The Impact of Generative AI on Code Expertise Models: An Exploratory Study

Otávio Cury
Computer Science Department, Federal University of Piauí
Teresina, Brazil
otaviocury@ufpi.edu.br

Guilherme Avelino
Computer Science Department, Federal University of Piauí
Teresina, Brazil
gaa@ufpi.edu.br

ABSTRACT

Generative Artificial Intelligence (GenAI) tools for source code generation have significantly boosted productivity in software development. However, they also raise concerns, particularly the risk that developers may rely heavily on these tools, reducing their understanding of the generated code. We hypothesize that this loss of understanding may be reflected in source code knowledge models, which are used to identify developer expertise. In this work, we present an exploratory analysis of how a knowledge model and a Truck Factor algorithm built upon it can be affected by GenAI usage. To investigate this, we collected statistical data on the integration of ChatGPT-generated code into GitHub projects and simulated various scenarios by adjusting the degree of GenAI contribution. Our findings reveal that most scenarios led to measurable impacts, indicating the sensitivity of current expertise metrics. This suggests that as GenAI becomes more integrated into development workflows, the reliability of such metrics may decrease.

KEYWORDS

generative artificial intelligence, source code expertise, truck factor, mining software repository

1 Introduction

In recent years, Generative Artificial Intelligence (GenAI) has attracted considerable attention from the Software Engineering community, revealing research gaps across multiple areas [29]. This growing interest is justified by GenAI's potential to transform a range of domains [20]. In software development, the increasing integration of GenAI into development environments has brought numerous benefits. Tools such as OpenAI's ChatGPT[1] and GitHub Copilot[2] have contributed to increased productivity and enhanced software quality [15, 40]. Despite these advantages, researchers have raised concerns related to adopting such technologies [16].

One significant drawback is the risk of developer over-reliance on GenAI tools, where users may accept generated code without fully understanding it [38, 39]. Such behavior can undermine problem-solving skills and hinder a deeper comprehension of the codebase [32]. Given this, we argue that *models* used to estimate developer expertise should reflect the potential impact of GenAI-assisted code generation on code understanding. Typically based on authorship data of *Version Control Systems*, these models estimate a developer's knowledge of the code and are applied across various stages of the software development [13, 22]. They also play a key role in knowledge loss mitigation, being used in metrics such as the *Truck Factor*, which estimates the concentration of knowledge within a development team, a well-established line of research [5, 12, 25].

Building on this premise, this paper investigates how developers' use of GenAI tools may impact source code expertise identification models. For this investigation, we searched for ChatGPT shared links embedded in source code files from GitHub projects. Using these links, we retrieved the corresponding conversations and identified the extent to which GenAI-generated code had been integrated into the project files. Based on statistical evidence of this integration, we simulated different scenarios to assess the potential impact on expertise identification models and a widely used Truck Factor algorithm, by attributing a portion of developer authorship to GenAI. This study is guided by the following overarching question: **How does the use of GenAI tools for source code generation affect the accuracy and reliability of models that identify developer expertise in source code?**

The key contributions of this study are as follows: *1)* Quantitative insights into how code generated by ChatGPT is integrated into open-source projects; *2)* A comprehensive analysis of how this integration may impact expertise models and their applications, such as knowledge concentration metrics; *3)* A publicly available dataset containing information on the integration of GenAI-generated code in open-source repositories. This work is organized as follows: Section 2 introduces key concepts. Section 3 reviews related studies on the relationship between GenAI tools and source code knowledge. Section 4 describes the data collection process used in this study. Section 5 presents the results, Section 6 discusses them, and Section 7 concludes the paper.

2 Background

2.1 Code Knowledge Models

Source code knowledge models are designed to identify developer expertise within software projects. This information supports a range of activities, including task assignment and bug fixing, and also assists project managers in monitoring knowledge concentration within the codebase [17, 34]. Most existing approaches rely on development history stored in *Version Control Systems*. For example, the *Degree of Knowledge* model, proposed by Fritz et al., combines two types of information: the developer's authorship of a file captured by the *Degree of Authorship* (DOA), and the number of interactions the developer has had with that file [18].

