Master's Thesis

Advancing Cognitive Modeling of Code Comprehension: Integrating Backtracking and Similarity Error Mechanisms in ACT-R

Timon Dörzapf

May 16, 2025

Advisor:
Dr. Marvin Wyrich Chair of Software Engineering

Examiners:
Prof. Dr. Sven Apel Chair of Software Engineering
Prof. Dr. Vera Demberg Chair of Computer Science and Computational Linguistic

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

UNIVERSITÄT
DES
SAARLANDES

Timon Dörzapf: *Advancing Cognitive Modeling of Code Comprehension: Integrating Backtracking and Similarity Error Mechanisms in ACT-R,* © May 2025

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,_____ _____

(Datum/Date) (Unterschrift/Signature)

Abstract

Code comprehension is a crucial task in software development, as a lot of time is spent reading and understanding code. The ability to simulate the thought process of a developer during code comprehension tasks could provide insights into the cognitive processes of developers. This in turn would enable the development of tools that support developers in their everyday work through more efficient code comprehension.

However, the cognitive processes involved in code comprehension are complex and not yet fully understood. Even though a lot of theoretical models have been proposed, they are difficult to use in practice due to their complexity. To this day, only models with major simplifications of the actual environment of a developer have been implemented, such as only being able to read one line at a time without the ability to return to previous lines.

We split the thesis into two parts: a replication study and an extension study. The replication study aimed to replicate the results of a previously developed cognitive model, enhanced with similarity errors, for code comprehension tasks, while the extension study extended this model with a backtracking mechanism to improve its generalizability. For both parts, we conducted an empirical evaluation with participants who were asked to read and understand code snippets and a simulation evaluation with a cognitive model trained on the empirical data that simulates the behavior of the participants.

We were able to replicate some results of the previously developed cognitive model for code comprehension tasks. Additionally, by extending this model with a backtracking mechanism and similarity errors, we were able to improve the generalizability of this model and provided a better foundation for subsequent studies in this area. Notably, our empirical evaluation showed that 70% of participants exhibited at least one instance of backtracking, supporting the view that code comprehension is an iterative and non-linear process. In addition, incorporating backtracking improved the model's fit to the empirical data, allowing it to more accurately simulate developers' cognitive processes during code comprehension.

The developed model is a good starting point for simulating the cognitive processes of developers during code comprehension tasks, but further improvements and refinements are still needed.

Contents

List of Figures

List of Tables

Listings

Acronyms

CD	Code Distance
CR	Repeated Code
DR	Declaration Redeclaration Distance
RP	Repeated Distance
ACT-R	Adaptive Control of Thought—Rational
KS	Kolmogorov-Smirnov
ans	activation noise s
lf	latency factor
rt	retrieval factor
mp	mismatch penalty
bll	base level learning

1 Introduction

Code comprehension plays an important role in the everyday work of developers, as they need to understand existing code to maintain and extend it. Being able to understand code efficiently is crucial for developers to remain productive and to avoid introducing errors. It is estimated that 30–70% of a developer's time is spent on understanding code [24, 42]. Thus, the need for a better understanding of how developers comprehend code has been recognized by the software engineering community [41].

To make source code easier to understand, software developers often refactor their code to improve its readability and maintainability [2]. Traditional approaches to evaluating code comprehension often rely on static code metrics provided by analysis tools, such as cyclomatic complexity or line counts. While such metrics can indicate potential difficulties, they remain at a surface level and do not directly capture the cognitive processes underlying human comprehension. These metrics may not be validated [27] or may not reflect the actual value that should be measured [26, 31]. Therefore, there is a need for models that go beyond static properties and simulate the cognitive processes involved in code comprehension more directly.

This leads us to the concept of mental models, which describe how developers internally represent code and how these representations influence comprehension performance. Such mental models can be formalized into cognitive models using cognitive architectures like ACT-R [4]. Cognitive models can simulate the behavior of developers during code comprehension tasks and provide fine-grained insights into the underlying cognitive mechanisms — such as memory load, attention shifts, and decision-making processes — offering a much deeper understanding than traditional metrics alone.

