communicated by:
Lehr- und Forschungsgebiet Informatik 9

Prof Dr. Ulrik Schroeder

The present work was submitted to Learning Technologies Research Group
Diese Arbeit wurde vorgelegt am Lehr- und Forschungsgebiet Informatik 9

Exploring the Impact of ChatGPT Assistance on Novice Programmers' Efficiency and Learning in Python Programming Tasks

Untersuchung der Auswirkungen der ChatGPT-Unterstützung auf die Effizienz und das Lernen von Programmieranfängern bei Python-Programmieraufgaben

Master-Thesis
Masterarbeit

presented by / von

Gherman, Codruța-Andreea
436923

Prof. Dr. Ulrik Schroeder

Prof. Dr. Jürgen Giesl

Aachen, June 9, 2024

Contents

List of Figures

List of Tables

List of Listings

Abstract

This thesis investigates implementing and testing a JupyterLab extension integrated with ChatGPT (GPT-3.5) to assist students in learning programming. The goal is to provide timely and comprehensible help while maintaining students' autonomy and enhancing their learning experience. The study addresses three research questions: the effectiveness and frequency of different types of help provided by ChatGPT, the impact of ChatGPT on time efficiency in completing programming tasks, and the perceived learning improvements among students using ChatGPT. The importance of this research lies in its potential to bridge gaps in previous studies by integrating AI-powered feedback systems within educational platforms, thereby enhancing the learning experience for novice programmers. A controlled experiment was conducted with a small group of participants over two weeks, collecting data through surveys and system logs.

The findings reveal that step-by-step guidance and error explanations were the most effective and frequently used forms of assistance. ChatGPT support notably enhanced time efficiency when present, but there was no significant difference when the AI assistance was removed. Students in the experimental group perceived more in-depth learning enhancements than those in the control group. These results indicate that integrating AI-generated feedback in programming education can boost learning efficiency and student satisfaction. Future research should fine-tune the types of feedback to aid novice students better and explore the impact of AI assistance on the quality of code produced. This study contributes to the understanding of the role of AI in education, offering insights into the effective integration of AI tools to support learning.

Chapter 1 Introduction

In recent technological advancements, using Artificial Intelligence (AI) in daily tasks has become a usual factor for most people, especially students. The wide availability of Large Language Models (LLMs) in the forms of OpenAi's ChatGPT, Codex, and Github's Copilot affects the educational landscape, providing students and teachers with new information, help, and, of course, new challenges. While students rejoice at the possibility of solving their tasks quicker and with less stress, teachers can't help but wonder if this is the right step for them, if this helps students, or if the AI is doing their homework for them.

Novice students' needs

Looking at novice programmers, it is commonly known that they require more help in grasping the programming concepts and the language they are using, making it a challenging task for a young student. Learning programming can be a daunting task for novices, and the role of mentors significantly influences their success or failure. While motivation and practice are essential, the human factor is vital in cultivating genuine passion. However, due to the limited availability of teachers and tutors, students must rely mostly on internet findings and help, such as StackOverflow, GeeksForGeeks, and, more recently, ChatGPT. Using LLMs for solving programming homework comes at a price; students won't get help; they will get their complete homework solved without learning outcomes. While schools and universities can ban specific sites internally, they cannot stop students from using these tools to complete homework quickly.

Research has already started to address this situation, asking questions about whether LLMs can be used to increase students' productivity and learning outcomes while maintaining a healthy learning environment. Novice programmers require constant feedback and guidance, and while teachers are not constantly available, AI can be a powerful tool that can provide insights about concepts, hints, and constructive feedback.

In academic settings, especially in Computer Science (CS) courses, one of the most commonly used tools for homework is JupyterLab. JupyterLab is extensively utilized for Python programming and data analysis due to its main feature - the ability to work with Jupyter Notebooks. One advantage of using JupyterLab, and Jupyter Notebooks in particular, is the absence of the need to install tools such as Python, pip, and various packages on personal computers, which would add to the challenges the students would face when trying to solve their homework. Due to these factors, JupyterLab was the prime candidate for implementing AI features.

The primary objective of this research was to develop a JupyterLab extension designed to offer timely and comprehensible assistance to students while preserving their autonomy and facilitating their learning. This extension integrates with ChatGPT (GPT-3.5), enabling students to request help for various scenarios and predefined problems—the subsequent experiment aimed to collect sufficient data to address several key research questions.
The following questions guided the research:

Research Question 1: Help Buttons What types of help provided by ChatGPT do students consider most effective for their learning, and which types are utilized more frequently versus less frequently?

Research Question 2: Time Efficiency Does the availability of help from ChatGPT lead to improved time efficiency in completing programming tasks among students, and do these time efficiency benefits persist in the final set of problems when the help is no longer available?

Research Question 3: Self-Satisfaction and Learning To what extent do students perceive that they have learned from using the help provided by ChatGPT, and is there a significant difference in self-reported learning improvements between students who used ChatGPT and those who did not?

Chapter 2 Background

This chapter provides a comprehensive overview of the various tools and methodologies utilized throughout this thesis, focusing on their application within the given context. The aim is to familiarize the reader with the foundational technologies and concepts that underpin the research and development processes discussed later in the thesis.

2.1 JupyterLab

JupyterLab [5] is an interactive, open-source, web-based environment that enables users to write and execute code without the need to set up any software on the user's machine. In the context of Python programming, JupyterLab provides a favorable environment for iterative development and experimentation. Users can execute Python code cells within notebooks, facilitating rapid prototyping and testing of algorithms and solutions.

What distinguishes JupyterLab is its extendability through customizations and integration of new features. Through the use of extensions and plugins, users can tailor the environment to suit specific requirements and workflows. Whether it be integrating additional programming languages, incorporating specialized visualization tools, or enhancing collaboration capabilities, JupyterLab's extensibility empowers users to adapt the environment to diverse contexts within computer science and beyond.

2.1.1 Extensions

Extensions play a big role in enhancing the functionality and adaptability of JupyterLab, allowing users to tailor the environment to their specific needs. This section looks into several extensions used within the context of this thesis, highlighting their features and applications.

Juxl (JupyterLab xAPI logging interface)

JupyterLab xAPI logging interface (Juxl) extends the functionality of JupyterLab by offering basic logging features tailored for the integration of Learning Analytics (LA) into the platform. Comprising four primary packages - juxl, juxt-extension, vocabulary, and logging - Juxl serves as the core of this extension, defining essential interfaces for the other packages to implement.[8]

The @juxl/logging package monitors specific events triggered by any JupyterLab extension. Each event is then converted into an Experience API (xAPI) statement. A dedicated translator facilitates the extraction of the verb and object for the xAPI statement. Meanwhile, the actor, context, and timestamp details remain consistent across all potential statements and are supplied by a centralized interface. The @juxl/vocabulary package furnishes the necessary vocabulary, streamlining the management of verbs, objects, and additional attribute names, types, and identifiers. This approach simplifies maintenance, particularly as some attributes are reused across different events.
To accommodate user privacy preferences, the actor attribute may contain either an anonymous actor, a pseudonym, or a placeholder for personalized actor information. Extensions to the context or other attributes, such as a result, are managed by a translator as required. This encapsulated structure enhances flexibility and scalability while maintaining a clear separation of concerns within the Juxl framework.

Juxl Cut

The Juxl Cut extension [4] provides the ability to add test cells to the code cells (Figure 2.2). An advantage posed by this extension is the possibility of hiding the test cells and presenting the user with just a button to run the said tests (Figure 2.1). In the context of the thesis, this feature has been used to allow students to test their code without having the bias of knowing the input and the expected output, such that no code is tailored for the exact test cases provided.

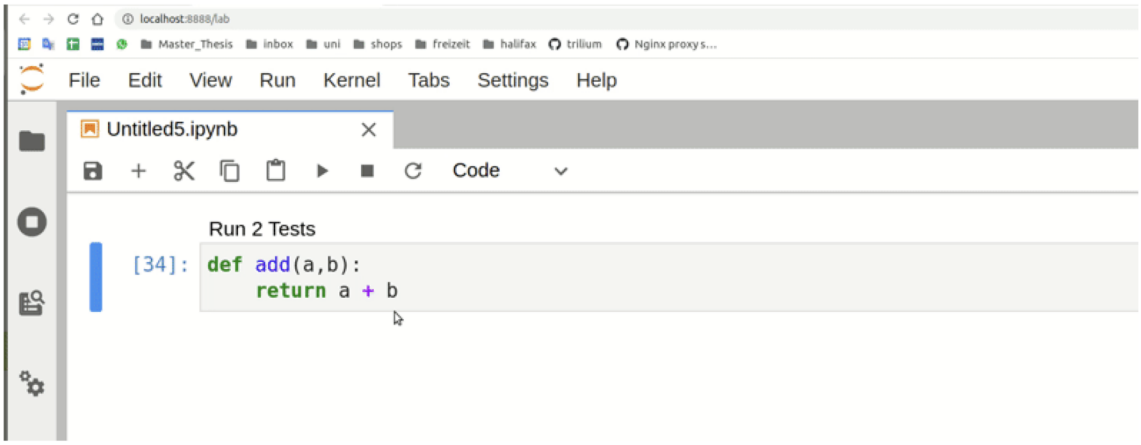


Figure 2.1: Example of the run button for the tests with the test cells hidden.[4]

Hint Buttons

The hint buttons have been developed for this thesis in particular.

2.2 ChatGPT

ChatGPT is an instance of OpenAI's Generative Pre-trained Transformer (GPT) model specifically fine-tuned and deployed for conversational interactions. It's designed to understand and generate human-like text based on the input it receives. Trained on a

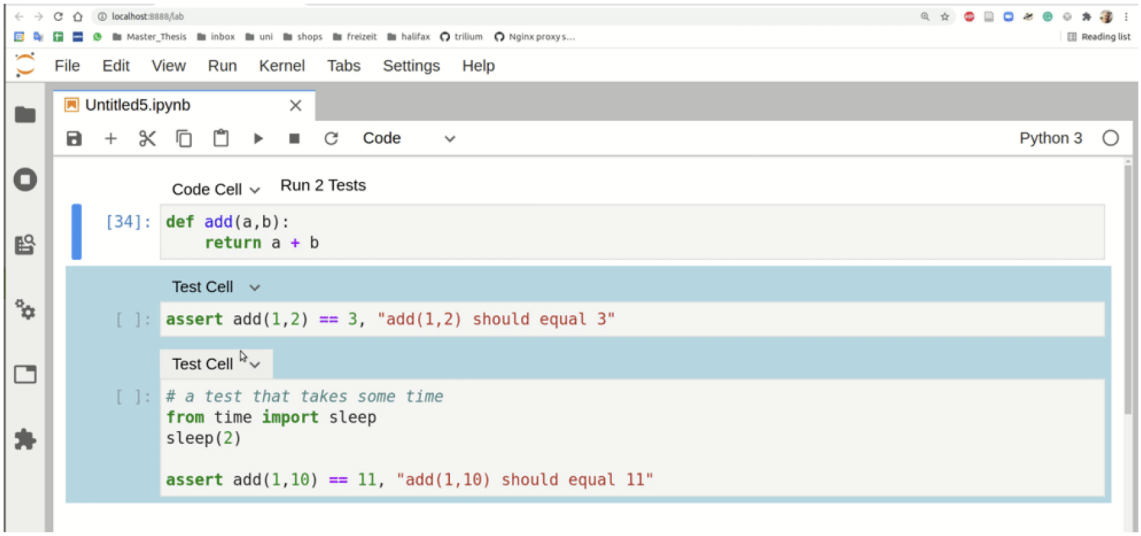


Figure 2.2: Example of the test cells[4]

vast corpus of text data from the internet, ChatGPT can engage in a wide range of conversations, answer questions, provide explanations, offer suggestions, and more. It leverages its understanding of context and language structure to generate responses that are coherent and relevant to the conversation. As a language model, ChatGPT continuously learns and adapts based on the interactions it has with users. [1]
In the context of this thesis, the ChatGPT API has been used to generate helpful hints based on the student's problem, existing code and the type of help requested. The use of the API is not free and requires an API key. The user pays in advance per batch of tokens. Roughly 4 characters represent a token.
For each of the five help types defined, the model is prompted with some rules to respect when providing help. The language style and writing style have been defined, as well as key rules - eg. do not provide code to the student. Listing 2.1

```
1 const guidelinesText: string = '**Role**: You are a programming teacher specializing in
      Python.If the FOCAL CELL is empty, the student might need more clarification on
     the REQUIREMENT CELL. Provide additional explanations or examples related to the
     requirements to help the student understand the problem better without presenting
     actual code. Remember to end each interaction with a list of the programming
     concepts that should be used for solving the problem, as specified in the
     REQUIREMENT CELL. This ensures that the student has clear guidance on the key
     concepts relevant to the task. Deliver responses in [Selected Language].';
2 const requirementText: string = '*Requirement Cell:*' + getProblemStatement();
3 const templateText: string = '*Template Cell:*' + '';
4 const studentCodeText: string = '*Focal Cell:*' + getStudentCode();
5 const language = 'english';
6 const generatedError: string = '*Generated Error:*' + (getError() ?? "No error
     generated");
```

Listing 2.1: Example setup for ChatGPT API call

2.3 The xAPI (Experience API)

Experience API[6], or xAPI, is a specification that facilitates communication between various software programs regarding educational activities. It is also referred to

as ADL (Advanced Distributed Learning) xAPI and Tin Can API. Conventional learning management systems (LMS) frequently monitor basic performance indicators including test scores, course completion rates, and activity completion time. On the other hand, xAPI broadens this tracking capacity to encompass more learning activities—both online and offline.

With the use of xAPI, information about a learner's actions may be gathered from various sources, including games, virtual reality experiences, simulations, and mobile learning. The assertions that make up this data are composed of an actor, a verb, and an object. A Learning Record Store (LRS), which serves as a central location for keeping and evaluating learning data, can receive these assertions.

One of the main advantages of xAPI is its capacity to offer a more thorough picture of a learner's experience across many contexts and platforms, enabling a more accurate evaluation of the efficacy of learning and customized learning opportunities. It is extensively used to monitor, assess, and enhance learning experiences in the e-learning, training, and educational technology industries.

In the context of the thesis, xAPI is used to store information about students' actions in the Notebooks. The basic verbs for opening or closing the notebook, or running code and modifying it are already defined by the Juxl extensions and are automatically logged when the extension is enabled. Additionally, five new verbs have been added to log when the students use the help buttons. The five buttons and their usage description can be found in Table 2.1.

Besides the five events added for button presses, seven more have been added to aid in the data analysis process. These additional events (found in Table 2.2) related to the events in Table 2.1 log the response of the LLM for the initial button press. The decision to have separate events for the initial button press and the response was made due to the timestamp which is added to each of the events. An important aspect of feedback is also the time required to wait for it.

2.4 Feedback

When applied skillfully, feedback may be a potent learning intervention. Feedback has consistently been shown to be effective in raising student learning performance, according to decades of educational research. Feedback may be divided into two categories: formative and summative, depending on the delivery time. While formative feedback is given to students before the assessment findings, summative feedback is given to students concurrently with the outcomes of the assessment [7]. Across a wide range of disciplines, formative feedback has consistently been proven to be more helpful than summative feedback [26].

2.4.1 Types of feedback

Narciss and Huth [17] [16] defined different levels and types of feedback. There are two types of feedback: simple feedback and elaborated feedback. Simple feedback does not improve solutions. It provides performance information (*Knowledge on Performance - KP*), which can be presented as a percentage of correct results; feedback on correctness (*Knowledge of result/response - KR*) presented as *correct/incorrect*;

Table 2.1: Help buttons descriptions

Button Name	Verb	Task Description
Hint	helpClicked	Help students complete their code by offering step-by-step information.
Explain Concepts	explainConceptsClicked	Provide additional explanations or examples related to the requirements to help the student understand the problem better. Offer a list of the programming concepts that should be used for solving the problem.
Explain Error	explainErrorClicked	Based on the code and the existing error suggest steps to overcome the problem and guide students in the logical thinking process.
Check code	checkCodeClicked	Test the code to ensure correctness based on the requirements. Identify any potential issues or edge cases that might not be handled by the current code. Offer guidance on how to address these issues.
Improve code	improveCodeClicked	Suggest improvements or optimizations to align it with the stated requirements.

