Identifying the Factors that Influence Trust in AI Code Completion

Adam Brown
adambrovvn@google.com
Google
USA

Sarah D'Angelo
sdangelo@google.com
Google
New Zealand

Ambar Murillo
ambarm@google.com
Google
Germany

Ciera Jaspan
ciera@google.com
Google
USA

Collin Green
colling@google.com
Google
USA

ABSTRACT

AI-powered software development tooling is changing the way that developers interact with tools and write code. However, the ability for AI to truly transform software development may depend on developers' levels of trust in these tools, which has consequences for tool adoption and repeated usage. In this work, we take a mixed-methods approach to measure the factors that influence developers' trust in AI-powered code completion. We found that characteristics about the AI suggestion itself (e.g., the quality of the suggestion), the developer interacting with the suggestion (e.g., their expertise in a language), and the context of the development work (e.g., was the suggestion in a test file) all influenced acceptance rates of AI-powered code suggestions. Based on these findings we propose a number of recommendations for the design of AI-powered development tools to improve trust.

CCS CONCEPTS

• **Human-centered computing → Empirical studies in HCI**.

KEYWORDS

Artificial Intelligence (AI), Code Completion, Logs based analysis, Software Engineering, Trust

ACM Reference Format:
Adam Brown, Sarah D'Angelo, Ambar Murillo, Ciera Jaspan, and Collin Green. 2024. Identifying the Factors that Influence Trust in AI Code Completion. In *Proceedings of the 1st ACM International Conference on AI-Powered Software (AIware '24), July 15–16, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3664646.3664757

1 INTRODUCTION

AI-powered software development tooling is transforming the way that developers write code [4, 17]. With the creation of tools like GitHub Copilot [10], Amazon CodeWhisperer [2], and Google Duet [11], generative AI has been placed directly into a developer's Integrated Development Environment (IDE) and the capabilities of this technology are expanding. Most, if not all, hypotheses around the impact of these technologies suggest that leveraging these systems will lead to sizable improvements in developer productivity. Many early results appear to support these hypotheses, either in terms of objective measurements (e.g., time to complete a development task) or self-reported productivity [5, 7, 25, 28]. While these early results are encouraging, the ability for AI to truly transform software development may depend on developers' levels of trust in these tools. Prior work has proposed that trust is a key factor for adopting tools [3, 12, 19], which suggests that developers may not reap the maximum benefit of these technological advances unless they develop an appropriate level of trust in AI-powered tools, leading to increased usage of them in day-to-day work.

It is critical that when we talk about trust in AI, we focus on helping developers build an appropriate level of trust. Too much trust when it is not warranted is risky [9, 15]. We still want developers to appropriately review AI-powered suggestions. For example, prior work has shown that developers can write less secure code when they use AI-powered coding tools [21, 22]. With this in mind, we should aim to support developers in building the appropriate level of trust, rather than simply increasing trust in AI without an upper bound. In order to achieve this goal, we need to understand trust in AI in the context of software engineering and, if possible, do so passively and at scale. In this work, we aim to answer the question: *"How do we measure the trustworthiness of AI-powered software development tools?"*

In order to tackle this question, we first sought to understand the factors that influence developers level of trust in AI. Based on prior work [12] and our own preliminary research, we identified three main drivers of trust in AI-powered development tools:

(1) characteristics of the suggestion (e.g. quality or length of suggestions)
(2) characteristics of the developer (e.g. their level of expertise or familiarity with AI)
(3) the development context (e.g. how complex or sensitive is the task)

Next, we aimed to determine which of these factors we could quantify from logs data to enable measurement at scale. In support of developing a logs-based measurement of trust to enable this analysis, we specifically examined trust in AI-generated code completion suggestions and leveraged the acceptance rate of suggestions as a proxy for trust. In this paper, we outline our mixed-methods approach to measuring trust in AI-powered code completion. Section 3 reports a preliminary study that defines the characteristics that developers consider when evaluating their trust in AI. Section 4 builds on the results of the preliminary study by extracting behavioral data from logs that correspond to these characteristics and identifies which ones are related to trust when interacting with an AI-powered tool during development work. Based on these results, we make recommendations for the design of AI-powered tooling that support building the appropriate level of trust.

2 RELATED WORK

The rapid growth of AI-powered development tools has generated a wealth of research aimed at understanding the impact of the presence of AI in development workflows. Much of this research has investigated how the inclusion of AI in developer tools impacts productivity outcomes and tool adoption [4, 7, 8]. For example, research has shown a positive impact of interacting with code completion suggestions on self-perceived productivity [25, 28] and coding iteration time [24]. While other studies have revealed that developers may spend more time reviewing suggestions than saving time [17] or that their interactions with code suggestions depend on whether they are engaged in exploratory work to understand how to complete a task or relying on AI to accelerate work that they already know how to complete [5]. These studies demonstrate that while there is an opportunity for AI to change the way developers work, it is not a panacea for productivity and there is considerable nuance in understanding how developers are engaging with AI-powered tools.

