1 Introduction

Large-scale generative models are increasingly being used in tooling applications. As one prominent example, code generation models, such as Copilot [8], CodeWhisperer [1], and AlphaCode [7], recommend code completions within an IDE to help programmers author software. However, since these models are imperfect, their erroneous recommendations can introduce bugs or even security vulnerabilities into a code base if not overridden by a human user [14]. In order to override such errors, users must first detect them. This can be challenging as even experts may be susceptible to automation bias and automation-induced complacency [13, 17]. To help users detect errors in medical [5, 11], legal [2, 9], and other high-stakes domains, conveying AI uncertainty and providing explanations has become of paramount importance [4, 3]. However, prior scenarios focus on decision support (e.g., a single classification or diagnosis), and it is not clear how to translate these strategies to generative scenarios where every generation may include dozens or hundreds of small decisions (e.g., each token of recommended code). Likewise, it is unclear how best to convey the uncertainty of generative models to human operators or if doing so will positively impact human-AI collaboration.

To make progress on these questions, through a mixed-methods, preregistered (https://osf.io/tymah) study with N = 30 participants, we explore the value of token-level highlighting in a code generation scenario. Similar to in-line spell-check in text editors, highlighted tokens are meant to draw operator attention to regions of the code that would benefit most from human oversight. In our study, we explore two possible highlighting strategies, together with a baseline without highlights. Our first strategy highlights tokens with lowest probabilities, as output directly from the underlying language model. The intuition is that low-probability sequences are non-conventional, and therefore may indicate an error. This highlighting strategy has been proposed in past work [15] and is implemented in OpenAI’s online “Playground” interface [12]. Conversely, our second highlighting strategy directly predicts the need for intervention by predicting which tokens of a suggestion are likely to be edited. For this paper, we learn an edit model for a closed-world set of programming tasks, but argue that there are clear paths to generalize the approach by learning from existing large-scale data, including telemetry. From our study, we find the edit model strategy results in significantly faster task completion time, significantly more localized edits, and was strongly preferred by participants.
2 Study and Results

We conducted an interview study with 30 participants. All were employees of a large technology company located in the United States and have experience coding in Python. Participation was voluntary and participants were paid $50. Interviews lasted approximately one hour. Participants were asked to complete three coding tasks, selected from Leetcode’s “easy” setting [10], with three AI-powered code completion tools. In actuality, all three tools used OpenAI’s Davinci-002 Codex model [6], and differed only in how completions were presented: One tool showed only the generated code completion (Prediction only). Another showed the completion with highlights on tokens that were most uncertain (threshold at 71%) as output by Codex (Generation probability). The final tool highlighted tokens that were most likely to be edited (threshold at 66%) according to our edit model (Edit model). The edit model was built using data collected from nine coders in a preliminary data collection phase: Coders were asked to edit the Codex output until the task was completed properly. We chose the highlighting thresholds such that the total number of highlights shown across all three tasks for each condition were equivalent. The order of tasks as well as assignments of tools to tasks were randomized. Participants were able to run their code for debugging, and also run a set of provided unit tests. Once participants were satisfied with their solution, or after a limit of 10 minutes, participants were asked a series of questions rating their experience.

Our results show that token-level highlighting meaningfully impacts user behavior, and further that the choice of highlighting strategy yields critical differences. For instance, participants were fastest in the edit highlighting condition ($\mu = 8.59$ minutes), and slowest in the generation probability condition ($\mu = 9.27$ minutes) (Fig 1a). The difference between the two highlighting conditions is highly significant ($p = 0.003$), while the difference between the edit condition and the prediction only condition shows a promising trend in this direction ($p = 0.06$). Echoing this finding, our results also indicate that our edit model steers people towards making more precise edits (Fig 1b). Non-highlighted tokens are significantly more likely to remain untouched by the participant in the edit model condition ($\mu_{\text{survives}} = 0.87$), than in both the generation probability condition ($\mu_{\text{survives}} = 0.81, p < .0001$), and the prediction only condition ($\mu_{\text{survives}} = 0.79, p < .0001$). Conversely, tokens that are highlighted in the edit model condition survive significantly less often ($\mu_{\text{survives}} = 0.35$) than tokens highlighted in the generation probability condition ($\mu_{\text{survives}} = 0.74, p < .0001$), meaning that the edit model is a stronger signal of what will be edited by people. Finally, this preference is echoed in participants’ subjective ratings to the following 7-point Likert items: “I found the AI’s highlights helpful in determining what to edit”; “I would be willing to pay to access the AI’s highlights”; and “I found the AI’s highlights distracting” (reverse coded). Here we find an average response of 3.94 for the edit conditions vs 2.88 for generation probability condition (Fig 1c), and this difference is again significant with $p = 0.001$.

3 Conclusion and Future Work

Together our findings demonstrate that highlighting appropriate tokens in generated code can meaningfully impact and improve operator behavior, and point a clear path in favor of edit models for this purpose. However, important research questions and challenges remain. First, our work relies on a closed-world edit model, learned on the very AI-generations evaluated in our study. This clearly represents a best-case perfectly-calibrated scenario. It remains to be demonstrated that we can
learn an open-world general-purpose edit model, and that such a model would similarly impact user interaction. Fortunately, systems like GitHub Copilot already consider edits to AI-generations as a form of performance metric [18] and this product-scale data stream could be directly repurposed to this end. Secondly, it remains to be demonstrated that the observed benefits (task completion time, targeted edits, preference), translate to important high-level outcomes such as more accurate human oversight and reduced automation bias. However, past work has shown that reducing the effort needed to interpret model explanations, or expressions of uncertainty, can increase the likelihood of people overriding AI-induced errors [16]. We hope to explore these questions in immediate future work.

References


