Improved inference and autotyping in EEG-based BCI typing systems

Andrew Fowler[†], Brian Roark[†], Umut Orhan[◦], Deniz Erdogmus[◦], Melanie Fried-Oken[†]
[†]Oregon Health & Science University [◦]Northeastern University

ABSTRACT

The RSVP Keyboard™ is a brain-computer interface (BCI)-based typing system for people with severe physical disabilities, specifically those with locked-in syndrome (LIS). It uses signals from an electroencephalogram (EEG) combined with information from an n-gram language model to select letters to be typed. One characteristic of the system as currently configured is that it does not keep track of past EEG observations, i.e., observations of user intent made while the user was in a different part of a typed message. We present a principled approach for taking *all* past observations into account, and show that this method results in a 20% increase in simulated typing speed under a variety of conditions on realistic stimuli. We also show that this method allows for a principled and improved estimate of the probability of the backspace symbol, by which mis-typed symbols are corrected. Finally, we demonstrate the utility of automatically typing likely letters in certain contexts, a technique that achieves increased typing speed under our new method, though not under the baseline approach.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: Input devices and strategies, natural language; I.2.7 [**Natural Language Processing**]: Language models; K.4.2 [**Social Issues**]: Assistive Technologies for Persons with Disabilities

General Terms

algorithms, experimentation

Keywords

brain computer interfaces, language models, text entry

1. INTRODUCTION

Communicating by text is a ubiquitous part of modern life, from composing text documents or emails via a QWERTY keyboard to SMS messages via mobile devices. Many people with physical disabilities also rely on text entry for face-to-face interaction, producing text that is then synthesized into speech by their speech generating devices. Often the disabilities preclude the use of standard text entry interfaces due to the physical requirements of directly selecting symbols from a keyboard. Without sufficient hand or pointer control to select individual keys in a keyboard or make other selecting gestures, such individuals often rely on keyboard emulation methods, whereby a single binary switch provides access to desired symbols via an interface that scans through alternatives. For example, letters can be arranged on a matrix or grid, and rows or columns highlighted until the intended symbol is identified by the selected row and column (known as row/column scanning). Such interfaces have been around since the 1970s, e.g., the Tufts Interactive Communicator (TIC) [2, 3].

The binary switch used to indicate selection of a scanned item can be placed to best leverage the capabilities of the individual, i.e., it can be triggered based on an eyeblink, movement of an eyebrow or jaw, or a breath, among other options. For those with the most severe movement impairments, e.g., those with locked-in syndrome (LIS) resulting from ALS or a brain stem stroke, no volitional switch may be available. In these cases, non-invasive brain-computer interfaces have been developed to allow for text entry [8, 14, 11]. For several such systems, electroencephalography (EEG) is used to discover user intent by detecting responses to visual stimuli, such as the row/column scanning described above. Detection of a positive response corresponds to a binary switch activation, allowing text entry much as with more conventional switches. However, in this case the activation does not require volitional effort; rather it is the result of involuntary responses to stimuli. This makes these technologies ideal for individuals who cannot effectively produce reliable volitional selection movements.

The main challenge for these BCI typing systems is that event detection from EEG signals is a very difficult classification task, even with state-of-the-art signal processing techniques. Consequently, typing with such a system can be both error-prone and slow. Current systems have methods for addressing the inherently low signal strength of the EEG. One is to make repeated observations of the brain signal, relying on certain independence assumptions to increase confidence [8]. Another is to use a language model prior to improve the posterior probability distribution over letters (or words) given the context in the typed string [11]. In this paper we examine inference methods making use of both techniques.

The RSVP Keyboard™ BCI typing system [11] combines a language model prior distribution over symbols with repeated EEG observations to facilitate accurate typing. Presentation follows a rapid serial visual presentation (RSVP) paradigm, which involves linear visual scanning of one symbol at a time. Figure 1 shows the interface of the system, with a single symbol in the center of the dis-

Figure 1: The RSVP Keyboard™ interface, with the candidate symbol in the center of the screen, and typed message to its left.

play rather than a grid. To perform text entry, one attends to a rapid sequence of individual symbols at the center of the screen, looking for the target symbol. When the user recognizes the target symbol an event related potential (ERP) is evoked – the well-known P300, occurring 300ms after stimulus onset – which can be detected from the EEG signals. This detection is made via classifier fusion with a language model: when the posterior probability of an event associated with a symbol is greater than a threshold, that symbol is typed; otherwise another sequence of EEG observations is obtained.

Currently, the RSVP Keyboard assigns posterior probabilities to the symbols in each context independently of observations taken earlier in the typing session. In this paper, we propose an approach that maintains the posterior probabilities over all observed contexts, continuing to update them and take them into account during inference, even if they are no longer consistent with what has actually been typed. High likelihood for the backspace symbol (used to delete previously typed symbols) in the current context is evidence for alternate continuations of earlier contexts, and we demonstrate how to efficiently maintain and allocate that probability mass. For example, if the typed string is `core` and there is high likelihood of backspace, then the letters other than `e` should have higher probability when returning to the context `cor`.

We efficiently maintain posterior probabilities of the symbols for all contexts up to the length of the longest typed context in the sentence, and update and use these posterior probabilities during inference. Among other things, this provides a principled estimate of the backspace probability. Further, we demonstrate – via simulation – additional typing speed improvements using *autotyping*, whereby symbols with probability above threshold are typed without requiring EEG observations in that context. Autotyping with the baseline system is shown to provide no gain in typing speed.