Table 2.2: Additional events with descriptions

Button Name	Verb	Verb Description
	helpClickedResponse explainConceptsClickedResponse explainErrorClickedResponse checkCodeClickedResponse improveCodeClickedResponse	Logs the response received from ChatGPT
Submit and close	formSubmitted	Logs the students' feedback related to a response they have received from the LLM
Finish	finishClicked	Collects and logs all information related to the notebook in the current state.

and lastly *Knowledge of the correct result - KCR* presented as a description of the correct result.

A short description of the elaborated feedback components is presented in Table 2.3.

Table 2.3: Classification of feedback components as described in [16]

Category	Description
Knowledge about task constraints (KTC)	Guides to assist students in navigating the demands and limitations unique to each assignment. Task requirements (TR) and task-processing rules (TPR) hints are two examples of subtypes.
Knowledge about concepts (KC)	Gives information to aid students in comprehending the underlying ideas. Examples illustrating concepts (EXA) and explanations of the subject matter (EXP) are subtypes.
Knowledge about mistakes (KM)	Provides information on the various kinds of mistakes that students may make. Subtypes include test failures (TF), compiler errors (CE), solution errors (SE), style issues (SI), and performance issues (PI).
Knowledge on how to proceed (KH)	Helps students make corrections and improve their solutions. Subtypes include bug-related hints for error correction (EC), task-processing steps (TPS), and suggestions for improvements (IM).
Knowledge about metacognition (KMC)	Contains feedback about the learning process.

Hao et. al.[12] found that students who received elaborated feedback (KCR + hint) on why the test cases failed and hints on how to fix the code outperformed the students who received only information on what test cases failed (KR).

Using the knowledge about these types of feedback the Feedback LLM extension was created. Each of the buttons described in Table 2.1 is related to at least one type of elaborated feedback in Table 2.3. More information on this will be presented in chapter 4.

2.5 Analysis tools and tests

The acquired data were analyzed using Python, employing several statistical tests to determine normality, homogeneity, and significance.

To assess the normality of the data, the Shapiro-Wilk test was performed. This test evaluates whether a sample comes from a normally distributed population, with the null hypothesis being that the data is normally distributed. A p-value greater than 0.05 suggests that the data is normally distributed, while a p-value less than 0.05 indicates that the data is not normally distributed, leading to the rejection of the null hypothesis.

In cases where the data was not normally distributed, the Mann-Whitney U test was used to determine if there was a statistical difference between the two groups. This non-parametric test is suitable when the data does not follow a normal distribution. It tests the null hypothesis that there is no difference between the two groups, providing a U-value and a p-value. A p-value greater than 0.05 indicates no significant difference between the groups, while a p-value less than 0.05 suggests a significant difference. The U-value, or Mann-Whitney U statistic, indicates the likelihood of the observed result occurring by chance; a smaller U-value implies a lower probability of the result being due to chance.
For normally distributed data with similar variances, the Independent T-test was utilized to compare the means of the two groups. This test also follows the same p-value interpretation as the Mann-Whitney U test, where a p-value greater than 0.05 suggests no significant difference and a p-value less than 0.05 indicates a significant difference between the groups.
To assess the equality of variances between the two groups, Levene's test was conducted. This test evaluates the null hypothesis that the variances of the samples are equal. If the test shows that variances are similar, it supports the use of parametric tests like the Independent T-test for further analysis.

Chapter 3 Related Work

The integration of ChatGPT and other large language models (LLMs) in educational contexts has garnered significant attention in recent years, particularly for their potential to support novice programmers in learning and problem-solving tasks. This section reviews the relevant literature, highlighting key findings and contributions from recent studies.

3.1 ChatGPT for Educational Feedback and Programming Assistance

The feasibility of using ChatGPT to provide feedback on student assignments has been a subject of interest among researchers. Dai et al. [10] conducted a study to evaluate the readability, alignment with instructor feedback, and effectiveness of ChatGPT-generated feedback in enhancing student learning. Their results indicate that ChatGPT is capable of producing detailed and coherent feedback that aligns well with instructor assessments. This suggests that ChatGPT could be a valuable tool to supplement traditional feedback mechanisms, potentially improving the scalability and consistency of educational feedback.

In another study, MacNeil et al. [15] explored the use of GPT-3 for generating explanations of programming code. Through practical application and feedback from students and educators, they assessed the effectiveness and challenges associated with AI-generated explanations. The findings reveal that while GPT-3 can enhance understanding and provide valuable learning aids, there are notable challenges regarding the accuracy and relevance of the explanations. The study offers recommendations for integrating AI tools into programming education, emphasizing the potential benefits and limitations of such technologies.

The study by Kiesler et al. [13] investigates the effectiveness of using ChatGPT to provide feedback on introductory programming tasks. The researchers used student submissions from a CS1 course as input for ChatGPT and analyzed the feedback generated. They aimed to explore how ChatGPT responds to students seeking help with their programming tasks and to identify the types of feedback provided. The study found that ChatGPT can generate useful feedback, including textual explanations of errors and suggested fixes, which can benefit students in understanding and correcting their code. However, the study also highlighted that ChatGPT's feedback could

be misleading at times, and its effectiveness varied depending on the task. The researchers concluded that while LLMs like ChatGPT show promise in providing automated feedback, there is a need for careful integration and guidance to ensure the feedback is reliable and beneficial for novice programmers.

Further advancing the application of LLMs in education, Sarsa et al. [22] focused on the automatic generation of programming exercises and code explanations using OpenAI Codex. Their work demonstrates the ability of Codex to generate meaningful programming exercises and accurate code explanations, which can be utilized as educational resources. They emphasize the potential of AI-driven tools to create diverse and effective learning materials for introductory programming courses, highlighting the capacity of such tools to enhance the educational experience for novice programmers.

Roest et al. [20] explore the application of large language models (LLMs) to generate next-step hints for students in introductory programming courses. The study focuses on the effectiveness of these LLM-generated hints in guiding students through programming tasks and compares the results with traditional data-driven hint generation methods.

The authors developed a dataset comprising sequences of steps that novice students typically take when solving programming problems. Using this dataset, they engineered prompts for generating next-step hints with LLMs, specifically utilizing OpenAI's GPT-3.5-turbo model. The workflow involved creating prompts that included a problem description and relevant context, enabling the LLMs to generate coherent and contextually appropriate next-step hints for the students.

The evaluation of the generated hints revealed several key findings. The LLM-generated hints were found to be effective in helping students progress through programming tasks. Students were able to use the hints to understand the next steps in their problem-solving process, which facilitated learning and task completion. The evaluation of the answers was provided by students through a feedback form for each of the interactions (see 3.1).The study compared LLM-generated hints with those generated by traditional data-driven methods. It was observed that LLMs provided more nuanced and contextually relevant hints, potentially due to their extensive training on diverse datasets. Similar to findings in other studies [18], the potential for over-reliance on LLM-generated hints was noted. Students might become dependent on these hints, which could hinder the development of their independent problem-solving skills. While the hints were generally helpful, there were instances where the LLMs provided incorrect or misleading information. This highlights the importance of incorporating validation mechanisms to ensure the quality and accuracy of the hints provided.

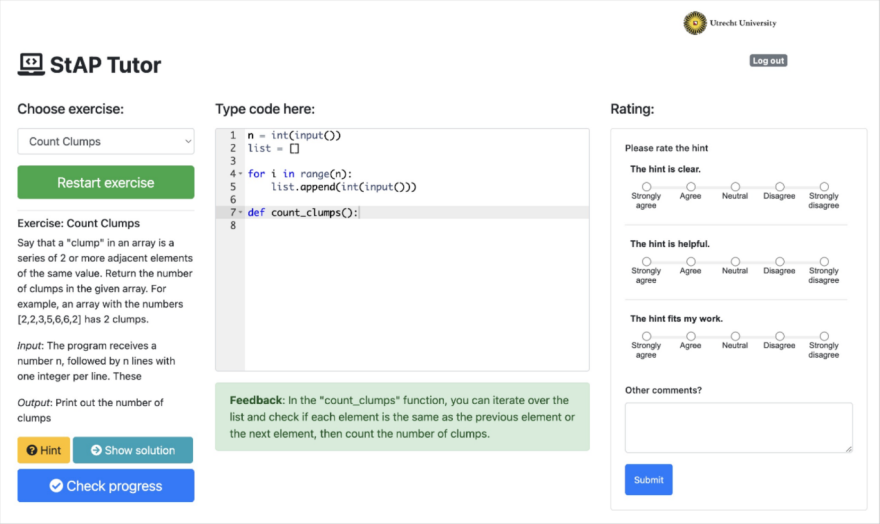


Figure 2: Interaction with the StAP-tutor: the student is asked to rate the hint.

Figure 3.1: Example of the hint and rating interaction taken from Roest et al. [20]

The authors concluded that LLMs hold significant promise for enhancing programming education by providing immediate, personalized assistance to students. However, they also emphasized the need for careful implementation to mitigate potential drawbacks, such as over-reliance and occasional inaccuracies in the generated hints.

This study aligns closely with the present research, as both investigate the use of LLMs for generating feedback and guidance in programming education. The insights from Roest et al. [20] underscore the potential benefits and challenges of employing LLMs in educational settings, providing a valuable reference point for the development and evaluation of similar educational technologies.

3.2 Prompt Design and Interaction Patterns

Prompt design is a critical aspect of optimizing the performance of LLMs like ChatGPT in educational settings. Cao et al. [9] conducted a comprehensive study on prompt design, examining the advantages and limitations of various prompting methods. They provide a systematic survey of prompting techniques and their applications, offering insights into how prompt design can be optimized to enhance the performance of LLMs. Their findings underscore the importance of carefully crafted prompts to achieve more accurate and contextually relevant outputs from LLMs.

Building on the importance of prompt design, White et al. [25] present a catalog of prompt patterns aimed at improving the quality of code generated by LLMs like ChatGPT. Their work categorizes different prompt patterns and provides detailed examples and motivations for each pattern. They highlight how structured prompts can lead to more accurate and relevant outputs, thereby enhancing the usability of LLMs in educational and programming contexts. This research underscores the necessity of developing effective prompting strategies to maximize the potential of LLMs in various applications.

3.3 Comparative Studies and Practical Implications

Empirical studies have been conducted to evaluate the practical applications and limitations of ChatGPT as a programming assistant. Tian et al. [24] conducted an extensive evaluation of ChatGPT's potential as a fully automated programming assistant. Their research focuses on tasks such as code generation, program repair, and code summarization. The findings indicate that ChatGPT performs comparably to state-of-the-art approaches in these areas, providing valuable insights into its practical applications. However, the study also highlights the need for further research to address current limitations and enhance the capabilities of ChatGPT. These findings suggest that while ChatGPT holds promise as a programming assistant, continuous improvement and adaptation are necessary to fully realize its potential.

3.4 Automated Feedback on Programming Assignments

A study by Pankiewicz and Baker [18] investigated the use of large language models (LLMs) like GPT to automate feedback on programming assignments. Their research focused on assessing the impact of GPT-generated hints on students' performance in solving C# tasks. The study found that the experimental group, which received GPT-generated hints, demonstrated a significantly higher success rate in completing tasks compared to the control group. Additionally, the experimental group showed improved efficiency, taking less time to solve tasks without hints due to the additional information provided previously.

However, the study also noted a potential downside: students might rely too heavily on GPT-generated feedback when facing complex tasks they do not fully understand. This observation underscores the need for balanced use of AI tools in education, ensuring that they complement rather than replace the learning process.

3.5 Jupyter Lab Implementations

There are several Jupyter Lab implementations that provide access to GPT models.

ChatGPT for Jupyter [3] is a browser extension designed to enhance the functionality of Jupyter Notebooks by integrating various AI helper features powered by ChatGPT and GPT-4. This extension aims to assist users in several programming tasks directly within the Jupyter environment.

The primary functions of the ChatGPT Jupyter Extension include (see 3.2): *Format*: Automatically add comments, docstrings, and formatting to code cells, improving code readability and documentation; *Explain*: Provide explanations of code cell content in a simple, easy-to-understand manner; *Debug*: Assist in debugging by analyzing error messages and suggesting possible fixes; *Complete*: Help complete code snippets, reducing the time and effort required to write code from scratch; *Review*: Offer code reviews, highlighting potential improvements and ensuring code quality.

The extension is designed to be user-friendly and integrates seamlessly with Jupyter Notebooks, providing real-time assistance to users. It supports both Chrome and Firefox browsers, with installation options available from the Chrome Web Store or through local installation procedures.
Despite its powerful capabilities, the developers caution users that ChatGPT is not infallible and may produce incorrect or unexpected results. Users are encouraged to verify the outputs and use the tool as an aid rather than a definitive solution provider.

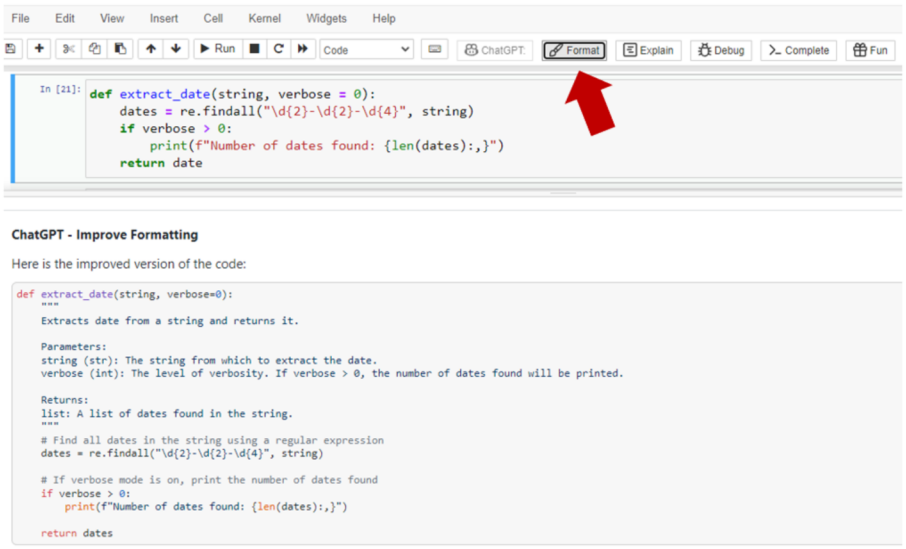


Figure 3.2: Example from ChatGPT for Jupyter extension [3]

Jupyter AI: A Generative AI Extension for JupyterLab developed as part of the JupyterLab organization, integrates generative AI capabilities directly into Jupyter Notebooks, significantly enhancing the productivity and functionality of the JupyterLab environment [23]. This extension is designed to provide a seamless way for users to explore and utilize generative AI models within their notebooks. It offers a native user interface that allows users to interact with generative AI as a conversational assistant, facilitating a more interactive and intuitive user experience.

Chapter 4 Research Design

This chapter will discuss and present all the aspects of the study, what was built and why, the research questions, and the study design.

4.1 JupyterLab Extension

As presented in chapter 2, an extension for JupyterLab was created. The Feedback LLM extension was developed to aid novice programmers. The central aspect of the extension is its ability to present the user with multiple choices regarding the type of help they need.

Using Narciss [16] feedback strategies, five buttons have been created in the Feedback LLM extension. Each button has a different function in helping novice programmers solve their tasks, as seen in Table 4.1. Further, the context and usage of the buttons will be presented.