2.1.1 Degree of Expertise. In a previous study, we proposed the *Degree of Expertise* (DOE), a knowledge model that uses four variables from development history to measure a developer's knowledge on a source code file [13]. Unlike existing models in the literature, *DOE* combines fine-grained measures of change, authorship, recency of

[1] https://chatgpt.com
[2] https://github.com/features/copilot

modification, and file size to achieve greater precision in knowledge estimation. The *DOE* model showed superior performance in identifying file experts [13], and was applied within a Truck Factor algorithm [10, 12]. The *DOE* of a developer d in version v of file f is calculated using Equation 1.

$$\begin{aligned}\textbf{DOE(d, f(v))} = {} & 5.28223 + 0.23173 \cdot \ln(1 + \textbf{Adds}^{d,f(v)}) \\ & + 0.36151 \cdot (\textbf{FA}^{f}) - 0.28761 \cdot \ln(\textbf{Size}^{f(v)}) \\ & - 0.19421 \cdot \ln(1 + \textbf{NumDays}^{d,f(v)}) \end{aligned} \quad (1)$$

Where, **Adds**: number of lines added by developers d on file f; **FA**: boolean if developer d is the creator of the file f; **Size**: number of lines of code (LOC) of the file f; **NumDays**: number of days since the last commit of a developer d on file f. In this work, we examine how the integration of GenAI-generated code impacts this knowledge model. The rationale behind selecting this particular model is further detailed on Section 4.

2.2 Truck Factor Algorithm

The *Truck Factor* measures how many developers would need to leave a software project for it to be critically affected [25]. This metric helps identify knowledge concentration risks and has been widely studied in the literature [4, 12, 17, 25]. Among the proposed methods for estimating Truck Factor, Avelino's algorithm [4] is notable for its strong performance in comparison studies [17] and for being adopted in subsequent validation studies [2, 9].

Avelino's algorithm estimates the Truck Factor using a strategy based on developer authorship. It identifies file experts through the *Degree of Authorship* (DOA) model and iteratively removes the developer who is the expert for the largest number of files. After each removal, the algorithm checks how many files are left without any expert. This process continues until more than half of the project's files are considered abandoned. The number of developers removed at that point is returned as the *Truck Factor*. The pseudo-code for this algorithm is provided in the original study [4].

In this study, we use the implementation provided by previous work [10, 12], which modifies the original Truck Factor algorithm by replacing the *DOA* model with the *DOE* model (Section 2.1.1) for expert identification. This adaptation enables us to investigate how the influence of GenAI-generated code on knowledge models affects knowledge concentration metrics, such as the Truck Factor.

3 Related Works

Some researchers have raised concerns about using Generative Artificial Intelligence (GenAI) to generate source code that might impact programming knowledge. Denny et. al also cite *learner over-reliance*, trusting in the generated code without fully understanding it—as a key risk in student learning [14].

Students also share this concern. Yilmaz et al. analyzed ChatGPT's limitations from the perspective of undergraduates and found that one common concern was its potential to impair algorithmic thinking skills [38]. Similarly, Ma et al. reported that students in a Python course feared ChatGPT could hinder their learning process [28]. Prather et al. interviewed students using Copilot in a programming class, and several participants expressed concern that relying on the tool might prevent them from fully understanding the code [32]. Among practitioners, similar concerns were observed. Russo reported that some engineers fear over-reliance on GenAI tools may reduce comprehension of the code they integrate [35].

Some studies have employed methodologies similar to ours. Grewal et al. analyzed ChatGPT shared conversation links to investigate how generated code is integrated into software projects [21]. Using the *Levenshtein Distance*, they measured the degree to which code from these conversations was incorporated into GitHub repositories. Similarly, Jin et al. examined the effectiveness of ChatGPT in assisting developers with code generation [26]. Applying a comparable approach, they found that in 16.8% of conversations, the generated code snippets had exact matches in the main branches of the analyzed projects, allowing for minor modifications.

Although we did not identify studies addressing the relationship between GenAI and *Knowledge Models*, the existing literature highlights a research gap. It suggests that using GenAI tools for source code generation may lead to developers not fully comprehending the code generated and integrated into their projects.

4 Study Design

Our study design consists of four main steps. First, we select source code files that contain ChatGPT links. Then, we mine the development history of these files and retrieve the code generated in the associated conversations with ChatGPT. By combining these two sources of information, we identify the extent to which code generated by Generative Artificial Intelligence (GenAI) was integrated into the files. Finally, we statistically analyze these integrations. Based on these insights, we simulate different scenarios to assess the impact of GenAI on knowledge models by attributing a portion of the authorship to GenAI rather than to developers. Figure 1 provides an overview of the study design.

4.1 Selecting Files with ChatGPT Shared Links

In May 2023, OpenAI introduced a feature that allows users to share their conversations with ChatGPT via shareable links[3]. This feature enables developers to share their chats with ChatGPT, containing specific solutions integrated into the source code. These shared links appear in GitHub artifacts, and they have already been extracted in related studies [21, 23, 26, 37]. Inspired by the study of Xiao et. al [37], in this study, we construct a dataset focused on shared links within GitHub source code files. We chose the *source code files* artifact because it is more closely related to the knowledge metrics used, facilitating the analysis.

