Structural Code Quality and Its Influence on AI Code Translation from Python to Java

4082

Abstract

As artificial intelligence becomes increasingly integrated into software development, AI code translation tools are being used to convert programs between programming languages to save time and reduce manual effort. While these tools have shown strong potential, translation accuracy remains inconsistent, and developers often must correct AI-generated outputs. This study examines whether the structural quality of source code influences the accuracy of AI code translation from Python to Java. The research question, “To what extent does the structural quality of code affect the accuracy of AI code translation from Python to Java?”, was investigated using an experimental method. Fifteen Python programming tasks were created, each written in two versions: a Clean version and a Messy version, resulting in thirty Python programs. All programs produced identical outputs prior to translation. Each program was translated into Java using three AI translators Claude AI, Google Gemini, and GitHub Copilot Chat. This resulted in a total of ninety Java translated programs. Translation accuracy was measured through compilation and five unit tests per program. A chi-square test of independence was used to analyze whether differences in translation success between Clean and Messy code were statistically significant. Results showed that Clean Python code consistently translated successfully across all three AI tools. Translation failures occurred only in Messy code and varied by translator. Statistical analysis revealed that structural code quality had a significant effect on translation accuracy for Google Gemini, while no statistically significant relationship was found for Claude AI. For GitHub Copilot Chat, all translations succeeded regardless of code quality, preventing statistical comparison. These findings suggest that structural code quality can influence AI code translation accuracy, but this effect is not consistent across all AI tools. As organizations increasingly rely on AI for software modernization, understanding when and how code structure impacts translation accuracy is essential for applying these tools effectively and responsibly.

Introduction

In computer science classes and real software teams, people switch between languages all the time. A student might write a small tool in Python and then need the same thing in Java for a class project. A team might keep an older Java service while adding new features in Python. AI code translators create a bridge between those worlds with one prompt. Recent progress shows that AlphaCode, an AI coding tool by DeepMind, competed against human programmers in ten coding contests. In the contests, the AlphaCode had been given problems which it had to solve by writing a working program. When the judges ran the official tests, its solutions passed many of the same tests as the human entries did, placing it around the top half of competitors (Li et al., 2022). Even though current AI code translation tools are advanced, translation is still not always accurate or complete, and developers often work with imperfect outputs. Interviews with 11 professional engineers report that they use AI translations often to save time, but they still have to edit the output because it can be inaccurate (Weisz et al., 2022). Although many studies explain how AI translates code and how to test it, there is not any direct evidence about whether the structural quality of the Python code affects the accuracy of translation to Java. This led to the research question “To what extent does the structural quality of code affect the accuracy of AI code translation from Python to Java?”.

Literature Review

AI Translation Processes

Research has shown that artificial intelligence can now translate programming languages in a similar way to how people use Google Translate for human languages. A study demonstrated that computers could be trained to understand code structure and rewrite it in another language without needing matched examples of the same program written in both languages (Roziere et

al., 2020). As this technology developed, researchers built larger and more advanced models which are now capable of learning from millions of code examples. Recent surveys explain that these large language models use a method called self-attention, which means the model looks at all parts of the code at once so it can remember the context and meaning while translating (Jiang et al., 2025). In code translation, context is especially important because one line of code often depends on variables, loops, or conditions that appear earlier in the program. If the model misunderstands one part of the structure, the final translated program may still look complete but behave differently from the original version. Other research found that giving the model only a few translation examples could still improve its performance, showing that newer techniques help AI learn faster with less data (Y. Chen et al., 2024). Additional work showed that creating extra and varied training examples, known as data augmentation, can further boost translation accuracy by teaching the AI to handle more code patterns (B. Chen et al., 2024). These studies show that strong code translation depends on understanding structure, context, and training strategies that give the model the right examples to learn from.

Unit Test Method

To measure how accurate AI code translations are, researchers rely on software testing methods that check whether the translated program still behaves correctly. One core challenge is the “oracle problem,” which means it can be hard to know the one right answer for every program, so reliable tests are needed to judge correctness (Barr et al., 2015). To handle cases where an exact expected output is unknown, metamorphic testing checks whether certain input changes lead to predictable output changes, which is useful for validating translations (Segura et al., 2016). Another approach is to use mutation-based analysis, which makes small faults in the translated code to see if the tests catch them and then turns that into a concrete score for

translation quality (Guizzo et al., 2023). Recent work shows that large language models can generate unit tests. These tests run most of the important parts of the program and include checks that confirm the results are correct, so you can use them to verify that the translated code still works (Schäfer et al., 2024). These methods provide a clear and practical way to test AI-translated code through measurable ways to determine “pass” or “fail” in code testing.

