Source Code Comprehension: An Experiment on Human Comments, AI Comments, and Conversational Chatbots

1st Marta Coiro
Università degli Studi di Salerno
Fisciano,Italia
m.coiro9@studenti.unisa.it

2nd Katia Buonocore
Università degli Studi di Salerno
Fisciano,Italia
k.buonocore1@studenti.unisa.it

***Abstract*—In this study, we explore how different forms of code documentation—human-written comments, comments generated by Artificial Intelligence (AI), or conversations with a virtual assistant—impact university students' understanding of source code. Through an experiment conducted with 18 participants, we analyzed the perceived and actual effectiveness of each approach. Our web-based system administered snippets in Java and Rust, measuring comprehension, preferences, and response times. The results reveal interesting patterns in individual preferences and the objective effectiveness of the different approaches.**

***Index Terms*—comment,AI,virtual-assistant**

1. INTRODUCTION

Understanding source code represents one of the most significant challenges for software developers, both in academia and industry. Despite increasing automation in development processes, much of a programmer's work still involves reading, understanding, and modifying code written by others. In this context, documentation plays a central role, serving as a cognitive aid to facilitate decoding the original design intentions and preventing interpretation errors.

Traditionally, understanding is supported through comments handwritten by expert developers. However, the emergence of Artificial Intelligence technologies has introduced new ways to support code reading: from the automatic generation of summaries and descriptions to the ability to interact directly with conversational agents capable of answering questions and explaining code snippets in real time.

These new methods promise to increase code accessibility, especially for students or junior programmers. However, a fundamental question remains: which of these strategies is truly more effective in facilitating understanding? Do users prefer the static support provided by comments (whether human or generated), or do they find greater benefit in active interaction with a conversational assistant?

In this paper, we propose a systematic investigation conducted through a controlled experiment, aimed at directly comparing three methods of code comprehension support:

Identify applicable funding agency here. If none, delete this.

-Comments written by humans.

-Comments generated by AI-based code summarization techniques.

-Conversation with a chatbot based on large linguistic models (LLM).

The objective is to analyze the impact of each method in terms of response accuracy, reading times, subjective preferences, and interaction modes. The study is aimed at university students, thus providing useful insights both for programming education and for the development of software development support tools.

2. RELATED WORKS

In recent years, the increasing accessibility and capabilities of Large Language Models (LLMs) have stimulated a growing body of research on how these tools can support source code comprehension. Several studies have investigated the effectiveness of both human-written and AI-generated explanations, as well as interactive approaches such as conversational agents.

Stapleton et al. (2020) conducted one of the earliest large-scale user studies comparing human-written summaries with machine-generated ones, showing that human comments significantly improve code comprehension ($p = 0.029$) compared to automatic approaches based on ASTs and sequence models. Wu et al. (2024) proposed CODERPE, an LLM-based evaluator that outperformed traditional automatic metrics (BLEU, ROUGE) in judging code summaries, highlighting the potential of LLMs not only for generation but also for evaluation.

Leinonen et al. (2023) compared student-written explanations with those generated by GPT-3 and GPT-4. Their results show that participants generally preferred LLM-generated explanations, particularly for clarity and conciseness. This suggests the increasing potential for integrating LLMs into educational tools. Similarly, Sun et al. (2024) conducted an automatic evaluation using GPT-4 and CodeLlama-Instruct, showing that fine-tuned open-source models can outperform general-purpose ones.

Finally, Nam et al. (2024) explored a hybrid approach, integrating LLM-based assistance into an IDE plugin. Their user study demonstrated that the LLM-enhanced environment improved task completion time, accuracy, and perceived usefulness, particularly when users were allowed to interact with the model in natural language.

While these studies focus primarily on comparing different types of comments or evaluating summarization quality, our work adds a new dimension by comparing three distinct modalities — human comments, AI-generated comments, and interactive chatbot assistance — in a controlled user study. Moreover, our counterbalanced design ensures a fair comparison across languages and presentation order, a factor often neglected in prior studies.

3. THE CODEINSIGHT PLATFORM

The CodeInsight Platform is a custom web platform developed to deliver our code comprehension experiment involving three different documentation modes: human-written comments, AI-generated summaries, and chatbot interaction. The platform was built using Flask (Python) for the backend and SQLite for persistent storage, while the frontend was developed with standard HTML, CSS, and JavaScript.