While the potential of creating cognitive models for code comprehension is promising, previous work has mostly been theoretical [19]. Closheim [10] developed a first executable ACT-R model simulating developers during code comprehension tasks, achieving promising results. However, the model still relies on many assumptions and simplifications, which may limit its accuracy in reflecting real-world developer behavior.

1.1 Goal of this Thesis

The goal of this thesis is to replicate the empirical evaluation and extend the ACT-R model of Closheim. By replicating the study of Closheim, we aim to validate the results of the model

and to understand the limitations of the model. This is an important step in ensuring that the model is accurate and can be used to simulate the behavior of developers in real-world scenarios.

Additionally, by performing an extension study, we aim to improve the model by incorporating a backtracking mechanism. As the current model can only show one line of code at a time and does not allow for backtracking, it is limited in its ability to accurately reflect the behavior of developers. This is a significant limitation, as developers often need to go back and forth between different lines of code to understand the overall structure and flow of the code. By incorporating a backtracking mechanism, we aim to increase the generalizability of the model and provide a better foundation for subsequent studies in this area. While this will still not be a perfect model of the behavior of developers, it will be a step in the right direction and will allow us to better understand the cognitive processes involved in code comprehension.

To achieve these goals, we will split each study into two evaluations: an empirical evaluation and a simulation evaluation. The empirical evaluation will consist of a study with participants who will be asked to read and understand code snippets. The simulation evaluation will consist of a simulation of the behavior of the participants using the ACT-R model. The results of the empirical part will be compared to the results of the simulation to evaluate the accuracy of the model.

1.2 Structure of this Thesis

The Background chapter will discuss the theoretical background of the proposal and explain the concepts that are important for the proposal. In Part I, we will discuss the methodology, research questions, and expectations of the replication study. In Part II, we will discuss the methodology, research questions, and expectations of the extension study. The first Evaluation chapter will discuss the evaluation of the empirical and simlation study results of the replication study. The second Evaluation chapter will discuss the evaluation of the empirical and simulation study results of the extension study. In the Related Work chapter, we will discuss the similar works that have been done in the past and how they are related to our thesis. Finally, the Conclusion chapter will conclude the thesis and provide an outlook for future work.

1.3 Methodological Overview

In this section, we will discuss the methodology of the thesis. The thesis consists of two parts: a replication study and an extension study. In Figure 1.1, we present the schematic representation of the research methodology.

In Part I, we focused on the replication study of the ACT-R model of Closheim. In this study, we performed an empirical evaluation and a simulation evaluation. The empirical

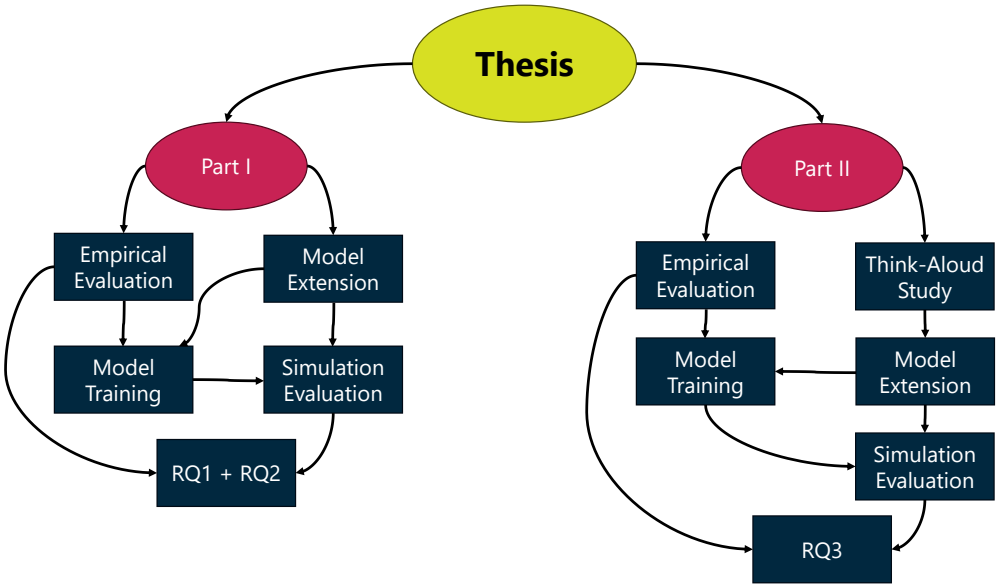


Figure 1.1: Schematic Representation of the Research Methodology

evaluation consisted of a crowdsourcing study with participants who were be asked to read and understand code snippets. In parallel, we extended the model of Closheim to include similarity errors. With the extended model, we used the results of the empirical evaluation to train the parameters of the model. The resulting model was then evaluated in regard to its accuracy in terms of processing time and error rates. The results of this simulation evaluation were used to answer RQ1 and RQ2.