Code Categorization

To compare “clean” and “messy” code in a fair way, researchers use objective measures instead of opinions. One approach is a learned readability score that rates how easy code is to read, which gives a consistent number for comparison (Buse & Weimer, 2010). Another approach measures how complicated code is by counting its decision points, like how many if statements and loops it has, and shows that higher counts make programs harder for people to understand (Feitelson, 2023). Code smells are common warning patterns in code that signal design or structure problems, not actual bugs. Studies catalog these smells and provide tools to detect them, so files can be labeled by the issues they contain (Paiva et al., 2017). Experimental work also shows that clear indentation reduces the time it takes to read code, which supports using formatting as part of structural quality (Hanenberg et al., 2024). While these studies each focus on a different aspect of code quality, they all point to the same idea that structure and readability can be measured in clear and consistent ways. This creates clear baselines for building clean and messy versions of code for translation tests.

Real-World Applications of AI Code Translation

Developers use code translation tools in the real world, and how well these tools work affects how quickly teams can move software across languages. An evaluation of AI-assisted translation found that teams worked faster and made fewer mistakes with AI assistance, but

human review was still needed to catch logic issues and edge cases (Weisz et al., 2022). This shows that even when AI translation helps, it is not fully reliable on its own. A survey of professional developers reported that many projects mix multiple languages and that better support is needed to move code and ideas between them (Mayer et al., 2017). Because of this need, researchers have worked on building better translation tools. A study explains that a code translation system was built and tested to reuse common building blocks of programs, like classes and methods, across languages, and it helped teams reuse existing code instead of having to rewrite it from scratch (Schaub & Malloy, 2016). In scientific computing, large language models help move older programs written in outdated languages into modern languages so researchers can keep long-used algorithms in current workflows (Dhruv & Dubey, 2025). Across all of these studies, the demand for accurate code translation is clear, which is why understanding what affects translation accuracy matters.

Current Gap

Considering the body of knowledge on AI code translation and software testing, there is little research that has directly measured whether the structural quality of the Python input changes the accuracy of translation to Java. Specifically, my research will focus on a controlled comparison that keeps the tasks, prompt, and tests the same while changing only code structure. This control is important because code translation accuracy can be affected by many factors, such as the difficulty of the task, the exact wording of the prompt, or the strength of the testing method. By keeping these parts the same, my study can focus more directly on whether structure itself plays a role in the translation outcome. Although prior work explains how translators operate and how to test code, there have not been studies that analyze the specific structural choices included in my research, which are readable names, clear indentation, simple logic, and

comments for the clean version of code, and then the opposite of these features for the messy version. With the literature review in mind, it is clear that there is a gap in the body of knowledge on whether code structure by itself changes translation outcomes, which is what the research question **“**To what extent does the structural quality of code affect the accuracy of AI code translation from Python to Java?” will investigate.

Justification

This research adds valuable knowledge about when AI translators are most reliable for students and developers who need to move code between languages. AI tools are now used in classes and real projects, so translations need to be more accurate to avoid wasting time. If structure changes accuracy, then knowing which features of code raise or lower success will help people plan their workflow and avoid avoidable errors. The improper use of translation tools without awareness of code quality can lead to wasted time fixing outputs and can spread mistakes into shared code. These concerns have posed an ongoing problem for software development teams and computer science classes, so the topic should be investigated. The research question has led me to hypothesize that clean Python code will produce more accurate translations to Java than messy Python code, meaning translations from clean code are more likely to compile and pass all unit tests than translations from messy code.

Methodology

Experimental Method

The research question, “To what extent does the structural quality of code affect the accuracy of AI code translation from Python to Java?” was explored through an experimental method. This approach was chosen because it allowed one variable to be changed while keeping all others constant. The independent variable was the structural quality of Python code, and the

dependent variable was translation success. A translation was considered successful only if the translated Java code compiled and passed all five unit tests. The experimental approach was most appropriate because it allowed me to directly observe how changes in code structure affected the accuracy of AI translation. By creating both clean and messy versions of each task, I was able to test the relationship between structure and outcome under identical conditions.