2. BACKGROUND

2.1 Language models in AAC Typing Systems

Communication with AAC devices can be significantly slower than written or spoken language. As a result, both users and designers of assistive technologies often place great value on improving communication speed and accuracy. Predictive language models are a natural fit for this problem, to the extent that they can retrieve likely continuations of a message in a way that requires fewer gestures (keystrokes) to select than if they were produced letter-by-letter. Word completion is a component of almost every commercial AAC typing system in use today, and this has been the case since the early days of such systems [6].

The idea behind word completion is to use an n-gram language model trained on a large corpus to help predict the next word (or current word in progress) based on the last few words of context in the typed string. Very often a specific area of the screen is reserved specifically for a set of probable next words, which can be selected directly by the user in order to type one of them without taking time on the intervening letters, or selected via scanning that region in a keyboard emulation system. Higginbotham [5] showed that such word completion lists can provide about 40-50% keystroke savings[1] over systems without word completion. One important caveat of word completion is that more cognitive effort is required on the part of the user to keep track of dynamically changing word predictions, a fact which [7] has shown to actually decrease user performance somewhat in such systems. Nevertheless, it is a ubiquitous feature in AAC typing systems and found by many to be of high utility.

Language modeling can also be leveraged to speed typing in binary switch scanning systems, of the sort described in the introduction. For row/column scanning methods, likely symbols can be moved to the upper left corner of the grid so that they require less scanning to select. The TIC system cited above [2, 3] organized the grid based on overall symbol frequency so that the grid remained static. Lesher et al. [9] described a technique for dynamically rearranging the grid based on contextual language models, but showed that the large cognitive load involved in keeping track of an ever-changing letter grid made this method dispreferred. *Huffman scanning* [13, 12] is a method for scanning according to Huffman codes derived from contextual language models without dynamically reordering the spelling matrix/grid, thus requiring fewer keystrokes to indicate likely letters.

In the systems described in this paper, the language model is used in two ways. First, it serves as a prior model for intent detection classification in the RSVP Keyboard, as described in Section 2.2. This can speed typing of likely symbols by reducing the number of symbol sequence presentations required for the posterior probability to rise above the threshold. Second, if we reduce the minimum number of presentations from 1 to 0, the system can type a symbol based solely on the language model probability, provided it is above the threshold, resulting in autocompletion. This is a similar function to word completion mentioned earlier in this section, and can speed typing by reducing the number of symbol positions requiring symbol scanning. Details on the linear scanning system used by the baseline RSVP keyboard are presented in the next section.

This approach is different from, and complementary to, the language modeling methods presented in [10], which also applied improved inference methods to the RSVP Keyboard. Briefly, that work focused on improved priors derived from the language model, while this work is focused on maintaining posterior probabilities for many possible alternative strings in case they are subsequently revisited.

2.2 The RSVP Keyboard

The RSVP Keyboard [11] is a binary typing system consisting of a cap of 10-20 electrodes for measuring brain signals on the scalp, and a visual display presenting candidate symbols, as shown in Figure 1. The symbol set consists of 26 letters in the alphabet plus a spacebar symbol ("_") and a backspace symbol ("←") that permits deletion of the previously typed symbol. All 28 symbols are pre-

[1]Keystroke savings is the percentage savings in keystrokes versus typing letter-by-letter.

sented in sequence in random order at 2.5 Hz, i.e., 400 ms on screen display for each symbol.[2] After the full sequence of symbols has been presented, a classifier is used to detect the presence of a P300 event, under the assumption that one such event has occurred (corresponding to the target symbol). A posterior probability is calculated, which combines, via naïve Bayesian fusion, the probability from each sequence presentation with the prior probability from the language model (see details below). If the posterior probability of any symbol is above a parameterized threshold value, then that symbol is typed.[3] Otherwise, another sequence of the symbol set is presented in random order.[4] Once a symbol is typed (or deleted), the display and internal context state is updated and scanning continues at the next position.

Beyond EEG signal capture and stimulus display, the RSVP Keyboard system consists of three key parts: the ERP classifier; the language model; and the fusion of the evidence for decision on the target symbol. In addition, there are several system meta-parameters that control functionality. We present each of these in turn.

2.2.1 ERP Classification

The input to the ERP classifier is a set of features extracted from the EEG signals. For each stimulus in the sequence (28 symbols), the signals are extracted from the onset of the symbol presentation until 500ms after presentation, since this window is expected to include the relevant signal components associated with the P300 ERP. Bandpass filters and linear dimensionality reduction methods (PCA) are applied to derive a feature vector to be given to the binary classifier (target/non-target).

Regularized Discriminant Analysis (RDA) [4] is used to generate a likelihood for both classes (target/non-target) for each of the 28 stimuli in the sequence. RDA is a modified version of a quadratic discriminant analysis model which is made less susceptible to singularities in the covariance matrices via regularization and shrinkage. The class feature vectors are assumed to conform to multivariate normal distributions. This model can be used to generate classification scores (via log-likelihood ratios) or a probability distribution over stimuli. For more details, see [11].