The platform consists of two primary components:

- The Code Viewer, which displays code snippets in either Java or Rust, with one of the three documentation modalities depending on the experimental condition assigned to the participant. The viewer includes syntax highlighting to improve readability and engagement during the comprehension phase.
- The Interaction Area, which dynamically adapts to the assigned condition:
  - In the Human Comments (HC) and AI Comments (AIC) modes, a pre-filled comment block is displayed directly above the code.
  - In the Chatbot (Chat) mode, the code appears without any comment, but the user can interact with an LLM-based chatbot (powered by OpenAI GPT-4o-mini) to ask questions before starting the quiz. This chat interface is positioned just below the quiz section, and it becomes read-only after the quiz begins, preserving the context but preventing further interaction.

Each snippet is followed by a quiz section, composed of two multiple-choice questions and one open-ended question aimed at assessing the user's understanding of the code,and only for codes with human comments and AI a multiple choice question is asked asking the user whether the comments in that code are written by humans or not

The interface is designed to ensure a smooth user experience: participants can freely navigate between quiz questions using Next and Previous buttons. However, answers can only be submitted collectively at the end of each snippet session using a dedicated Submit Quiz button. A confirmation dialog is shown before the quiz starts to make sure the user is ready.

All user actions—such as opening a snippet, interacting with the chatbot, navigating questions, and submitting answers—are timestamped and logged in the database for later analysis. The platform is session-based and fully anonymized, using manual participant IDs not linked to institutional emails to ensure privacy and GDPR compliance.

CodeInsight supports real-time response saving, structured data loading from a pre-defined JSON dataset, and a counterbalanced snippet assignment matrix, ensuring that each participant sees a unique and balanced combination of documentation types across Java and Rust.

Fig. 1: "Welcome" home page

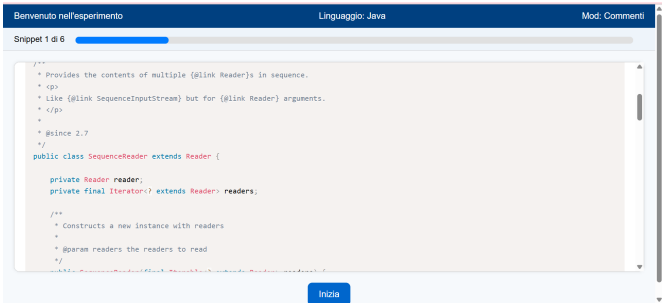

Fig. 2: View snippets in one of three modes

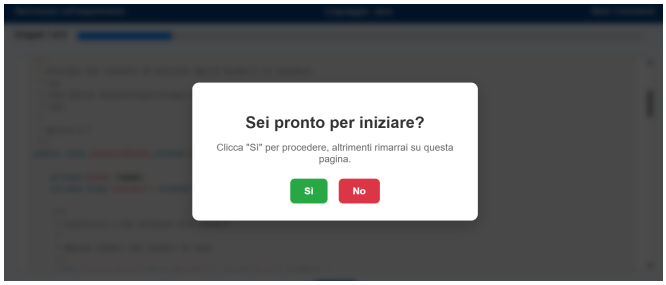

Fig. 3: Experiment start confirmation banner

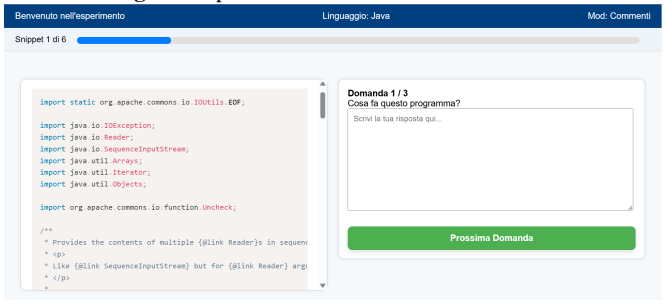

Fig. 4: Open question display

Fig. 5: Viewing quiz display

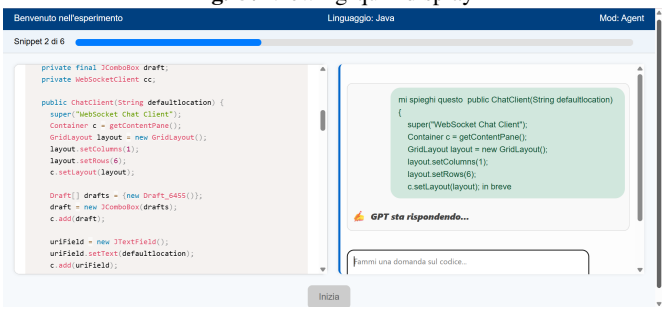

Fig. 6: Viewing chat with Chat-gpt

Fig. 7: Final page

4. USER STUDY

This experiment aims to answer the following research questions:

1) Do participants better understand code when supported by human-written comments, AI-generated summaries, or a conversational agent?
2) Which documentation modality leads to higher quiz performance (accuracy and time)?
3) What are the perceived advantages and limitations of each approach from the participants' point of view?

To explore these questions, we conducted a user study with 18 participants using the CodeInsight platform. Each participant was assigned 6 code snippets: 3 in Java and 3 in Rust, across three documentation modalities (Human Comments – HC, AI Comments – AIC, and Chatbot Interaction – Chat). Each session included structured quiz tasks and evaluation surveys.

4.1. Partecipants

Eighteen students (15 M and 3 F), aged between 21 and 26, took part in the study on a voluntary basis. All were enrolled in Bachelor or Master programs in Computer Science or Computer Engineering. The participants had heterogeneous backgrounds but a common familiarity with basic programming concepts. Each participant was assigned a unique manual ID to access the experiment anonymously.

TABLE I: Participants detail for the experiment

Participants	M	F	NB	Mean age
18	15	3	-	*23,78 (SD 1,66)*

4.2. Design

The experiment follows a within-subjects design, in which all participants are exposed to the three documentation modalities. Each participant views one Java and one Rust snippet in each of the three conditions (HC, AIC, Chat), following a fully counterbalanced assignment matrix **(Table II)**.

TABLE II: Fig. 1. Assignment of Java and Rust snippets to participants across the three modes (AIC, HC, Chat).

Partecipante	Java			Rust		
	AIC	HC	Chat	AIC	HC	Chat
1	TJ1	TJ2	TJ3	TR1	TR2	TR3
2	TJ2	TJ3	TJ1	TR2	TR3	TR1
3	TJ3	TJ1	TJ2	TR3	TR1	TR2
	HC	Chat	AIC	HC	Chat	AIC
4	TJ1	TJ2	TJ3	TR1	TR2	TR3
5	TJ2	TJ3	TJ1	TR2	TR3	TR1
6	TJ3	TJ1	TJ2	TR3	TR1	TR2
	Chat	AIC	HC	Chat	AIC	HC
7	TJ1	TJ2	TJ3	TR1	TR2	TR3
8	TJ2	TJ3	TJ1	TR2	TR3	TR1
9	TJ3	TJ1	TJ2	TR3	TR1	TR2
Rust ↔ Java						
	AIC	HC	Chat	AIC	HC	Chat
10	TR1	TR2	TR3	TJ1	TJ2	TJ3
11	TR2	TR3	TR1	TJ2	TJ3	TJ1
12	TR3	TR1	TR2	TJ3	TJ1	TJ2
	HC	Chat	AIC	HC	Chat	AIC
13	TR1	TR2	TR3	TJ1	TJ2	TJ3
14	TR2	TR3	TR1	TJ2	TJ3	TJ1
15	TR3	TR1	TR2	TJ3	TJ1	TJ2
	Chat	AIC	HC	Chat	AIC	HC
16	TR1	TR2	TR3	TJ1	TJ2	TJ3
17	TR2	TR3	TR1	TJ2	TJ3	TJ1
18	TR3	TR1	TR2	TJ3	TJ1	TJ2

4.3. Apparatus

The experiment was conducted using the CodeInsight platform. Each participant accessed the experiment through a web browser on their laboratory computer. The system tracked all user interactions, including timestamps for code visualization, chat interaction, and quiz responses. The chat interface was only available for Chat-mode snippets and was locked once the participant started the quiz.