Sample Collection

To begin my investigation, I first created fifteen Python programming tasks that represented different areas of basic coding, including strings, numbers, arrays, sorting, and logic. For each task, I wrote two Python programs: one Clean version and one Messy version. Both versions were designed to produce the same correct outputs before translation. This resulted in a total of thirty working Python programs that served as the sample for the study. Each program's structural quality was evaluated using two predefined checklists: one for Clean code and one for Messy code. The Clean checklist required proper indentation, clear and descriptive variable naming, helpful comments, straightforward and readable logic, and an overall organized structure. The Messy checklist required the opposite characteristics, including inconsistent or incorrect indentation, unclear variable naming, no comments, confusing or repetitive logic, and disorganized structure. A program was labeled Clean only if it met all Clean criteria and did not meet any Messy criteria. A program was labeled Messy only if it met all Messy criteria and did not meet any Clean criteria. Any program that did not clearly meet one checklist was revised until it fit its intended category. This revision process helped make the categories stronger because it prevented a program from being placed in the Clean or Messy group when it only partly matched the checklist. As a result, the final sample had clearer differences between the two structural quality levels. The digital tools used during this investigation included a personal

computer to create, organize, and store all program files. ChatGPT was used to assist in the creation of the original Python programs prior to translation. The free online CodeHS platform was used to run and test both the Python programs and the translated Java programs. AI translators Claude AI, Google Gemini, and GitHub Copilot Chat were used to convert Python programs into Java during the translation phase of the experiment. After all Python programs were created and categorized, I wrote five unit tests for each programming task. Each unit test included a specific input and its exact expected output. These unit tests were first run on all Python programs using CodeHS to confirm that both the Clean and Messy versions produced identical and correct outputs. This step ensured that any errors observed later in the Java programs would be caused by the translation process rather than the original Python code. Once all Python programs passed their five unit tests, I began the translation process. Each Python program was entered into the three AI translators one at a time using the same fixed translation prompt: “Translate this Python program into Java. Put everything in a single file. Use public class MyProgram as the top-level class name, and include a standard public static void main(String[] args) method as the entry point. Do not create any other public classes.” Each translation was completed in a new chat window to prevent context carry over between programs. The resulting Java code was then copied into a new CodeHS Java project labeled with the task name, code quality (Clean or Messy), and translator used. This procedure was repeated for all thirty Python programs across all three translators, resulting in ninety translated Java programs.

Data Collection

Once all translations were completed, I tested each Java program in CodeHS using the same five unit tests that were originally used for the Python versions. Each Java file was first

compiled. If the program failed to compile, it was immediately marked as unsuccessful. If the program compiled successfully, the five unit tests were executed one at a time, and the program's outputs were compared to the expected outputs. A translation was marked successful only if the program compiled and produced all five expected outputs correctly. If the program failed even one test, it was marked unsuccessful. For each trial, I recorded the task name, code type (Clean or Messy), translator used, compilation result, number of correct outputs, and final success status. To maintain control and fairness, the same translation prompt, test cases, and CodeHS environment were used for every trial. No manual edits or corrections were made to any translated code. The order in which programs were translated and tested was randomized to reduce bias. The only difference between trials was the structural quality of the original Python code.

Chi-Square Analysis

After all trials were completed, the collected data were organized by translator. For each translator, I created a 2 by 2 contingency table comparing code quality (Clean versus Messy) with translation outcome (Successful versus Unsuccessful). Translation success was defined as compiling and passing all five unit tests, while failure included either compilation errors or failing one or more tests. A chi-square test of independence was then conducted for each translator to determine whether there was a statistically significant relationship between code quality and translation success. The observed frequencies in each category were compared to the expected frequencies assuming no relationship between the variables. The chi-square value was calculated using the standard formula that sums the squared differences between observed and expected values divided by the expected values. The resulting chi-square value and corresponding p-value were used to determine statistical significance. A p-value less than 0.05

indicated that the difference in translation success between Clean and Messy code was unlikely due to chance. The results from all three translators were then compared to evaluate whether the effect of code quality on translation accuracy was consistent across different AI tools.

