Sequential Decision-Making for Inline Text Autocomplete

Rohan Chitnis
Meta AI

Shentao Yang[*]
UT Austin

Alborz Geramifard
Meta AI

Abstract

Autocomplete suggestions are fundamental to modern text entry systems, with applications in domains such as messaging and email composition. Typically, autocomplete suggestions are generated from a language model with a confidence threshold. However, this threshold does not directly take into account the cognitive load imposed on the user by surfacing suggestions, such as the effort to switch contexts from typing to reading the suggestion, and the time to decide whether to accept the suggestion. In this paper, we study the problem of improving inline autocomplete suggestions in text entry systems via a sequential decision-making formulation, and use reinforcement learning to learn suggestion policies through repeated interactions with a target user over time. This formulation allows us to factor cognitive load into the objective of training an autocomplete model, through a reward function based on text entry speed. We acquired theoretical and experimental evidence that, under certain objectives, the sequential decision-making formulation of the autocomplete problem provides a better suggestion policy than myopic single-step reasoning. However, aligning these objectives with real users requires further exploration. In particular, we hypothesize that the objectives under which sequential decision-making can improve autocomplete systems are not tailored solely to text entry speed, but more broadly to metrics such as user satisfaction and convenience.

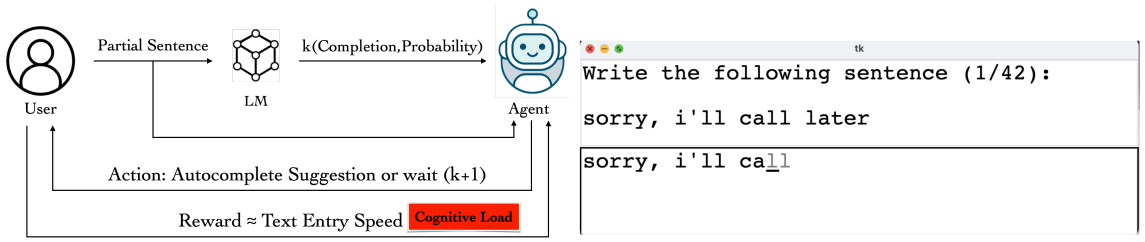
1 Introduction

The ability to enter text is essential in today's world, spanning technology such as keyboards, phone/tablet screens, and smart watches. Autocomplete suggestions are fundamental to modern text entry systems, with applications in domains such as messaging and email composition. Autocomplete provides a mechanism for users to transmit more information without needing to enter too many more characters. For example, the user may simply enter "`how a`", upon which the autocomplete system suggests "`how are you?`", which can be accepted by the user in one additional stroke.

Among many possible instantiations, we choose to focus on *inline* autocomplete systems, which may provide the user with up to a single suggestion at each timestep, displayed inline, e.g., "`how are you?`". Such systems have been shown to be more effective at encouraging users to enter text than other approaches, such as displaying multiple suggestions in a different on-screen location (Azzopardi & Zuccon, 2016). An inline autocomplete system must decide not only *what* to suggest to the user, but also *when* to make a suggestion to avoid interrupting users' focus (Quinn & Zhai, 2016).

Existing autocomplete systems generally follow a two-stage process: 1) generate a ranked list of candidate completions of the current text entered by the user, 2) decide which, if any, to surface to the user based on a pre-defined fixed confidence threshold (Cai et al., 2016; Mitra & Craswell, 2015; Quinn & Zhai, 2016; Fowler et al., 2015). The downside of these systems is that they do not directly consider the cognitive load that surfacing autocomplete suggestions will impose on the user. This may come from several aspects of the autocomplete process, such as switching contexts from typing

[*] Work done while author was an intern at Meta AI. Correspondence to: `{ronuchit, alborzg}@meta.com`


Partial Sentence
k(Completion,Probability)
User
LM
Agent
Action: Autocomplete Suggestion or wait (k+1)
Reward ≈ Text Entry Speed
Cognitive Load
tk

```
Write the following sentence (1/42):

sorry, i'll call later

sorry, i'll call
```

Figure 1: *Left:* An overview of the workflow of our RL agent for inline text autocomplete. A language model (LM) generates k candidate completions of the current text. The RL agent decides which, if any, to give the user as an inline suggestion. The agent is rewarded based on the user's text entry speed, which takes into account the cognitive load of showing suggestions. *Right:* The interface for our user study (Section 6.3). In this example, the user was asked to type the sentence "sorry, i'll call later" on a keyboard. Currently, they have typed "sorry, i'll ca", and a suggestion was made that completes the last word as "call", which the user can accept by pressing the `tab` key.

to reading a suggestion, mentally processing the suggestion, and deciding whether to accept it. Too many suggestions, even if usually accurate, can therefore lead to bad user experience.

We tackle the problem of generating inline autocomplete suggestions while minimizing the user's cognitive load by formulating text autocomplete as a sequential decision-making problem, and apply reinforcement learning (RL) methods (Kaelbling et al., 1996; Sutton & Barto, 2018) to train the autocomplete model. RL solves the problem of learning an optimal sequential decision-making policy through environment interaction, without assuming the differentiability of the objective/reward function *w.r.t.* the model parameters. RL methods have demonstrated impressive success in domains such as Atari (Mnih et al., 2013) and Go (Silver et al., 2016). Using RL methods to train an autocomplete model on top of a language model (LM) allows us to learn suggestion policies through repeated interactions with a target user, and define a reward function based on text entry speed that captures the cognitive load of surfacing suggestions (Figure 1). We note that RL is suitable here since the cognitive load takes an unknown functional form *w.r.t.* the autocomplete model.