Thus, we should think holistically about developers' interactions with AI. While productivity is one of the desired outcomes of these interactions, we propose that trust is a core driver of this outcome: trust influences adoption and repeated use of tools both in the context of AI-powered tools and software development more generally [3, 12, 15, 19, 20]. Accordingly, a deeper understanding of trust may lead to a better understanding of the impact of AI more broadly. Prior work has leveraged various behavioral indicators in order to measure trust in the output from ML- and AI-based systems [14, 26, 27]. Specifically, these studies have used measures of how often an individual uses the information provided by these systems as their own answer, suggesting that the act of accepting a prediction from a model may represent a form of trust. In terms of the relationship between trust and productivity, recent work has shown the acceptance rates for AI-generated code suggestions, which may represent a proxy for trust in this context, predict developers' perceptions of their productivity [28]. Together, these findings suggests that if we can understand what influences whether or not a developer trusts AI output, leading them to accept suggestions from these systems, we may be able to develop a better understanding of how to increase productivity with AI.

Recent work has introduced a framework for building trustworthy software tools [12] that proposes trustworthy tools have specific qualities that span five pillars: Personal, Interaction, Control, System, and Expectations (PICSE). This framework aligns with other research that has demonstrated how showing additional information regarding model confidence can increase trust in AI [27], as well as information about the individual interacting with these tools (e.g., gender or level of expertise) can influence trust [3, 23]. From a more granular perspective, researchers have investigated which suggestions are most likely to be accepted by developers based on the characteristics of the suggestion (e.g., length, quality, etc.) and how to optimize models for these characteristics in order to increase acceptance rates, which may be one method for increasing trust in these systems [18]. However, there is an opportunity to further expand our understanding of the characteristics of the developers and contextual factors of the code, which are a critical part of how developers experience AI-powered suggestions.

Building on the prior work in this space, we aim to identify the logs-based factors that influence acceptance rates for Google's internal AI-powered code completion [24]. Based on these findings we will expand our understanding of how developers interact with AI, which will enable us to make informed design decisions that support developers in building an appropriate level of trust in AI and potentially unlocking large benefits to productivity.

3 PRELIMINARY STUDY: UNDERSTANDING DEVELOPER TRUST IN AI

In order to measure trust and quantify the factors that influence trust, we first need to understand how developers describe aspects of interacting with AI-powered technologies that they believe shape trust. For example, what makes them more or less likely to trust the outputs of AI powered developer tooling and incorporate these outputs into their own work? To address these questions, we conducted a preliminary study in which we asked developer what influences their level of trust in AI. We then analyzed the results with a focus on factors that could be quantified using logs data.

3.1 Method

We recruited 12 software developers employed at Google to participate in a sixty-minute remote-moderated semi-structured interview. In the interviews, we asked participants a series of questions on their experience with and perceptions of AI-powered developer tooling. Prior to participating in the study, participants gave written consent. The interviews were transcribed and we conducted a thematic analysis with inductive coding [13] to identify factors that were common across participants.

3.2 Results

Based on our interviews we identified three factors that influence developers' levels of trust in AI-powered developer tools that appeared to be good candidates for measurement: (1) characteristics of the suggestions, (2) characteristics of the individual developer and (3) the development context. These are consistent with prior work [12], which also highlights personal and system-level factors. Compared to prior work, we have a adopted a more limited set of factors that are good candidates for logs-based measurement and more tailored to interactions with code completion tools.

3.2.1 *Characteristics of the Suggestion.*
The most common factor from the interviews was characteristics of the suggestion. Specifically, when determining their level of trust in AI-powered suggestions, a majority of developers commented on the quality of the suggestion. Many expressing that they are more likely to trust high quality or accurate suggestions and less likely to trust incorrect suggestions. For example one participant commented:

> "I think it's the more errors that I see the more likely I will trust it less."

In addition to quality, developers also mentioned suggestion length, expressing that longer suggestions have a higher risk associated with them because they take a long time to review.

3.2.2 *Characteristics of the Developer.*
Participants also commented on personal characteristics that played a role in their level of trust. For example, their understanding of particular language and their confidence in their ability to evaluate an output influenced their level of trust. If they didn't feel confident in their knowledge it limited their ability to trust AI. For example, one participant commented:

> "Something that would help enhance my trust is my ability to evaluate. In the C++ case, my ability to evaluate was low and so I couldn't determine if this was a good suggestion or not. So I had very low trust in it."

In addition to level of expertise in a given language or domain, developers also cited their familiarity with the tool playing a role in their level of trust; the more familiar they were with how the tool performed, the more appropriate their level of trust became.

3.2.3 *Development Context.*
Lastly, developers mentioned that the context in which they encounter a suggestion matters, for example, if they are working on more critical or complex tasks compared to simple tasks. Developers expressed that they were more likely to trust AI for simple tasks and less likely to trust AI for complex or sensitive tasks. For example, one participant commented:

> "It's very much specific to a particular domain, right? When it's something more general, I'm fine giving away some control. If it's something very critical to the domain or it needs a lot of domain expertise, in that case, I would prefer keeping the control with me."

Ultimately, the combination of the characteristics of the suggestion, the individual developer, and the context inform trust. To successfully measure trust, we should incorporate signals of all three and identify the relative strengths of their importance for trust. As one participant explained:

> "It's a mix of individual ability to evaluate and more experiences with it going well."