For this paper, the ERP classifier remains unchanged from the current RSVP Keyboard system. For simplicity of later notation, let $\mathit{ERP}_{ij}(x)$ be the likelihood from sequence presentation j that x is the target symbol at position i in the typed string, as determined by the ERP classifier. Then, after k sequences have been presented at position i, $\mathit{ERP}_i^k(x)$ is the likelihood of symbol x at position i,

$$\mathit{ERP}_i^k(x) \quad = \quad \prod_{j=1}^{k} \mathit{ERP}_{ij}(x) \tag{1}$$

2.2.2 Language Modeling

The language model used in the RSVP Keyboard is a 6-gram character-based language model, i.e., the probability of symbol x at position i (denoted x_i) is conditioned on the previous 5 symbols. Let $h_i^{i-k} = x_{i-k} \ldots x_{i-1}$ be the context (or history) of length k of the typed string at position i. Then

$$\begin{aligned} \mathrm{P}(x_i \mid h_i^{i-k}) \quad &= \quad \lambda(h_i^{i-k})\, \widehat{\mathrm{P}}(x_i \mid h_i^{i-k}) \\ &+ \quad (1-\lambda(h_i^{i-k}))\, \mathrm{P}(x_i \mid h_i^{i-k+1}) \end{aligned} \tag{2}$$

where $\widehat{\mathrm{P}}(x_i \mid h_i^{i-k})$ is the raw relative frequency as found in the training corpus, and $\lambda(h_i^{i-k})$ is a mixing parameter between 0 and 1, estimated using the version of Witten-Bell smoothing [16] from [1], as detailed in [12].

For the current RSVP Keyboard system, and unchanged in this paper, the language model was trained on a subset of the New York Times portion of the English Gigaword corpus (LDC2007T07). Also included in the corpus were 112 thousand words from the CMU Pronouncing Dictionary,[5] and several repeated lines of text generated by an actual user of AAC technology. The training data was text-normalized to de-case letters, and several other procedures were performed in order to reduce the possibility of duplicate sentences or articles appearing in the corpus, as presented in [12].

Each symbol is scored by the language model using a likelihood ratio of the symbol being the target versus the symbol not being the target, and then normalized over the symbol set. Since this language model prior is not being changed for this paper, for notational simplicity let $\mathit{LM}_i(x)$ be the prior that symbol x is the target symbol at position i, given what has already been typed.

2.2.3 Language Model and EEG Fusion

At position i in the typed string, a posterior probability P of each symbol x in the symbol set can be calculated via Bayesian fusion of the language model and ERP classifier scores, as follows

$$P_i^j(x) \quad = \quad \mathit{ERP}_i^j(x) \cdot \mathit{LM}_i(x) \tag{3}$$

Note that $P_i^0(x) = \mathit{LM}_i(x)$, and that

$$P_i^j(x) \quad = \quad P_i^{j-1} \cdot \mathit{ERP}_{ij}(x) \tag{4}$$

Once the normalized posterior $P_i^k(x)$ is greater than the decision threshold, the symbol x is typed.

2.2.4 System meta-parameters

The RSVP Keyboard relies upon several meta-parameters for operation. In this section, we introduce each of these parameters, and the settings in the current version of the system.

- **Decision threshold**: If the posterior probability of a symbol is greater than the decision threshold θ, then that symbol is typed. In the current system, that threshold is set at $\theta = 0.9$.
- **Minimum Sequences**: The minimum number of sequences to show before typing. In the current system, this parameter is set to 1, meaning that at least one sequence must be presented to the user prior to typing a symbol. If this parameter is set to 0, it means that if $P_i^0(x) > \theta$ then the symbol will be typed based solely on evidence from the language model (autotyping).
- **Maximum Sequences**: The maximum number of sequences to show before typing. If this number is reached and no symbol exceeds the decision threshold θ, the most probable symbol is typed. The current system sets this to 3.

[2] Presenting subsets of the full set of symbols and at a faster rate of presentation (5 Hz) are methods currently under exploration, beyond the scope of this paper.

[3] The backspace ("←") symbol is not typed, rather it causes the last typed letter to be deleted.

[4] Note that the randomness of the sequence is important, since the strength of the P300 response is partially based on the unpredictability/surprise on the part of the user when the target stimulus appears.

[5] www.speech.cs.cmu.edu/cgi-bin/cmudict

- **Backspace Probability**: The current system uses a fixed probability $\beta = 0.05$ of the backspace symbol in every context.
- **LM Damping**: This is a damping factor used on the language model probability distribution. A value of λ means we multiply log probabilities from the language model by λ. In the current system, $\lambda = 0.5$.

3. METHODS

In this paper, we present new methods of inference for the RSVP Keyboard. In particular, we present new ways to calculate the posterior probability of different possible letter strings. In this section, we present our approach algorithmically, then step through a simple example to illustrate its operation.

3.1 Improved Inference

In the current RSVP Keyboard system, once a letter is typed (or deleted), the system advances to the new position and reconsiders the 28 possible options (26 letters, whitespace and backspace), without taking into account previous events. In our updated approach, once a possible string has been considered, its posterior probability is maintained even if another symbol is typed. The typed symbol at position i in the string generally has a posterior probability less than 1.0, meaning that other options retain some probability mass. Use of the backspace symbol as a deleting mechanism allows the system to go back and potentially access those alternate strings, hence we maintain their non-zero posterior probability. In fact, if we accrue additional evidence for the backspace symbol through ERP detection, that probability mass should be allocated to these alternate strings, since it is evidence that the current string is not the intended string but rather one of the alternates.

We present our algorithm formally in Figure 2, and describe it here less formally, with references to specific lines in the algorithm. To copy a given text T using a symbol vocabulary V, we provide three metaparameters: a threshold parameter θ to determine the required probability of a symbol to type it; and a minimum and maximum number of iterations of sequence presentation to gather ERP evidence for the symbols, as discussed earlier. Note that the set V includes every symbol that will be typed, i.e., not backspace, which is a special symbol denoting that a previously typed symbol should be deleted. In our case, V includes the letters a-z plus whitespace.