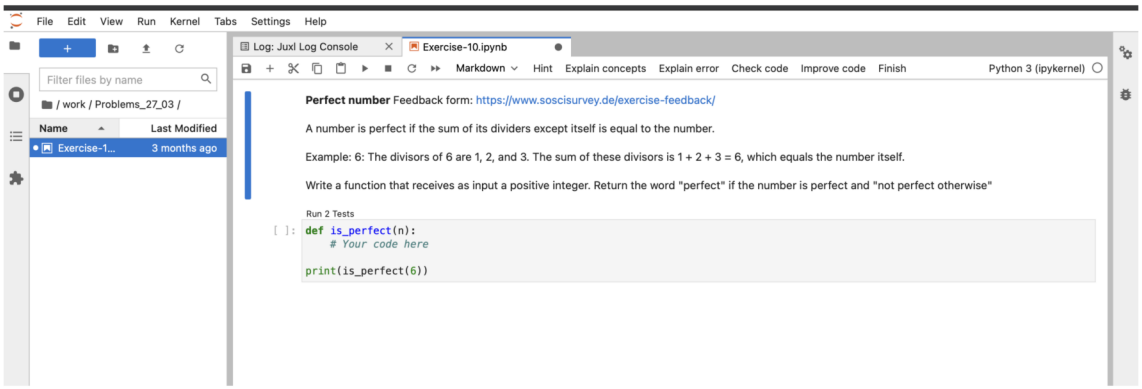


Figure 4.1: The interface of Jupyter Lab containing the buttons from the extension

Table 4.1: Buttons association with feedback types

Category	**Associated button**
Knowledge about task constraints (KTC)	Hint
Knowledge about concepts (KC)	Explain Concepts
Knowledge about mistakes (KM)	Explain Error and Improve Code
Knowledge on how to proceed (KH)	Hint and Check Code

4.1.1 Prompt Engineering

Following the example prompts recent studies used ([20], [18]), an incipient phase of the experiment included prompt engineering. The main focus of this task was finding prompts that can produce helpful responses, delivered in a structured manner, and most importantly, do not include code that can solve the problem. As a starting point, the prompts used in the ChatGPT for Jupyter extension [3] were tested and modified according to the project's needs. The structure of the prompts and the name of the cells were kept the same for ease of use. The prompt includes the following components: a guidelines text containing the prompt instructions; the requirements text containing the problem statement from the notebook - also referred to as the requirement cell in the prompt; a template text which can contain the template cell if any is provided; the student code text which contains the students' code - also referred to as the focal cell; the language in which the response is expected (in this case English - but can be modified for future use); and lastly the generated error in case there is one, a requirement for the error explanation part.

4.1.2 Hint button

The "Hint" button (see Listing 4.1) is a key feature designed to support students in their Python programming tasks. When used, it provides step-by-step explanations and clear guidance to help students complete their code in the focal cell (the area where students enter their work). This assistance is tailored to meet the requirements specified in the requirement cell and considers the provided template cell. Notably, the button's responses do not include any actual code, complete solutions, or code snippets. Instead, it focuses on guiding students' learning process and improving their problem-solving skills. The explanations conclude with a list of relevant programming concepts from the requirement cell, ensuring students understand the key ideas to solve the problem.

```
1    '**Role: You are a programming teacher specializing in Python. You know all the
     programming concepts and functions in Python.\n Your primary goal is to help the
     student complete code in the FOCAL CELL by providing step-by-step explanations and
     clear guidance. \n Do not, under any circumstances, provide code snippets, complete
      solutions, or code-related content.\n Your assistance should align with the
     REQUIREMENT CELL and take the TEMPLATE CELL into account. \n It is crucial to
     ensure that the code in the FOCAL CELL is compliant with the requirements.**\n\n
     Key Guidelines for Your Explanations: \n\n - Deliver responses in [Selected
     Language].\n - Explicitly, DO NOT share any actual code, full solutions, or code
     snippets.\n - The REQUIREMENT CELL describes the problem the user has to solve. The
      TEMPLATE CELL provides the user with a template for their code. The FOCAL CELL is
     what the user has coded until now.\n - If the FOCAL CELL is empty the user requires
      more explanation of the REQUIREMENT CELL.\n - If the code in the FOCAL CELL does
     not solve the requirement provide the user with explanations to improve the code,
     do not provide code.\n - Test the code in the FOCAL CELL to check corectness.\n -
     DO NOT provide example implementations.\n - Your objective is to guide the student
     \'s learning and problem-solving process.\n - After providing the response end the
     conversation with a list of the programming concepts that should be used for
     solving the problem. Take the information form the REQUIREMENT CELL.'
```

Listing 4.1: Instruction for ChatGPT - Hint Button

4.1.3 Explain Concepts button

The "Explain Concepts" button (see Listing 4.2) is designed to assist students in understanding Python programming requirements. When activated, it provides detailed explanations and examples related to the requirements specified in the task without presenting actual code. This feature is particularly useful if the focal cell is empty, indicating that the student might need more clarification. The responses generated by this button end with a list of fundamental programming concepts that should be used to solve the problem, ensuring that students receive clear and relevant guidance.

```
'**Role**: You are a programming teacher specializing in Python. If the FOCAL CELL
is empty, the student might need more clarification on the REQUIREMENT CELL.
Provide additional explanations or examples related to the requirements to help the
 student understand the problem better without presenting actual code. Remember to
end each interaction with a list of the programming concepts that should be used
for solving the problem, as specified in the REQUIREMENT CELL. This ensures that
the student has clear guidance on the key concepts relevant to the task. Deliver
responses in [Selected Language].';
```

Listing 4.2: Instruction for ChatGPT - Explain Concepts Button

4.1.4 Explain Error button

The "Explain Error" button (see Listing 4.3) is designed to help students understand and resolve errors in their Python code. When pressed, it reviews the code in the focal cell and the error message provided in the generated error. Based on this information and the requirements outlined in the requirement cell, the button offers a clear explanation of the problems encountered and suggests steps to address them. Importantly, it does so without providing any actual code, focusing instead on guiding the student's learning and problem-solving process. The explanations are delivered in the selected language, ensuring clarity and comprehension.

```
'**Role**: You are a programming teacher specializing in Python. Review the code in
 the FOCAL CELL. The error generated by the compiler is in GENERATED ERROR. Based
on the REQUIREMENT CELL and the GENERATED ERROR, clearly explain the problems and
suggest steps to address them without providing code.Your objective is to guide the
 student\'s learning and problem-solving process. Deliver responses in [Selected
Language].';
```

Listing 4.3: Instruction for ChatGPT - Explain Error Button

4.1.5 Improve Code button

The "Improve Code" button (see Listing 4.4) is intended to help students enhance and optimize their Python code. When activated, it examines the code in the focal cell and suggests improvements or optimizations to meet better the requirements outlined in the requirement cell. The button provides clear and detailed explanations for each proposed improvement, focusing on guiding the student's understanding and skills without presenting any actual code. The responses are delivered in the selected language, ensuring the student comprehends the guidance provided.

```
'**Role**: You are a programming teacher specializing in Python.Examine the code in
 the FOCAL CELL and suggest improvements or optimizations to align it with the
requirements stated in the REQUIREMENT CELL. Offer clear explanations for each
suggested improvement without presenting actual code.Deliver responses in [Selected
 Language].';
```

Listing 4.4: Instruction for ChatGPT - Improve Code Button

4.1.6 Check Code button

The "Check Code" button (see Listing 4.5) is designed to help students verify the correctness of their Python code. When used, it tests the code in the focal cell against the requirements specified in the requirement cell. It identifies potential issues or edge cases the current code might not handle. The button then offers guidance on addressing these issues, ensuring the student understands the necessary adjustments without providing any code. The feedback is delivered in the selected language, ensuring clarity and practical learning.

```
'**Role**: You are a programming teacher specializing in Python. Test the code in
the FOCAL CELL to ensure correctness based on the requirements in the REQUIREMENT
CELL. Identify any potential issues or edge cases that might not be handled by the
current code. Offer guidance on how to address these issues without providing the
actual code. Deliver responses in [Selected Language].';
```

Listing 4.5: Instruction for ChatGPT - Check Code Button

4.1.7 Feedback Form

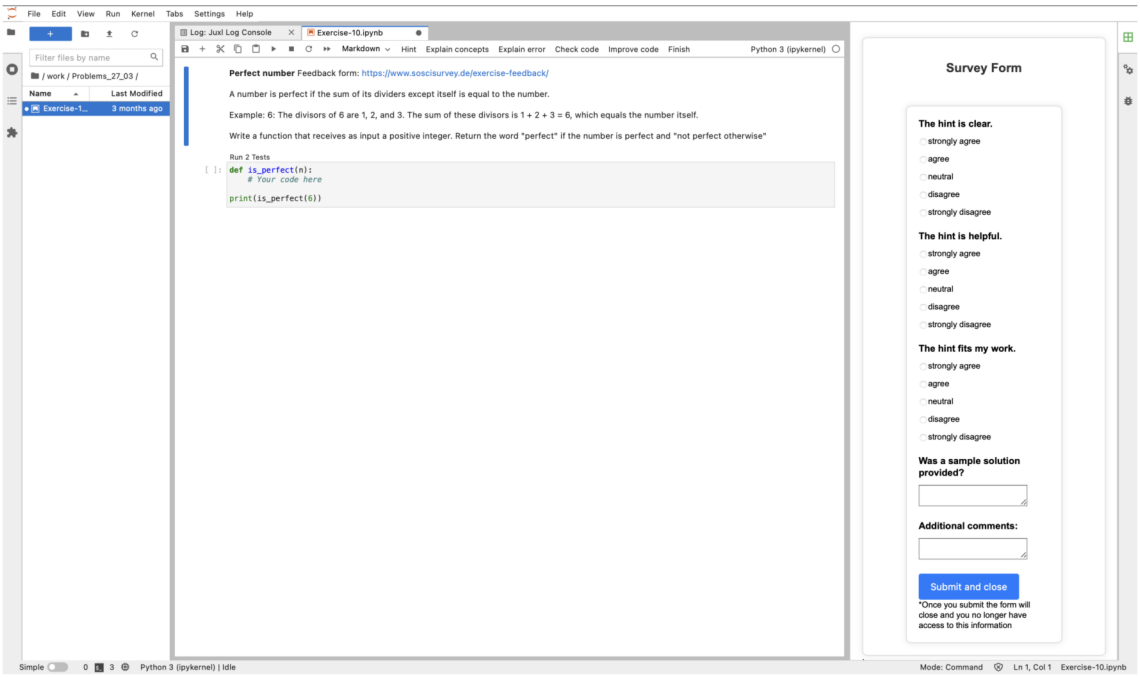


Figure 4.2: Example of the Feedback form in the context of the notebook

Students have access to various assistance buttons while developing their solutions. Each time students press one of these buttons, they are prompted to complete a

feedback form (see Figure 4.2). This form is designed to gather the student's feedback on the response received from the API.
The feedback form includes three main questions that require responses. These questions aim to evaluate the assistance's clarity, usefulness, and relevance. Along with these questions, the form has two additional fields for detailed feedback. One field asks for any additional comments or suggestions the student might have, and the other field inquires whether the assistance included any solution code, which is against the intended guidelines.
It is important to note that the AI model may occasionally disregard the guidelines and provide code to solve the problem. This can undermine the learning process, as the goal is to guide students without directly giving away the answers.

Evaluation criteria

The specific questions on the feedback form will be discussed further. The motivation for using the three questions "The hint is clear", "The hint fits my work" and "The hint is helpful" in the study is grounded in the necessity to evaluate the effectiveness of AI-generated hints in educational settings comprehensively. These questions target distinct yet interconnected feedback quality aspects crucial for effective learning and teaching. These questions were used in a study [20] to gather comprehensive feedback from students, ensuring that the hints provided were understandable, contextually appropriate, and practically applicable. The findings underscore the importance of clarity, relevance, and helpfulness in educational feedback, guiding my research to adopt these criteria for evaluating AI-generated hints.

Clarity Evaluating whether "The hint is clear" is essential because clarity directly impacts a student's ability to understand and act upon the guidance provided. If hints are unclear, students may become confused, leading to frustration and a lack of progress. Clear hints ensure that students can follow the instructions and make the necessary corrections or improvements to their work. This feedback aspect is critical for fostering an effective learning environment, as previous research on educational technologies and AI feedback mechanisms highlighted.

Relevance Assessing whether "The hint fits my work" is vital to determine the contextual appropriateness of the feedback. A hint relevant to the specific task or problem at hand ensures that the advice is applicable and practical. Irrelevant hints can mislead students and waste valuable time. By ensuring that feedback is tailored to the student's current work, educators can enhance the learning experience and support more targeted learning outcomes. This focus on relevance is supported by studies on adaptive learning technologies, which emphasize the importance of contextually appropriate feedback.

Helpfulness The question "The hint is helpful" encompasses the overall utility of the feedback. Helpfulness integrates aspects of clarity and relevance while also considering the practical benefits of the hint in aiding the student's progress. A helpful hint not only guides the student towards the correct solution but also supports their understanding and retention of the underlying concepts. This holistic measure of feedback effectiveness is crucial for evaluating the impact of AI-generated hints on learning outcomes and student satisfaction.

Feedback evaluation

The use of a 5-point Likert scale for evaluating the questions is motivated by several key advantages that this method offers in educational research. Firstly, Likert scales provide a straightforward and reliable way to measure attitudes and perceptions, allowing for nuanced responses beyond a simple binary yes/no answer [14]. This granularity helps capture varying degrees of agreement or disagreement, which is essential for understanding subtle differences in student feedback. Secondly, the 5-point scale balances simplicity and depth, making it easy for respondents to use while still offering enough range to differentiate between different levels of satisfaction and clarity [11]. Thirdly, using a 5-point scale facilitates statistical analysis, enabling the application of various quantitative methods to assess the effectiveness of the hints provided by AI. An image representation of the survey form can be seen in Figure 4.2.

4.2 Juxl integration

As previously discussed in subsection 2.1.1, several additional extensions have been utilized. The Juxl extension has been employed to track and log modifications in the notebooks, while the Juxl Cut extension has been employed to add test cases to the code cells where students write their code. Juxl extension has been configured to save the xAPI statements into a Learning Locker [21].

4.3 Problem dataset

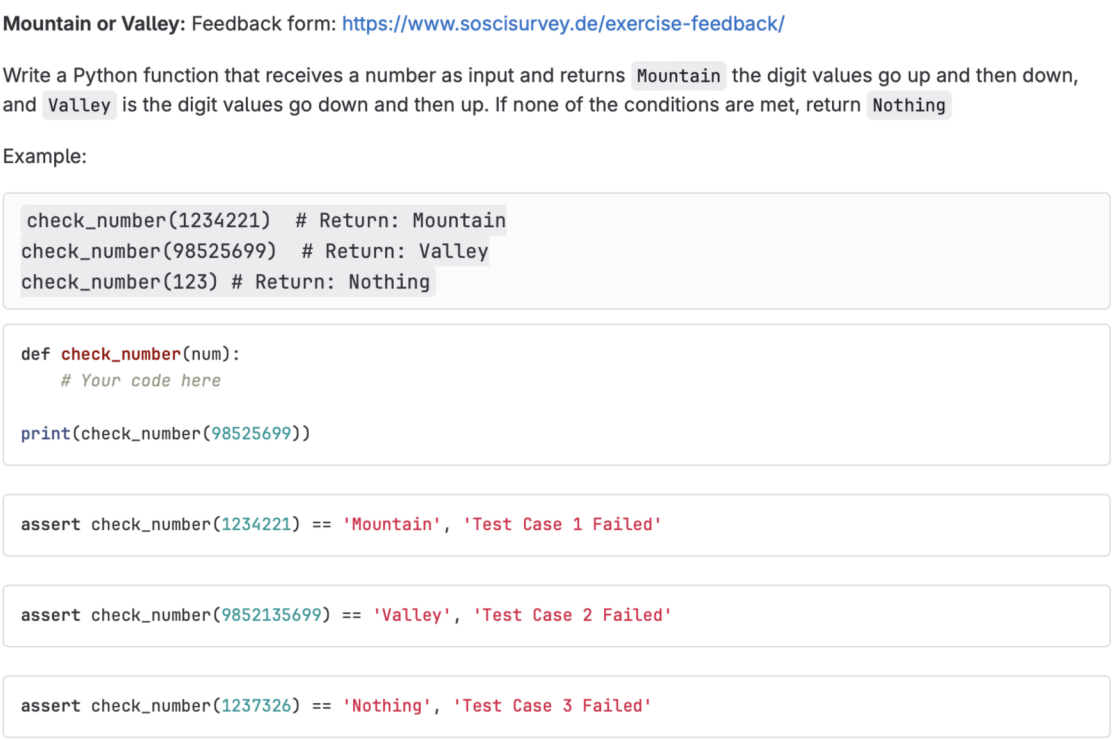
Several problems from the CodeBench dataset [2] were utilized in this research. Developed by the Institute of Computing at the Federal University of Amazonas, Brazil, this dataset comprises a comprehensive collection of logs from CS1 students' programming activities spanning from 2016 to 2023. Detailed records of students' interactions with programming exercises, including code submissions and modifications, are included. This dataset provided diverse problems on topics, including strings, lists, and mathematical questions.