4 LOGS BASED ANALYSIS OF TRUST

Building on the findings from the preliminary study, we approached identifying factors that influence trust that represented all three themes: the characteristics of the suggestion, the developer, and the context. We also required a reasonable proxy for developers' trust in AI. For this analysis, we used accepted suggestions as this proxy. We propose that this is a valid proxy because accepting a suggestion demonstrates that the developer is being vulnerable to taking on code written by AI that they believe will benefit them, which is consistent with prior definitions of trust [16]. Additionally, as described in Section 2, a number of studies have used acceptance of output from ML/AI systems as a measure for trust [14, 26, 27]. We acknowledge the limitations of this approach in our discussion.

Developer behavior is complex and no single feature is likely to capture the complete story about what influences trust in code completion suggestions. However, we wanted to build on the insights presented in the preliminary research and attempt to better understand whether there are certain situations or characteristics of development or developers that are associated with higher levels of trust in code completion suggestions. In order to address these questions, we constructed a dataset using data from developers at Google that combined information about the developer, the suggestion, and the broader context of the work being completed to model how this information influences the acceptance of suggestions. In selecting these factors we prioritized data availability and alignment with insights from our preliminary study. We recognize this is only a subset of all possible factors but it is reflective of our logs coverage and qualitative research.

4.1 Method

4.1.1 *Data.*
To construct our dataset for this research, we combined logs-based signals from Google's internal code completion service [24] with existing data about developers and aspects of their work. We sampled 1M multi-line code completion suggestions[1] that were seen by 59k individual developers. Each suggestion was required to be associated with an eventual code change. When available, the issue tracking information associated with the change was also matched back to an individual suggestion[2]. At a high level, we aimed to generate model features that were about the characteristics of the code completion suggestion, characteristics of the developer, and characteristics of the code change and issue tracking information. In each case, we placed an emphasis on generating features that were related to quality, control, expertise, and familiarity. The full list of features can be found in Table 1.

For the completion suggestion information, we extracted the model quality score associated with the suggestion (the model produced log-probability of the suggestion being accepted), the character length of the suggestion, the character position in a line of code where the suggestion was displayed, and the line position in the file where the suggestion was displayed. For the developer information, we included the number of code completion suggestions that they had previously seen overall, within the file they were working on, on the current day, and in the current file on the current day. These features were designed to get at the concept of familiarity of interacting with this technology at different levels of granularity. We developed a number of features related to developer language and codebase expertise. First, we created a measure of language expertise by identifying the percentage of total Lines of Code (LOC)

[1] These code completions, as the name suggest, span multiple lines of code. We focused on these suggestions specifically because they were a newer addition to development workflows, which we hypothesized would make trust a more central construct when interacting with them.

[2] 85% of the code suggestions were successfully associated back to an issue. Models were constructed such that we introduce a categorical variable indicating that this information was missing.

Table 1: Complete list of features generated for analysis.

Metric Category	Metrics
Code Completion Suggestion	Model Quality Score Character Length Line Position in File Character Position in Line
Developer Characteristics and Experience	Familiarity with Code Completion • Number of Suggestions Seen Overall • Number of Suggestions Seen on day • Number of Suggestions Seen within current file • Number of Suggestions Seen on day within current file Language Experience • % LOC in Suggestion Language • Readability in Suggestion Language (Y/N) Familiarity with Current Code • Pct. Recent Code Changes in Codebase • Pct. Recent Code Changes in File Job Level Tenure
Development Context	Test File (Y/N) Change Size Bug Information (Type & Priority)

submitted in different languages in the 90 days leading up to when a suggestion was shown. These percentages were split into different buckets that mapped onto different levels of experience (e.g., "No Usage" = 0%; "Primary Usage" = >25%). Second, our company enforces mandatory style guidelines for specific languages, such that all code in these languages is subject to review for adherence to these guidelines. Developers can demonstrate their knowledge of the best practices and function as reviewers for style; this process is known as *readability*. If a changelist author modifies a file in a language for which they do not have readability, the changelist must be approved by a reviewer with readability in that language. For each code completion suggestion, we identified whether the developer had readability in the language the code suggestion was shown in. We derived two features related to how familiar the developer was with the code they were interacting with using data from the 30 days leading up when they saw a suggestion: (1) the percentage of their code changes that included the file the suggestion appeared in and (2) the percentage of their changes that were associated with the codebase project the current file was associated with. We also included each developer's employment tenure and job level as covariates. In terms of changelist and issue tracking data, we classified whether the code suggestion was shown in a test file, the size of the eventual code change associated with each suggestion (as Small, Medium, Large sizes based on LOC), the type of issue listed (e.g., bug fix, feature, etc.) and the priority of the issue.

4.1.2 Model. The features from the previous section were entered in a mixed-effects logistic regression that modeled individual developers as a random effect (intercepts) and included a temporal spline to account for changes over time. Two completion information features (suggestion length and line position in file) were modeled non-linearly due to patterns revealed during exploratory data analysis, as well as reasonable hypotheses that these relationships could take on more of an inverted-U shape than a linear pattern. In each case, these features were split into Low/Medium/High buckets using terciles and modeled as categorical variables using the Medium level as a reference. All p-values were adjusted to control for the false discovery rate with the Benjamini & Yekutieli method [6].

5 RESULTS

The regression results are shown in Table 2. Reference groups are specified for each categorical input to the model. The model intercept represents a multi-line code completion with the mean model quality score at the reference levels shown in the table.