The algorithm maintains a set of context strings S, initialized with the empty string (with probability 1) on line 2 of the CopyText algorithm. The set S is the set of strings that are one symbol extensions of strings that have actually been typed at some point in the session. Each of the strings in the set is maintained with a posterior probability by the algorithm. Until the input text T is correctly typed, we must decide at each point which of the 28 symbols to choose. If each member of V is given an index from 1 to $|V|$, and we let 0 be the index of the backspace symbol, we can establish the prior probability for each of these in the current context. Scanning through all strings s (with associated probabilities p) in the set S (lines 6-15 of the CopyText algorithm in Figure 2), there are three possible cases for each stored string: 1) the typed string t is not a prefix of s (line 7), in which case the posterior probability p of that string is allocated to backspace (index 0); 2) the string s is exactly the typed string t (line 9), in which case s has never been expanded before and must be expanded (see below); or 3) the string s is longer than t (line 14), meaning that the posterior probability p of s should be added to that of whatever symbol from V comes after the prefix t in s. For case 2, the string s is removed from the set S, and the set of strings resulting from concatenating each symbol $x \in V$ to s are added to S, with the appropriate probability as assigned by the language model. Finally, these prior probabilities are normalized for use by the LM and ERP fusion decision process.

```
UpdSymbolProbs(V, P, i, θ, min_iter, max_iter)
1  j ← 0
2  for k = 0 to |V| do                          ▷ Let v_0 = backspace
3    L[k] ← 1                                   ▷ Initialize likelihoods to 1
4  while j < max_iter and (max(P) < θ or j < min_iter) do
5    j ← j + 1                                  ▷ Present another sequence to user
6    for k = 0 to |V| do                        ▷ Let v_0 = backspace
7      L[k] ← L[k] · ERP_ij(v_k)                ▷ Update likelihoods to return
8      P[k] ← P[k] · ERP_ij(v_k)                ▷ See Equation 4
9    P ← NORMALIZE(P)
10 return (P, L)

CopyText(T, V, θ, min_iter, max_iter)           ▷ Given a text T to copy
1  t ← ε                                        ▷ Initialize copied text to empty string
2  S ← {(t, 1.0)}                               ▷ Initialize set of context strings (string, prob)
3  while T ≠ t do
4    for k = 0 to |V| do                        ▷ Let v_0 = backspace
5      P[k] ← 0                                 ▷ Initialize probabilities to zero
6    for each (s, p) ∈ S do                     ▷ for all existing contexts
7      if s[1, |t|] ≠ t then                    ▷ t is not a prefix of s (inconsistent)
8        P[0] ← P[0] + p                        ▷ contributes to backspace probability
9      else if s = t then                       ▷ t has not been extended before
10       REMOVE (s, p) FROM S
11       for each v_k ∈ V  (1 ≤ k ≤ |V|) do
12         P[k] ← LM_{|t|+1}(v_k) · p           ▷ Initialize with LM prior
13         S ← S ∪ {(s · v_k, P[k])}            ▷ extend s by each symbol
14     else for v_k = s[|t| + 1, |t| + 1]       ▷ s[0, |t| + 1] = t · v_k
15       P[k] ← P[k] + p                        ▷ Add probability (already extended)
16   P ← NORMALIZE(P)
17   (P, L) ← UpdSymbolProbs(V, P, |t| + 1, θ, min_iter, max_iter)
18   for each (s, p) ∈ S do                     ▷ Update all context probs
19     if s[1, |t|] = t then                    ▷ s is consistent with t
20       for v_k = s[|t| + 1, |t| + 1]          ▷ v_k is symbol following t
21         (s, p) ← (s, p · L[k])
22     else (s, p) ← (s, p · L[0])              ▷ s is inconsistent with t
23   NORMALIZE(S)
24   v̂ ← argmax_{v_k ∈ V}(P)                    ▷ System detected target symbol
25   if v̂ = v_0 then                           ▷ Backspace was selected
26     t = t[1, |t| − 1]                        ▷ Remove final symbol from t
27   else t = t · v̂                            ▷ Type symbol v̂ in copied text t
```

Figure 2: Improved inference algorithm for copying text. For strings s, t, $s \cdot t$ denotes string concatenation. For string $s = s_1 \dots s_n$ let $s[i, j]$ be the substring $s_i \dots s_j$. For simplicity, if $j > i$, then $s[i, j] = \epsilon$; and if $j > |s|$, then $s[i, j] = s[i, |s|]$.

The algorithm UpdSymbolProbs at the top of Figure 2 takes as input the symbol vocabulary, the probability array calculated as described above, the position in the string and the relevant metaparameters, and returns an updated (normalized) posterior probability array and an ERP classifier-based likelihood array $\mathcal{L}$, which does not include the prior probabilities. After the symbol sequences have been presented the requisite number of times, the strings in set S are scanned again, to update their posterior probabilities with the new information (lines 18-22 of the CopyText algorithm in Figure 2). For strings s which have t as a prefix, the posterior probability is updated with the relevant symbol's likelihood at that position from the recent round of symbol sequence presentations. Otherwise, the posterior probability is updated with the likelihood of the backspace symbol. Finally, the posterior probabilities of all strings in the set are normalized.

typed string	context string	LM prob.	ERP observation 1	2	3	normalized posterior
ϵ	A	0.4	0.2			0.14
	B	0.6	0.8			0.86
Action: type B						
B	A	0.4	0.2	0.1		0.03
	BA	0.4	0.8	0.7		0.85
	BB	0.2	0.8	0.2		0.12
Action: type A						
BA	A	0.4	0.2	0.1	0.95	0.17
	BAA	0.3	0.8	0.7	0.03	0.11
	BAB	0.1	0.8	0.7	0.02	0.03
	BB	0.2	0.8	0.2	0.95	0.69
Action: delete A						

Table 1: Probabilities for example

After the posterior probabilities have been calculated for all strings in the set, the system detected symbol is determined. If that symbol is backspace, then the typed string t is shortened by removing its last symbol. Otherwise, the detected symbol is appended to t.