For our initial search, we used the GitHub REST API[4] to look for keywords that indicate the presence of ChatGPT shared links within the contents of the source code file. The following endpoint was used to locate these chat links in the source code: *https://api.github.com/search/code?q=#chatgpt_url+language:#language*. We searched for links using two keywords (#chatgpt_url). The first, *https://chat.openai.com/share/*, was originally introduced by OpenAI [37]. The second, *https://chatgpt.com/share/*, emerged in 2024. To

[3]https://help.openai.com/en/articles/7925741-chatgpt-shared-links-faq
[4]https://docs.github.com/en/rest?apiVersion=2022-11-28

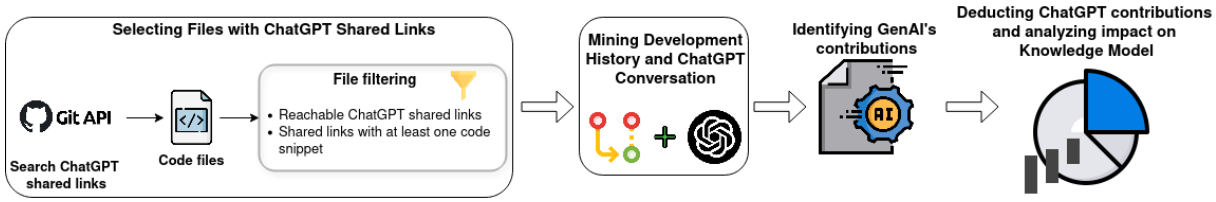

Figure 1: Overview of the methodology used to assess the impact of GenAI on a knowledge model.

focus on source code files, we applied a programming language filter (#language), targeting the 10 most popular languages on GitHub in 2022[5]: JavaScript, Python, Java, TypeScript, C#, C++, PHP, Shell, C, and Ruby. This search, conducted in November 2024, identified 2579 files with shared links. Due to the limitations of the API, which do not allow regex searches, we manually filtered the results to identify artifacts containing links that exactly match the shared link pattern using the following regular expressions: *^https://chat.openai.com/share/[a-zA-Z0-9-]{36}$*, and *^https://chatgpt.com/share/[a-zA-Z0-9-]{36}$*. After applying these to the file contents, we refined our dataset to 2502 source code files from 1354 repositories, totaling 2036 shared links.

Following the initial search and regex filtering, we applied two additional filters. First, since shared links can be disabled after being created[6], we checked the reachability of each valid shared link by fetching the page content and checking for errors. Second, we verified that each successful fetch contained at least one code snippet from ChatGPT. We found 198 shared links without code snippets and 172 disabled links. As a result, our dataset was reduced to 2094 files in 1135 repositories, containing 1666 shared links. The number of source code files and shared links per programming language before and after applying these filters is shown in Table 1.

Table 1: Number of source code files and shared links per programming language before and after applying the filters.

	Before Filtering		After Filtering	
Language	**Files**	**Shared Links**	**Files**	**Shared Links**
Python	988	803	781	629
JavaScript	542	483	476	414
C++	251	195	223	169
Java	247	183	210	146
TypeScript	178	134	146	108
Shell	100	90	88	79
C#	80	62	69	51
C	68	63	59	53
PHP	45	27	41	22
Ruby	3	4	1	1

The results show a significantly higher number of links in Python files, 60% more than in the second most common language, JavaScript. This finding aligns with Xiao [37]. Additionally, Python and JavaScript are among the languages whose users most frequently utilize ChatGPT and Copilot [31, 40].

After applying these filters, the distribution of the number of files with shared links per repository has a first quartile (Q1) of 1, a median (Q2) of 1, and a third quartile (Q3) of 1. Similarly, the shared link distribution per repository has Q1 = 1, Q2 = 1, and Q3 = 2. These statistics indicate sparse distributions, where most repositories contain only one file with shared links, and most of these files contain only a single shared link.

4.2 Extracting Development History and GenAI Conversations

After filtering all relevant links and associated files, we constructed a dataset to analyze how the solutions provided in the links were integrated into their respective files. This step allowed us to assess the extent to which the generated code was adopted and how much of the developers' contributions could be attributed to ChatGPT.

Using the successfully fetched shared links described in Section 4.1, we first extracted the developers' prompts with the corresponding ChatGPT responses that included generated code snippets. We then cloned 1,135 repositories to collect the development history of the files containing the shared links. The vast majority of these repositories were relatively small, with a median of 1 contributor (Q1 = 1, Q3 = 2), 24 commits, and 34 files.

4.3 Identifying GenAI's contributions

In addition to mining data to calculate the author's contributions to the files that contain the links, we also needed to assess how much of the code from each link was integrated into the file. Following the study by Grewal et al. [21], we identified *matched lines* between code snippets and lines added in the same commit as the shared link. While their study used *Levenshtein Distance* for fuzzy matching, we take a more conservative approach and consider *only exact matches*. Our analysis is based on two assumptions, explained below with their supporting rationale.