Ethics

This study did not require IRB approval because it did not involve human subjects or personal data. All programming tasks were originally written by me and later refined with the assistance of AI tools. The experiment analyzed only code behavior and AI-generated outputs, and no private or sensitive information was collected. Because the study involved only software and posed no risk to individuals, ethical approval was not required.

Findings

This study examined the relationship between structural code quality and AI translation outcomes. The independent variable was the structural quality of the Python code (Clean vs. Messy), and the dependent variable was translation outcome. The results were separated by AI translator because each tool produced different outcomes. This made it possible to see whether code quality affected all translators in the same way or only affected certain tools. Instead of combining all translations into one general result, organizing the findings by translator made the pattern clearer and showed where the failures actually occurred.

Claude AI

For Claude AI, 15 out of 15 Clean programs translated successfully. For Messy programs, 14 out of 15 translated successfully, while 1 program was unsuccessful. All observed failures for this translator occurred when the original Python code was Messy.

Google Gemini

For Google Gemini, 15 out of 15 Clean programs translated successfully. For Messy programs, 10 out of 15 translated successfully, while 5 programs were unsuccessful. All unsuccessful translations for this translator occurred in the Messy code condition.

GitHub Copilot Chat

For GitHub Copilot Chat, 15 out of 15 Clean programs translated successfully. Similarly, 15 out of 15 Messy programs translated successfully. No translation failures were observed for this translator under either code quality condition.

Summary of Observed Outcomes

Across all three AI translators, Clean Python code consistently resulted in successful translations, with no failures observed. Translation failures occurred only in Messy Python programs, and the frequency of these failures varied by translator. Google Gemini exhibited the highest number of unsuccessful translations, followed by Claude AI, while GitHub Copilot Chat showed no unsuccessful translations.

Analysis

This study set out to examine to what extent the structural quality of code affected the accuracy of AI code translation from Python to Java. Based on the results, this experiment provides a new understanding that structural code quality influences translation accuracy to a moderate extent, but this effect is not universal across all AI translators. To support this, the analysis performed on each AI translator showed how clean code consistently produced successful translations, and how statistical significance was only observed for one out of the three translators.

Clean Code Consistency

To further support this, I visualized the results by comparing the number of successful translations for Clean and Messy code across each AI translator. **Figure 1** (below) shows that Clean code programs consistently achieved full success, while Messy code performance varied depending on the translator.

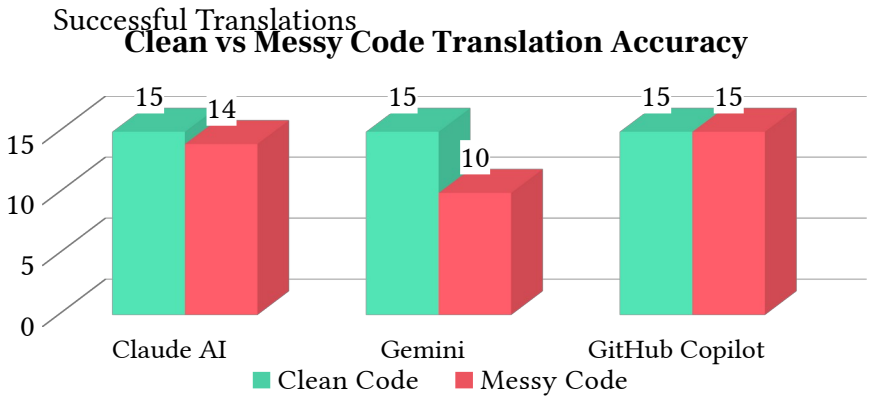

Figure 1

Statistical Significance

After the translation outcomes were recorded and organized, chi-square statistical analysis was conducted to determine whether the observed differences between Clean and Messy code translations were meaningful or due to chance.

Claude AI

For Claude AI, Clean code produced slightly more successful translations than Messy code. The chi-square test produced a chi-square value of 1.03, with a corresponding p-value of approximately 0.31. Because this p-value is greater than 0.05, the difference between Clean and Messy code is not statistically significant. This means that although Messy code had one failed translation while Clean code had none, the difference is small enough that it could reasonably have occurred by chance. Therefore, for Claude AI, the structure of the original Python code did not have a meaningful effect on translation success.