For this paper, we extended the above algorithm by pruning extremely unlikely strings from the set S. Any string with a probability less than e^{-30} was removed from S, resulting in increased efficiency and no typing performance difference.

3.2 Example of improved inference

For this example, we will assume an alphabet of two symbols (A and B), and our observations from the EEG will be of the form

$$\text{ERP observation } j = [\mathcal{L}(\texttt{A}), \mathcal{L}(\texttt{B}), \mathcal{L}(\texttt{backspace})],$$

where $\mathcal{L}(x)$ is the normalized conditional likelihood that the next symbol is x based on the ERP classifier. Table 1 presents the probabilities used to maintain the normalized posterior for each context string in the set S. For this example, $\theta = 0.8$ will be the decision threshold.

We begin with no letters currently typed, and no observations made. The first step is to query the language model for priors across potential next letters. The first two rows of Table 1 show example LM probabilities for the two symbols in the alphabet. Since no letter has a posterior probability above the threshold value, we must now make an observation. Note that when no letters have been typed, the probability of backspace must be zero.

$$\text{ERP observation } 1 = [0.2, 0.8, 0.0]$$

We add this to our table, calculate the product across each row, then normalize to get an updated posterior probability. The probability of the string B now exceeds threshold, so we type it. We can also now calculate the posterior probability of backspace, which is the sum of all posterior probabilities that do not have our currently typed string (B) as a prefix. In this case, it is equal to 0.14.

This leads to rows 3-5 of Table 1. Since B has been typed, we expand the context strings starting with B and re-query the language model. We need not expand the A row, but we retain it for our calculations – it now falls in the 'inconsistent' class of context strings referred to in the COPYTEXT algorithm in Figure 2. Note that score of B from ERP observation 1 is shared by both new expanded context strings BA and BB.

Since no string probability exceeds threshold based just on the LM probability and ERP observation 1, we make another ERP observation:

$$\text{ERP observation } 2 = [0.7, 0.2, 0.1]$$

When adding this observation to our table, we apply $\mathcal{L}$(backspace) $= 0.1$ to A, since it falls in the set of strings that are inconsistent with the currently typed string.

Based on posteriors calculated after ERP observation 2, A is the symbol above threshold that is typed, making the currently typed string BA. Then we expand again. Note that the probability of backspace – calculated by summing the posteriors of all context strings that are inconsistent with the typed string (lines 7-8 in the COPYTEXT algorithm in Figure 2) – is now $0.03 + 0.12 = 0.15$.

Suppose the next observation strongly favors typing backspace:

$$\text{ERP observation } 3 = [0.03, 0.02, 0.95]$$

The newly expanded table with ERP observation 3 can be seen in rows 6-9 of Table 1. The probability of backspace is now $0.17 + 0.69 = 0.86$, so we type it, taking our typed string back to B. For the next step of the algorithm after deleting the symbol, the prior probabilities for A and B would now be calculated by summing over already expanded context strings (lines 14-15 in the algorithm).

3.3 Testing and Simulation

For the experiments in this paper, we used a simulation system that generates artificial EEG score vectors from prerecorded EEG data. These are sampled from a kernel density estimation of the target/non-target classes generated by the RDA classifier on data from real users of the system. A separate model is trained for each individual user during a calibration phase, and the area under the receiver operating characteristic curve (AUC) is calculated using cross-validation. This value generally corresponds to the ease with which the ERP classification system can classify the EEG signals from that particular user, hence we use the score to characterize the user in terms of how many symbol presentations are typically required to reach threshold. We used five different user sessions as simulation data, each with a different AUC value, ranging from quite high (0.90) to relatively low (0.71), thus simulating several diverse use scenarios. In each condition, we ran 100 Monte Carlo simulations using this data.

We used three test sets: The main test set was a set of personal emails set aside for our use from a person with LIS, who we will refer to as GB. This set contained 1,128 lines of text, about 10,000 words. We also tested two other sets, one with around 35,000 lines of New York Times newswire text, and 10,000 words from a "simulated" AAC-like corpus created through crowdsourcing [15].

Grid search parameter optimization was carried out for both the baseline (current) system and the system with improved inference (see results in the next section), for all of the system meta-parameters other than minimum sequence presentations. We tried 1960 combinations of parameters, each varying by regular intervals within their natural range, over five simulations each on the held-out set, which consisted of 50 reserved lines from the GB email corpus.