> ***Assumption 1:*** *The exact matching lines identified correspond to instances of code directly copied and pasted from ChatGPT-generated outputs.*

Rationale: Since there is no known direct integration of ChatGPT with development environments, we consider lines that match the generated code as instances of *copying*. In the context of Copilot, matched lines are often referred to in research as *accepted* lines. A previous study by Dohmke found that developer satisfaction and productivity increase as the acceptance rate rises [15]. With

[5]https://octoverse.github.com/2022/top-programming-languages
[6]https://help.openai.com/en/articles/7925741-chatgpt-shared-links-faq

the growing use of ChatGPT for code generation [8, 30], we can assume similar trends, although this may reduce developers' autonomy over the code [7]. Moreover, research on Copilot has already shown an increase in copied and repeated code [24], while studies on ChatGPT indicate that more than 50% of generated code snippets are integrated without modifications [21].

> ***Assumption 2:*** *Developers do not acquire the same level of knowledge from integrating copied ChatGPT-generated code.*

Rationale: Passively accepting solutions may impact the understanding of integrated code [32] and create cognitive dissonance regarding one's problem-solving abilities, leading to an illusion of competence, as demonstrated by Prather's study with students [33]. We believe this effect should be reflected in knowledge models. Research suggests that actively retyping a solution enhances comprehension and learning [19, 36]. For example, active code retyping has been a key component of proposed methodologies for improving cognitive engagement with AI-generated code [27].

Building on these assumptions, we quantify the extent of copied code for each *file–shared link* pair. This provides statistical evidence of how ChatGPT-generated code is integrated into open-source projects and allows us to simulate different usage scenarios by attributing varying levels of authorship to GenAI.

4.4 Analyzing GenAI Conversations and Code Integration

Among the 2,235 file–shared link pairs selected and filtered (Section 4.1), 1,699 (76%) contained at least one matched line. However, manual inspection revealed that some of these matches consisted only of single-character overlaps, such as individual symbols or keys. To improve data quality, we applied an additional filter to retain only pairs with at least one matched line containing more than one character. After this refinement, 1,672 pairs remained (74%), and only these will be considered in the subsequent analysis.

With this filter applied, we primarily analyzed the distribution of the percentage of code copied from ChatGPT by developers and how this varies across programming languages. Additionally, we assessed the number of conversation turns present in these solutions, a characteristic frequently explored in related studies.

4.5 Impact Simulation Design

As demonstrated in the previous sections, the dataset we constructed is sparse regarding the link–file–repository relationship. In most cases, a single link is added in a single commit to a single file within a repository. As a result, a direct analysis of the impact of copied code from these links on files and repository history would be limited and might not reflect realistic usage scenarios. Therefore, drawing on the statistical findings from the real data (presented in Section 5.1), we performed a *simulated* analysis to assess the potential impact of copied code.

Instead of simulating the impact on the repositories containing shared links, most of which are small and have limited relevance, we focused on more prominent repositories. Specifically, we selected the five most-starred GitHub projects for each of the ten chosen programming languages. We excluded 14 non-software repositories (e.g., code collections and roadmaps) due to their limited relevance for knowledge concentration analysis, and 12 additional projects were excluded based on size, following methodologies from related studies [4, 13]. Ultimately, 24 repositories were selected, as shown in Table 4 (Section 5).

We simulated the impact by applying a *uniform code copy rate*, derived from the statistical analysis presented in Section 5.1. For this simulation, we adopted the Degree of Expertise (DOE) model for two primary reasons. First, *DOE* estimates a developer's expertise in a file partially based on the number of lines they have added, which enables us to discount lines attributed to GenAI. This level of granularity is not supported by models like Degree of Authorship (DOA), which rely on commit counts. Second, previous research has demonstrated that *DOE* outperforms other models in identifying developer expertise and in computing the Truck Factor of software projects [12].

We analyzed the impact of code copying on two key aspects within the scope of this work. First, we investigated how code copying affects the *DOE* values by evaluating the impact of excluding these contributions on the expertise scores of developers. Second, we examined how these changes influence the Truck Factor of the projects, analyzing alterations in the ranking of Truck Factor developers and changes in the Truck Factor values themselves.

For the Truck Factor analysis, we evaluated the impact by varying the percentage of files affected by code copying to 10%, 20%, 30%, 40%, and 50%. In each iteration, the affected files for each developer were randomly selected. Then, in every commit made by that developer on these files, we deducted the uniform code copy rate of the added lines, simulating GenAI authorship. Figure 2 illustrates the procedure. In each iteration, the Truck Factor was compared to the original results without GenAI impact. For the first analysis, which examines the impact on the expertise values calculated by the *DOE* model, we focus exclusively on the 50% impact scenario. This choice is justified by our interest in analyzing statistical metrics, such as the overall average impact on *DOE* values.