Ten worksheets were devised for the experiment, each focusing on a single problem. Each worksheet adhered to the following structure, as illustrated in Figure 4.3:

1. Problem statement: This section provided a clear problem statement and a hyperlink to access a feedback form dedicated to resolving the specific problem (elaborated in subsection 4.4.2). Additionally, each problem statement included an illustrative example to assist students in testing their code.

2. Code cell: This section contained a predefined function name and its parameters, offering a starting point for students to work on the problem.

3. Test cells: These cells, although concealed from the students, served as a mechanism for assessing the correctness of their code.



Mountain or Valley: Feedback form: https://www.soscisurvey.de/exercise-feedback/

Write a Python function that receives a number as input and returns `Mountain` the digit values go up and then down, and `Valley` is the digit values go down and then up. If none of the conditions are met, return `Nothing`

Example:

```
check_number(1234221)  # Return: Mountain
check_number(98525699)  # Return: Valley
check_number(123) # Return: Nothing
```

```
def check_number(num):
    # Your code here

print(check_number(98525699))
```

```
assert check_number(1234221) == 'Mountain', 'Test Case 1 Failed'
```

```
assert check_number(9852135699) == 'Valley', 'Test Case 2 Failed'
```

```
assert check_number(1237326) == 'Nothing', 'Test Case 3 Failed'
```

Figure 4.3: This is an example of an exercise sheet the students have solved

4.4 Experimental setup

Phase 1: Initial Phase (3 days) Participants began by completing a pre-questionnaire (see subsection 4.4.1). Following this, they undertook a pretest comprising 5 problems to acclimate to the experimental environment and familiarize themselves with the problem format.

Phase 2: Experimental Phase (1 week) Participants were given different instructions in this phase based on their group assignment. The experimental group used a separate server for each phase and had access to the Feedback LLM extension. All participants were instructed to use the *Finish* button upon completing each worksheet and to utilize the provided test cases for testing their code. During this period, they were required to solve seven problems, released in sets of 2 every two days, with the final three problems released together.

Phase 3: Final Phase (1 week) In the final phase, participants solved the last three problems, divided into sets of 2 and 1, without the assistance of the Feedback LLM extension. Upon completing these tasks, participants were required to complete a post-questionnaire by the end of the week.

4.4.1 Pre and post questionnaires

Pre-questionnaire

The pre-questionnaire employed in this research aimed to gather comprehensive insights into the participants' experiences and attitudes toward learning Python programming. Key questions included rating the clarity, relevance, and helpfulness of the hints provided by the AI using a 5-point Likert scale. Participants were asked to evaluate their confidence in solving programming tasks, proficiency with Python, and familiarity with online programming resources. Additionally, the survey explored their perceptions of AI in education, their motivations for learning programming, and their satisfaction with traditional learning tools. This structured approach thoroughly assessed the participants' learning processes and the effectiveness of AI-generated hints.

Post-questionnaire

Two distinct post-surveys were administered to evaluate participants' experiences in both the experimental and control groups. The participants in the experimental group, who received AI-generated feedback during their programming tasks, were asked to provide detailed feedback on the AI assistance they received. They were prompted to assess the effectiveness, helpfulness, and clarity of the AI-generated feedback, as well as its impact on their confidence and understanding of programming concepts. Additionally, they were asked to compare the AI feedback to human instructor feedback and other learning resources they have used.

In contrast, the control group participants, who did not receive AI-generated feedback, were asked to describe their overall experience with the programming tasks. They rated their confidence in programming skills before and after the tasks, their preference for receiving feedback in future programming tasks, and their perceptions of incorporating AI-generated feedback into formal education.

4.4.2 Exercise feedback forms

After each of the ten exercises and the initial pretest, participants were required to complete an exercise feedback form to evaluate their experiences. This form asked participants to identify the exercise they were reviewing and to provide their thoughts on the proposed exercise. They rated the complexity of the exercise on a scale from "very easy" to "very hard" and assessed whether they learned something from solving the exercise. Additionally, participants compared the actual time taken to solve the exercise with their initial expectations. They were also encouraged to provide qualitative feedback on the content and their overall experience. This structured feedback aimed to gather detailed insights into the participants' learning processes and the effectiveness of the exercises.

4.4.3 Participants

A cohort of 30 subjects was initially recruited for this experiment. However, 8 participants withdrew, resulting in a final sample size of 22 participants. Before the commencement of the experiment, participants were randomly assigned to either the control group or the experimental group. Each participant was assigned a unique identifier in the format 'user_xxxx,' where 'x' represents a digit. To ensure comparable conditions, all participants were from the same class at the Technical University of Cluj-Napoca, possessing similar backgrounds and knowledge levels.

Upon verifying the collected data, it was found that certain students did not utilize the Finish button in most instances. Subsequent verification confirmed that these students had not completed the exercises and were thus excluded from the experiment. Additionally, two other students were omitted because they did not participate in the experiment at all. After excluding students who only participated in the Pre-test, the user base consisted of 22 individuals. In some cases, students merely forgot to press the Finish button. Figure 4.4 represents with green the users who clicked the Finish button on the specified notebook, otherwise in white.

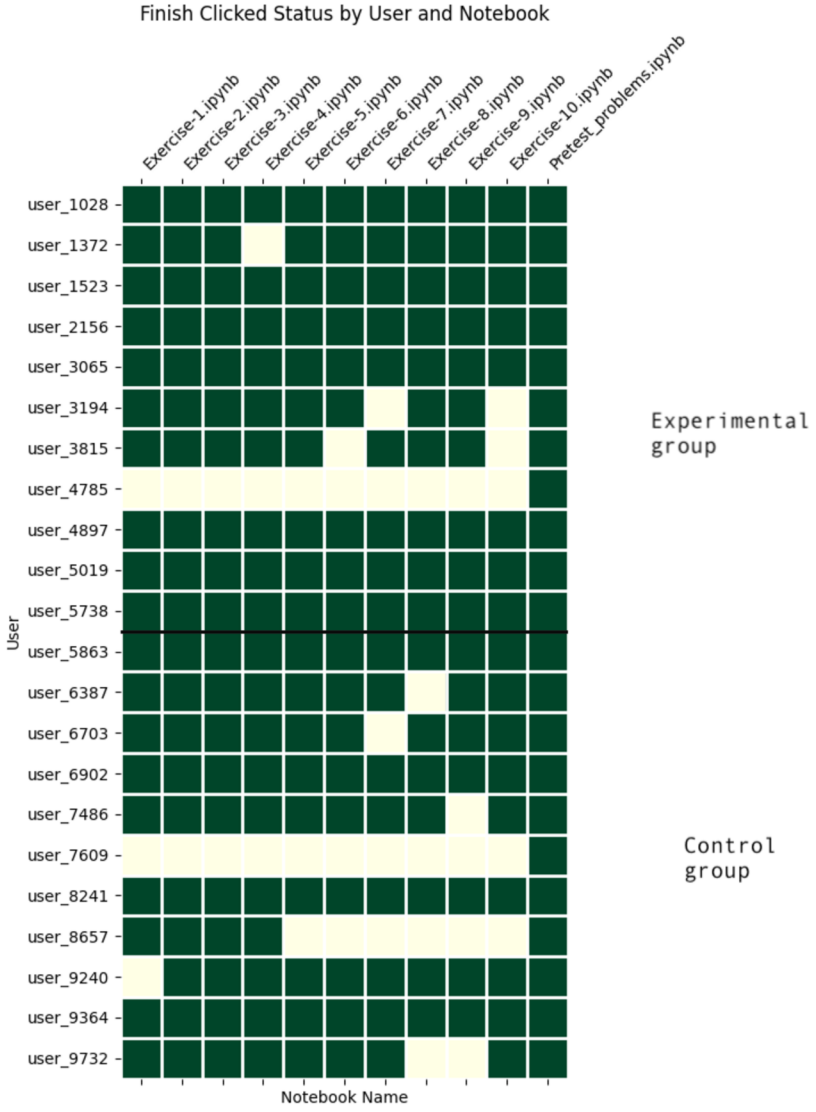


Figure 4.4: This figure represents whether the users clicked the Finish button. The green boxes mark that the button was pressed, while the white ones represent the button was not pressed. The users in the top half of the figure represent the experimental group, and the bottom half represents the control group.

An important mention: after the 8 participants withdrew, the groups remained balanced with 11 participants each. No participants were reassigned to a different group.

The study had 14 male participants and eight female participants, with a median age of 19 years old, studying for a Computer Science bachelor's degree. All participants indicated that they frequently use technology for learning.
From the self-reported levels of skills and confidence, the following can be reported:

Programming Skills Participants were asked to rate their programming skills on a scale from 1 to 5, with 1 being beginner and 5 being expert. The distribution of responses is shown in Table 4.2. One person reported their programming skills as beginner, five reported their skills as a lower intermediate, eleven reported intermediate skills and five upper intermediate, while none of them reported their skills as expert.

Rating	Frequency
1 (Beginner)	1
2	5
3	11
4	5
5 (Expert)	0

Table 4.2: Programming Skills Frequency

Confidence Level Participants rated their confidence in solving programming tasks independently on a scale from 1 to 5, with one being not confident and 5 being very confident. The distribution of responses is shown in Table 4.3. On a confidence level, one person reported lower confidence, eleven reported medium confidence in solving tasks independently, eight of them reported a higher lever of confidence and two reported as being very confident.

Rating	Frequency
1 (Not Confident)	0
2	1
3	11
4	8
5 (Very Confident)	2

Table 4.3: Confidence Level Frequency

Python Skills Participants rated their programming skills with Python on a scale from 1 to 5, with 1 having no experience and 5 being expert. The distribution of responses is shown in Table 4.4. When it comes to rating their Python programming skills, more than half (14 participants) reported having no experience, seven reported a little experience and one of them reported high skills.
Overall, the data suggests that while participants have general programming experience and moderate confidence in their abilities, there is a clear need for improvement in Python-specific skills. This indicates that future training programs should emphasize Python proficiency to better equip participants for related tasks.

Rating	Frequency
1 (No Experience)	14
2	7
3	0
4	1
5 (Expert)	0

Table 4.4: Python Skills Frequency

4.4.4 Research Questions

This thesis aims to explore the effectiveness of using ChatGPT as a helper tool for novice programmers to enhance their learning outcomes. The following research questions guide the research:

Research Question 1: Help Buttons What types of help provided by ChatGPT do students consider most effective for their learning, and which types are utilized more frequently versus less frequently?

Research Question 2: Time Efficiency Does the availability of help from ChatGPT lead to improved time efficiency in completing programming tasks among students, and do these time efficiency benefits persist in the final set of problems when the help is no longer available?

Research Question 3: Self-Satisfaction and Learning To what extent do students perceive that they have learned from using the help provided by ChatGPT, and is there a significant difference in self-reported learning improvements between students who used ChatGPT and those who did not?

4.4.5 Variables

The study investigates several variables to understand the impact of using ChatGPT on programming education. The **independent variable** is the use of ChatGPT, which is enabled for the experimental group and disabled for the control group.
The **dependent variables** include the task completion time, which is used to assess efficiency, and satisfaction with ChatGPT as a learning tool, which measures the users' contentment with the tool. Additionally, the confidence level of participants is measured both before and after the experiment to assess changes in self-efficacy. Programming skills are also evaluated through ratings to determine the skill levels of participants before and after the experiment.
Control variables are included to ensure the comparability and validity of the results. These control variables consist of participant demographics such as age, gender, and educational background. Furthermore, the field of study is controlled by ensuring that all participants come from similar academic backgrounds, which helps to manage variability in programming knowledge.

Chapter 5 Results and Analysis

This chapter presents the findings from the study, which aimed to evaluate ChatGPT's effectiveness as a helper tool for novice programmers. The analysis covers various aspects, including the usage patterns of help buttons, task completion times, correctness of solutions, self-reported satisfaction, and learning outcomes. section 5.1 presents the analysis of the button usage and received feedback. section 5.2 presents the findings related to the time component of the experiment, providing insights into the completion times of the students as well as statistical analysis and discussion. Lastly, section 5.3 dives into the students' perception of their abilities and newly gained knowledge.

The data comprises metrics such as button presses and feedback forms. Participants interacted with different help features, such as the Hint, Explain Error, Improve Code, Explain Concepts, and Check Code buttons. The frequency and distribution of these interactions provide insights into the preferred types of assistance and their perceived usefulness.

Performance differences between the experimental group, which had access to ChatGPT assistance, and the control group, which did not, are examined. Metrics such as task completion time, correctness of solutions, and self-reported confidence levels are analyzed to understand the impact of AI assistance on learning efficiency and outcomes.

Feedback on the AI tools, including satisfaction levels and qualitative comments, is analyzed to assess the overall user experience and identify areas for improvement. This section provides a detailed account of the findings and their implications for using AI in educational settings.

5.1 Button Usage and Feedback Analysis

This section will analyze the usage patterns of help buttons and the feedback provided for the AI assistance features. In the Table 5.1 and Table 5.2, the mapping of button usage is done by button type and by exercise. Out of 88 button presses, only 65 feedback forms were submitted by the students.

Some students had a higher preference for some buttons than others; the Check Code button was used the most with a number of 30 presses, second the Hint button with 22 presses, and third the Explain Error button with 21 presses. The least used are the Improve Code button, which has 11 presses, and the Explain Concepts button, which has 4 presses. Figure 5.1 is a stacked bar chart representing the usage of each button across the students. Some students used the available help, with uses over 16, and

Metric	Count
Total number of presses	88
Total number of feedback forms submitted	65
Hint Button presses	22
Explain Error Button presses	21
Improve Code Button presses	11
Explain Concepts Button presses	4
Check Code Button presses	30

Table 5.1: Summary of Button Usage and Feedback Forms

Exercise	Hint	Explain Error	Improve Code	Explain Concepts	Check Code
1	8	2	1	1	6
2	3	3	1	1	7
3	0	3	2	1	3
4	0	0	0	0	6
5	0	5	0	0	1
6	5	4	1	0	3
7	2	0	1	0	3
9	3	2	3	1	1
10	1	2	2	0	0

Table 5.2: Help Buttons Usage Across Exercises

others tried once or twice, but all of the students in the experimental group tried the help at least once.