In Part II, we focused on the extension study of the ACT-R model of Closheim. In this study, we also performed an empirical evaluation and a simulation evaluation. The empirical evaluation consisted of a crowdsourcing study with participants who were be asked to read and understand code snippets. In parallel, we carried out a think-aloud study with three participants to gain insights into the cognitive processes of developers during backtracking to previous lines of code. The results of the think-aloud study were used to understand the cognitive processes of developers during backtracking and to extend the model of Part I to include backtracking. With the extended model, we used the results of the empirical evaluation to train the parameters of the model. The resulting model was then evaluatued in regards to its accuracy in terms of processing time and error rates. The results of this simulation evaluation were used to answer RQ3.

2

Background

In this chapter, we provide an overview of the cognitive architecture ACT-R and its components, as well as the concepts of letter similarity and reading mechanisms. These topics are fundamental to understanding the cognitive processes involved in source code comprehension, which is the focus of this thesis.

The ACT-R architecture is particularly relevant because it provides a framework for modelling human cognition, including how knowledge is represented, retrieved and applied. By using ACT-R, we aim to simulate the cognitive processes of programmers as they read and understand source code.

We will also discuss letter similarity to highlight how visual and cognitive similarities can lead to errors or increased cognitive load, a phenomenon that is also observed in source code comprehension when variable names or structures are similar. Understanding these effects helps us design more realistic cognitive models.

Finally, we will explore reading mechanisms, including backtracking and multi-line reading, to establish their importance in understanding source code. These mechanisms are critical for simulating how programmers navigate and interpret code, which directly informs the enhancements made to the ACT-R model in this work.

By connecting these topics, this chapter lays the groundwork for the subsequent development and evaluation of a cognitive model that incorporates backtracking to better simulate human code comprehension.

2.1 Cognitive Architectures

Cognitive architectures are defined in the following way by Anderson [5]:

> "A cognitive architecture is a specification of the structure of the brain at a level of abstraction that explains how it achieves the function of the mind."

Therefore, cognitive architectures are models that attempt to explain how the human mind works by simulating cognitive processes such as perception, memory, and problem-solving. These models are not intended to be exact replicas of the brain, they should provide a framework for understanding how cognitive functions are organized and how they interact with each other while also enforcing biological constraints. These models are used to study

human behavior, predict performance, and understand the underlying mechanisms of cognition [19].

While there are many cognitive architectures, one of the most well-known and widely used is the Adaptive Control of Thought-Rational (ACT-R) architecture developed by Anderson [3].

2.2 ACT-R

ACT-R is domain-specific programming language built on top of LISP that is used to model cognitive processes. It is based on assumptions about how the human mind works and is designed to simulate human behavior in a variety of tasks. The simulations can be used to predict several psychological dimensions, such as reaction times and task accuracy[19]. ACT-R consists of eight modules representing different aspects of cognition which are associated with a particular brain region. These modules interact to simulate cognitive processes and behavior. Each module has its own „buffer“, which contains „chunks“of information that represent knowledge. In addition, ACT-R uses a production system to model how knowledge is represented and retrieved in the brain [19].

Buffers

Buffers are temporary storage areas that hold information being processed by the cognitive system. Buffers are also used to communicate between modules and to represent the current state of the system such as failure, empty or full [19].

Chunks

Chunks are the basic unit of knowledge in ACT-R. ACT-R uses chunks to represent knowledge in the system, such as facts, rules, and concepts. They consist of a set of slots, which are key-value pairs that represent different aspects of the knowledge [19]. Chunks can be declared before running a simulation to represent the existing knowledge or created dynamically during the simulation to represent new information.

