LLM. To probe this, we ran a small zero-shot comparison across several open LLM families on easy tasks.

A qualitative pattern emerged: encoder–decoder models from the T5 family behaved markedly better than decoder-only models when asked to emit a single corrected word. In our runs, Qwen-7B [3], Gemma-7B [48], and Llama-3.1-8B [38] (all decoder-only) produced low accuracy, while FLAN–T5 variants ranked higher despite far fewer parameters. This suggests that for noisy-to-clean, single-word normalization, architectural fit matters at least as much as raw scale.

Why architecture matters. **Cross-attention fits "map-and-say" tasks.** T5-style models first encode the noisy sequence and then let the decoder attend over a fixed representation, closely matching our "map-and-say" objective. This separation reduces copy bias and instruction leakage: the prompt and target are distinct streams, and the decoder can focus on reconstructing the word. Decoder-only models instead treat prompt and target as a single left-to-right sequence; when the desired output is short, their strong language priors over the prompt often dominate, yielding safe, prompt-like words instead of strict spellings.

Instruction tuning aligns behavior for concise outputs. FLAN variants are instruction-tuned and, even without task-specific fine-tuning, follow constraints like "output only one word" more reliably than general decoder-only baselines. However, without domain-specific fine-tuning they still lack a robust "noisy to clean spelling" mapping, which explains their modest zero-shot performance.

Scale helps less than match. Increasing parameter count in decoder-only models did not close the gap with T5 on this task: larger variants continued to exhibit prompt-following and over-generation behaviors. By contrast, scaling within the same family (e.g., from FLAN–T5-small to FLAN–T5-XL) improved zero-shot accuracy, suggesting that for spelling-like normalization the family match is more important than raw size across architectures.

These observations reinforce our conclusion: architectural alignment to the problem class matters. A compact fine-tuned FLAN–T5-small can excel while remaining small enough to support real-time use and flexible deployment.

7.3 Cloud vs. On-device Decoder Implementation

A practical keyboard decoder must keep latency low, protect privacy, and run across heterogeneous devices. In our setting, the latency constraint is particularly strict: ten-finger typing relies on pre-decoding after each touch and delivers real-time visual feedback, so any additional delay would be directly perceived as lag. At the same time, we aim to support commodity tablets today and in the longer term, other passive surfaces and form factors.