Table 2: Table containing all Odds Ratios (ORs) and 95% CIs for ORs

	Dependent variable:
	Accepted Suggestion?
Intercept	1.039 (0.999, 1.080)
Model score (Z-transformed)	1.231*** (1.225, 1.236)
Suggestion Length (ref = p33-p66)	
p0-p33 Suggestion Length	0.784*** (0.776, 0.793)
p66-p100 Suggestion Length	0.785*** (0.776, 0.793)
Line Position (ref = p33-p66)	
p0-p33 Line Position	0.859*** (0.849, 0.868)
p66-p100 Line Position	0.938*** (0.927, 0.949)
Character Position	0.991*** (0.991, 0.992)
Overall Suggestion Number	1.000*** (1.000, 1.000)
Suggestion Number on Day	0.994*** (0.993, 0.995)
Suggestion Number in File on Day	1.008*** (1.006, 1.009)
Is Test File (Y)	0.792*** (0.784, 0.800)
Pct. Code Changes with Codebase	1.000*** (0.999, 1.000)
Pct. Code Changes with File	1.000 (0.999, 1.000)
Language Usage (ref = Working Usage (10% - 25%)	
No Language Usage	0.909*** (0.875, 0.942)
Some Language Usage (<10% LOC)	0.959** (0.939, 0.980)
Primary Language Usage (>25% LOC)	1.041*** (1.022, 1.060)
Readability in Language (Y)	1.305*** (1.262, 1.349)
Change Size (ref = Size S)	
Change Size XS	0.870*** (0.832, 0.909)
Change Size M	1.095*** (1.074, 1.116)
Change Size L	1.146*** (1.124, 1.169)
Change Size XL	1.148*** (1.117, 1.178)
Change Size U	0.875 (0.194, 1.555)
Bug (ref = No Bug Information)	
Bug Priority P0	1.002 (0.972, 1.032)
Bug Priority P1	0.999 (0.977, 1.022)
Bug Priority P2	1.016 (0.994, 1.037)
Bug Priority P3	0.954 (0.914, 0.994)
Bug Priority P4	1.054 (0.962, 1.146)
Bug Type - Bug Fix	0.959*** (0.942, 0.977)
Bug Type - Feature Request	1.007 (0.989, 1.024)
Bug Type - Internal Cleanup	0.845*** (0.817, 0.873)
Job Level (ref = Level 4)	
Job Level 3	0.975 (0.946, 1.005)
Job Level 5	1.027 (0.995, 1.058)
Job Level 6	1.043 (0.986, 1.100)
Job Level 7+	1.176 (1.025, 1.328)
Tenure (ref = 1-3yrs)	
Tenure<6 months	1.073** (1.034, 1.111)
Tenure 6 months-1yr	0.996 (0.967, 1.025)
Tenure 3-5yrs	0.956* (0.926, 0.986)
Tenure 5-8yrs	0.958 (0.920, 0.996)
Tenure 8+yrs	1.011 (0.963, 1.058)
Spline	0.554*** (0.541, 0.567)
Pseudo R^2	0.182
Observations	1,000,098
Note: p-values are adjusted for FDR	*p<0.05; **p<0.01; ***p<0.001

5.1 Code Suggestion Characteristics

All four features generated from characteristics of the code completion suggestion were significant predictors of accepting a suggestion. The model quality score, which was Z-transformed, was a significant positive predictor of accepting a suggestion; a one standard deviation increase in model score was associated with a 23.1% increase in the odds of accepting a suggestion, *OR* = 1.231, 95% CI [1.225, 1.236]. This was one of the strongest predictors of accepting a suggestion. The position in a line was a negative predictor of accepting the suggestion, with each additional character position leading to a decrease in the odds of accepting the suggestion *OR* = 0.991, 95% CI [0.991, 0.992]. Suggestions that were in the bottom 33rd percentile and top 33rd percentile in length were less likely to be accepted than suggestions of intermediate length. A similar pattern (i.e., an inverted-U) emerged for suggestions that were in the top and bottom 1/3 of a file.

5.2 Developer Characteristics

5.2.1 Familiarity with Code Completion. Developer's familiarity with multi-line code completion, as measured by the number of times they were exposed to suggestions, was a positive predictor of accepting suggestions. The current analysis found that the overall count of multi-line suggestions that a developer had seen was a small, but significant positive predictor of accepting a suggestion, *OR* = 1.0002, 95%CI = [1.0001, 1.0003]. A one standard deviation increase for this input (*SD* = 101.15) would lead to a 2% increase in the odds to accept a suggestion. Additionally, we saw that the number of code completion suggestions a developer saw on a given day within a given file was a positive predictor of this behavior, *OR* = 1.008, 95%CI = [1.006, 1.009], suggesting that familiarity can have a positive impact both globally and more locally within specific files on specific days. However, we also saw that seeing more suggestions on a specific day was negatively associated with accepting suggestions, *OR* = 0.994, 95%CI = [0.993, 0.995], which may suggest seeing many suggestions across a number of different files actually comes at a cost to trust. Familiarity was one of the key aspects for developing trust in AI assistance tooling that was found in our qualitative research. This finding was largely replicated at scale using more quantitative data, however it is worth noting that these effects are relatively small compared to some of the other relationships found in this research. This is revisited in the discussion.