We used two evaluation metrics. The first is sequences presented per letter. This records the number of observations (sequences shown) divided by the total number of characters in the test set. The second is letters per minute, which is an estimated typing rate

System Configuration	AUC 1.00	AUC 0.90	AUC 0.83	AUC 0.80	AUC 0.75	AUC 0.71
Current	1.00	2.34	2.87	7.77	FAILURE	FAILURE
Current Opt.	1.00	2.05	2.39	3.74	5.22	6.18
Current Opt+Auto	FAILURE	2.09	2.46	3.83	5.27	6.29
Current Opt+Back	1.00	1.98	2.32	3.64	5.10	6.14
Current Opt+Auto+Back	FAILURE	1.84	2.17	3.51	4.89	6.03
Best Configuration Pct. Improvement over Current	0.0%	21%	24%	54%	N/A	N/A

Table 2: Simulated Sequences per Letter on the GB email test set for various AUC values and RSVP Keyboard parameterizations, using current system inference. 100 Monte Carlo simulations in every condition.

for the test set, based on an RSVP paradigm of 200 milliseconds per symbol, 28 symbols, and five seconds between sequences.

4. RESULTS

4.1 Current system parameter optimization

Before examining the impact of our improved inference algorithm on the efficiency of typing with the RSVP Keyboard, we first examined the impact of optimizing system meta-parameter settings, along with some other system adjustments suggested by our algorithm. In such a way, we can compare against a number of baseline system configurations, to better judge the ultimate utility of our algorithm in the space of possible configurations.

Table 2 presents the mean number of sequence presentations required per letter in the GB email testset under five configurations of the current system, for varying AUC values. We include AUC=1.0 (a perfect classifier, always giving all of the probability mass to the correct symbol) to illustrate behavior of the system under ideal conditions. The first row shows the current system, using meta-parameters as described in Section 2.2.4. Even though the maximum sequences parameter is set to 3, more sequences may be required per letter because of errors in typing followed by repair. Note that, under this scenario, the simulations for users with AUC less than 0.8 fail to type the test corpus, due to the inability to recover from errors. In such a scenario, the fixed backspace probability is likely too low to adequately recover.

We next performed a grid search for optimized meta-parameter settings for each AUC level on a held-out data set, as described in the previous section. The minimum sequences parameter is left at 1 for the Current Opt. row of Table 2; setting the minimum sequences parameter to 0 yields an autotyping system, and results for that condition are shown in the Current Opt+Auto row. Optimizing these meta-parameters yields large improvements, particularly for the lower AUC scenarios – halving the number of sequence presentations for AUC=0.8 and providing a system configuration for the lower AUC values that permit task completion. Interestingly, autocompletion yields no gains in any condition, and in fact causes failure in the perfect classifier condition (AUC=1.0). This is due to endless loops that occur when an incorrect symbol is autotyped, followed by a (correct) backspace – which again yields a state that will autotype the same incorrect symbol. This scenario illustrates the importance of keeping track of prior events in assigning probabilities to subsequent events in this system.

Finally, we examined the current RSVP Keyboard system in a configuration where the probability of backspace is not fixed, but rather is related to the posterior probability when the previous symbol was typed. If p was the posterior probability of that last-typed symbol (either above threshold or highest probability after the maximum number of sequences have been presented), then the probability of delete at the next position is set to $1-p$. This configuration is presented in the Current Opt+Back row of Table 2; and is combined with autocompletion in the Current Opt+Auto+Back row. Other than for the hypothetical perfect classifier, this dynamic backspace probability yields modest improvements over the standardly optimized configuration; and with the dynamic backspace probability, autocompletion now yields some additional improvements. If we look at the percentage improvement of the best system configuration over the current system configuration, we find robust improvements for all subject-derived AUC levels, with dramatic improvements for the lower AUC levels. We can now use this as our baseline against which to compare our improved inference performance.

4.2 Improved inference

Table 3 shows the results achieved with our improved inference algorithm versus both the Current Opt. configuration and the best performing configuration from Table 2 of the current RSVP Keyboard system. For all conditions other than the perfect classifier (AUC=1.0), our algorithm with no autotype (minimum number of presented sequences set to 1) achieves modest improvements over the current configurations, with more gains at the lower levels of AUC. When we add autotyping to our improved inference by reducing the minimum number of presented sequences to 0, we get additional gains across the board, even for the AUC=1.0 condition. In this case, gains due to autotyping are greater for higher AUC conditions than they are for lower AUC conditions.

Table 3: Sequences per Letter for various AUC values and inference algorithms. Values average 100 Monte Carlo simulations.

System Configuration	AUC 1.00	AUC 0.90	AUC 0.83	AUC 0.80	AUC 0.75	AUC 0.71
Current Opt.	1.00	2.05	2.39	3.74	5.22	6.18
Current Best Configuration	1.00	1.84	2.17	3.51	4.89	6.03
Improved Inference	1.00	1.78	2.08	3.10	4.21	4.80
Improved Inference+Auto	0.67	1.48	1.78	2.82	3.93	4.56
Improvement Pct. over Current Opt.:	33%	28%	26%	25%	25%	26%
Current Best Configuration:	33%	20%	18%	20%	20%	24%