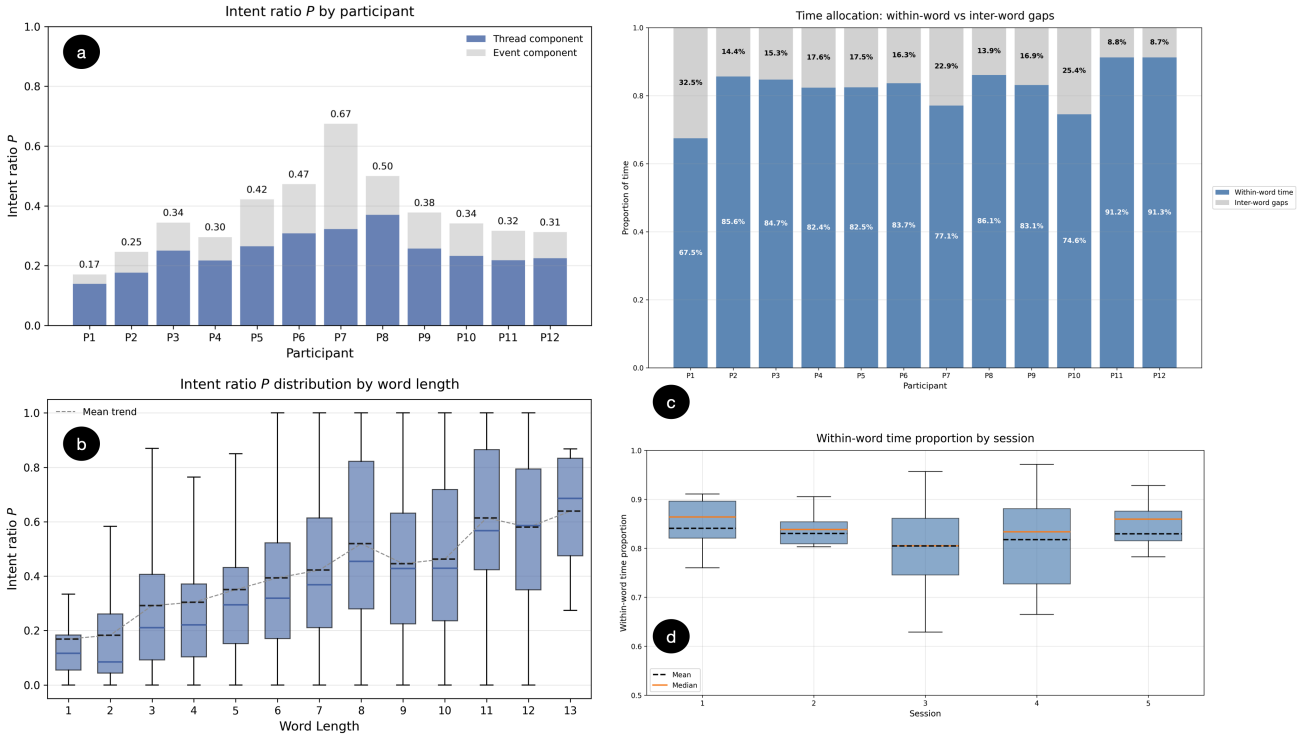

Figure 11: Ten-finger typing behavior. (a) Intent ratio P by participant, decomposed into thread and event components; labels show totals. (b) Distribution of P by word length: longer words show higher P as resting fingers stabilize. (c) Time allocation by participant: proportion of time spent within words vs. inter-word gaps. (d) Within-word time proportion by session: distributions remain stable.

On-device inference offers stable low latency, full offline privacy, and the opportunity for user-specific adaptation (e.g., private LoRA [26] adapters). In our case, a quantized FLAN–T5-small (77M parameters) can plausibly fit on modern tablets and phones (roughly 100 MB in int8) and matches our single-word inputs well. The drawbacks are energy cost on mobile CPUs/NPUs, tighter memory limits that preclude larger checkpoints, and fragmented hardware/software stacks that complicate maintenance. One plausible deployment profile is therefore a pure on-device mode for privacy-sensitive or offline settings, at the cost of using a smaller model and additional engineering for hardware-specific optimization.

Cloud inference, in contrast, trades network latency for accuracy headroom and operational agility. Hosting the decoder on a server allows us to leverage larger or rapidly updated checkpoints while maintaining uniform behavior across devices. In our deployment, the additional end-to-end delay introduced by cloud decoding averaged about 56 ms, well below perceptual thresholds for real-time visual feedback and consistent with prior on-cloud decoders [36, 61]. To mitigate privacy and bandwidth concerns, our pipeline keeps all pre-decoding on-device (threading, time clustering and bounded-edit candidate generation) and uploads only a compact package: the noisy letter sequence, not raw touch logs.

Given these trade-offs and our goal of robust accuracy under highly ambiguous inputs, our current implementation deploys the LLM decoder in the cloud while keeping sensing and pre-decoding local. This split—heavy, privacy-sensitive processing on-device; compact symbolic queries to a neural backend—offers a flexible pattern for LLM-powered text entry: the user interface and touch-processing pipeline remain stable, while the decoder itself can evolve over time or migrate between cloud and on-device execution as application constraints demand.

7.4 Limitations and Future Work

Our study deliberately targets a slice of the design space: word-level correction on a fixed QWERTY layout, in English. This yields clear attributions for our decoder comparison but also omits several factors that matter in everyday typing: multi-word context, out-of-vocabulary content (e.g., names, slang, mixed-language phrases), more extreme or bursty errors, and long-term use in the wild. We highlight four concrete next steps:

From word-level to context-aware decoding. All decoders in this work operate at the word level and ignore sentence context, even though longer-range semantics could help disambiguate difficult cases and stabilize corrections over time. A natural extension is to lift the LLM decoder to sentence- or phrase-level operation. This would bring hands-down typing closer to continuous text entry, and allow future work to study how context-sensitive correction interacts with user trust, error visibility, and learning.