5.2.2 Language and Codebase Expertise. Another of the stronger predictors of whether or not a developer would accept a multi-line code completion suggestion was whether or not they had readability in the language the suggestion was shown in, *OR* = 1.305, 95%CI = [1.262, 1.349]. In order to gain readability, developer must go through a lengthy review process in which their code changes are specifically critiqued for adherence to the company style guide[3]. Given that code completion models are trained only on code that has made it into the codebase, and likely follows the style guide due to this, it appears possible that individuals that are trained to look for certain style-based signatures might be more likely to accept suggestions due to their presence. It also seems likely that these individuals are heavy users and experts in this language.

We also found that experience working with a given language was a predictor of accepting or rejecting suggestions. Compared to developers who showed Working Usage Experience (between 10% and 25% LOC in the previous 90 days), developers who were Primary Users of a language (>25% LOC) were more likely to accept a suggestion *OR*=1.041, 95% CI =[1.022, 1.060], whereas developers that had Some Usage (between 0% and 10%) and No Usage in the previous 90 days were less likely to accept a suggestion, *OR* = 0.959, 95% CI = [0.939, 0.980] and *OR* = 0.909, 95% CI = [0.875, 0.940], respectively.

The percentage of code changes in the 30 days leading up to the suggestion that were linked to the codebase project of that change showed a very small relationship with accepting a suggestion, *OR* = 1.000, 95% CI = [0.999, 1.000]. Given the sample size of this analysis and the range of possible values that this feature can take, we have decided not to interpret this effect due to its potentially limited practical significance. The percentage of code changes that involved the file that a suggestion was shown in did not show a significant relationship with accepting a suggestion. Together, these findings suggest that familiarity with the code that is being worked on does not influence how likely a developer is to accept code completion suggestions.

5.3 Development Context

As a proxy for the context of the work being completed when a developer saw a suggestion (e.g., how critical a code change was) we collected information about the type of work being completed, its priority, and its scope. We found that the priority of the work, according to the company's issue tracking software, was not a significant predictor of accepting code completion suggestions. Work priority is one proxy for criticality, but it is possible that this information does not quite capture what developers think about when they describe a task as "critical." We return to this in the discussion.

When a developer was working on a code change that was listed as a Bug Fix or an Internal Cleanup, both representing smaller scopes, they were less likely to accept suggestions than when this information was not present, *OR* = 0.959, 95% CI = [0.942, 0.977] and *OR* = 0.845, 95% CI = [0.817, 0.873], respectively. In each case, we hypothesize that the code change being worked on is potentially more constrained (i.e., a small, specific behavior is targeted), which makes longer code completion suggestions undesirable. This finding appears consistent with the ones above related to the size of the change being made.

We also found that code completions that occurred within in test files were less likely to be accepted *OR* = 0.792, 95% CI = [0.784, 0.800]. This was one of the strongest negative predictors of accepting a suggestion. Similar to the effects described above regarding smaller scopes, it is possible that tests are also more constrained than feature code (i.e., tests may look to elicit one narrow behavior, which a developer may have stronger plans about earlier in the development process).

5.3.1 Work Demographics. We included two covariates dealing with the developer's work tenure and job level as additional measures of experience, but also to generally control for these features.

[3]All production code is subject to readability review, but not all reviewers are granted the ability to provide approvals for readability

Many analyses for developer productivity tend to control for these features when possible and often researchers make hypotheses surrounding how being more senior or having a longer tenure might influence various developer experience metrics. Here, we did not find significant effects of job level, but found that tenure had some impact on acceptance rate, such that very new developers with less than 6 months of tenure were more likely to accept suggestions than those with 1 to 3 years of tenure at the company, *OR* = 1.073, 95% CI = [1.034, 1.111]. Developers with 3 to 5 years of tenure, were less likely to accept suggestions than this reference group, *OR* = 0.956, 95% CI = [0.926, 0.986]. No other tenure groups showed a difference compared to this reference group. This pattern of results may suggest a trend that newer employees are more susceptible to code completion suggestions, whereas more tenured employees are less susceptible to them. We suggest that these findings should be considered in the broader context of learning a new skill, which is one of the proposed areas of benefit of working with these technologies. However, this finding is potentially at odds with other experience and expertise results described above, where we found having *more experience* was positively related to accepting suggestions, not less.

6 DISCUSSION

In our preliminary study, through interviews with developers, we identified three aspects that influence trust in AI-powered tools: (1) characteristics of the suggestions, (2) characteristics of the developer and (3) the development context. We developed a dataset that collected signals that aligned with each of these characteristics and modeled how this information influenced trust in the suggestion, as proxied by whether or not the suggestion was ultimately accepted.

Our results underscore that developer behavior is complex and no single feature is likely to capture the complete story about what influences trust in AI-powered tools. However, these findings expand prior research on predicting when code completion suggestions are useful to developers by including a number of features about the developers themselves, as well as the contexts in which they work [18]. Much attention in this area has been given to the systems and models involved in surfacing AI-generated code to developers, while little work has investigated which individual and situational aspects of the development process may play a role in how much trust there is in these systems. Our findings introduce the notion that these additional features are important when thinking about trust of these technologies and suggest that certain design decisions may be available to bolster trust in AI-powered tooling; we discuss these opportunities below.