4.4. Procedure

At the beginning of the session, each participant received a unique access code and an introductory explanation of the study's objectives. Participants were asked to complete an initial questionnaire containing:

- Consent to data processing pursuant to EU Regulation 2016/679 (GDPR).
- ID.
- Age.
- Gender.
- Education level.
- Degree program.
- Programming experience.
- Java proficiency (Likert scale 1–5).
- Rust proficiency (Likert scale 1–5).
- Frequency of LLM use.
- LLM use for coding tasks.

Before starting, participants watched a short demo that illustrated how the experiment worked.

Task Execution Phase

After watching the demo, the experiment officially began. Each participant worked on six tasks (one per fragment). For each task the participant viewed a code fragment (in one of three documentation modes: Human Comments, AI Comments or Chat Interaction). In the Chat condition, participants could interact with a conversational assistant before starting the quiz. Once the quiz began, chat was disabled and the user could only see the messages exchanged with the agent. Each task had a maximum duration of 8 minutes: up to 6 minutes for understanding the code and approximately 2 minutes for answering the questions. If code reading was completed early, the remaining time was added to the response time. Participants answered three questions:

- 2 multiple-choice questions, and only for the human and AI comment modes, there was an additional question that required them to identify the type of comments.
- 1 open-ended question.

Participants could move between questions using the ”Next” and ”Back” buttons. A dialog box required confirmation before starting each quiz. All actions and times were automatically recorded.

Post-task evaluation

After completing all six snippets, participants were shown a final summary indicating, for each snippet, whether the documentation was produced by a human or by AI comments. Subsequently, participants completed three NASA-TLX-inspired questionnaires, one for each mode.

Finally, participants completed a final free form comments to assess usability and satisfaction with the three modes. The free form comments included:

- Preference ranking by mode.
- Perceived usefulness for understanding the code.
- Open-ended justification for choosing the most useful mode.
- Open-ended response regarding the time given.
- Free comments and observations.

The entire session lasted approximately 30 minutes for participant.

4.5. Dataset

We manually constructed a dataset consisting of a total of 12 code snippets: 6 in Java and 6 in Rust. Each snippet was chosen from existing open-source projects, selecting those that would be understandable and meaningful to computer science students. We did not classify them by structure or type, but rather ensured a variety of usage scenarios and balanced coverage across the two languages (Java and Rust), striving to maintain similar length and logical complexity across the snippets.

Each snippet was duplicated in three different versions:

-With comments written by human experts (HC), designed to clearly explain the logical and functional intent of the code;

-With comments automatically generated by an AI model specialized in code summarization (AIC), via adaptive prompts on ChatGPT;

-With no comments, but with the option of interacting via a chatbot for clarification before asking questions (Chat mode).

It was not possible to maintain complete consistency in length or complexity across all versions, as the snippets are derived from open source projects and were selected based on their educational relevance and clarity. Some snippets are more complex than others, but they were kept within a difficulty range compatible with the experiment’s target audience (university students with basic/intermediate programming knowledge). Each file contains:

- The snippet’s unique identifier (e.g., TJ1, TR3).
- The language (Java or Rust).
- The formatted source code.
- Relevant comments (HC and AIC versions only).

The dataset was saved in JSON format and dynamically loaded by the experimental system for each participant. The chatbot versions were designed to stimulate the ability to ask questions and obtain contextualized answers, in order to test comprehension in an interactive and flexible manner.

5. RESULTS

All 18 participants fully completed the experiment, interacting with six code snippets each. Every snippet was presented in one of the following three comprehension support modes:

- AI-generated comments.
- Human-written comments.
- Interaction with a conversational assistant (Chatbot, Agent mode).

TABLE III: Average Correct Answers for User (by Mode and Language)

Mode	Language	Total Answers	Total Correct Answers	Avg. Correct/User	Avg. Incorrect/User
agent	Java	54	21	1.17 (2.2%)	1.83 (3.4%)
agent	Rust	54	15	0.83 (1.5%)	2.17 (4.0%)
comment_ai	Java	54	19	1.06 (2.0%)	1.94 (3.6%)
comment_ai	Rust	54	16	0.89 (1.6%)	2.11 (3.9%)
comments_human	Java	54	26	1.44 (2.7%)	1.56 (2.9%)
comments_human	Rust	54	13	0.72 (1.3%)	2.28 (4.2%)

TABLE IV: Total Comprehension + Response Time (by Mode and Language)

Mode	Language	Avg. Total Time	Standard Deviation
agent	Java	4:21	1:48
agent	Rust	4:48	1:51
comment_ai	Java	3:25	1:50
comment_ai	Rust	4:38	2:34
comments_human	Java	4:08	2:10
comments_human	Rust	5:04	2:24

The primary goal was to evaluate whether and how the type of support influenced participants' performance. A one-way analysis of variance (ANOVA) was conducted to compare the three conditions with respect to two key variables:

- Accuracy of responses (based on multiple choice questions).
- Comprehension time (extracted from system logs).