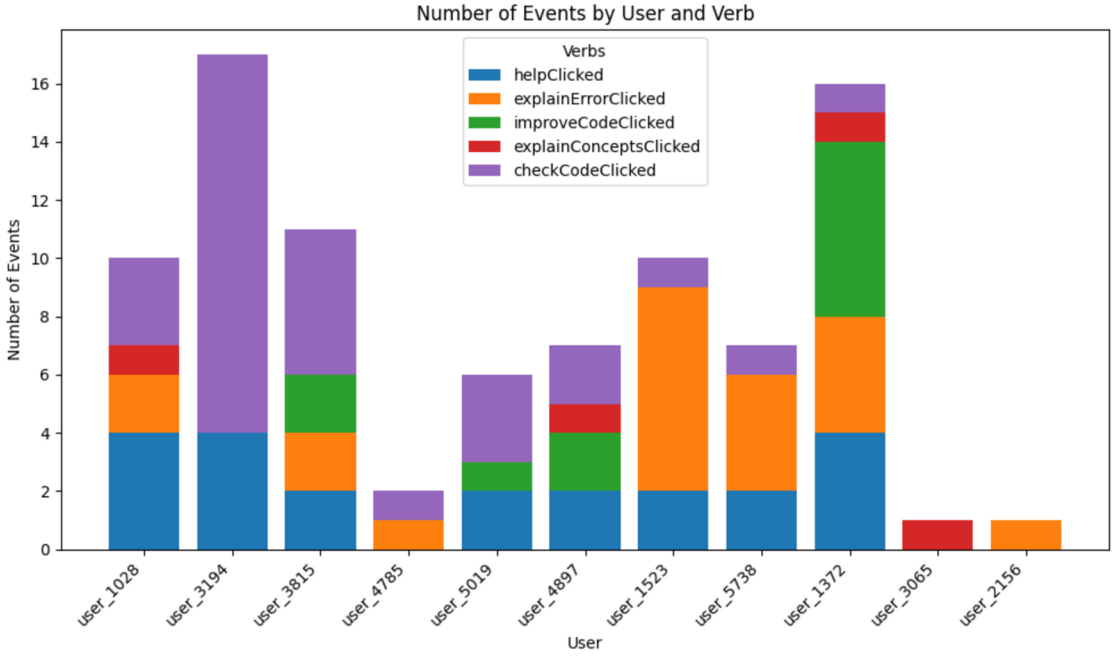


Figure 5.1: Stacked bar chart representing how many times students asked for help by pressing the buttons. A color represents each button.

This presentation includes a detailed breakdown of the feedback and ratings for the different help buttons, as well as comparative tables for clarity.

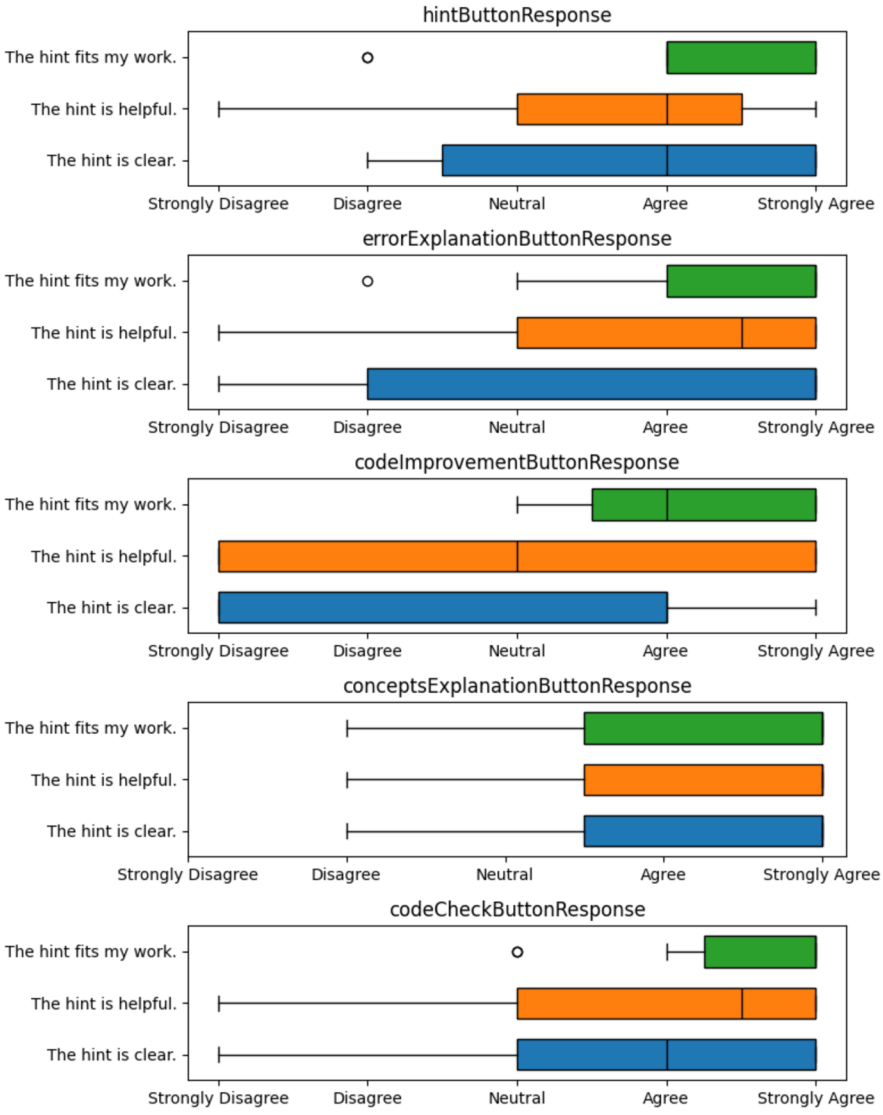


Figure 5.2: Comparison of the buttons with their respective feedback from the students

Hint button feedback: Table 5.3 The Hint button received mixed feedback from students, with clarity rated highly but helpfulness rated lower. Comments appreciated the guidance steps rather than direct solutions. Some students found the guidance straightforward and easily understandable.

Error Explanation button feedback: Table 5.4 Feedback for the Error Explanation button indicated a need for more detailed examples, particularly regarding syntax usage. Some students found the hints not helpful, indicating possible gaps in error explanation clarity.

Rating Criteria	Average Rating
The hint is clear	4.2
The hint is helpful	3.53
The hint fits my work	3.87
Total Average Rating	3.87

Table 5.3: Hint Button Feedback Ratings

Rating Criteria	Average Rating
The hint is clear	4.23
The hint is helpful	3.83
The hint fits my work	3.69
Total Average Rating	3.92

Table 5.4: Error Explanation Button Feedback Ratings

Code Improvement button feedback: Table 5.5 The Code Improvement button received the lowest ratings across all criteria. Comments revealed dissatisfaction with incorrect or unhelpful hints, though positive feedback was given when the advice improved code quality and encouraged thinking beyond the provided examples.

Rating Criteria	Average Rating
The hint is clear	4.14
The hint is helpful	3.0
The hint fits my work	2.86
Total Average Rating	3.33

Table 5.5: Code Improvement Button Feedback Ratings

Concepts Explanation button feedback: Table 5.6 The Concepts Explanation button received consistently high ratings across all criteria. Students found the explanations helpful in understanding problem requirements clearly.

Rating Criteria	Average Rating
The hint is clear	4.0
The hint is helpful	4.0
The hint fits my work	4.0
Total Average Rating	4.0

Table 5.6: Concepts Explanation Button Feedback Ratings

Code Check button feedback: Table 5.7 The Code Check button received the highest ratings for clarity and helpfulness. Comments indicate it often suggested aspects beyond the problem requirements, which was seen as helpful for considering additional edge cases. Specific praise was given for aiding in understanding method functionalities, such as deleting the last character from a string.

Rating Criteria	Average Rating
The hint is clear	4.64
The hint is helpful	4.05
The hint fits my work	3.95
Total Average Rating	4.21

Table 5.7: Code Check Button Feedback Ratings

Rating Criteria	Highest Rating	Lowest Rating
The hint is clear	4.64 (Code Check)	4.0 (Concepts Explanation)
The hint is helpful	4.05 (Code Check)	3.0 (Code Improvement)
The hint fits my work	4.0 (Concepts Explanation)	2.86 (Code Improvement)

Table 5.8: Highest and Lowest Rating Comparison

One aspect of the Code Check type of help worth mentioning is that the students found it very helpful because it provided them with corner cases to address to improve their code and make it more error-proof.
Additional Insights:

- Edge Case Considerations: Positive feedback often correlated with hints that prompted students to consider edge cases, enhancing their problem-solving skills.
- Syntax and Implementation Details: Requests for more detailed syntax and implementation guidance suggest that students value specific, actionable feedback over general advice.

5.1.1 Analysis and Discussion

Further, an analysis of student feedback on the support tools provided during the Python exercises will be presented. This section includes detailed evaluations of the Code Check button, Hint button, Error Explanation button, Code Improvement button, and Concepts Explanation Button. Each tool's effectiveness will be discussed regarding clarity and helpfulness, supported by specific student comments. Key insights related to edge case considerations and the importance of detailed syntax and implementation guidance will also be highlighted.
The analysis shows that the Code Check button received the highest average ratings for clarity and helpfulness (see Table 5.8), indicating that students found this type of assistance most beneficial. On the other hand, the Code Improvement button received the lowest ratings, suggesting room for improvement in providing actionable and relevant hints.
The comments also reflect the varying degrees of satisfaction and usefulness of the hints provided. For instance, while some students appreciated the detailed advice on code improvement (e.g.: *"helped me improve my code and think of other examples that were not given in the problem.", "it has helped me to understand exactly what the problem wants me to do"*), others found specific hints to be unhelpful or incorrect (e.g.: *"This hint is straight up wrong"*).

Mixed Feedback
Step-by-Step Guidance

Hint Button Response The Hint button received mixed feedback, with high ratings for clarity but lower ratings for helpfulness. This suggests that while students understood the hints, they did not always find them directly applicable or useful in solving their problems. Positive feedback often highlighted the value of step-by-step guidance rather than direct solutions. This preference for incremental support indicates that while hints are clear, they need to be more aligned with the students' immediate problem-solving needs to enhance their perceived usefulness.

> "very helpful and understandable for everybody"

> "helpful and straight forward"

Need for More Detailed Examples

Error Explanation Button Response While the Error Explanation button was rated highly for clarity, its helpfulness rating was comparatively lower. This discrepancy highlights a critical area for improvement: the need for more detailed examples, especially regarding syntax usage. Some students found the hints provided by this button insufficient, indicating potential gaps in the error feedback's clarity and comprehensiveness. Enhancing the detail and applicability of these examples could bridge this gap, making error explanations more useful for students.

> "I think it would have been helpful if I could've seen how the append function is used(how I should write it, the syntax)"

Lowest Ratings
Need for Relevance

Code Improvement Button Response The Code Improvement button had the lowest ratings across all criteria, reflecting widespread student dissatisfaction. Comments pointed to incorrect or unhelpful hints as the primary source of discontent. Despite this, there were instances of positive feedback when the advice provided enhanced code quality and encouraged students to think beyond the provided examples. This suggests that while the potential for helpful feedback exists, the button needs significant refinement to offer relevant and accurate suggestions consistently. Improving the precision and applicability of the feedback could increase its value to students.

> "it offers great advice in improving the code"

> "helped me improve my code and think of other examples that were not given in the problem"

> "This hint is straight up wrong"

Consistent High Ratings
Clear Explanations

Concepts Explanation Button Response The Concepts Explanation button received consistently high ratings across all criteria, underscoring its effectiveness in helping students grasp problem requirements. The clarity of the explanations provided by this button was particularly appreciated, suggesting that it breaks down complex concepts into understandable parts. This button's ability to deliver concise and coherent explanations appears to be instrumental in enhancing student comprehension and application of coding principles.

> "it has helped me to understand exactly what the problem wants me to do"

Code Check Button Response The Code Check button received the highest ratings for clarity and helpfulness, indicating that students found the feedback relevant and easy to understand. The positive response suggests that this button effectively meets student needs by providing comprehensive and detailed feedback. Comments revealed that students appreciated the button's ability to offer insights beyond the immediate problem requirements, including considerations for additional edge cases. This broader context likely enhances students' understanding of their solutions and improves their problem-solving skills.

Highest Ratings

Detailed Feedback

> "it is great, suggesting tho some aspects that the problem does not require."

> "it made me take more cases into consideration"

Additional Insights Positive feedback often correlated with hints that prompted students to consider edge cases, indicating that such prompts are valuable in developing students' problem-solving abilities. Additionally, the recurring request for more detailed syntax and implementation guidance highlights students' need for specific, actionable feedback. Providing more thorough and precise explanations of syntax and implementation details could significantly enhance the overall effectiveness of these support tools.

Edge Case Considerations

Detailed Syntax Guidance

Conclusion The analysis reveals that while the Code Check and Concepts Explanation buttons are highly effective, the Error Explanation, Hint, and Code Improvement buttons can be improved. Addressing the identified issues—such as the need for more detailed examples, clearer guidance, and more relevant feedback—could enhance the overall utility and student satisfaction with these support tools. By refining these elements, educators can better support students in developing robust programming skills.

5.2 Time performance

This section will present the findings related to the time variable. To determine the total time a student spent solving an exercise, the following steps were performed: All xAPI statements were parsed and organized into files, each representing a specific user and exercise. These statements include timestamps indicating when specific actions occurred. The total time spent on a notebook was calculated by summing all the time intervals between events. Since there are events representing the opening and the closing of a notebook, the time period between closing a notebook and opening it was not counted (since it was not active working time). Otherwise, if there are different types of events, the time between them is added up, only if there is a time difference smaller than 20 minutes. Similarly, Price et al. [19] define in their study a session as a series of events with no more than a 20-minute gap between any two consecutive events.

Some students reported using a different environment to write their code. This might have influenced their total time, resulting in a lower amount of time registered. Since the students did not work in a controlled environment but in their own time, the experiment aimed to reproduce real-life situations, where interruptions appear and the students' focus might not be 100% on the task.

Pre-test time analysis As a baseline, a pre-test with five problems has been solved by all of the participants. None of the participants had access to help and they solved the exercises in their own time. To assess whether there are any differences between the two groups several analysis tests were employed. The Shapiro-Wilk test was performed for both groups to assess the normality of the data. For the experimental group, the Shapiro-Wilk test indicated that the data is normally distributed ($p = .209$). For the control group, the Shapiro-Wilk test indicated that the data is normally distributed ($p = .223$). Given the normality of the data, an Independent T-test was performed to evaluate whether there is any significant difference between the experimental group ($M = 2563.0$, $SD = 713.6$), and the control group ($M = 3609.6$, $SD = 2264.6$) timings before the start of the experiment. The test results indicate that there are no significant differences ($t(20) = -1.393$, $p = .178$).

For the 5.3(b) and 5.3(a) were considered the notebooks in which the users reported finishing the exercise. The definition of finished, in this case, is given only by pressing the *Finish* button on the notebook interface.

5.3(a) represents the times considered for the first group of exercises (1-7), while the 5.3(b) considered only the last three exercise sheets: exercises 8, 9, and 10. The distinction is made due to the nature of the experiment, to separate the exercises in which students had access to the help buttons and in which they did not - as is the case for the last three exercises.

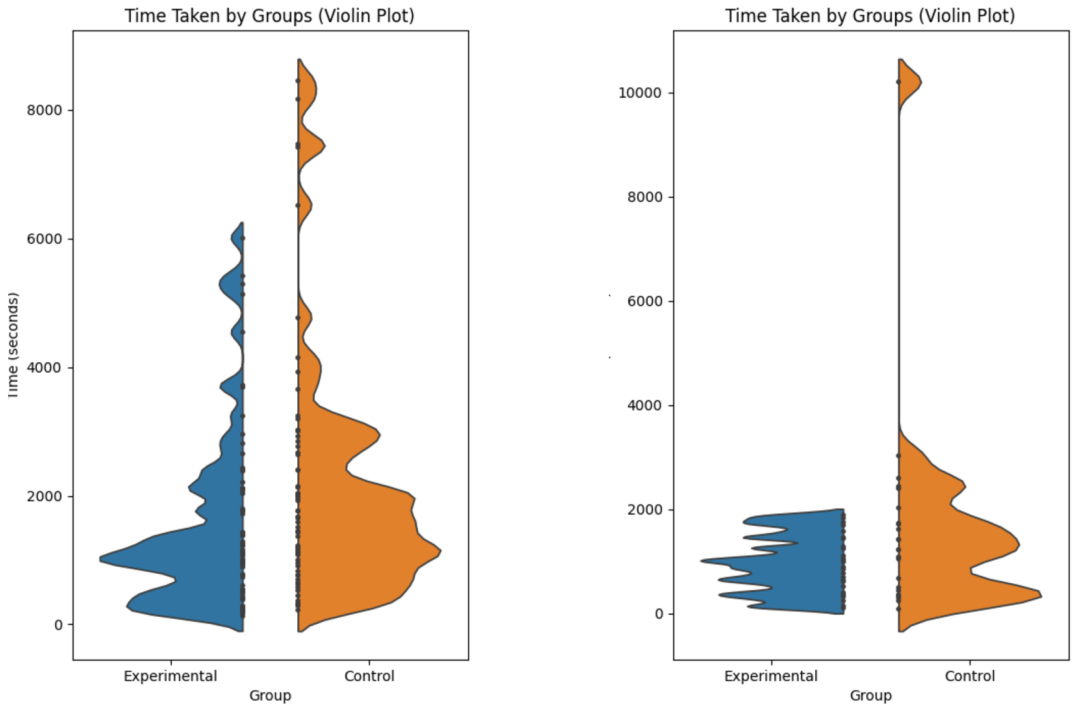
Next, the results of the data analysis will be presented in two sections. The first section will contain exercises 1 to 7, and the second section will contain exercises 8 to 10.