We found that one of the strongest positive predictors of accepting a suggestion was the model quality score for that completion (i.e., whether the underlying AI model thought the code would be accepted). This finding makes sense intuitively and replicates prior analyses that suggest model score is an important feature when predicting acceptance [18]. It seems likely that performance of code generation models will only continue to improve over time, however, when considering the impact of quality on trust it seems worthwhile to think about the behaviors at either extreme of the quality spectrum (i.e., at very low and very high quality). Specifically, we see that even when the model predicts with extremely high confidence that a developer will accept a suggestion, that the acceptance rate is only 50%. This pattern of behavior introduces the importance of personal preferences in the code completion space. Specifically, some developers may be bothered or distracted by seeing a high number of these suggestions, even when they are high in quality, while other developers may not mind evaluating suggestions that do not meet a higher threshold for being accepted. On the other end of the quality spectrum, developers are being shown suggestions that are *predicted* to be unlikely to be accepted. In both cases, we should consider delivering more control to developers that allow them to control how often they are exposed to these suggestions and, potentially, let them define how regularly they wish to see suggestions of various quality. Furthermore, similar to prior work estimating trust in AI output, there is an opportunity here to provide information within the IDE that shows the predicted accuracy (or model confidence) in the AI-generated code suggestion.

The three additional characteristics of the code completion suggestion were also significant predictors of whether or not a developer would accept a suggestion. Together these findings replicate and extend the importance of features of the suggestion itself seen in prior work [18]. Additionally, these findings suggest that there are certain places within a line of code or a file that developers are less susceptible to AI generated code completions. One hypothesis for the pattern exhibited for character position in a line revolves around the idea that a developer may be planning a few steps ahead of what they are currently typing, and once the mental work is done, the benefit of accepting a suggestion is decreased. On the one hand, these features could be tuned by teams that train code completion models in an attempt to optimize against whether developers accept a suggestion. On the other hand, these features may introduce an interesting opportunity for personalization and user controls. Users settings could dictate the length, position in a line, or where in a file developers wish to see code suggestion generated by AI.

A number of effects suggested expertise plays an interesting role in trusting AI-generated code completions. The strongest positive predictor of accepting a multi-line code suggestion was whether or not the developer had been awarded readability in the language of the suggestion. We also saw that recent usage of a language had a positive impact on accepting suggestions. These effects replicate the results of our preliminary study using quantitative data at scale. However, the two features we generated to measure expertise with the code being generated (both in terms of experience with the codebase and experience with the specific file being edited) did not reveal significant relationships with accepting a suggestion. We hypothesize that the disconnect between language-based effects and codebase effects may stem directly from what AI-generated code completion models aim to achieve (i.e., to best predict what code comes next, rather than codebase-specific outcomes). It is also worth noting that our operationalization of codebase expertise was generated based on data availability, but could likely be defined in many other ways. Again, with an eye towards designing these systems for trust, there appears to be an opportunity to leverage information about the developer's expertise in a given language when surfacing suggestions. It is possible that this finding could extend to other types of expertise where we would anticipate the

AI-generated material to be more aligned with the expertise being considered.

A number of features that were related to the scope of the work being completed were significant predictors of accepting a suggestion. Developers working on smaller code changes were less likely to accept multi-line completion suggestions and became more likely to accept them as the size of the change grew. One hypothesis for these results is that smaller changes are more straightforward to implement, which leads developers to have clearer ideas on the functionality they are adding, making multi-line suggestions less impactful. One of the strongest negative predictors of accepting a suggestion was whether or not the developer was editing a test file. Similar to the influence of smaller scopes, test code may be more constrained than feature code, making multi-line code suggestions less valuable. Another possible explanation for this pattern is that there may be misalignment between what the AI is attempting to complete and what the developer is hoping to achieve when writing tests. Code suggestions may ignore *local* codebase specific practices and attempt to provide suggestions that leverage more *global* best practices. More research is needed to understand why developers are less likely to accept code completion suggestions when writing tests, especially given the growing in interest in having generative AI tooling write tests for developers [1]. Similar to what we have described above, designing a system that allows developers to indicate the scope of what they are working on or how important more local practices are to the eventual goal of the code may help reduce the number of suggestions that a developer sees in contexts where they are unlikely to accept them.

There were two effects that did not replicate our initial study. First, the priority of the work being completed, as captured by our issue tracking software, was not a significant predictor of accepting a suggestion. We hypothesized that higher priority work is more critical work and therefore should be related to how developers interact with AI-generated code. However, similar to the our discussion of codebase familiarity, we acknowledge that this definition may not fully capture the criticality of the work. For example, a developer may be working on a top priority new feature that does not have an urgent deadline. Future work should continue to explore how task urgency and/or criticality influences interactions with AI-powered developer tools. Second, developers described their familiarity with these tools are being highly important in their interactions with them. However, the effects in our quantitative study were quite small, suggesting that while this relationship may exist, it may not be as important as initially described. Additionally, we saw that familiarity (as measured by exposures to code completion suggestions) did not always have a positive impact. Specifically, seeing more suggestions on a given day spread across a wider number of files was potentially associated with decreases in accepting a suggestion. This finding may suggest that developers may be interested in controlling when and where (in terms of files) that AI-generated code is shown. From a design perspective, this file-level settings could be surfaced or simply an option to turn these suggestions on or off depending on the file being edited.