In addition, the following were analyzed:

- Open-ended responses ("What does this program do?").
- The ability to recognize the origin of comments (human or AI).
- The influence of the programming language (Java vs Rust).

5.1. RQ1 Response Accuracy

To evaluate whether the support modality influenced the accuracy of multiple-choice answers, we analyzed the average number of correct and incorrect answers per participant, grouped by both modality and programming language (Table 1).

The results show small differences between conditions. A one-way ANOVA was performed to determine whether these differences were statistically significant. The outcome was:

$F(2, 51) = 0.375$

$p = 0.689$

The type of comprehension support (AI, human, or chatbot) did not significantly affect overall user accuracy on multiple-choice questions.

Although not statistically significant, some descriptive patterns emerge:

With Java, the best performance was observed in the human comment condition (1.44 correct answers/user).

With Rust, users performed slightly better with AI-generated comments (0.89 correct/user) compared to human comments (0.72).

The Agent condition yielded intermediate results in Java, but was the least effective in Rust. These trends suggest that while accuracy remains relatively stable across modalities, there may be nuanced interactions with programming language and familiarity that warrant further investigation.

The results are show in Table III

5.2. RQ2 Comprehension Time

To assess whether the support modality had an impact on how quickly participants understood and responded to the code, we analyzed the total time spent on each snippet (comprehension + response), grouped by modality and programming language.

The results are shown in Table IV, which reports both the average total time and standard deviation.A one-way ANOVA was conducted to determine whether the observed differences were statistically significant across modalities. The results revealed a significant main effect of support type on total comprehension time:

$F(2, 51) = 3.217$

$p = 0.048$

The type of support used by participants significantly influenced the total time needed to understand and answer each snippet.

Key Findings For Java, the shortest average total time was achieved with AI-generated comments (3:25), followed by human comments (4:08), and lastly by the Agent condition (4:21).

For Rust, times were generally longer across all modalities:
Human comments resulted in the longest total time (5:04),
Followed by Agent (4:48),
And AI (4:38).

The AI mode consistently reduced total comprehension time, especially for Java, suggesting a more efficient understanding process in familiar environments. In contrast, the Agent mode, although possibly cognitively supportive, resulted in longer total times, particularly in Rust, where users may have needed additional clarification or interaction to complete the task.

5.3. RQ4 Recognition of Comment Origin

To explore participants' ability to identify the source of code comments, we asked them the question: *"Do you think these*

TABLE V: Responses to "Were these comments written by a human?" grouped by support modality and programming language. The table reports the number of correct and incorrect answers, mean and standard deviation per participant.

Modality	Language	Total	Correct	Incorrect	Mean C.	SD C.	Mean I.	SD I.
comment_ai	Java	18	11	7	0.611	0.502	0.389	0.502
comment_ai	Rust	16	7	9	0.438	0.512	0.562	0.512
comments_human	Java	18	6	12	0.333	0.485	0.667	0.485
comments_human	Rust	15	9	6	0.600	0.507	0.400	0.507

TABLE VI: Performance by programming language. The table reports the average number of correct and incorrect answers per user, along with standard deviations.

Language	Avg. Correct	SD Correct	Avg. Incorrect	SD Incorrect
Java	0.09	0.28	0.91	0.28
Rust	0.06	0.23	0.94	0.23

comments were written by a human?"
Table V summarizes the total number of correct and incorrect answers, the mean per participant, and standard deviations, categorized by modality and programming language.

- In the *AI-generated comments* condition for **Java**, participants correctly recognized the non-human origin in 61.1% of cases, but 38.9% still believed the comments were written by a human.
- For **Rust**, the correct recognition rate dropped to 43.8%, suggesting greater difficulty in identifying AI-generated content in less familiar codebases.
- In the *human-written comments* condition, the lowest recognition rate was observed in **Java** (33.3%), meaning participants frequently mistook human comments for AI-generated ones.
- For **Rust**, recognition improved to 60.0%, suggesting that human-authored comments were more distinguishable.