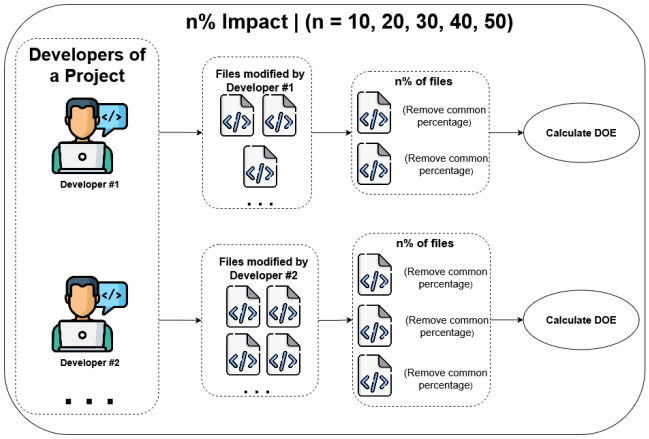

Figure 2: Overview of the approach used to simulate the impact of GenAI code copying across different usage scenarios.

5 Results

Following the study design described, we present the results in two parts. First, we provide a statistical analysis that characterizes the conversations with ChatGPT and the code integration. Second, we evaluate the impact of GenAI usage on developer expertise values, as calculated using the Degree of Expertise (DOE) model, followed by an assessment of its effect on the Truck Factors.

5.1 Code Integration Statistics

First, to characterize the conversations in the shared links, we analyzed the number of *turns* involved in the proposed solutions. A *turn* consists of a user prompt followed by a ChatGPT response, following the same terminology used by Hao et. al [23], for example. The distribution of the number of turns in the shared solutions has a first quartile (Q1) of 4, a median (Q2) of 8, a third quartile (Q3) of 16, and a mean of 14.29.

Second, we analyzed the distribution of the percentage of copied code. The distribution has a mean of 39%, with the first quartile (Q1) at 14%, the median (Q2) at 31%, and the third quartile (Q3) at 66%. Table 2 presents a breakdown of the mean copy percentage and the number of file–shared link pairs, by programming language. As there is only one Ruby file in the dataset, we excluded it from this analysis and the table.

As described in Section 4.5, the impact analysis requires a code copy rate to simulate the attribution of authorship to GenAI. For this purpose, in the following sections, we adopt the mean of the code copy rate distribution of 39% as the *uniform code copy rate*.

Table 2: Percentage of copied code by programming language.

Language	Percentage	Num
C	59%	53
C#	51%	59
Shell	44%	69
Java	42%	175
Python	41%	612
C++	38%	186
JavaScript	34%	373
TypeScript	33%	109
PHP	32%	35

5.2 Impact on Degree of Expertise

In this initial analysis, we focus exclusively on the 50% impact scenario, as described in Section 4.5. This choice is justified by our interest in the overall average impact on *DOE* values rather than individual cases explored in the Truck Factor analysis. Table 3 presents the distribution of *DOE* values across developer-file pairs affected by GenAI usage, both with and without code copying. The third column (**Difference**) shows the distribution of the differences between the *DOE* values in these two conditions.

The project-level analysis also did not reveal any significant differences. We calculated the mean difference in *DOE* values for each project with and without copied code. The standard deviation of these means was low (0.0017), indicating that the values were tightly clustered around the overall mean of 0.125. Similarly, when grouping projects by programming language, the mean differences also showed no notable variation.

Table 3: Quartile distribution of original DOE values, DOE values affected by code copying, and their differences.

	Original DOE	Copy Affected DOE	Difference
Q1	2.248	2.118	0.113
Q2	2.793	2.666	0.118
Q3	3.397	3.277	0.160
Mean	2.842	2.716	0.126

We also segmented the analysis by comparing the differences in *DOE* values between core and peripheral developers. Core developers, defined here as those identified in the Truck Factor, experienced smaller losses, with an average reduction of 0.124. In contrast, peripheral developers showed a slightly higher average loss of 0.126. A *Wilcoxon Signed-Rank Test* comparing the *DOE* differences between these two groups yielded a statistically significant result ($p < 0.005$), suggesting a systematic disparity in the impact.

We also applied the *Wilcoxon Signed-Rank Test* to the distribution of *DOE* variations within each project, finding a statistically significant difference from zero in all cases ($p < 0.005$). This suggests that, despite the small magnitude of individual changes, there is a consistent and systematic effect across projects. The following sections explore how these differences affect higher-level metrics.

5.3 Impact on Truck Factor

First, we present a Truck Factor analysis of the selected projects. The distribution of Truck Factor values shows a mean of 108.75 and a median of 17. Due to the presence of outliers, such as *torvalds/linux* and *ohmyzsh/ohmyzsh*, the median offers a more representative measure of central tendency. The original (unimpacted) Truck Factor values are listed in the **TF** column of Table 4.