5.2.1 Statistical Analysis - Exercises 1-7

In these exercises, the students from the experimental group had access to the help tools. A graphical representation of the times can be seen in 5.3(a).

Normality Test

The Shapiro-Wilk test was performed for both groups to assess the normality of the data. For the experimental group, the Shapiro-Wilk test indicated that the data is not normally distributed ($p = 2.621 \times 10^{-7}$). For the control group, the Shapiro-Wilk test indicated that the data is not normally distributed ($p = 2.689 \times 10^{-8}$).



(a) Violin chart representing the time students took to solve the first seven exercises.

(b) Violin chart representing the time students took to solve the last three exercises.

Figure 5.3: Comparison of the time students took to solve the first seven exercises and the last three exercises.

Mann-Whitney U Test

Given the violation of the normality assumption, the Mann-Whitney U test was also performed to evaluate whether the usage of help influences the time the students take to solve the exercises. The Mann-Whitney U test indicated that the experimental group took significantly less time in solving the exercises than the control group ($U = 1662.0$, $p = .019$).

Homogeneity of Variances Test

Levene's test was conducted to assess the equality of variances between the two groups. It indicated that the variances are equal across the groups ($p = .202$).

Conclusion

The non-parametric tests indicate that there is a significant difference in the times taken to solve the exercises between the two groups.

5.2.2 Statistical Analysis - Exercises 8-10

A graphical representation of the times can be seen in 5.3(b).

Normality Test

To assess the normality of the data, the Shapiro-Wilk test was performed for both groups. For the experimental group the Shapiro-Wilk test indicated that the data is normally distributed ($p = .371$). For the control group the Shapiro-Wilk test indicated that the data is not normally distributed ($p = 8.207 \times 10^{-7}$).

Mann-Whitney U Test

Given the violation of the normality assumption, the Mann-Whitney U test was also performed to evaluate whether the previous usage of help influences the time the students take to solve the exercises. The Mann-Whitney U test indicated that there is no significant difference between the two groups when solving the problems without any help ($U = 234.0$, $p = .196$).

Homogeneity of Variances Test

Levene's test was conducted to assess the equality of variances between the two groups. It indicated that the variances are equal across the groups ($p = .095$).

Conclusion

Both parametric and non-parametric tests indicate that there is no significant difference in the times taken to solve the exercises between the two groups. Despite the data's non-normality, the tests consistently show no significant difference, suggesting that the two groups' performance is statistically similar.

5.2.3 Discussion

The findings highlight the immediate efficiency benefits of using ChatGPT as a support tool in programming tasks. For exercises where assistance was available, students completed tasks more quickly, reflecting the tool's ability to provide timely and relevant help. However, the transition to unassisted tasks did not show a sustained improvement, indicating that the tool's impact on developing independent speed in problem-solving requires further investigation.

While the efficiency gains are evident, it is important to consider the implications on learning and understanding. The assistance from ChatGPT, while beneficial in reducing task completion time, must be balanced to ensure that students are not overly reliant on the tool. The findings align with previous research by Pankiewicz and Baker [18], which highlighted the potential risk of students becoming dependent on AI-generated feedback. Ensuring that students develop their problem-solving skills remains a priority, and future implementations should consider integrating mechanisms to encourage independent thinking.

5.3 Self-reported levels of improvement

This section deals with the responses the students gave after solving each exercise. They were asked to fill in a questionnaire related to their view on the current exercise, rating the complexity, their view on the amount of time it took to solve it, and the outcome of their learning journey - whether they learned some new concepts and what are they.
Moreover, this section will examine the pre- and post-surveys, which contain valuable information about the students' learning outcomes, their views on the provided assistance, and their overall impressions of AI usage in education.
For this section, the answers from all the participants are considered.

5.3.1 Students' view of the exercises

After completing each exercise, the users were asked to rate its complexity, whether they considered they learned something by solving it, and the amount of time it took them to solve it compared to their initial impression. In addition to the rating questions, they were asked to describe what they had learned by solving the exercise.

Complexity ratings

Figure 5.4 represents the students' responses to the question ***How do you rate the complexity of the proposed exercise?***. Table 5.9 represents the mean ratings of the students for the question. The students tended to rate most of the exercises as medium complexity, and the means were similar across the exercises when comparing the two groups. The experimental group found the exercise harder than the control group in 3 exercises out of 10.
Across all exercises, the statistical analysis consistently shows no significant differences in perceived complexity between the experimental and control groups. While some exercises show minor numerical differences in mean ratings, these are not statistically significant. Both groups found the exercises similarly challenging, suggesting that the interventions did not substantially affect their perception of exercise complexity.

Table 5.9: Mean Ratings of Exercise Complexity

Exercise	Mean Rating (Experimental)	Mean Rating (Control)
Pre-test	1.82	1.45
1	3.22	3.27
2	2.33	2.50
3	2.40	2.09
4	2.00	2.09
5	2.00	2.00
6	3.20	3.60
7	1.80	1.36
8	2.00	2.10
9	3.18	3.20
10	1.60	1.55

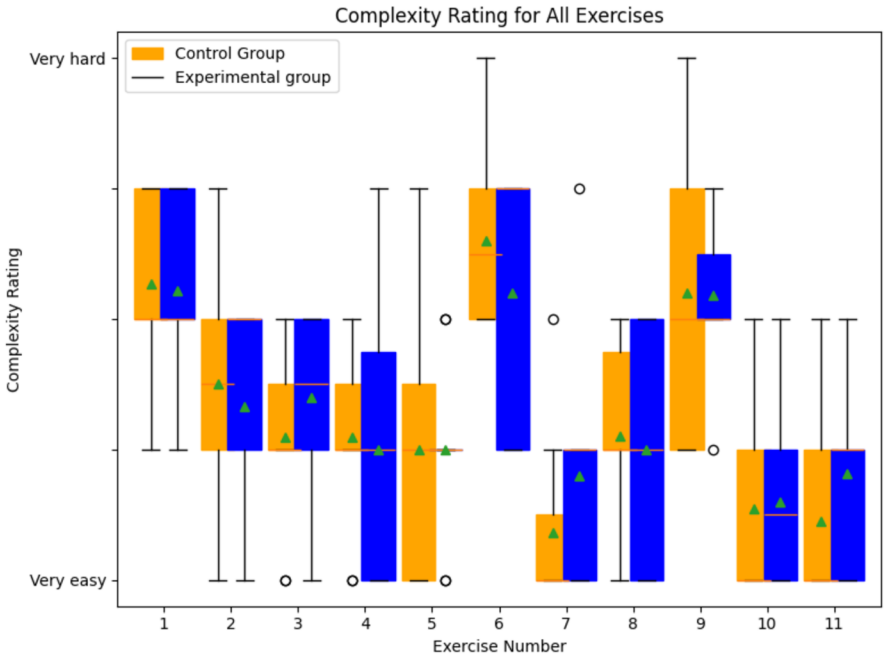


Figure 5.4: Responses for the question: How do you rate the complexity of the proposed exercise?

Table 5.10: Shapiro-Wilk and Mann-Whitney U Test p-values

Exercise	Shapiro-Wilk p-value		Mann-Whitney U p-value
	Experimental	Control	
Pre-test	0.0183	0.0005	0.23323
1	0.0254	0.0091	0.93471
2	0.0052	0.1062	0.81753
3	0.0085	0.0184	0.31664
4	0.0739	0.0184	0.68080
5	0.0219	0.0637	0.84975
6	0.0002	0.0085	0.53940
7	0.0036	0.00004	0.18772
8	0.0060	0.0359	0.84118
9	0.0043	0.0456	0.85242
10	0.0085	0.0025	0.87503

The analysis across various exercises indicates no statistically significant difference in the perceived complexity between the experimental and control groups. For most exercises, both groups exhibited non-normality in their ratings, as indicated by the Shapiro-Wilk test ($p < .05$) (see Table 5.10). The Mann-Whitney U test suggested no significant difference in complexity ratings between the two groups ($p > .05$). Levene's test indicated that the variances were generally equal between the groups, with p-values mostly above 0.05 (see Table 5.11).

These findings suggest that the interventions did not substantially affect the students' perception of exercise complexity.

Table 5.11: Levene's Test and Interpretations

Exercise	Levene's Test p-value	Interpretation
Pre-test	0.73058	No significant difference
1	0.89476	No significant difference
2	1.0	No significant difference
3	0.45547	No significant difference
4	0.18664	No significant difference
5	0.21850	No significant difference
6	0.56545	No significant difference
7	0.44019	No significant difference
8	0.17688	No significant difference
9	0.02261	No significant difference
10	0.82102	No significant difference

Learning ratings

Figure 5.5 represents the students' responses for the question ***Do you consider you learned something by solving the proposed exercise?***. Table 5.12 presents the means across exercises for the two groups.

The analysis of the mean ratings for learning perception across various exercises reveals that, in general, there are no significant differences between the experimental and control groups. The data suggests that both groups had similar experiences and perceptions of learning something new from the exercises, as indicated by the consistently non-significant p-values in the Mann-Whitney U tests (see **??**) and Levene's tests (see **??**) for most exercises.

Specifically, the results for Exercise 2 are noteworthy, where the control group reported a significantly higher perception (mean of 4.17) of learning compared to the experimental group (mean of 2.44). This suggests that, in this particular case, the intervention provided to the experimental group did not enhance their learning perception as effectively as the approach experienced by the control group. However, this was an isolated instance, as the other exercises did not show such a significant difference.

Overall, these findings imply that the feedback intervention, while having potential benefits, did not consistently lead to a higher perception of learning among students in the experimental group compared to the control group. The intervention's impact may vary depending on the specific context or type of exercise, as evidenced by the variation in Exercise 2. Further research might explore the factors contributing to this variability and how the feedback mechanism can be optimized to enhance learning perceptions consistently across different exercises.

Time ratings

Figure 5.6 represents the students' responses for the question ***How would you rate the amount of time it took you to solve the task as compared to the initial impression?***. **??** represents the mean ratings for each of the exercises.

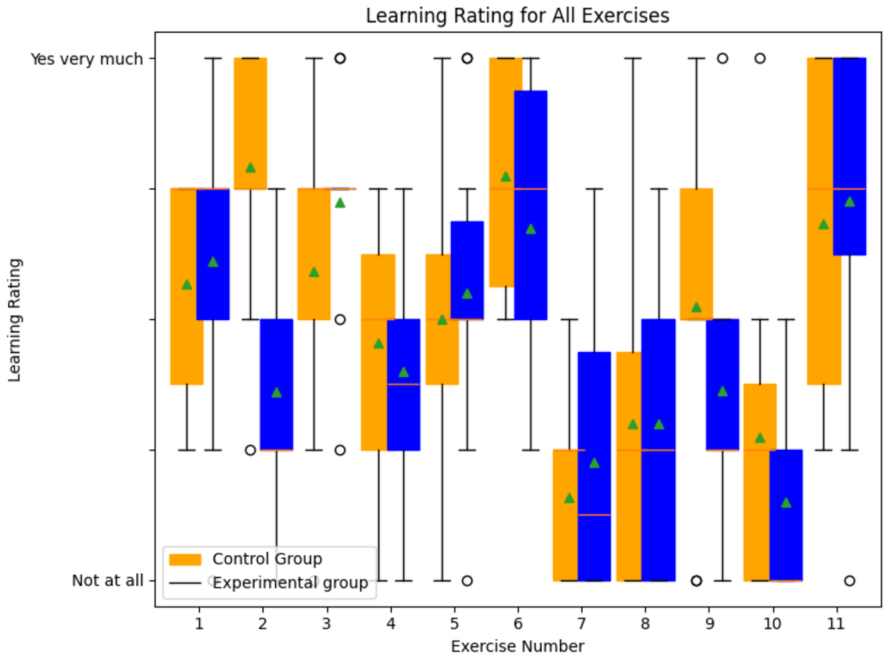


Figure 5.5: Responses for the question: Do you consider you learned something by solving the proposed exercise?

Table 5.12: Mean Ratings for Learning Perception

Exercise	Mean Rating (Experimental)	Mean Rating (Control)
Pre-test	3.91	3.73
1	3.44	3.27
2	2.44	4.17
3	3.90	3.36
4	2.60	2.82
5	3.20	3.00
6	3.70	4.10
7	1.90	1.64
8	2.20	2.20
9	2.45	3.10
10	1.60	2.09

The analysis of the mean ratings for the amount of time it took to solve the tasks, compared to the initial impression, reveals that there are no significant differences between the experimental and control groups. The experimental group generally reported feeling that they took less time compared to their initial impression, but this difference was not statistically significant across the exercises.

For most exercises, the Shapiro-Wilk tests indicated non-normality in at least one of the groups, yet the Mann-Whitney U tests consistently showed no significant differences between the two groups. Additionally, Levene's tests for homogeneity of variances confirmed that there were no significant differences in the perception of time taken to complete the tasks.

Table 5.13: Shapiro-Wilk and Mann-Whitney U Test p-values

Exercise	Shapiro-Wilk p-value		Mann-Whitney U p-value
	Experimental	Control	
1	0.4074	0.0012	0.65853
2	0.2485	0.0113	0.00338
3	0.0256	0.1405	0.23575
4	0.2449	0.1651	0.60607
5	0.2122	0.1696	0.76430
6	0.1240	0.0167	0.47713
7	0.0190	0.0062	0.76050
8	0.0060	0.0383	1.00000
9	0.0971	0.1269	0.16515
10	0.0014	0.0177	0.34355
11	0.0107	0.0164	0.70441

Table 5.14: Levene's Test and Interpretation

Exercise	Levene's Test p-value	Interpretation
1	0.53026	No significant difference
2	0.48595	Significant difference
3	0.40904	No significant difference
4	0.77535	No significant difference
5	0.79170	No significant difference
6	0.48249	No significant difference
7	0.20093	No significant difference
8	0.54795	No significant difference
9	0.83276	No significant difference
10	0.56809	No significant difference
11	1.0	No significant difference

These findings suggest that the feedback intervention did not significantly alter the students' perception of the time required to solve the exercises. Both the experimental and control groups had similar experiences regarding the time it took to solve the tasks, indicating that the intervention did not lead to a noticeable change in their time perception.