Participants showed a limited ability to distinguish between human and AI-generated comments—especially in Java, where AI-generated comments were more often mistaken for human-written. This suggests that AI comments may effectively mimic human naturalness in familiar contexts. In contrast, Rust yielded more accurate classifications, particularly for human-written content.

5.4. RQ5 Language Effects

To evaluate whether the programming language influenced participants' performance, we compared results across snippets written in **Java** and **Rust**. Table VI reports the average number of correct and incorrect answers per participant, along with the corresponding standard deviations.

- The average number of correct answers was slightly higher for **Java** (0.09 vs 0.06), although the difference is minimal.
- Participants made nearly the same number of incorrect answers in both languages (0.91 in Java, 0.94 in Rust).
- Variability across participants was slightly higher for Java, as indicated by the standard deviations.

Although the differences are small, the data suggest a marginally better familiarity with Java. However, the number of incorrect answers remains high across both languages. This result aligns with prior findings in open-ended responses and comprehension times: while Java tends to be processed slightly faster and more accurately, Rust may demand more cognitive effort, making the presence of conversational support (e.g., the Agent) more beneficial in this context.

5.5. Overall Analysis of Multiple-Choice Responses

Participants answered multiple-choice questions with an overall accuracy rate of **40.4% correct** and **59.6% incorrect**, highlighting a general difficulty in understanding the code across all support conditions.

When aggregating responses by support modality, the *comments_human* condition achieved the highest accuracy, with **47.2%** correct answers. The *comment_ai* and *agent* conditions both resulted in **37.0%** accuracy, suggesting similar but lower effectiveness than human-written comments. Although these differences were not statistically significant (as discussed in RQ1), they offer a descriptive trend that could inform future investigations.

At the individual level, users showed an average accuracy of approximately **38%**, with a **standard deviation of 12%**, reflecting moderate variability in participant performance. Some users were able to adapt more effectively to different support modes, while others maintained a more stable yet less accurate response pattern.

TABLE VII: Aggregated accuracy by support modality

Modality	Correct	% Correct
comments_human	51	47.2%
comment_ai	40	37.0%
agent	40	37.0%

In summary, although no statistically significant differences were found among support modalities, the aggregated results suggest that human-written comments still offer a slight advantage in facilitating code comprehension—particularly in familiar contexts such as Java. The comparable performance of AI-generated and chatbot-based support, however, opens up promising directions for hybrid approaches to assist program understanding.

5.6. NASA-TLX

The analysis of perceived mental workload using the weighted NASA-TLX showed similar average values across the three code comprehension support modes: Agent (mean = 10.16, standard deviation = 3.06), AI Comments (mean = 10.00, standard deviation = 4.80), and Human Comments (mean = 11.06, standard deviation = 4.60). These results suggest that the overall mental workload perceived by users is relatively balanced across the modes, with a slight tendency toward higher workload in the Human Comments mode. To further investigate, the analysis of user preferences expressed across the 15 pairs of TLX dimensions helped identify which aspects of mental workload are most influential in each mode. Specifically, for each question, the "winning" dimension was determined based on the count of user preferences. The winning responses charts reveal distinct patterns among the modes: for example, the Agent mode tends to highlight "Mental Demand" and "Performance" as the predominant dimensions, while the Human Comments mode also emphasizes dimensions related to "Frustration" and "Temporal Demand." The AI Comments mode shows an intermediate distribution, with balanced preferences among various dimensions. This dual analysis—quantitative (weighted TLX) and qualitative (preferences on dimension pairs)—provides a more comprehensive view of the mental workload associated with different support modes. It is useful for guiding improvement efforts and optimizing the user experience. This analysis offers important insights for evaluating the effectiveness and usability of the interfaces studied.