To evaluate the impact of GenAI usage on these values, we consider two main aspects: (1) whether the Truck Factor value itself changes, and (2) whether the value remains stable but the developer ranking is affected. To quantify changes in developer rankings, we apply the Kendall Rank Correlation Coefficient, which measures the similarity between two ranked lists [1].

We computed 120 Truck Factor values, covering 24 projects under 5 scenarios. The results are shown in the **TF_*N*** columns, where *N* indicates the proportion of impacted files, as illustrated in Figure 2. Of these, 87 values (73%) changed, impacting 20 of the 24 projects. Among the 87 changes, 86 showed a reduction in the Truck Factor, with a median decrease of 2 developers. Only one case showed an increase, by a single developer.

Additionally, 85 Truck Factors (71%) exhibited differences in their ranking order across 21 projects. The distribution of τ (tau) values, representing ranking similarity, has a mean of 0.43, with the first quartile (Q1) at 0.26, the median (Q2) at 0.37, and the third quartile (Q3) at 0.60. Among the 21 affected projects, 12 began to show differences, either in the value of the Truck Factor or in the order, in the first 10% impact scenario. Seven projects exhibited changes

Table 4: Truck Factor values of the target repositories across different impact scenarios.

Repository	TF	TF_10	TF_20	TF_30	TF_40	TF_50
discourse/discourse	23	22	22	21	21	21
electron/electron	16	15	15	15	15	15
facebook/react	6	6	7	6	6	6
facebook/react-native	68	66	66	65	64	60
freeCodeCamp/ freeCodeCamp	15	14	13	13	13	13
godotengine/godot	58	56	55	56	52	52
huginn/huginn	7	6	7	6	6	6
jellyfin/jellyfin	13	13	12	11	11	11
laravel/framework	212	189	188	189	181	173
mastodon/mastodon	5	5	5	4	4	4
microsoft/PowerToys	21	20	20	19	19	18
microsoft/terminal	6	6	6	6	6	6
microsoft/vscode	18	17	17	17	16	16
netdata/netdata	3	3	3	3	3	3
ohmyzsh/ohmyzsh	845	632	615	649	630	641
PowerShell/ PowerShell	13	13	12	12	12	12
rails/rails	455	399	400	391	380	374
redis/redis	33	31	31	30	30	30
Significant-Gravitas/ AutoGPT	6	6	6	6	6	6
tensorflow/tensorflow	149	146	146	144	144	144
torvalds/linux	604	593	589	580	574	564
twbs/bootstrap	22	18	19	17	17	19
vercel/next.js	10	10	10	10	10	9
vuejs/vue	2	2	2	2	2	2

specifically at the 20% impact scenario, while one project showed changes only at the 30% scenario, and another at the 50% scenario.

We found a moderate positive correlation of $\rho = 0.41$ between the original Truck Factor size and the frequency of changes across scenarios. However, this does not mean that changes were exclusive to projects with higher Truck Factors. To investigate whether projects with lower Truck Factors were also affected, we selected those with a Truck Factor less than or equal to 7, approximately the first quartile (6.75) of the original Truck Factor distribution. This subset represents 35 (29%) of the 120 calculated values. Even within this group, 17 (49%) exhibited changes, either in their ranking order or in their Truck Factor value.

6 Discussion

Firstly, regarding the number of turns in the mined conversations, our results differ from those reported by Hao et al., where most conversations consisted of only a single turn [23]. A key difference lies in the datasets: our study focuses specifically on code generation that was at least partially integrated into real projects. A more comparable reference is the work by Jin et al., who reported an average of 10.4 turns per code generation conversation [26]. In contrast, our dataset shows a considerably higher average of 14.291 turns. However, the 100th percentile of our distribution is 326, indicating the presence of outliers. In this context, the median value of 8 is a more representative central tendency and aligns more closely with results from other studies.

The data on the percentage of copied code by programming language suggests that lower-level languages, such as *C* and *Shell*, exhibit the highest rates. In contrast, higher-level languages like *JavaScript* and *TypeScript* show lower percentages. This trend may be associated with the level of abstraction provided by each language, with higher-level languages tending to be more concise, potentially reducing the need for directly copying code snippets.

Regarding the impact analysis, the difference in developers' Degree of Expertise values within the files they contributed to was consistent but small, as shown in Section 5. This outcome is expected, given that the simulation removed 39% of only one variable in the *DOE* model, which is not the most significant for assessing knowledge, as discussed in Section 2. Among the stratifications conducted, none revealed a notable difference, except that core developers appeared to lose slightly less knowledge and were less affected overall. However, even these small changes were reflected in the Truck Factor calculations, impacting both high and low values, with a strong tendency toward decreasing the Truck Factor. In such scenarios, the algorithm tends to suggest a greater concentration of knowledge among key developers.