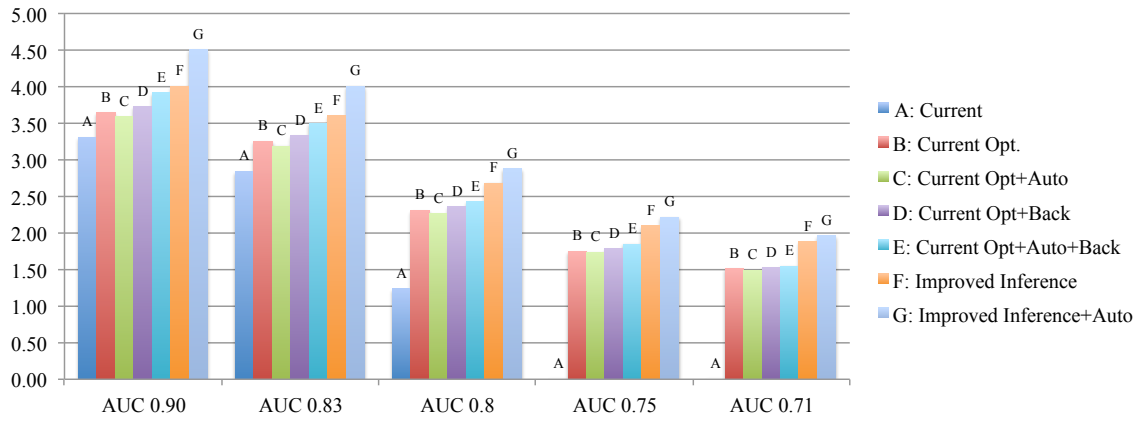

Figure 3: Letters typed per minute for various simulated AUC values.

Figure 3 presents all conditions reported in Tables 2 and 3, with the sequences per letter converted to letters per minute, following the conversion as presented in Section 3.3. In this graph, it is easy to visualize the relatively higher impact of autotyping with improved inference at the higher AUC levels versus the lower levels; and the improvements due to just improved inference at those lower levels.

Table 4 shows the performance of the best current RSVP Keyboard configuration compared to our improved inference with autotype for AUC=0.90, when evaluated on the different test sets mentioned in Section 3.3. Comparable results (indicating generalization across different text genres) were found for other AUC levels as well. Due to the number of simulations and the size of the test sets, all reported differences are statistically significant at $p < 0.001$.

Table 4: Comparison of sequences per letter across different test corpora (AUC = 0.90).

Test Corpus	Current Opt+Auto+Back	Improved Inference+Auto	Pct. Improvement
GB	1.84	1.48	20%
AAC	2.01	1.69	16%
NYT	1.85	1.50	19%

4.3 The backspace tradeoff

We find that the optimal setting for the decision threshold with our improved algorithm is at 0.5. Thus, if the probability is 0.5 or higher, then it is more efficient to go ahead and type the symbol than to spend the time to achieve higher certainty. One potential problem with setting the decision threshold this low is that a higher proportion of typing actions is a backspace than when the threshold is set at, say, 0.9, as in the current RSVP Keyboard system. Table 5 shows the ratio of backspaces to all typed symbols in different simulations. Our improved inference algorithm requires backspace more than twice as often as the fastest method under the current system, and over three times as often for the highest AUC level.

Because frequent backspaces may be confusing and frustrating, users might choose to sacrifice some performance by reducing the frequency of backspace. This tradeoff can be managed by varying the decision threshold parameter: a higher threshold will be suboptimal from a typing speed standpoint, but will result in fewer backspaces. The dynamics of this tradeoff are illustrated in Figure 4. The decision threshold values used for this figure were 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99 and 0.999. By choosing a particular operating point, it is possible to cut the backspace percentage in half and only slightly decrease typing performance.

AUC	Current Opt+Auto+Back	Improved Inference	Improved Inference+Auto
0.71	18.5	31.6	37.2
0.75	14.1	28.0	34.3
0.80	13.1	24.6	32.7
0.83	10.0	18.3	28.9
0.90	9.7	16.4	29.2

Table 5: Percentage of symbols typed that are backspace across varying AUC values.

4.4 Autotyping and autodelete

One aspect of autotyping that we have not discussed much up to now is that the backspace symbol can be autotyped just like other symbols, which results in what we call autodeletion. In fact, complex combinations of automatic typing and deleting can lead to what we have termed "autorevision." For example, Figure 5 presents a sequence of autotype and autodelete events that were actually observed in our simulated trials. Once the string "my care" was typed, autotyping produced the rest of the word "carefully" with following whitespace, before reaching a state where EEG observations were required. The result of ERP detection was a backspace symbol, leading to deletion of the whitespace. Once this deletion occurred, autotyping of the backspace symbol (autodeleting) deleted the suffix of "carefully" back to "care", at which point autotyping produced a new full word candidate "caregivers." This entire sequence of actions required just a single round of ERP detection.

For AUC=0.90, we find that 41% of symbols are autotyped, and that around 18% of autotyping events complete a word. For lower AUC values, those percentages are lower, 31% autotyped and 13% of autotyping events complete a word. Table 6 presents statistics on the frequency of the various kinds of autotyping events for different AUC levels with our improved inference algorithm. We see that forward autotyping is the most common, from 70-80% of the autotyping events, versus 20-30% being autodeletions. More than half of autotyping is a single symbol.

5. SUMMARY AND FUTURE WORK

We have shown that our improved inference algorithm results in significant improvements in simulated typing speed. We have shown

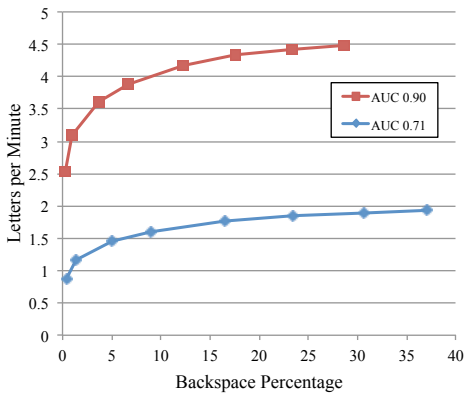

Figure 4: Backspace percentage versus typing speed for the improved inference with autotype at two simulated AUC values.