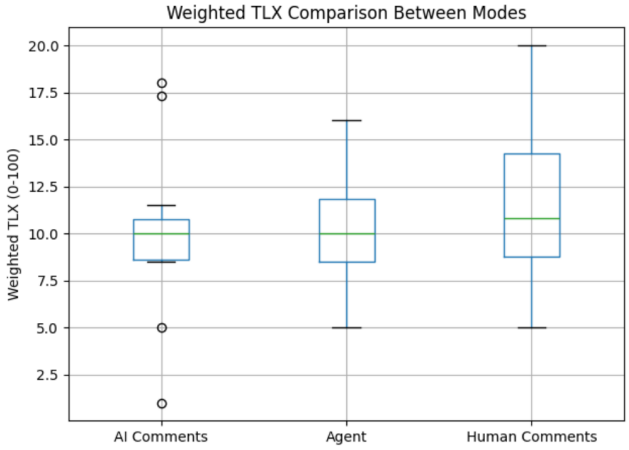

Fig. 8: Weighted TLX

5.7. Free form comments

5.7.0.1. User Preferences on Code Comprehension Modalities In the final free form comments, users were asked to rank the three code comprehension modalities based on their preferences: human-written comments (HC), automatically generated comments (AIC), and interaction with a virtual assistant (Chat). The results show a clear preference for the Chat modality, which was ranked first by 11 out of 18 participants. This modality was the most appreciated, likely due to its interactivity and the ability to ask clarifying questions in real time. The automatically generated comments (AIC) modality was more frequently selected as the second choice, indicating moderate but not dominant interest. Finally, human-written comments (HC) were most often ranked as the third choice, making it the least preferred modality overall. These results suggest that, within the context of this experiment, users found the dynamic and adaptive nature of the virtual assistant to be more effective for understanding code compared to more static approaches.

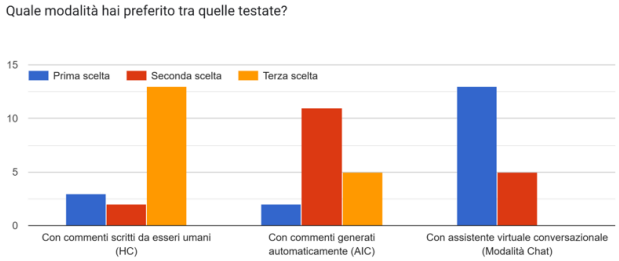

Fig. 9: Favorites mode

5.7.0.2. Most Useful Modality for Understanding Code Users were also asked which of the three modalities they found most useful for understanding the code. Once again, the virtual assistant (Chat) stood out as the most effective, receiving 10 first-choice votes out of 18 participants. This confirms the perceived effectiveness of dynamic interaction in clarifying concepts and guiding comprehension. The automatically generated comments (AIC) received a balanced number of first-choice votes, with 6 participants selecting it as the most useful, suggesting that while it is less interactive, it still provided meaningful support in some cases. Conversely, the human-written comments (HC) received only 2 first-choice votes and were often ranked second or third. This result may seem counterintuitive but suggests that the static nature or variable quality of human comments may have limited their perceived effectiveness compared to the other two modalities. Overall, a consistent picture emerges: direct interaction with an assistant was not only the most preferred modality but also the one deemed most useful for understanding code.

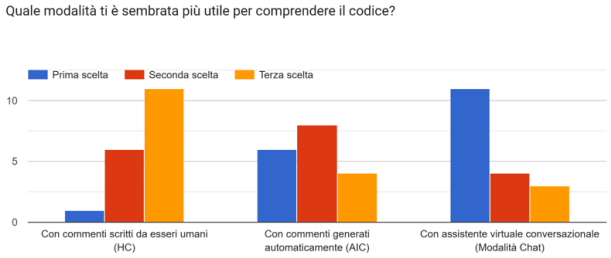

Fig. 10: Useful way to understand the code

5.7.0.3. Open-Ended Responses Reinforcing the Findings The open-ended responses reinforce the evidence that the virtual assistant modality was perceived as the most useful: many participants emphasized how the ability to "ask specific questions" or receive "line-by-line" explanations greatly facilitated

understanding, especially with complex languages like Rust. Others appreciated the automatically generated comments for their clarity and distribution throughout the code, although some found them unhelpful or too generic. Human-written comments, on the other hand, were sometimes perceived as imprecise or overly subjective.