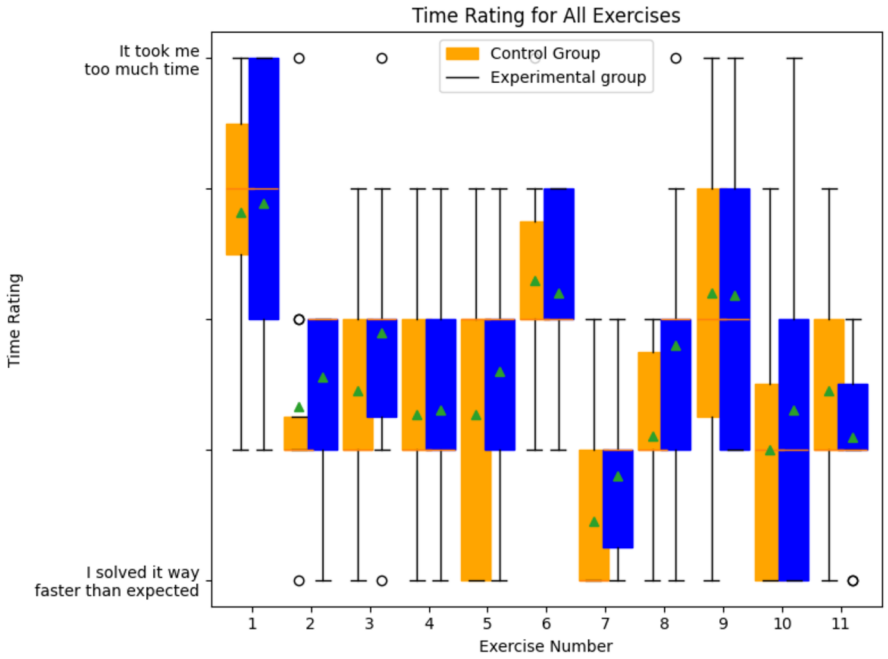


Figure 5.6: Responses for the question: How would you rate the amount of time it took you to solve the task as compared to the initial impression?

Table 5.15: Mean Ratings of Time Taken Compared to Initial Impression

Exercise	Mean Rating (Experimental)	Mean Rating (Control)
Pre-test	2.09	2.45
1	3.89	3.82
2	2.56	2.33
3	2.90	2.45
4	2.30	2.27
5	2.60	2.27
6	3.20	3.30
7	1.80	1.45
8	2.80	2.10
9	3.18	3.20
10	2.30	2.00

Table 5.16: Shapiro-Wilk and Mann-Whitney U Test Results

Exercise	Shapiro-Wilk p-value		Mann-Whitney U p-value
	Experimental	Control	
Pre-test	0.0184	0.1504	0.31638
1	0.1942	0.0317	0.93633
2	0.0009	0.0011	0.21410
3	0.3591	0.1504	0.31209
4	0.2869	0.1807	0.97027
5	0.1716	0.0149	0.54649
6	0.0251	0.0449	0.96723
7	0.0123	0.0005	0.19789
8	0.4788	0.0359	0.14961
9	0.0474	0.5745	0.97101
10	0.0866	0.0637	0.76848

Table 5.17: Levene's Test and Interpretation

Exercise	Levene's Test p-value	Interpretation
Pre-test	0.48766	No significant difference
1	0.88028	No significant difference
2	0.88155	No significant difference
3	0.84779	No significant difference
4	0.83136	No significant difference
5	0.40850	No significant difference
6	0.72219	No significant difference
7	0.84066	No significant difference
8	0.33056	No significant difference
9	0.76455	No significant difference
10	0.27793	No significant difference

5.3.2 What did students learn by solving the proposed exercise?

In the same questionnaire, the students were asked to describe what did they learn by solving the proposed exercise.
This section presents a summary of the learning feedback collected from both the experimental and control groups across all exercises. The responses have been grouped based on similar themes and concepts learned by the students. The exercises could be solved in multiple ways, using different approaches. Some students approached an easier manner, while others decided to use more complicated concepts. Due to this factor, each student's learning journey and reports are different.

Programming Syntax and Basic Operations Across several exercises, students in both groups gained insights into basic Python syntax and operations. They learned about integer division using the '//' operator, the importance of indentation in Python, and how to use loops and conditional statements effectively.

> "I learned that its easier to iterate through a number by transforming it into a string."

> "I refreshed my Python syntax skills after several years of only using C/C++."

> "I learned how to use the while loop in Python and practiced the indentation feature of the language."

String Manipulation String manipulation was a recurring theme where students learned various methods for handling strings. They discovered how to determine the length of a string using `len(string)`, replace characters with `replace()`, and reverse strings. Additionally, they explored string slicing and concatenation.

> "I learned how slicing in Python works."

> "The length of a string is determined by `len(string)`, you cannot assign a value to a character of a string."

> "I learned how to concatenate strings."

List and Array Operations Both groups learned about creating, manipulating, and iterating through lists and arrays. They learned to append items, remove elements, and sort lists. These exercises highlighted the differences between Python and other programming languages, particularly in handling lists.

> "I learned that even if I use the remove function, when iterating through an array some elements may be skipped."

> "I learned how to create a new vector and use lists in Python."

Mathematical and Logical Operations Students gained a deeper understanding of mathematical and logical operations in Python. They learned to use functions such as `max()`, `round()`, and `int()` for various calculations. Some exercises also reinforced the importance of understanding mathematical formulas and their implementation in code.

> "I've learned how to use the `round()` function."

> "I learned a new way to calculate faster the maximum of some numbers and how to display the results."

Dictionaries and Advanced Functions Some students explored how to use dictionaries, perform string splitting, and apply lambda functions in their solutions.

> "I learned a little bit about dictionaries in Python and about the lambda function."

> "I learned how to use dictionaries and how to split a string."

Error Handling and Debugging Several students reported learning about Python's error handling mechanisms, specifically using the try-except block.

> "I learned about the try-except block in Python."

Problem-Solving and Algorithm Development The exercises encouraged students to think critically and develop algorithms to solve problems. They learned to structure their code, anticipate potential issues, and implement efficient solutions.

> "I learned that I should have a structure in mind before I write the code to be sure that I understood the problem."

> "The exercise made me be more careful with all the cases that can occur, so my solution can be good for every one of them."

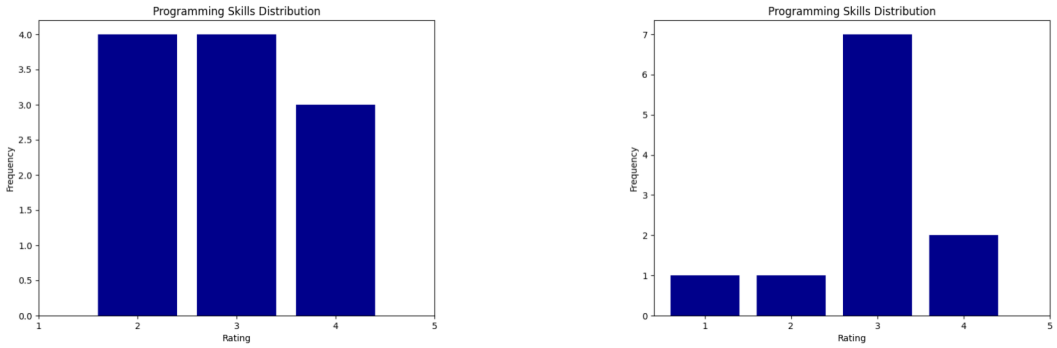
In summary, the exercises provided a comprehensive learning experience, covering a wide range of fundamental and advanced programming concepts. Both groups covered a broad range of topics including basic syntax, string manipulation, list operations, mathematical functions, and error handling. The experimental group reported more diverse and specific instances of learning advanced concepts like dictionaries and lambda functions. Also they often mentioned deeper insights into how to handle specific programming challenges and reflected on their problem-solving approaches. The control group focused more on fundamental programming skills and general coding practices. Only the experimental group mentioned learning about try-except blocks, indicating exposure to error handling concepts.

Based on the responses, the experimental group appears to have learned a wider range of advanced concepts and provided more detailed insights into specific programming challenges, while the control group focused more on fundamental programming skills and coding practices.

5.3.3 Pre- and post-survey confidence ratings

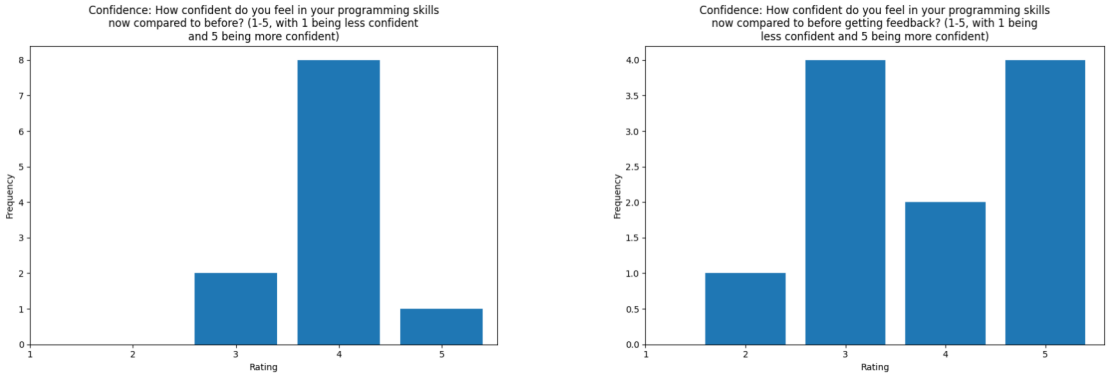
In both the pre-and post-surveys, students were asked to rate their confidence in their programming skills. The responses from the control group are shown in 5.7(a) and 5.8(a). In contrast, the responses from the experimental group are displayed in 5.7(b) and 5.8(b).

In the pre-survey the control group had four participants reporting as lower-intermediate, four participants as intermediate and 3 participants as advanced. The experimental group had one participant reporting as beginner, one as lower-intermediate, seven as intermediate and two as advanced. In the post-survey, the control group reported levels of the same to higher confidence, while the experimental group reported higher levels of confidence. Comparing the initial and final both groups report increasing confidence in their programming skills.



(a) Control Group ***(b)*** Experimental Group

Figure 5.7: Responses for the question **Rate your current programming skills (1-5, with 1 being beginner and 5 being expert)**



(a) Control Group ***(b)*** Experimental Group

Figure 5.8: Responses for the question **How confident do you feel in your programming skills now compared to before? (1-5, with 1 being less confident and 5 being more confident)**

Views on AI integration in education

The students in the experimental group were also asked about their views on the AI help they have received and their general views about the integration of AI in education. The ratings can be seen in Figure 5.9. Overall, the box plots reveal a generally positive reception to AI-generated feedback, with medians for most questions leaning towards the favorable end of the scale.

One interesting question to look at is A801_1, where the median for the responses is at the highest value - 5. This suggests a highly positive reception, with the majority of students willing to recommend this feedback system to peers. This indicates strong approval and perceived benefit of the feedback.

Legend: A201_01: Feedback: To what extent do you believe the feedback helped you in completing programming tasks more efficiently? (1-5, with 1 being not effective at all and 5 being very effective)

A202_01: Helpfulness: To what extent were explanations and assistance helpful for understanding programming concepts?(1-5, with 1 being not helpful at all and 5 being

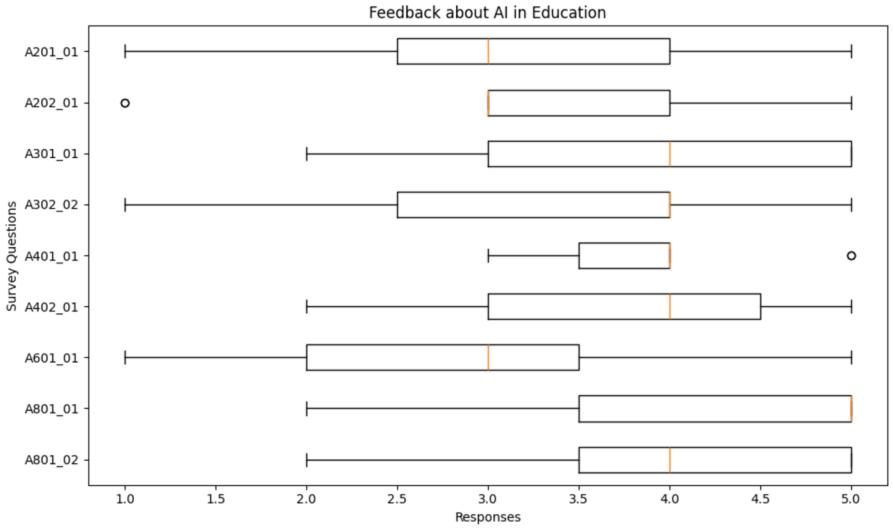


Figure 5.9: Several responses for questions related to the usage of AI in education

very helpful)
A301_01: Confidence: How confident do you feel in your programming skills now compared to before getting feedback? (1-5, with 1 being less confident and 5 being more confident)
A302_02: Understanding: To what extent did the feedback contribute to improving your understanding of programming concepts? (1-5, with 1 being not at all and 5 being very much)
A401_01: Preference: To what extent would you prefer getting feedback for future programming tasks? (1-5, with 1 being not at all and 5 being very much)
A402_01: Satisfaction: How satisfied are you with the support provided by the feedback in your learning journey? (1-5, with 1 being very dissatisfied and 5 being very satisfied)
A601_01: Impact: Did the feedback you received impact your problem-solving approach in programming tasks?
A801_01: Recommend: To what extent would you recommend the use of this type of feedback to other novice programmers for learning and problem-solving? (1-5, with 1 being not at all and 5 being very much)
A801_02: Recommend: Do you see potential for incorporating AI generated feedback into formal programming education? (1-5, with 1 being not at all and 5 being very much)

5.3.4 Feedback from the students

In the post-survey, the students from the control group were asked ***"How would you describe your overall experience with the programming tasks?"*** Here are the responses from them:

> "it was easy for me to code and to think how to solve them"
>
> "Some tasks were easy, but some of them were a bit challenging. Overall was all good and I'm glad that I took part of this."

> "It was a pleasant way of spending my time and learning new things about python"
>
> "It was interesting to see what new task was waiting for me; on the technical side, it also went smoothly"
>
> "Overall, I think this experience gave me a clear vision of how Python works and why it is considered a language for beginners."
>
> "It was an enjoyable experience for learning programming and Python"
>
> "While the tasks were easy and bite-sized daily challenges, I often found myself viewing them as a chore, rather than a learning opportunity"
>
> "They were really interesting and it was something new and a very good method of studying"
>
> "I had a fun experience. The task was clear and designated for beginners, and I believe it helped my grasp the basics of Python"
>
> "I really enjoyed it. I am happy that I accepted to take part of this. I learned a lot of new things."
>
> "Great overall fun and experience"

The feedback from the control group reveals a generally positive reception of the programming tasks. Many students described the tasks as a pleasant and enjoyable way to learn Python, appreciating the balance between easy and challenging problems. They found the tasks engaging and effective in imparting foundational programming knowledge, noting that the experience gave them a clear understanding of how Python works and why it is considered beginner-friendly. However, one student mentioned that the tasks occasionally felt like chores rather than learning opportunities, suggesting a potential area for improvement in maintaining motivation. Overall, the structured and novel approach to studying was well-received, with students expressing that the tasks were a fun and effective method for grasping the basics of Python. This positive feedback indicates that the programming tasks successfully facilitated learning and were appreciated for their educational value and engagement.
At the same time, the students in the experimental group were asked ***How would you describe your overall experience with feedback aid for programming?*** Here are the responses from them:

> "I really enjoyed this experiment. Learning a new programming language was an exciting experience, and the exercises you provided were exactly what we needed to grasp the basics of Python."
>
> "I did not use it that much, but the time I have made use of it the feedback was not helpful at all."
>
> "It was good and useful"
>
> "I enjoyed the experience and also i found it useful because it helps with understanding some things in python"
>
> "I feel like it wouldn’t have been necessary to put a feedback to every problem, overall, it was nice"

> "I think the feedback helped me learn or improve on different aspects because all feedback aid was clear and on point"
>
> "It was an interesting and unique experience"
>
> "didn’t use it that much"
>
> "It was a nice experience, well prepared and much work put into it. "
>
> "It wasn’t all that useful, I ended up figuring things out by myself or using standard resources on the Internet (not ChatGPT)"
>
> "I did in fact try to use it to see what it does and it actually helped me with the python syntax a lot plus it had good explanations on how to solve the problems so its really good for a beginner"

The feedback from the experimental group provides mixed insights into the overall experience with the feedback aid for programming. Many students appreciated the feedback, describing it as a useful and engaging tool that helped them understand Python better. Comments like "learning a new programming language was an exciting experience" and "the feedback helped me learn or improve on different aspects" reflect the positive impact of the feedback aid on their learning. Some students found the feedback particularly helpful for understanding Python syntax and problem-solving, indicating its value for beginners.
However, there were also critiques. A few students mentioned that they did not use the feedback aid extensively, and when they did, it was not always helpful. One student felt that feedback for every problem was unnecessary, while another preferred figuring things out independently or using other resources. These mixed responses suggest that while the feedback aid was beneficial for some, it did not meet the needs or preferences of all students. Overall, the feedback aid was appreciated for its clarity and usefulness by many, but its effectiveness varied among students.