Handling OOV words. Similar to other intelligent decoders, an LLM-based decoder also struggles with out-of-vocabulary (OOV) words, especially when they appear for the first time. One potential solution is to preserve the literal string, the exact characters typed by the user, on the suggestion strip as a viable fallback before applying auto-correction. This allows users to select it in case a false correction is made. Moving forward, once the decoder correctly identifies an OOV word, it should incorporate that word into its dictionary to support smoother and more accurate future input.

Hybrid decoding and longer-term evaluation. Our results show a clear pattern: the statistical decoders perform strongly on very short words, while the LLM dominates on medium and long ones. A promising avenue is to explore hybrid decoders that route inputs based on length or uncertainty: using the statistical decoder for simple short words and the LLM for ambiguous or longer ones. Such hybrids could improve overall accuracy and reduce computation for easy cases. Finally, our five-session user study captures only the early phase of skill acquisition; a longitudinal, in-the-wild deployment would allow us to test these hybrid strategies under realistic conditions and to measure how comfort, speed, and correction patterns evolve over days or weeks.

Cross-user variability and fixed timing. As users' typing behavior varies, personalization could further improve the performance. For example, our 100 ms cognitive window was chosen from prior data and works well on average, but individual traces show substantial variation. A static τ_c therefore limits how well the system can match each user's rhythm, hinting at further speed and comfort headroom with online personalization of temporal and spatial priors.

Together, these directions move from a controlled prototype toward a context-aware, coverage-robust, and field-tested system that can support high-fluency ten-finger typing on everyday surfaces.

8 Conclusion

We set out to enable comfortable, ten-finger typing on commodity, pressure-insensitive touchscreens without adding new hardware. KeySense addresses this challenge by combining a lightweight predecoder that interprets hands-down multi-touch input with a task-specialized FLAN–T5-small decoder trained on human-informed synthetic errors, yielding robust word-level corrections from noisy letter sequences. Our component evaluations and user study suggest that this combination is already practically viable: the LLM decoder substantially outperforms statistical baselines and participants quickly attain higher speeds with lower physical demand than conventional hover typing. More broadly, KeySense treats resting-finger contacts as structured signal rather than nuisance, pointing toward ubiquitous, surface-agnostic typing—where any flat surface can serve as a high-performance keyboard, combining ergonomic comfort with competitive speed.

Acknowledgments

We thank the anonymous reviewers for their insightful feedback. This work was supported by the Google Inclusion Research Award, NSF Award 2153056, and NIH Award R01EY035688.

References

[1] Philip A Allen, Albert F Smith, Heli Vires-Collins, and Susan Sperry. 1998. The psychological refractory period: evidence for age differences in attentional time-sharing. *Psychology and Aging* 13, 2 (1998), 218.

[2] Apple Inc. 2021. iPad Pro 11-inch (3rd Generation). https://www.apple.com/ipad-pro/.

[3] Jinze Bai, Shizhe Bai, Yunfei Li, Yuxuan Zhou, Kun Song, et al. 2023. Qwen Technical Report. *arXiv preprint arXiv:2309.16609* (2023).

[4] Xiaojun Bi, Shiri Azenkot, Kurt Partridge, and Shumin Zhai. 2013. Octopus: evaluating touchscreen keyboard correction and recognition algorithms via. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 543–552.

[5] Xiaojun Bi, Ciprian Chelba, Tom Ouyang, Kurt Partridge, and Shumin Zhai. 2012. Bimanual gesture keyboard. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. 137–146.

[6] Nicholas Ryan Bonaker, Emli-Mari Nel, Keith Vertanen, and Tamara Broderick. 2022. A performance evaluation of nomon: A flexible interface for noisy single-switch users. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–17.

[7] William Buxton et al. 1990. A three-state model of graphical input. In *Human-computer interaction-INTERACT*, Vol. 90. Citeseer, 449–456.