5.7.0.4. Analysis of Open-Ended Responses from the Final Questionnaire The open-ended responses provide further insights into the quantitative findings regarding users' preferences across the comprehension modalities. Many participants praised the Chat modality for its interactivity and the ability to ask specific questions while reading the code, as reflected in comments such as: "The chat mode is more efficient and quicker, getting straight to the point clearly and without unnecessary digressions" and "Being able to talk with Chat helps resolve doubts, especially with complex languages like Rust." This modality was seen as the most user-friendly, thanks to its more natural and flexible interaction compared to static comments. Automatically generated comments (AIC) were generally considered clear and frequent across different parts of the code, offering step-by-step support. However, some users felt they were too generic or lacked depth in certain cases. Human comments (HC) received mixed evaluations: while some recognized their value in conveying personal reflections and improving code readability, others found them less precise or slower to interpret compared to AI-based or interactive approaches. Some noted that the static nature or inconsistent quality of human comments might have limited their perceived usefulness. Overall, the open-ended responses highlight how dynamic interaction and the possibility of contextual clarification offered by the virtual assistant represent a significant advantage in code comprehension, especially for users dealing with unfamiliar or complex programming languages.

6. 6 CONCLUSIONS AND FURTHER WORKS

In this study, we conducted a user study to investigate how different forms of code documentation—namely human-written comments, AI-generated comments, and chatbot interaction—affect university students' ability to comprehend source code. To this end, we developed CodeInsight, a web-based experimental platform designed to deliver code snippets in Java and Rust under controlled conditions, enabling us to measure both objective performance (accuracy and time) and subjective perceptions (usability and preference).

The findings revealed that, although no statistically significant differences emerged in terms of multiple-choice accuracy across support modalities, descriptive trends suggest that human-written comments resulted in slightly better comprehension performance, while AI-generated comments enabled faster processing times, especially in Java. Chatbot interaction, while associated with longer completion times, was consistently rated as the most useful and preferred modality by participants, particularly in handling more complex code fragments in Rust.

The open-ended responses further emphasized the perceived value of interactivity: participants highlighted the ability to ask clarifying questions, receive contextual answers, and adapt the explanation to their own needs as key factors in favor of the chatbot modality. This suggests that dynamic support mechanisms may offer cognitive advantages that static documentation lacks, particularly in educational or less familiar programming contexts.

As future work, we plan to expand the sample size to improve statistical power and generalizability. We will also refine the CodeInsight platform by including more diverse programming languages and more complex tasks, enabling a deeper evaluation of how support modalities scale with task difficulty. Additionally, we aim to integrate and compare other LLMs (e.g., Claude, LLaMA, Gemini) to evaluate potential differences in interaction quality, responsiveness, and pedagogical effectiveness.

Finally, we envision adapting this system to classroom settings and incorporating it into programming courses as a tool for formative assessment and personalized learning. Exploring hybrid modes—combining static comments with interactive clarification agents—may also represent a promising direction to enhance both usability and learning outcomes in code comprehension.

References

1. S. Stapleton, Y. Gambhir, A. LeClair, Z. Eberhart, W. Weimer, and K. Leach,
A Human Study of Comprehension and Code Summarization,
in *Proc. of the 28th International Conference on Program Comprehension (ICPC)*, 2020.
https://doi.org/10.1145/3387904.3389263

2. Y. Wu, Y. Wan, Z. Chu, W. Zhao, Y. Liu, H. Zhang, X. Shi, and P. S. Yu,
Can Large Language Models Serve as Evaluators for Code Summarization?,
arXiv preprint, 2024.
https://arxiv.org/abs/2405.14089

3. J. Leinonen, P. Denny, S. MacNeil, and A. Luxton-Reilly,
Comparing Code Explanations Created by Students and Large Language Models,
in *Proc. of ITiCSE 2023*, ACM, 2023.
https://doi.org/10.1145/3568813.3600118

4. W. Sun, Y. Miao, Y. Li, H. Zhang, and Z. Chen,
Source Code Summarization in the Era of Large Language Models,
to appear in *ICSE 2025*.
https://wssun.github.io/papers/2025-ICSE-LLMs4CodeSum.pdf

5. D. Nam, A. Macvean, V. Hellendoorn, C. Zhang, and K. Leach,
Using an LLM to Help With Code Understanding,
in *Proc. of ICSE 2024*, IEEE/ACM, 2024.
https://arxiv.org/abs/2402.18225