

recently popularized direction is offline RL (Levine et al., 2020), in which the agent does not get to interact with the environment, but rather must learn from a static dataset of prior interactions.

4 Problem Formulation: Autocomplete as Sequential Decision-Making

In this section, we describe how to formulate inline text autocomplete as an MDP, which can then be solved via RL. As shown in Figure 1, we assume access to a language model (LM) that can generate k candidate completions of a partial sentence, along with their probabilities. We build the MDP on top of this LM. A state in the MDP is the current *context* and the k candidates from the LM (with their probabilities). The context is all characters entered so far in the sentence, which includes the full previous words and the prefix of the current word. The action space has size $k+1$: surfacing one of the k candidates to the user, or a special `wait` action that does not surface anything. Note that in this formulation, the word corresponding to each action $a \in \{1, \dots, k\}$ (except for `wait`) changes on each step based on the LM output. While the agent should do better with higher k since it has more candidates to choose from, increasing k would also increase the problem complexity.

The transition model is defined by the user and LM. On each timestep, after the RL agent acts, the user enters a character. The character can be a special acceptance key (e.g., `tab`), or the next English character the user wants to enter. After that, the LM generates k new candidates from the updated context. In our computational experiments (Section 6.2), we will consider an *idealized user* who behaves as follows. They sample a target sentence at the start of each episode, unknown to the RL agent. On each timestep, the idealized user accepts the suggestion if and only if it matches any prefix of the remaining sentence. In case of no suggestion or a mismatched suggestion, the idealized user enters the next English character of the current word without making any typos.¹ An episode ends when all characters of the target sentence have been entered.

Finally, the reward function is proportional to the time saved/lost due to the suggestions:

$$R = \begin{cases} 0 & \text{no suggestion made by agent} \\ (1 - \alpha) \cdot \text{len(suggestion)} - \beta & \text{suggestion accepted by user} \\ -\alpha \cdot \text{len(suggestion)} - \beta & \text{suggestion ignored by user} \end{cases}.$$

Here, $\alpha, \beta \in [0, 1]$ are parameters controlling the degree of penalty on the agent when it makes a suggestion, due to the user's cognitive load. α is the cognitive load proportional to suggestion length, while β is a constant cognitive load incurred for moving the gaze. Refer to Appendix A for the derivation of the reward function. By default, we use $\alpha = 40/521$ (character reading time / character writing time) and $\beta = 60/521$ ($2\times$ saccade time / character writing time). If the user is idealized, then under this reward function, `wait` actions give zero reward, correct suggestions accepted by the user give positive reward, and incorrect suggestions ignored by the user give negative reward. The choice of γ can heavily influence the optimal solution; our experiments will explore this further.

5 Theoretical Analysis

We begin with a theoretical study of a simple setting where a user may be entering one of two words with equal probability. Our goal is to answer: *Do there exist conditions such that the sequential decision-making formulation of text autocomplete is beneficial over myopic reasoning?* To answer this question, we study a restricted setup and derive conditions on the reward function under which the optimal policy in the MDP with $\gamma = 1$ obtains higher return at a particular timestep than the optimal (myopic) policy in the MDP with $\gamma = 0$, holding all other components fixed. In particular, we aim to show that the farsighted agent *waits* for more information in certain cases where the myopic agent would make an incorrect suggestion, which gives the farsighted agent higher return.

¹This transition model is stochastic from the perspective of the RL agent because it does not know the target sentence. We could alternatively model this using a partially observable MDP (Kaelbling et al., 1998), but we stick with the fully observable formulation due to its simplicity and popularity in the RL field.

5.1 Analysis Setup

We focus on an idealized user (Section 4) and $k = 1$, meaning there are two actions to choose from at each timestep: *show* the top LM candidate suggestion (S) or *wait* (W). To keep the analysis simple, we consider two arbitrary words u and v of length n . Both words share the first m letters, e.g., for words “this” and “they” we have $n = 4$, $m = 2$. The user picks the target word uniformly at random across both at the start of each episode. We aim to find α, β in our reward function such that the optimal policy for $\gamma = 1$ waits while the optimal policy for $\gamma = 0$ shows, at the last common letter (position m). Note that this is sufficient to answer our central question stated above, because the Q -value of the $\gamma = 1$ policy is precisely the sum of future rewards, our evaluation criteria. Due to greedy action selection, the $\gamma = 1$ policy will find a better action than the $\gamma = 0$ policy, should they behave differently.

We consider a specific form of the LM that outputs the non-target word as the candidate when $\leq m$ letters are entered, and the target word after that. Let u_t be the state of the t^{th} step consisting of the first t letters of u , and similarly for v_t , for any $t \leq n$. For notational simplicity, we use “ $Q(u_t, \cdot)$ ” to refer to the optimal Q -value of the state u_t (similarly with v_t) and action either S or W . Note that this notation hides the LM candidates portion of the state.