Production System

The production system is a set of rules that pattern-match against the contents of the buffers to determine the next action to take. When a rule matches the contents of a buffer, it is fired, and the corresponding action is executed. This action can involve updating the buffers, creating new chunks, or modifying/removing existing chunks [19].

Symbolic and Subsymbolic Layer

ACT-R consists of two layers: the symbolic layer and the subsymbolic layer. The symbolic layer represents knowledge in a symbolic form, using chunks and rules to represent concepts and relationships. The subsymbolic layer represents human performance in terms of cognitive tasks. It uses facts derived from psychological experiments to predict human behavior [19].

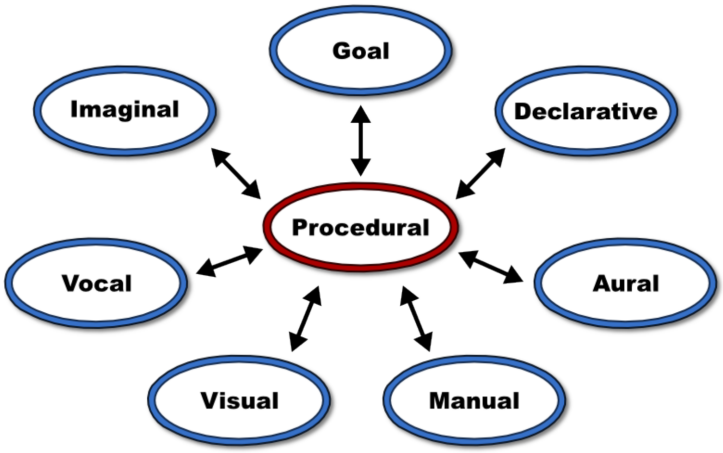


Figure 2.1: ACT-R 6.0 modules and their communication paths [19]

Modules

ACT-R consists of several modules that can be seen in Figure 2.1. The visual, aural, vocal, and manual modules are able to interact with the environment. The remaining modules are explained below:

Declarative Memory

Declarative memory is the long-term memory system that stores facts, concepts, and rules. Information can be retrieved via the retrieval buffer, which is used to search for chunks in declarative memory that match a given pattern.

Procedural Memory

Procedural memory is the center of the communication between the modules. Procedural memory is responsible for controlling the flow of information between the modules and executing the actions specified by the production rules. Thus, it is the bottleneck of the system, as only one production may fire at any time.

Imaginal Module

The imaginal module is used to maintain the model's current problem state. It is used to store and manipulate information not currently in the environment, such as mental images or representations of the current task.

Goal Module

The goal module is used to represent the current goal of the model. It is responsible for selecting the next action to take based on the current goal and the contents of the buffers.

Partial Matching

Partial matching is a key feature of ACT-R that allows it to retrieve information from memory based on a partial pattern by using a similarity metric instead of an exact match during the matching process. This allows the model to recognize patterns that are similar to those stored in memory, even if they are not identical. Additionally, it also enables the model to make human-like errors, such as mistaking similar letters or numbers for each other [19].

Activation and Retrieval

Activation is computed based on the setting of several parameters, such as base-level activation, spreading activation, and noise. The general equation for activation is as follows:

$$A_i = B_i + P_i + \epsilon_i$$

where A_i is the activation of chunk i, B_i is the base-level activation, P_i is the partial matching value and ϵ_i is the noise term.

The base-level activation is computed as follows:

$$B_i = \ln(\frac{n}{1-d}) - d * \ln(L)$$

where n is the number of times the chunk has been retrieved, d is the decay parameter, and L is the lifetime of the chunk i.

The partial matching value is computed as follows:

$$P_i = \sum_k PM_{ik}$$

where P is the match scale parameter that reflects the added weight of the similarity and M_{ik} is the similarity between the value k in the retrieval specification and the value of slot k in chunk i.

The noise term is a random value drawn from a logistic distribution with a mean of o and a variance of:

$$\sigma^2 = \frac{\pi^2}{3}s^2$$

where s is defined with the noise parameter :ans.
To calculate the retrieval time, we use the following equation:

$$RT = Fe^{-f*A_i}$$

where RT is the retrieval time, F is the latency factor parameter :lf, f is the latency exponent parameter :le and A_i is the activation of chunk i. If no chunk can be retrieved, A_i is replaced by the value of the retrieval threshold parameter :rt[18].