[8] Shanqing Cai, Subhashini Venugopalan, Katie Seaver, Xiang Xiao, Katrin Tomanek, Sri Jalasutram, Meredith Ringel Morris, Shaun Kane, Ajit Narayanan, Robert L MacDonald, et al. 2023. Using large language models to accelerate communication for users with severe motor impairments. *arXiv preprint arXiv:2312.01532* (2023).

[9] Stuartk Card, THOMASP MORAN, and Allen Newell. 1986. The model human processor- An engineering model of human performance. *Handbook of perception and human performance*. 2, 45–1 (1986), 1–35.

[10] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, et al. 2022. Scaling Instruction-Finetuned Language Models. *arXiv preprint arXiv:2210.11416* (2022).

[11] Wenzhe Cui, Rui Liu, Zhi Li, Yifan Wang, Andrew Wang, Xia Zhao, Sina Rashidian, Furqan Baig, IV Ramakrishnan, Fusheng Wang, et al. 2023. Glancewriter: Writing text by glancing over letters with gaze. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–13.

[12] Wenzhe Cui, Suwen Zhu, Mingrui Ray Zhang, H Andrew Schwartz, Jacob O Wobbrock, and Xiaojun Bi. 2020. Justcorrect: Intelligent post hoc text correction techniques on smartphones. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 487–499.

[13] Abigail C Evans, Katie Davis, James Fogarty, and Jacob O Wobbrock. 2017. Group Touch: Distinguishing tabletop users in group settings via statistical modeling of touch pairs. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 35–47.

[14] Shariff AM Faleel, YiI-Keyboardshuo Liu, Roya A Cody, Bradley Rey, Linghao Du, Jiangyue Yu, Da-Yuan Huang, Pourang Irani, and Wei Li. 2023. T-Force: Exploring the Use of Typing Force for Three State Virtual Keyboards. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–15.

[15] Anna Maria Feit, Daryl Weir, and Antti Oulasvirta. 2016. How we type: Movement strategies and performance in everyday typing. In *Proceedings of the 2016 chi conference on human factors in computing systems*. 4262–4273.

[16] Leah Findlater, Ben Lee, and Jacob Wobbrock. 2012. Beyond QWERTY: augmenting touch screen keyboards with multi-touch gestures for non-alphanumeric input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2679–2682.

[17] Leah Findlater and Jacob Wobbrock. 2012. Personalized input: improving ten-finger touchscreen typing through automatic adaptation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 815–824.

[18] 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*. 2453–2462.

[19] first20hours. 2015. google-10000-english: List of the 10,000 most common English words. https://github.com/first20hours/google-10000-english.

[20] Shaona Ghosh and Per Ola Kristensson. 2017. Neural networks for text correction and completion in keyboard decoding. *arXiv preprint arXiv:1709.06429* (2017).

[21] 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*. 194–195.

[22] Mitchell Gordon, Tom Ouyang, and Shumin Zhai. 2016. WatchWriter: Tap and gesture typing on a smartwatch miniature keyboard with statistical decoding. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 3817–3821.

[23] Yizheng Gu, Chun Yu, Xuanzhong Chen, Zhuojun Li, and Yuanchun Shi. 2021. TypeBoard: Identifying Unintentional Touch on Pressure-Sensitive Touchscreen Keyboards. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 568–581.

[24] Sandra G Hart. 2006. NASA-task load index (NASA-TLX); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 50. Sage publications Sage CA: Los Angeles, CA, 904–908.

[25] 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.

[26] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR* 1, 2 (2022), 3.

[27] Robert JK Jacob. 1995. Eye tracking in advanced interface design. *Virtual environments and advanced interface design* 258, 288 (1995), 2.

[28] Jon L. Jacobi. 2019. Sensel Morph: A Brilliant, Ever so Near Miss. https://jonljacobi.com/sensel/

[29] Sunjun Kim and Geehyuk Lee. 2016. Tapboard 2: Simple and effective touchpad-like interaction on a multi-touch surface keyboard. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 5163–5168.

[30] Sunjun Kim, Jeongmin Son, Geehyuk Lee, Hwan Kim, and Woohun Lee. 2013. TapBoard: making a touch screen keyboard more touchable. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 553–562.