Google Gemini

For Google Gemini, Clean code produced more successful translations than Messy code. The chi-square test produced a chi-square value of 6.00, with a corresponding p-value of approximately 0.014. Because this p-value is less than 0.05, the difference between Clean and Messy code is statistically significant. This indicates that the higher number of failures observed in Messy code translations is unlikely to be due to chance. Therefore, for Google Gemini, the structure of the original Python code did have a meaningful effect on translation success.

GitHub Copilot Chat

For GitHub Copilot Chat, Clean and Messy code produced the same translation outcomes, with all programs translating successfully and no failures observed in either group. Because there was no variation in translation outcomes, a chi-square test of independence could not be performed. With all translations succeeding regardless of code quality, there was no difference to test for statistical significance. Therefore, for GitHub Copilot Chat, the results indicate that structural code quality did not affect translation success within this dataset.

Summary of Analysis

When comparing results across all three translators, my analysis shows that the effect of structural code quality on translation success was not consistent. Google Gemini demonstrated a statistically significant relationship between code quality and translation accuracy, while Claude AI showed no statistically significant relationship, and GitHub Copilot Chat showed no difference in outcomes due to a lack of variation. These differences suggest that AI translators vary in how sensitive they are to the structural quality of the input code. The results demonstrate that code quality may be one factor that affects translation accuracy, but it is not the only factor. The specific AI tool being used also matters, since each translator may handle unclear structure,

missing comments, or confusing logic in a different way. The original hypothesis stated that Clean Python code would result in more accurate translations to Java than Messy Python code. The analysis partially supports this hypothesis, because while Clean code consistently produced higher success rates across translators, statistical significance was only observed for Google Gemini. Therefore, the hypothesis was supported for one translator but was not supported across all AI tools tested.

Conclusions and Future Direction

Implications

The findings of this study have implications for developers, educators, and organizations that rely on AI code translation to modernize software systems. The results suggest that structural code quality can influence translation accuracy, particularly for certain AI translators. This implies that writing clean, well-structured code before translation may increase the chances of successful and accurate output when using some AI tools. For organizations working with legacy systems, these findings highlight the importance of code readability and structure as part of the modernization process, rather than relying solely on AI to correct poorly written code. Additionally, educators teaching programming and software engineering may emphasize clean coding practices not only for human readability, but also for improved compatibility with AI-based tools. Overall, this research suggests that AI code translation is not uniformly reliable across all tools, and that users should consider both the translator and the quality of the input code when applying these systems in real-world settings.

Limitations

This study contains several limitations that should be acknowledged. First, the sample size was limited to fifteen programming tasks, which may limit how broadly these findings can

be applied. The tasks were also shorter programs based on basic coding concepts, so they may not fully represent the complexity of larger software systems. Real-world programs often include multiple files, outside libraries, and more complicated dependencies, which could create additional translation challenges. A larger and more diverse set of programs could produce different results. Second, the study focused only on translation from Python to Java, meaning the results may not apply to other programming language pairs. Additionally, translation success was measured using compilation and unit testing, which evaluates functional correctness but does not account for code efficiency or readability. Another limitation is that only three AI translators were tested, all during a specific time frame. Because AI tools are frequently updated, future versions of these translators may produce different outcomes.

Future Direction

Future research could expand on this study in several ways. One direction would be to increase the number and complexity of programming tasks, including larger programs or real-world software modules. Using more complex programs would help determine whether the pattern found in this study becomes stronger, weaker, or stays the same when the code is closer to real software development. This would make future results more useful for developers who rely on AI translation in larger projects. Researchers could also examine additional programming language pairs, such as COBOL to Java or C++ to Python, to determine whether structural code quality has a similar effect across different translations. Another possible extension would be to analyze qualitative aspects of translated code, such as readability, efficiency, or maintainability, rather than focusing solely on functional correctness. Additionally, future studies could test a wider range of AI translation tools or examine how updates to existing models affect translation accuracy over time.

Final Remarks

As software systems continue to age and reliance on artificial intelligence increases, understanding the limits and conditions of AI code translation becomes increasingly important. This study contributes to that understanding by showing that structural code quality can influence translation accuracy, but that this effect is not consistent across all AI tools. As AI continues to play a growing role in software development and modernization, research that evaluates both its capabilities and limitations will be essential to ensuring reliable and secure technological infrastructure.