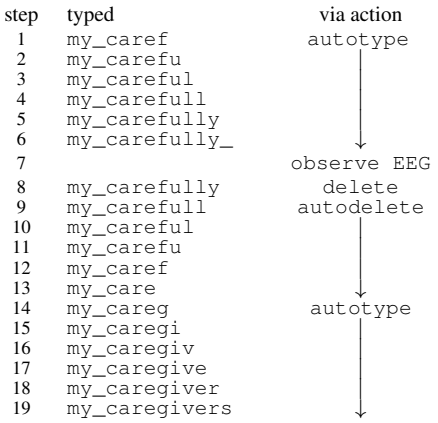

Figure 5: Example of autorevision

that this gain is consistent across different text corpora, and that autotyping is both fast and well-suited to our method. We have introduced the possibility of autodeletion and autorevision, interesting varieties of autotyping that we believe to be novel to this research. We have examined the tradeoffs involved with these higher typing speeds, specifically those involved with the frequency of typing backspace versus typing speed.

For future work, it is clear that we must validate these algorithms with real typing experiments, and with participants that fall on a broad range of AUC levels. Autotyping, including autodeletion and autorevision, will likely carry some additional processing burdon on users that will impact the ultimate utility of these algorithms. The system must be configured to clearly convey to the user when the system autotypes, so that they can focus on a target letter that is appropriate for the new context. In addition, we are currently working to analyze the behavior when presented sequences are less than the full set of symbols.

Acknowledgments

Thanks to the rest of the RSVP Keyboard team at OHSU and Northeastern. This research was supported in part by NIH Grant #1R01DC009834-01. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NIH.

6. REFERENCES

[1] B. Carpenter. Scaling high-order character language models to gigabytes. In *Proceedings of the ACL Workshop on Software*, pages 86–99, 2005.

[2] W. Crochetiere, R. Foulds, and R. Sterne. Computer aided motor communication. In *Proceedings of the 1974 Conference on Engineering Devices in Rehabilitation*, pages 1–8, 1974.

[3] R. Foulds, G. Baletsa, and W. Crochetiere. The effectiveness of language redundancy in non-verbal communication. In *Proceedings of the Conference on Devices and Systems for the Disabled*, pages 82–86, 1975.

[4] J. H. Friedman. Regularized discriminant analysis. *Journal of the American statistical association*, 84(405):165–175, 1989.

[5] D. J. Higginbotham. Evaluation of keystroke savings across five assistive communication technologies. *Augmentative and Alternative Communication*, 8(4):258–272, 1992.

[6] J. Higginbotham, B. Moulton, G. Lesher, and B. Roark. The application of natural language processing to augmentative and alternative communication. *Assistive Technology*, 24(1):14–24, 2012.

[7] H. H. Koester and S. Levine. Effect of a word prediction feature on user performance. *Augmentative and Alternative Communication*, 12(3):155–168, 1996.

[8] D. Krusienski, E. Sellers, D. McFarland, T. Vaughan, and J. Wolpaw. Toward enhanced P300 speller performance. *Journal of neuroscience methods*, 167(1):15–21, 2008.

[9] G. W. Lesher and G. J. Rinkus. Leveraging word prediction to improve character prediction in a scanning configuration. In *Proceedings of the RESNA 2002 Annual Conference*, Reno, 2002.

[10] U. Orhan, D. Erdogmus, B. Roark, B. Oken, S. Purwar, K. E. Hild II, A. Fowler, and M. Fried-Oken. Improved accuracy using recursive bayesian estimation based language model fusion in ERP-based BCI typing systems. In *34th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBCÕ12)*, 2012.

[11] U. Orhan, K. E. Hild II, D. Erdogmus, B. Roark, B. Oken, and M. Fried-Oken. RSVP keyboard: an EEG-based typing interface. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 645–648, 2012.

[12] B. Roark, R. Beckley, C. Gibbons, and M. Fried-Oken. Huffman scanning: using language models within fixed-grid keyboard emulation. *Computer Speech and Language*, in press.

[13] B. Roark, J. de Villiers, C. Gibbons, and M. Fried-Oken. Scanning methods and language modeling for binary switch typing. In *Proceedings of the NAACL HLT 2010 Workshop on Speech and Language Processing for Assistive Technologies*, pages 28–36, 2010.

[14] M. Treder and B. Blankertz. (C)overt attention and visual speller design in an ERP-based brain-computer interface. *Behavioral and Brain Functions*, 6(1):28, 2010.

[15] K. Vertanen and P. O. Kristensson. The imagination of crowds: Conversational aac language modeling using crowdsourcing and large data sources. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 700–711. ACL, 2011.

[16] I. H. Witten and T. C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *Information Theory, IEEE Transactions on*, 37(4):1085–1094, 1991.

Table 6: Frequency of different types of autotyping: forward only, backward only, and autorevision, under various AUC values. Algorithm is Improved Inference+Auto.

Type of autotyping event	AUC 0.90	AUC 0.80	AUC 0.71
Forward only	77.3	73.0	68.0
Forward only (1 symbol)	40.7	39.1	39.0
Forward only (2+ symbols)	36.6	33.9	29.1
Backward only	19.6	25.2	30.7
Backward only (1 symbol)	10.7	14.2	18.1
Backward only (2+ symbols)	8.9	11.0	12.7
Forward and backward (autorevision)	3.2	1.8	1.3