This finding demonstrates that the Truck Factor is sensitive to changes in developers' expertise, *raising concerns about its reliability when GenAI-generated code is involved*. However, this issue extends beyond the Truck Factor, as authorship metrics are widely employed in various software engineering tasks. For instance, measures such as *commits* and *lines of code* are used to assess technical expertise for recruitment [3], to identify appropriate code reviewers [22], and even as indicators of developer productivity [6]. We anticipate that all of these applications may be impacted as the integration of GenAI tools and automated code generation becomes more prevalent in software development.

7 Conclusion and Future Work

In this paper, we presented a study on the impact of using Generative Artificial Intelligence (GenAI) for code generation in a developer expertise identification model and a Truck Factor algorithm. Based on statistics regarding the percentage of code copied from ChatGPT, our simulations suggest that expertise identification models are sensitive to attribution loss. Although the quantitative reduction in expertise values is relatively small, applications such as the Truck Factor algorithm are still affected, even under scenarios of limited GenAI usage. These results highlight a potential lack of confidence in such metrics as GenAI tools become more prevalent in software development.

This is an exploratory study that opens several avenues for future research. We plan to investigate other knowledge models and Truck Factor algorithms using a similar methodology. Furthermore, we aim to explore developers' perceptions of this issue. To that end, we intend to survey to understand how developers integrate GenAI-generated code into their projects and how they perceive its impact on source code knowledge and expertise attribution.

ARTIFACT AVAILABILITY

We have made all artifacts from our study publicly available at https://doi.org/10.5281/zenodo.16969856 [11].

REFERENCES

[1] Hervé Abdi. 2007. The Kendall rank correlation coefficient. *Encyclopedia of measurement and statistics* 2 (2007), 508–510.

[2] Nuri Almarimi, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. 2021. csDetector: an open source tool for community smells detection. In *29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1560–1564.

[3] Daniel Atzberger, Nico Scordialo, Tim Cech, Willy Scheibel, Matthias Trapp, and Jürgen Döllner. 2022. CodeCV: Mining expertise of GitHub users from coding activities. In *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 143–147.

[4] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2016. A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE, 1–10.

[5] Guilherme Avelino, Marco Tulio Valente, and Andre Hora. [n. d.]. What is the Truck Factor of popular GitHub applications? A first assessment. *PeerJ PrePrints* 3 ([n. d.]), e1233v1.

[6] Moritz Beller, Amanda Park, Karim Nakad, Akshay Patel, Sarita Mohanty, Ford Garberson, Ian G Malone, Vaishali Garg, Henri Verroken, Andrew Kennedy, et al. 2025. What's DAT? Three Case Studies of Measuring Software Development Productivity at Meta With Diff Authoring Time. *arXiv preprint arXiv:2503.10977* (2025).

[7] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2023. Taking flight with copilot. *Commun. ACM* 66, 6 (2023), 56–62.

[8] Michelle Brachman, Amina El-Ashry, Casey Dugan, and Werner Geyer. 2025. Current and Future Use of Large Language Models for Knowledge Work. *arXiv preprint arXiv:2503.16774* (2025).

[9] Fabio Calefato, Marco Aurelio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. 2022. Will you come back to contribute? Investigating the inactivity of OSS core developers in GitHub. *Empirical Software Engineering* 27, 3 (2022), 1–41.

[10] Otávio Cury and Guilherme Avelino. 2024. Knowledge Islands: Visualizing Developers Knowledge Concentration. In *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, 789–795.

[11] Otávio Cury and Guilherme Avelino. 2025. The Impact of Generative AI on Code Expertise Models: An Exploratory Study. doi:10.5281/zenodo.16969856

[12] Otávio Cury, Guilherme Avelino, Pedro Santos Neto, Marco Túlio Valente, and Ricardo Britto. 2024. Source code expert identification: Models and application. *Information and Software Technology* (2024), 107445.

[13] Otávio Cury, Guilherme Avelino, Pedro Santos Neto, Ricardo Britto, and Marco Túlio Valente. 2022. Identifying source code file experts. In *16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 125–136.

[14] Paul Denny, James Prather, Brett A Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing education in the era of generative AI. *Commun. ACM* 67, 2 (2024), 56–67.

[15] Thomas Dohmke, Marco Iansiti, and Greg Richards. 2023. Sea change in software development: Economic and productivity analysis of the ai-powered developer lifecycle. *arXiv preprint arXiv:2306.15033* (2023).

[16] Neil A Ernst and Gabriele Bavota. 2022. Ai-driven development is here: Should you worry? *IEEE Software* 39, 2 (2022), 106–110.

[17] Mívian Ferreira, Marco Tulio Valente, and Kecia Ferreira. 2017. A comparison of three algorithms for computing truck factors. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 207–217.