Improvements

The students from the experimental group were also asked ***Are there specific aspects of the feedback that you think could be improved to better assist novice programmers?*** Here are the answers:

> "Nothing."
>
> "I think the feedback is as good as it will ever be, unless it actually shows you all/most of the relevant alternatives for solving a problem."
>
> "I think it’s good enough the way it is"
>
> "I don t think so"
>
> "i don t know"
>
> "I don’t think so"
>
> "Yes"
>
> "not really"
>
> "Not really, it has worked pretty good for me."

> "Novice programmers need more detailed feedback, including the explanation of certain programming concepts, where necessary"
>
> "I can’t think of anything at the moment"

The responses varied, but many students did not suggest significant changes. Several students felt that the feedback was already effective, with comments like "I think the feedback is as good as it will ever be" and "I don’t think so." Others echoed this sentiment, stating that they found the feedback sufficient as it was.
However, some students did suggest potential improvements. One student mentioned the need for feedback to show more relevant alternatives for solving problems, which could help in understanding different approaches. Another highlighted that novice programmers would benefit from more detailed feedback, including explanations of certain programming concepts where necessary. These insights suggest that while the existing feedback was generally well-received, enhancing its depth and variety could further support novice programmers in their learning process.

Study Design Feedback

Both groups were asked to give feedback on the study design and exercises: ***Do you have any feedback on the study design or the tasks assigned during the experiment?*** Here are their answers:

Control group:

> "they were perfect for beginners and for people learning the basics in python"
>
> "When the cases failed, it would be better to get an explanation why this thing happened."
>
> "Some of the latest tasks were much more easier than the ones in the beginning of the study. I think it would be nice to increase the difficulty over time so i can push my limits and to stimulate my brain harder:)"
>
> "The study design was quite easy to understand, although for some tasks I had to search on the internet for a more complex implementation of the simple examples that were provided"
>
> "No, everything was alright. Maybe I should mention that having an example of input and output for every problem helped a lot."
>
> "For me almost was perfect except I had some problems with the compiling and I had to copy paste from an external IDE the code to compile it"
>
> "I wish there was more teaching involved, rather than just blindly doing the exercises, although figuring it out by myself definitely made me remember things better"
>
> "I think at the end the problems should have been a little harder"
>
> "Not at all. The platform was easy to use and the tasks weren’t a problem. The number of task was also nicely given and well time spaced between each other"

"I think the tasks were very good for this experiment."

"All good"

Experimental group:

"I would have preferred to have the feedback option available throughout the entire experiment. The last link you provided did not include that option."

"I think the exercises were ok."

"I think they were quite good and not very difficult"

"I don t think so"

"no"

"Maybe the problems/exercises could introduce more programming concepts"

"No"

"sometimes more examples would have been useful"

"It were good tasks, very well presented and understandable."

"The tasks required a pre-existing set of programming and algorithm skills that novice programmers lack"

"The tasks were fine and contained about all the basics for programming, definitely some hard problems there for beginners i would say but overall helpful"

The control group generally found the tasks well-suited for beginners learning the basics of Python. Many praised the tasks for being perfect for novices and appreciated the examples of input and output provided. However, some students suggested improvements, such as receiving explanations when test cases failed and ensuring a gradual increase in difficulty to better challenge their skills. One student noted that while the study design was easy to understand, they often had to look up more complex implementations online, suggesting a need for more comprehensive teaching materials. Another highlighted issues with compiling code within the platform, indicating a technical area that could be refined. Overall, the feedback was positive, but it emphasized the need for more teaching support and a better progression in task difficulty.
The experimental group also had generally positive feedback but highlighted some specific areas for improvement. One student mentioned the desire for the feedback option to be available throughout the entire experiment, not just in the first part. There was a consensus that the tasks were appropriate and not overly difficult, but some students suggested introducing more programming concepts and providing additional examples to aid understanding. One student felt that the tasks required a level of programming and algorithmic knowledge that might be beyond novice programmers, suggesting a need to better align tasks with beginners' skills. Despite these points, the tasks were seen as well-presented and understandable, contributing to a positive learning experience.

Chapter 6 Conclusion and Future Work

This thesis aimed to explore the effectiveness of using ChatGPT as a helper tool for novice programmers to enhance their learning outcomes. The following research questions guided the research:

Research Question 1: Help Buttons What types of help provided by ChatGPT do students consider most effective for their learning, and which types are utilized more frequently versus less frequently?

Research Question 2: Time Efficiency Does the availability of help from ChatGPT lead to improved time efficiency in completing programming tasks among students, and do these time efficiency benefits persist in the final set of problems when the help is no longer available?

Research Question 3: Self-Satisfaction and Learning To what extent do students perceive that they have learned from using the help provided by ChatGPT, and is there a significant difference in self-reported learning improvements between students who used ChatGPT and those who did not?

RQ1: Help Buttons Students have interacted with multiple help buttons, and the Check Code button was the most used. The primary function of this button is to check the code against the problem requirements, find potential issues and edge cases, and offer guidance on how to address those issues. This type of help also received the highest ratings from the students for clarity and helpfulness. It also comes in second for the suitability of the help for the current problem.

On the other hand, the least used was the Concepts Explanation button. The primary function of this button is to provide additional explanations or examples to clarify the requirements of the exercise. Among all other buttons, this one received the lowest score for clarity, with an overall grade of 4 out of 5, which is not bad, but being the least used button, there was not a significant amount of feedback. This button also scored best for the information suitability with a 4 out of 5.

The least effective button for helpfulness and suitability was Code Improvement. Since the primary purpose of this button was to provide improvements and optimizations of the code, the students did not appreciate the advice. This type of help would be considered suitable only in cases where the students have already finished the code but would want to improve its readability and efficiency. This was not the case here, while students aim to finish their exercise most of the time and do not care about optimizations if the code is running. This type of help would be helpful for more advanced students, where code quality plays a more important role.

RQ2: Time Efficiency The availability of help from ChatGPT appears to lead to improved time efficiency in completing programming tasks. Statistical analysis of task completion times indicates that students in the experimental group, who had access to ChatGPT assistance, generally completed tasks faster than those in the control group. This improvement in time efficiency is particularly noticeable in the initial set of exercises, where students were still getting accustomed to the problem-solving environment and benefited greatly from immediate help.
However, the time efficiency benefits did not persist uniformly across all exercises. In the final set of problems, where ChatGPT assistance was no longer available, the time taken by students in the experimental group was comparable to that of the control group. This suggests that while ChatGPT assistance can significantly enhance time efficiency during its availability, students might not retain these efficiency gains when the aid is withdrawn. It might also suggest an overreliance on the help. At the same time, because of the length of the experiment, students might have been tired or bored with the daily tasks of the experiment.

RQ3: Self-Satisfaction and Learning While both groups reported an increase in their confidence in their programming skills, the concepts they have learned differ based on their group. The students from the experimental group reported more diverse and deeper insights than the control group, which reported only fundamental programming skills and coding practices. Due to each participant's own knowledge and perceptions, the significance of these learning outcomes cannot be measured, but all the participants certainly increased their programming skills.

Combining the insights from these research questions, it is evident that while ChatGPT's assistance features significantly enhance immediate task performance and satisfaction, there are challenges related to long-term retention and independence in problem-solving skills. Future implementations should consider incorporating elements that promote strategic learning and problem-solving independence alongside immediate assistance to address this. By doing so, educational tools can provide not only short-term support but also foster long-term skill development and confidence in students.

6.0.1 Limitations

The study was conducted with a small group of participants in a setting beyond the researcher's control. This lack of power led to several challenges. For instance, the timing of when students completed the exercises could not be regulated, resulting in

delays. The small sample size further exacerbated this issue, as delays were common among most participants, potentially affecting the accuracy of problem-solving time measurements. Additionally, the researcher had to be available almost constantly to assist students unfamiliar with the platform, leading to a high demand for immediate support.
The duration of the experiment, which spanned just over two weeks with daily tasks, likely contributed to participant fatigue. Some students might have grown tired of the daily requirements, leading them to occasionally overlook critical steps, such as pressing the Finish button or completing the post-exercise surveys. These factors introduced variability that could influence the study's outcomes.

6.0.2 Future work

The experiment generated a substantial amount of data that required extensive processing and analysis. Given these constraints, the study focused on specific aspects of the collected data. Future research should include a detailed analysis of the students' code, testing the code for performance and efficiency, and assessing whether ChatGPT's assistance significantly impacted the quality of the code produced. It would be particularly valuable to determine if the experimental group employed more advanced programming concepts than the control group, as suggested in subsection 5.3.2.
Another critical area for future work is refining the types of help provided to students. The study found that some forms of assistance were more beneficial than others. For novice students, the primary focus should be on their immediate learning needs rather than code efficiency and style. Conversely, more advanced students would benefit from assistance emphasizing best practices and code optimization. Additionally, students expressed a desire for code examples, indicating that incorporating code snippets for basic syntax usage could be beneficial.
Exploring using an integrated chatbot to simulate a student-teacher interaction could provide more dynamic and context-specific assistance. Such a feature would allow students to request help based on their unique problems and code, potentially offering a more personalized learning experience. This direction could provide valuable insights into the effectiveness of real-time, interactive assistance in programming education.

Appendix A Bibliography

[1] What is ChatGPT? URL https://help.openai.com/en/articles/6783457-what-is-chatgpt. [Online; Accessed 12.05.2024].

[2] Codebench dataset. URL https://codebench.icomp.ufam.edu.br/dataset/. [Online; Accessed 20.05.2024].

[3] ChatGPT for Jupyter. URL https://github.com/TiesdeKok/chat-gpt-jupyter-extension/tree/main. [Online; Accessed 12.05.2024].

[4] Juxl Cut. URL https://www.npmjs.com/package/@juxl/cut. [Online; Accessed 12.05.2024].

[5] Project Jupyter. URL https://jupyter.org/. [Online; Accessed 12.05.2024].

[6] xAPI (Experience API) Overview. URL https://xapi.com/overview/. [Online; Accessed 12.05.2024].

[7] Kathleen T. Brinko. The practice of giving feedback to improve teaching. *The Journal of Higher Education*, 64(5):574–593, 1993. doi: 10.1080/00221546.1993.11778449. URL https://doi.org/10.1080/00221546.1993.11778449.

[8] Annabell Brocker, Sven Judel, Ulrik Schroeder, and Yanik Söltzer. Juxl: Jupyterlab xapi logging interface. In *2022 International Conference on Advanced Learning Technologies (ICALT)*, pages 158–160, 2022. doi: 10.1109/ICALT55010.2022.00054.

[9] Lixin Cao, Yongjun Wang, Jie Huang, and Yang Li. A study on prompt design, advantages and limitations of chatgpt. *Journal of Artificial Intelligence Research*, 56(3):123–145, 2023. doi: 10.1613/jair.1.12836.

[10] Wei Dai, Jionghao Lin, Flora Jin, Tongguang Li, Yi-Shan Tsai, Dragan Gašević, and Guanliang Chen. Can large language models provide feedback to students? a case study on chatgpt. *Centre for Learning Analytics at Monash, Monash University*, 2023.

[11] John Dawes. Do data characteristics change according to the number of scale points used? an experiment using 5-point, 7-point and 10-point scales. *International Journal of Market Research*, 50(1):61–77, 2008.

[12] Qing Hao, David Smith, Li Ding, Andrew Ko, Carleton Ottaway, James Wilson, Kayoko Arakawa, Alexandra Turcan, Ted Poehlman, and Thomas Greer. Towards understanding the effective design of automated formative feedback for programming assignments. *Computer Science Education*, 2021. doi: 10.1080/08993408.2020.1860408. URL https://doi.org/10.1080/08993408.2020.1860408.

[13] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. Exploring the potential of large language models to generate formative programming feedback. In *Proceedings of the 2023 IEEE ASEE Frontiers in Education Conference*, College Station, Texas, 2023. IEEE. Accepted for publication.

[14] Rensis Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

[15] Scott MacNeil, Albert Tran, Daniel Mogil, Sarah Bernstein, Eric Ross, and Zhen Huang. Experiences from using code explanations generated by gpt-3. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 2*, pages 37–39, 2022. doi: 10.1145/3501385.3543972.

[16] Susanne Narciss. Feedback strategies for interactive learning tasks. In *Feedback in E-Learning: Types, Challenges, and Solutions*, pages 125–144. 2008.

[17] Susanne Narciss and Katja Huth. Fostering achievement and motivation with bug-related tutoring feedback in a computer-based training for written subtraction. *Learning and Instruction*, 16(4):310–322, 2006. ISSN 0959-4752. doi: https://doi.org/10.1016/j.learninstruc.2006.07.003. URL https://www.sciencedirect.com/science/article/pii/S0959475206000521.

[18] Marta Pankiewicz and Ryan S. Baker. Large language models (gpt) for automating feedback on programming assignments. In *Proceedings of the 2023 International Conference on Learning Analytics and Knowledge (LAK 2023)*, pages 210–219, 2023. doi: 10.1145/3170358.3170402.

[19] Thomas W. Price, David Hovemeyer, Kelly Rivers, Ge Gao, Austin Cory Bart, Ayaan M. Kazerouni, Brett A. Becker, Andrew Petersen, Luke Gusukuma, Stephen H. Edwards, and David Babcock. Progsnap2: A flexible format for programming process data. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 356–362, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368742. doi: 10.1145/3341525.3387373. URL https://doi.org/10.1145/3341525.3387373.

[20] Lianne Roest, Hieke Keuning, and Johan Jeuring. Next-step hint generation for introductory programming using large language models, 2023.

[21] RWTH Aachen University. Lrs - rwth aachen university. URL https://lrs.elearn.rwth-aachen.de. [Online; Accessed 20.05.2024].

[22] Javier Sarsa, David Aguilera, Andres Garcia, Albert Prats, and Angeles Vázquez. Automatic generation of programming exercises and assessment. *IEEE Transactions on Learning Technologies*, 15(1):1–15, 2022. doi: 10.1109/TLT.2022.3151324.

[23] JupyterLab Team. Jupyter ai: A generative ai extension for jupyterlab. https://github.com/jupyterlab/jupyter-ai, 2024. Accessed: 2024-06-03.

[24] James Tian, Emily Lee, and Michael Chen. Is chatgpt the ultimate programming assistant? a comprehensive evaluation. *Journal of Machine Learning Research*, 24(5):456–478, 2023. doi: 10.1109/JMLR.2023.1234567.

[25] Robert White, Alice Green, and Tim Black. A prompt pattern catalog to enhance prompt engineering with chatgpt. *Journal of AI Research*, 58(4):201–225, 2023. doi: 10.1613/jair.1.12848.

[26] Mantz Yorke. Formative assessment in higher education: Moves towards theory and the enhancement of pedagogic practice. *Higher Education*, 45:477–501, 2003. doi: 10.1023/A:1023967026413. URL https://doi.org/10.1023/A:1023967026413.

Appendix B Digital Appendix

The digital appendix is submitted on a SD card. To get access to this material they shall contact the author or LuFG i9.

Appendix C Use of AI Tools

ChatGPT (https://chatgpt.com) and Grammarly (https://app.grammarly.com/) have been used to check the grammar, spelling, and stylistics of the whole paper.

Eidesstattliche Versicherung

Gherman, Codruța-Andreea	436923
Name, Vorname	Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

Exploring the Impact of ChatGPT Assistance on Novice Programmers' Efficiency and Learning in Python Programming Tasks

ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen , June 9, 2024	
Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen , June 9, 2024	
Ort, Datum	Unterschrift