This paper's contributions are as follows:

- We formulated the inline text autocomplete problem as sequential decision-making.
- We performed theoretical analysis on when this formulation can improve over myopic reasoning.
- We ran simulated experiments on how an RL autocomplete agent performs on an idealized user.
- We performed a user study to understand the gaps between an idealized user and a real one.

Our key findings are as follows:

(1) We acquired theoretical and experimental evidence that, when optimizing certain objective functions, RL can provide a better suggestion policy than myopic reasoning, but aligning these objectives with real-world users requires further exploration. We believe these objectives are not tailored solely to text entry speed, but more broadly to metrics like user satisfaction and convenience, which aligns with earlier findings in the text entry literature (Quinn & Zhai, 2016).

(2) Given an idealized user that always accepts correct suggestions and inputs each character without typos, we could not find any evidence with a real dataset and language model that RL-trained models improve text entry speed over the classical threshold-based approach with a fixed threshold value.

(3) Our user study ($N = 9$) reveals that for keyboard typing, the cognitive load of looking at a suggestion is independent of its length, but is dependent on whether the suggestion matches what the user is trying to type (10ms if it matches, 50ms if it does not). Also, we could not find any evidence that suggestion acceptance rate is affected by the number of past surfaced suggestions.

Recommendation. Based on these findings, we recommend that further research into sequential decision-making for inline text autocomplete should *not* pursue the goal of solely increasing text entry speed for idealized users. Instead, more realistic scenarios (stochastic user behavior, typos in their input, etc.), with a focus on user experience, may provide better opportunities for RL methods.

2 Related Work

Autocomplete has a rich history, especially in the domain of search queries (Cai et al., 2016), where methods that rely on neural language models have become increasingly popular (Wang et al., 2020; Park & Chiba, 2017; Wang et al., 2018). For example, Wang et al. (2020) demonstrate how to make use of context in both the generation and ranking phases to improve end-to-end performance. These methods can also be used to obtain personalized models by embedding user IDs (Fiorini & Lu, 2018; Jiang et al., 2018). Like these works, we also use a neural language model in this paper for candidate suggestion generation, but for selection, we use a policy trained using deep reinforcement learning.

A separate line of work has investigated the empirical performance of existing autocomplete systems (Fowler et al., 2015; Quinn & Zhai, 2016; Azzopardi & Zuccon, 2016). Fowler et al. (2015) found that the text entry enhancements found in modern touchscreen phones greatly reduce word error rate. Quinn & Zhai (2016) discovered, surprisingly, that even though autocomplete suggestions can impair text entry speed, users often prefer them subjectively because they lowered cognitive and physical burden. Azzopardi & Zuccon (2016) studied the cost-benefit tradeoffs that users make when conducting searches, concluding that inline autocomplete systems are an effective way to increase the amount of text entered. This entire line of work complements our paper, providing rationale for why inline autocompletion is a useful problem to study, and why cognitive load is important.

Reinforcement learning methods have demonstrated impressive success in domains such as Atari (Mnih et al., 2013), Go (Silver et al., 2016), and control tasks (Brockman et al., 2016), but it is not commonly applied to text autocomplete systems. The two closest works to ours are Wang et al. (2017) and Lee et al. (2019). The former studies query autocomplete as opposed to inline autocomplete; the authors formulated the problem as a multi-armed bandit (Katehakis & Veinott, 1987), a stateless simplification of the Markov decision process framework we adopt. The latter studied autocomplete as a two-agent communication game solved via unsupervised learning and optimized with policy gradients (Sutton et al., 1999). However, they assumed the user only enters keywords drawn from the target phrase, while we allow the user to input any English characters.

3 Background: Reinforcement Learning

In this section, we give background for sequential decision-making and reinforcement learning in the Markov decision process (MDP) framework (Puterman, 2014). An MDP is given by $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, with state space $\mathcal{S}$; action space $\mathcal{A}$; transition model $T(s_t, a_t, s_{t+1}) = P(S_{t+1} = s_{t+1} \mid S_t = s_t, A_t = a_t)$, where t is the time-step, $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$, and S_t, A_t, S_{t+1} are random variables; reward function $R(s_t, a_t, s_{t+1}) = r_t \in \mathbb{R}$; and discount factor $\gamma \in [0, 1]$. The optimal solution to an MDP is a policy $\pi^* : \mathcal{S} \to \mathcal{A}$, a mapping from states to actions, such that acting under π^* maximizes *return*, the expected sum of discounted rewards: $\pi^* = \operatorname{argmax}_\pi \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R\left(s_t, \pi(s_t), s_{t+1}\right)\right]$. Here, H is the horizon of an episode, a sequence of state-action-rewards from an initial to a terminal state. The optimal Q-value of a state-action pair, $Q^*(s_t, a_t)$, is the expected return of taking action a_t from state s_t, and acting optimally afterward: $Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim T(s_t, a_t, \cdot)}\left[R(s_t, a_t, s_{t+1}) + \gamma V^*(s_{t+1})\right]$. The optimal value of a state, $V^*(s_t)$, is the maximum Q-value over actions: $V^*(s_t) = \max_a Q^*(s_t, a)$.

In reinforcement learning, an agent interacts with an MDP where T and R are unknown, and attempts to discover π^* through these interactions. There are many families of algorithms for RL (Kaelbling et al., 1996; Arulkumaran et al., 2017; Sutton & Barto, 2018), but some of the most popular are policy gradient methods and value-based methods. The former optimize a policy directly (Sutton et al., 1999; Schulman et al., 2017; 2015), while the latter learn the value of each state-action pair and use that to infer the policy (Mnih et al., 2013; Watkins & Dayan, 1992). Another