6.1 Quality, Familiarity, and Control

When taken together, our results indicate that the relationships between trust in AI-powered developer tooling and characteristics of the developer and development task are multi-faceted and depend on a number of features. Quality of suggestions was the most important for predicting the acceptance. This proposes that as quality increases, developers may be more willing to cede control to these systems when the context is appropriate. Critically, we are still learning what those contexts are and in the short-term should consider delivering more control to developers that allow them to configure how often they are exposed to suggestions that are lower quality or do not match their current goals (e.g., seeing suggestions when working on smaller scoped tasks). When considering the role of familiarity, we hypothesize that exposure to these tools may be largely beneficial early after a developer begins interacting with them, but once they tune their expectations, aspects of expertise and control become more relevant. Our interview study was conducted when some of these technologies were first being introduced to developers, which would support this notion. Alternatively, it is possible that attitudes regarding familiarity are over-estimated relative to the behavioral impact. Collectively, these results give insights into opportunities to improve trust while highlighting areas that may require additional user research to better understand how developers might interact with future AI offerings.

7 LIMITATIONS

This research was conducted at Google which has specific developer tooling and best practices. It is possible that some of these findings would not generalize to other development contexts and with other populations (e.g., student developers). That said, this setting allowed for this research to incorporate signals of the developer into these analyses, which may have been previously unexplored in AI-generated code completion studies.

The data leveraged in our analysis, specifically the features that were generated for modeling, were largely limited by the preliminary study and the existing literature. Due to this, our model may be missing a number of factors that could influence our interpretations of the results. A wider scope could have been adopted, however, we decided to place an emphasize on signals that developers described when discussing trust in these technologies rather than including a wider array of metrics for the sake of exploring hypotheses. That said, future work in this space should continue to evolve and expand model inputs. Additionally, we leveraged the acceptance of AI-generated code as a definition of trust, which was aligned with prior definitions of the construct, but may not tell a comprehensive story about this behavior. For example, a developer may accept a suggestion and then immediately delete it from the file they are working on. We did not explore the persistence of AI-generated code, but propose that future research should explore whether the features investigated in this research influence the persistence of code, both in the short-term (i.e., before the code is submitted) and in the longer-term (i.e., after it is submitted, does it persist in the codebase?).

Additionally, despite covering a fairly large number of inputs, these results are strictly correlational in nature. Future studies may work towards generating causal estimates of these effects in order to establish where in the developer workflow we can intervene to influence developer trust in AI-powered tools. We anticipate that the current work provides a starting point for these efforts.

Finally, our study focused solely on multi-line code completions. Code completion suggestions also exist as single-token or single-line completions, but were not investigated in the current research. It is an open question which features described in this research carryover to impact single-line suggestions.

8 FUTURE WORK

8.1 Evaluating Design and Model Changes

With this approach in place, future work is proposed to evaluate the impact of model updates and design changes in terms of trust measurements. This research has proposed a number of possible design updates that place more control into the hands of developers that is anticipated to lead to higher acceptance rates and, by extension, increases in developer productivity. We propose that design changes in this space should pay attention to how acceptance rates change, whether the same predictors described in the current work continue to show similar relationships, and, critically, how developers' sentiments and descriptions of trust are shaped by these changes.

8.2 Looking at AI-generated Code Persistence

As described in the discussion section, the current work placed an emphasis on accepting a code completion suggestion, but did not evaluate whether code that was accepted persisted through to being submitted. One potential pattern of behavior would be that a developer accepted a code completion suggestion only to heavily edit this code and move on. Future work will be designed to investigate how often this behavior occurs and if the features described in our current analysis shape the whether or not code generated by AI persists.

8.3 Expanding to Other AI-powered Tools

The current work focused specifically on code completion behavior, but AI-powered tools exist in many different parts of the development workflow. For example, developers are using these tools to gather information via chat interactions and to generate unit tests, to name a few. Trust is proposed to play a role in how developers choose to engage with these tools and how they incorporate the information they receive from these tools into their work. The current research has started to identify what developers consider when they think about trusting AI-powered tools and developing features that quantitatively capture these components. We propose that future research should continue to build on this approach for other interactions between developers and AI-powered tools.

9 CONCLUSION

Developing trust in AI is a critical precursor to unlocking anticipated productivity gains of adopting and using this technology, eventually transforming the field of software development. In this work, we take a mixed-methods approach to measuring the factors that influence developers trust in AI-powered code completion. We present evidence that features about the developer (e.g., their expertise) and the development context (e.g., fixing a bug) shape their levels of trust in AI-generated code. Based on the findings we highlight the importance of personalization in AI powered developer tools, particularly when considering future designs of these tools. Allowing developers to have more control over when and what suggestions they are shown will enable them to customize their experience based on their personal preferences and get the most out of AI offerings.

ACKNOWLEDGMENTS

We would like to thank Micheal Bachman, Andrew Macvean, Maxim Tabachnyk, and the Engineering Productivity Research team for their valuable feedback.