References

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The oracle problem in software testing: a survey. *IEEE Transactions on Software Engineering*, *41*(5), 507–525. https://doi.org/10.1109/tse.2014.2372785

Buse, R. P. L., & Weimer, W. R. (2010). Learning a metric for code readability. *IEEE Transactions on Software Engineering*, *36*(4), 546–558. https://doi.org/10.1109/tse.2009.70

Chen, B., Golebiowski, J., & Ziawasch Abedjan. (2024). Data augmentation for supervised code translation learning. *ACM Digital Library*, 444–456. https://doi.org/10.1145/3643991.3644923

Chen, Y., Yuan, S., Gu, X., Li, X., & Shen, B. (2024). Few-shot code translation via task-adapted prompt learning. *Journal of Systems and Software*, *212*, 112002. ScienceDirect. https://doi.org/10.1016/j.jss.2024.112002

Dhruv, A., & Dubey, A. (2025). Leveraging large language models for code translation and software development in scientific computing. *Proceedings of the Platform for Advanced Scientific Computing Conference*, 1–9. https://doi.org/10.1145/3732775.3733572

Feitelson, D. G. (2023). From code complexity metrics to program comprehension. *Communications of the ACM*, *66*(5), 52–61. https://doi.org/10.1145/3546576

Guizzo, G., Sarro, F., Harman, M., Zhang, J. M., & Treude, C. (2023). Mutation analysis for evaluating code translation. *Empirical Software Engineering*, *29*(1). https://doi.org/10.1007/s10664-023-10385-w

Hanenberg, S., Morzeck, J., & Gruhn, V. (2024). Indentation and reading time: A randomized control trial on the differences between generated indented and non-indented if-

statements. *Empirical Software Engineering*, *29*(5), 134. Springer Nature Link. https://doi.org/10.1007/s10664-024-10531-y

Jiang, J., Wang, F., Shen, J., Kim, S., & Kim, S. (2025, July 8). *A survey on large language models for code generation*. ACM Digital Library. https://dl.acm.org/doi/10.1145/3747588

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., & Molloy, J. (2022). Competition-level code generation with AlphaCode. *Science*, *378*(6624), 1092–1097. https://doi.org/10.1126/science.abq1158

Mayer, P., Kirsch, M., & Le, M. A. (2017). On multi-language software development, cross-language links and accompanying tools: A survey of professional software developers. *Journal of Software Engineering Research and Development*, *5*(1). https://doi.org/10.1186/s40411-017-0035-z

Paiva, T., Damasceno, A., Figueiredo, E., & Sant'Anna, C. (2017). On the evaluation of code smells and detection tools. *Journal of Software Engineering Research and Development*, *5*(1). Springer Open. https://doi.org/10.1186/s40411-017-0041-1

Roziere, B., Lachaux, M. A., Chanussot, L., & Lample, G. (2020). Unsupervised translation of programming languages. *ACM Digital Library*, 20601–20611. https://dl.acm.org/doi/10.5555/3495724.3497454

Schaub, S., & A. Malloy, B. (Eds.). (2016). The design and evaluation of an interoperable translation system for object-oriented software reuse. *JOT*, *15*(4), 1–33. AITO. https://doi.org/10.5381/jot.2016.15.4.a1.

Schäfer, M., Nadi, S., Eghbali, A., & Tip, F. (2024). An empirical evaluation of using large language models for automated unit test generation. *IEEE Transactions on Software Engineering*, *50*(1), 85–105. https://doi.org/10.1109/tse.2023.3334955

Segura, S., Fraser, G., Sanchez, A. B., & Ruiz-Cortes, A. (2016). A survey on metamorphic testing. *IEEE Transactions on Software Engineering*, *42*(9), 805–824. IEEE Xplore. https://doi.org/10.1109/tse.2016.2532875

Weisz, J. D., Muller, M., Ross, S. I., Martinez, F., Houde, S., Agarwal, M., Talamadupula, K., & Richards, J. T. (2022). Better together? An evaluation of ai-supported code translation. *27th International Conference on Intelligent User Interfaces*. https://doi.org/10.1145/3490099.3511157