[18] Thomas Fritz, Gail C Murphy, Emerson Murphy-Hill, Jingwen Ou, and Emily Hill. 2014. Degree-of-knowledge: Modeling a developer's knowledge of code. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 2 (2014), 1–42.

[19] Adam M Gaweda, Collin F Lynch, Nathan Seamon, Gabriel Silva de Oliveira, and Alay Deliwa. 2020. Typing exercises as interactive worked examples for deliberate practice in cs courses. In *Proceedings of the Twenty-Second Australasian Computing Education Conference*. 105–113.

[20] Roberto Gozalo-Brizuela and Eduardo C Garrido-Merchán. 2023. A survey of Generative AI Applications. *arXiv preprint arXiv:2306.02781* (2023).

[21] Balreet Grewal, Wentao Lu, Sarah Nadi, and Cor-Paul Bezemer. 2024. Analyzing Developer Use of ChatGPT Generated Code in Open Source GitHub Projects. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. IEEE, 157–161.

[22] Christoph Hannebauer, Michael Patalas, Sebastian Stünkel, and Volker Gruhn. 2016. Automatically recommending code reviewers based on their expertise: An empirical comparison. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 99–110.

[23] Huizi Hao, Kazi Amit Hasan, Hong Qin, Marcos Macedo, Yuan Tian, Steven HH Ding, and Ahmed E Hassan. 2024. An Empirical Study on Developers Shared Conversations with ChatGPT in GitHub Pull Requests and Issues. *arXiv preprint arXiv:2403.10468* (2024).

[24] William Harding. 2025. *AI Copilot Code Quality: Evaluating 2024's Increased Defect Rate via Code Quality Metrics*. White Paper. GitClear.

[25] Elgun Jabrayilzade, Mikhail Evtikhiev, Eray Tüzün, and Vladimir Kovalenko. 2022. Bus factor in practice. In *44th International Conference on Software Engineering: Software Engineering in Practice*. 97–106.

[26] Kailun Jin, Chung-Yu Wang, Hung Viet Pham, and Hadi Hemmati. 2024. Can ChatGPT Support Developers? An Empirical Evaluation of Large Language Models for Code Generation. *arXiv preprint arXiv:2402.11702* (2024).

[27] Majeed Kazemitabaar, Oliver Huang, Sangho Suh, Austin Z Henley, and Tovi Grossman. 2024. Exploring the Design Space of Cognitive Engagement Techniques with AI-Generated Code for Enhanced Learning. *arXiv preprint arXiv:2410.08922* (2024).

[28] Boxuan Ma, Li Chen, and Shin'ichi Konomi. 2024. Enhancing Programming Education with ChatGPT: A Case Study on Student Perceptions and Interactions in a Python Course. In *International Conference on Artificial Intelligence in Education*. Springer, 113–126.

[29] Anh Nguyen-Duc, Beatriz Cabrero-Daniel, Adam Przybylek, Chetan Arora, Dron Khanna, Tomas Herda, Usman Rafiq, Jorge Melegati, Eduardo Guerra, Kai-Kristian Kemell, et al. 2023. Generative Artificial Intelligence for Software Engineering–A Research Agenda. *arXiv preprint arXiv:2310.18648* (2023).

[30] Stack Overflow. 2024. Overflow: 2024 State of Development Survey. *https://survey.stackoverflow.co/2024/* (2024).

[31] Alan Peslak and Lisa Kovalchick. 2024. AI for coders: An analysis of the usage of ChatGPT and GitHub CoPilot. *Issues in Information Systems* 25, 4 (2024), 252–260.

[32] James Prather, Brent N Reeves, Paul Denny, Brett A Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Transactions on Computer-Human Interaction* 31, 1 (2023), 1–31.

[33] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The widening gap: The benefits and harms of generative ai for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*. 469–486.

[34] Martin P Robillard. 2021. Turnover-induced knowledge loss in practice. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1292–1302.

[35] Daniel Russo. 2024. Navigating the complexity of generative ai adoption in software engineering. *ACM Transactions on Software Engineering and Methodology* (2024).

[36] James Skripchuk, Neil Bennett, Jeffrey Zhang, Eric Li, and Thomas Price. 2023. Analysis of Novices' Web-Based Help-Seeking Behavior While Programming. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 945–951.

[37] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. Devgpt: Studying developer-chatgpt conversations. In *Proceedings of the 21st International Conference on Mining Software Repositories*. 227–230.

[38] Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. 2023. Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning. *Computers in Human Behavior: Artificial Humans* 1, 2 (2023), 100005.

[39] Beiqi Zhang, Peng Liang, Xiyu Zhou, Aakash Ahmad, and Muhammad Waseem. 2023. Demystifying practices, challenges and expected features of using github copilot. *arXiv preprint arXiv:2309.05687* (2023).

[40] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2024. Measuring GitHub Copilot's Impact on Productivity. *Commun. ACM* 67, 3 (2024), 54–63.