[31] Ue-Hwan Kim, Sahng-Min Yoo, and Jong-Hwan Kim. 2019. I-keyboard: Fully imaginary keyboard on touch devices empowered by deep neural decoder. *IEEE Transactions on Cybernetics* 51, 9 (2019), 4528–4539.

[32] Frank Chun Yat Li, Richard T Guy, Koji Yatani, and Khai N Truong. 2011. The 1line keyboard: a QWERTY layout in a single line. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 461–470.

[33] Zhuojun Li, Chun Yu, Yizheng Gu, and Yuanchun Shi. 2023. ResType: Invisible and Adaptive Tablet Keyboard Leveraging Resting Fingers. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–14.

[34] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[35] Yan Ma, Tony Li, Zhi Li, and Xiaojun Bi. 2025. LLM-powered Text Entry in Virtual Reality. In *2025 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. IEEE, 1628–1629.

[36] Yan Ma, Dan Zhang, IV Ramakrishnan, and Xiaojun Bi. 2025. LLM Powered Text Entry Decoding and Flexible Typing on Smartphones. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–14.

[37] 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.

[38] AI @ Meta. 2024. The Llama 3 Herd of Models. https://ai.meta.com/research/publications/the-llama-3-herd-of-models/.

[39] ngrok Inc. 2014. ngrok: Secure tunnels to localhost for testing and debugging. https://ngrok.com/. Accessed: 2025-09-08.

[40] Jakob Nielsen. 1993. Response times: the three important limits. *Usability Engineering* (1993).

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[42] 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.

[43] Helena Roeber, John Bacus, and Carlo Tomasi. 2003. 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*. 712–713.

[44] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709* (2015).

[45] Weinan Shi, Chun Yu, Xin Yi, Zhen Li, and Yuanchun Shi. 2018. TOAST: Ten-finger eyes-free typing on touchable surfaces. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 1 (2018), 1–23.

[46] 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.

[47] Paul Streli, Mark Richardson, Fadi Botros, Shugao Ma, Robert Wang, and Christian Holz. 2024. TouchInsight: Uncertainty-aware Rapid Touch and Text Input for Mixed Reality from Egocentric Vision. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–16.

[48] Gemma Team and Google DeepMind. 2024. Gemma: Open Models Based on Gemini Research and Technology. https://storage.googleapis.com/deepmind-media/gemma/gemma-report.pdf.

[49] 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 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[50] 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*. 1–14.

[51] 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*. 659–668.

[52] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652* (2021).

[53] Jason Wei and Kai Zou. 2019. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196* (2019).

[54] Daryl Weir, Henning Pohl, Simon Rogers, Keith Vertanen, and Per Ola Kristensson. 2014. Uncertain text entry on mobile devices. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 2307–2316.

[55] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 38–45.

[56] Ziang Xie, Sida I Wang, Jiwei Li, Daniel Lévy, Aiming Nie, Dan Jurafsky, and Andrew Y Ng. 2017. Data noising as smoothing in neural network language models. *arXiv preprint arXiv:1703.02573* (2017).

[57] Zheer Xu, Shanqing Cai, Mukund Varma T, Subhashini Venugopalan, and Shumin Zhai. 2024. SkipWriter: LLM-Powered Abbreviated Writing on Tablets. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–13.

[58] 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 & Technology*. 539–548.

[59] Mingrui Ray Zhang, He Wen, and Jacob O Wobbrock. 2019. Type, then correct: Intelligent text correction techniques for mobile text entry using neural networks. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 843–855.

[60] 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.

[61] Maozheng Zhao, Michael Xuelin Huang, Nathan G Huang, Shanqing Cai, Henry Huang, Michael G Huang, Shumin Zhai, IV Ramakrishnan, and Xiaojun Bi. 2025. Tap&Say: Touch Location-Informed Large Language Model for Multimodal Text Correction on Smartphones. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–17.

[62] Suwen Zhu, Tianyao Luo, Xiaojun Bi, and Shumin Zhai. 2018. Typing on an invisible keyboard. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.