5.2 Derivation for $\gamma = 0$

We begin by deriving conditions on α and β under which the optimal policy for $\gamma = 0$ would prefer to suggest (S) at $t = m$. The base case is $Q(u_n, W) = Q(u_n, S) = Q(v_n, W) = Q(v_n, S) = 0$. Then:

Case 1: $m < t \leq n$
$Q(u_t, W) = Q(v_t, W) = 0$
$Q(u_t, S) = Q(v_t, S) = (n - t)(1 - \alpha) - \beta$
Case 2: $t \leq m$
$Q(u_t, W) = Q(v_t, W) = 0$
$\begin{aligned} Q(u_t, S) &= Q(v_t, S) = 0.5[(n - t)(1 - \alpha) - \beta] + 0.5[-(n - t)\alpha - \beta] \\ &= (n - t)(0.5 - \alpha) - \beta \end{aligned}$

The condition for preferring suggesting (S) at $t = m$ under $\gamma = 0$ is $Q(u_m, S) > Q(u_m, W)$, which implies that $(n - m)(0.5 - \alpha) - \beta > 0$.

5.3 Derivation for $\gamma = 1$

Now, we derive conditions on α and β under which the optimal policy for $\gamma = 1$ would prefer to wait (W) at $t = m$. This derivation is more involved as the Q -value expressions are recursive, due to lookahead. Again, the base case is $Q(u_n, W) = Q(u_n, S) = Q(v_n, W) = Q(v_n, S) = 0$. Now:

Case 1: $m < t \leq n$
$Q(u_t, W) = \max(Q(u_{t+1}, W), Q(u_{t+1}, S))$
$Q(v_t, W) = \max(Q(v_{t+1}, W), Q(v_{t+1}, S))$
$Q(u_t, S) = Q(v_t, S) = (n - t)(1 - \alpha) - \beta$
Case 2: $t \leq m$
$\begin{aligned} Q(u_t, W) &= Q(v_t, W) = 0.5 \max(Q(u_{t+1}, W), Q(u_{t+1}, S)) + 0.5 \max(Q(v_{t+1}, W), Q(v_{t+1}, S)) \\ Q(u_t, S) &= Q(v_t, S) = 0.5[(n - t)(1 - \alpha) - \beta] + 0.5[-(n - t)\alpha - \beta + \max(Q(v_{t+1}, W), Q(v_{t+1}, S))] \end{aligned}$

With some algebra, we can solve the recursion for the $t = m$ case that we care about:

$$\begin{aligned} Q(u_m, W) &= (n - m - 1)(1 - \alpha) - \beta \\ Q(u_m, S) &= (n - m)(1 - 1.5\alpha) - 0.5(1 - \alpha) - 1.5\beta \end{aligned}$$

The condition for preferring wait (W) at $t = m$ under $\gamma = 1$ is $Q(u_m, S) < Q(u_m, W)$, which implies that $(n - m + 1)\alpha + \beta > 1$.

5.4 Final Constraints on α and β

Putting together the work in Section 5.2 and Section 5.3, we finally obtain the following constraint on α and β which would lead to different optimal policies for $\gamma = 0$ and $\gamma = 1$ at position m :

$$\boxed{\frac{1 - \beta}{n - m + 1} < \alpha < \frac{0.5(n - m) - \beta}{n - m}}.$$

With a fixed value of β , an α value close to 0.5 can satisfy the above. To go beyond the two-word setup empirically, we obtained the most likely 500 words from our LM (described in Section 6.1), calculated the optimal policy using backward dynamic programming for $\gamma = 0$ and $\gamma = 1$ (other MDP components are the same), and measured the number of states for which the two policies differed. Figure 2 below depicts this difference for various α . We set β to the default value 60/521.

We can see that the gap is largest when α is between 0.3 and 0.4. Hence, for these values of α , we should see more benefit to solving the autocomplete problem via the sequential decision-making formulation; however, the disagreement (y -axis) only corresponds to $\sim 2\%$ of the state space. Another intuition for using larger α values is to more strongly penalize long but incorrect suggestions, which incentivizes an RL agent to wait for additional input from the user before being informed enough to make a suggestion. Notice that as α gets closer to 1, both policies become very conservative due to the high potential penalty of making wrong suggestions, and hence the number of disagreements reduces drastically.

5.5 Limitations of our Analysis

Although we derived constraints under which text autocomplete benefits from sequential reasoning, these constraints may not reflect the goal of text entry speed improvement. In our user study (Section 6.3), we will see that the ideal values of α and β for faster text entry are near zero, which violates the constraints we derived. Additionally, our analysis depends on a simple form of LM which is unlikely to reflect those used in practice. The degree to which the sequential decision-making formulation can benefit our problem depends on the user's goal and what LM we use to implement the MDP.

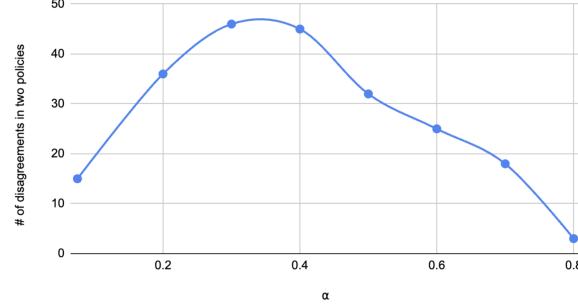


Figure 2: Number of states where the optimal far-sighted policy ($\gamma = 1$) and the optimal myopic policy ($\gamma = 0$) disagree, for various values of α .