REFERENCES

[1] Nadia Alshahwan, Jubin Chheda, Anastasia Finegenova, Beliz Gokkaya, Mark Harman, Inna Harper, Alexandru Marginean, Shubho Sengupta, and Eddy Wang. 2024. Automated Unit Test Improvement using Large Language Models at Meta. arXiv:cs.SE/2402.09171

[2] Amazon. 2024. ML-powered coding companion for developers - Amazon CodeWhisperer Features. https://aws.amazon.com/codewhisperer/features/

[3] Matin Amoozadeh, David Daniels, Daye Nam, Aayush Kumar, Stella Chen, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Trust in Generative AI among students: An exploratory study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 67–73.

[4] Johannes Bader, Sonia Seohyun Kim, Frank Sifei Luan, Satish Chandra, and Erik Meijer. 2021. AI in Software Engineering at Facebook. *IEEE Software* 38, 4 (2021), 52–61. https://doi.org/10.1109/MS.2021.3061664

[5] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2022. Grounded copilot: How programmers interact with code-generating models. *arXiv preprint arXiv:2206.15000* (2022).

[6] Yoav Benjamini and Daniel Yekutieli. 2001. The control of the false discovery rate in multiple testing under dependency. *The Annals of Statistics* 29, 4 (2001), 1165 – 1188. https://doi.org/10.1214/aos/1013699998

[7] Alexia Cambon, Brent Hecht, Ben Edelman, Donald Ngwe, Sonia Jaffe, Amy Heger, Mihaela Vorvoreanu, Sida Peng, Jake Hofman, Alex Farach, et al. 2023. *Early LLM-based Tools for Enterprise Information Workers Likely Provide Meaningful Boosts to Productivity.* Technical Report. MSFT Technical Report. https://www.microsoft. com/en-us/research ....

[8] Omer Dunay, Daniel Cheng, Adam Tait, Parth Thakkar, Peter C Rigby, Andy Chiu, Imad Ahmad, Arun Ganesan, Chandra Maddila, Vijayaraghavan Murali, et al. 2024. Multi-line AI-assisted Code Authoring. *arXiv preprint arXiv:2402.04141* (2024).

[9] Mary T Dzindolet, Scott A Peterson, Regina A Pomranky, Linda G Pierce, and Hall P Beck. 2003. The role of trust in automation reliance. *International journal of human-computer studies* 58, 6 (2003), 697–718.

[10] Nat Friedman. 2021. Introducing github copilot: Your AI pair programmer. https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/

[11] Google. 2024. AI-assisted application development. https://cloud.google.com/duet-ai

[12] Brittany Johnson, Christian Bird, Denae Ford, Nicole Forsgren, and Thomas Zimmermann. 2023. Make Your Tools Sparkle with Trust: The PICSE Framework for Trust in Software Tools. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 409–419.

[13] Michelle E Kiger and Lara Varpio. 2020. Thematic analysis of qualitative data: AMEE Guide No. 131. *Medical teacher* 42, 8 (2020), 846–854.

[14] Vivian Lai and Chenhao Tan. 2019. On human predictions with explanations and predictions of machine learning models: A case study on deception detection. In *Proceedings of the conference on fairness, accountability, and transparency*. 29–38.

[15] John D Lee and Katrina A See. 2004. Trust in automation: Designing for appropriate reliance. *Human factors* 46, 1 (2004), 50–80.

[16] Roger C Mayer, James H Davis, and F David Schoorman. 1995. An integrative model of organizational trust. *Academy of management review* 20, 3 (1995), 709–734.

[17] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2022. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. *arXiv preprint arXiv:2210.14306* (2022).

[18] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2023. When to Show a Suggestion? Integrating Human Feedback in AI-Assisted Programming. arXiv:cs.HC/2306.04930

[19] Emerson Murphy-Hill, Chris Parnin, and Andrew P Black. 2011. How we refactor, and how we know it. *IEEE Transactions on Software Engineering* 38, 1 (2011),

5–18.

[20] John O'Donovan and Barry Smyth. 2005. Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*. 167–174.

[21] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the keyboard? assessing the security of github copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 754–768.

[22] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do users write more insecure code with AI assistants?. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2785–2799.

[23] Crystal Qian and James Wexler. 2024. Take It, Leave It, or Fix It: Measuring Productivity and Trust in Human-AI Collaboration. *arXiv preprint arXiv:2402.18498* (2024).

[24] Maxim Tabachnyk and Stoyan Nikolov. 2022. ML-enhanced code completion improves developer productivity. https://ai.googleblog.com/2022/07/ml-enhanced-code-completion-improves.html

[25] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–7.

[26] Ming Yin, Jennifer Wortman Vaughan, and Hanna Wallach. 2019. Understanding the effect of accuracy on trust in machine learning models. In *Proceedings of the 2019 chi conference on human factors in computing systems*. 1–12.

[27] Yunfeng Zhang, Q. Vera Liao, and Rachel K. E. Bellamy. 2020. Effect of confidence and explanation on accuracy and trust calibration in AI-assisted decision making. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency (FAT* '20)*. Association for Computing Machinery, New York, NY, USA, 295–305. https://doi.org/10.1145/3351095.3372852

[28] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 21–29.

Received 2024-04-05; accepted 2024-05-04