2.3 Similarity of Letters and Numbers

Letter similarity plays a central role in several cognitive processes, especially in reading, letter recognition, and visual perception. Letter similarity refers to the degree to which different letters resemble each other in shape, form, and visual properties. The implications of letter similarity are significant in fields such as psycholinguistics, educational psychology, and human-computer interaction. By studying how similar or dissimilar letters affect cognition, we can better understand how people read, learn, and interact with written language.

There exist several motivations for studying letter similarity [32]:

- Practical attempts to make written text more comprehensible
- Empirical understanding of the visual system functions
- Theory of representation of letters in the visual system
- Theory of representation of the abstract form of letters in the cognitive system

Visual Perception and Letter Recognition

Letter recognition is a fundamental component of reading proficiency depending heavily on the visual distinctiveness of letters. The process of visual perception entails the capacity to detect, interpret, and categorize visual stimuli, which is an important aspect of the recognition of letters. The difficulty of distinguishing between two or more letters with similar visual features can decrease reading speed and increase the probability of identification errors. Research has demonstrated that letters with analogous shapes are frequently confused with one another, particularly when they are presented in suboptimal conditions, such as low contrast, in a noisy environment or under time pressure. The Feature Integration Theory, proposed by [36], suggests that the brain processes disparate visual features (e.g., lines, curves, and angles) serially before combining them to recognize entire objects, including letters. The integration process becomes more complex when letters share multiple visual features, increasing the likelihood of perceptual errors. For example, the letter ”p” and the letter ”q” are mirror images of each other, and individuals, particularly children or those with dyslexia, often confuse them due to their similar visual properties [35].

2.4 Source Code Comprehension

Source code comprehension has had many definitions over the years, but it is generally understood as the process of understanding and interpreting programming code. A more precise definition is provided by Wyrich [40]:

> "Source code comprehension describes a person's intentional act and degree of accomplishment in inferring the meaning of source code."

The degree of accomplishment refers to the level of understanding achieved by the person, which can range from a superficial understanding to a deep, detailed understanding. It should be noted that depending on the kind of comprehension task, different levels of understanding may be required, as finding a bug in a program requires a different level of understanding than having an in-depth understanding of the program itself.

The meaning of the source code can be separated into three categories, which may overlap depending on the context:

- Functional: What does the code do?
- Specification: What the code is supposed to do?
- Context: What is the intention of the author?

Code Comprehension Processes

Two popular theories of code comprehension are the top-down and bottom-up process. The top-down theory posits that programmers start by understanding the overall structure and purpose of the code before delving into the details. This approach is often used by expert programmers who have a deep understanding of common patterns and algorithms.

The bottom-up theory, on the other hand, suggests that programmers start by understanding the individual components of the code before piecing them together to understand the overall structure. This approach is often used by novice programmers who lack experience with common patterns and algorithms [15].

The ACT-R model developed by Closheim [10] implemented the bottom-up theory by simulating the slow learning process of novices as they build up their chunks of declarative knowledge. The model is limited by being only able to simulate a small part of the code execution as chunks are limited to the name of the variable and its value and that the starting knowledge of the model is limited to basic arithmetic operations.

Implementations of the top-down theory are more complex as they require the model to have a more detailed understanding of the code structure and logic. Further developed models such as the integrated metamodel by Von Mayrhauser and Vans [38] would complicate an implementation even further as they require the model to be able to switch between different comprehension models depending on the interaction and progression of the different code comprehension processes.

2.5 Reading Mechanisms

Reading is a complex cognitive process that involves the integration of multiple cognitive functions, such as visual perception, attention, memory, and language processing. The process of reading involves several stages, including visual processing, word recognition, sentence parsing, and comprehension [29].

Reading one line at a time without the ability to go back to previous lines is not how a programmer would read code. Programmers often read code in a non-linear fashion, jumping back and forth between different parts of the code to understand the overall structure and logic [23].

We present two ways to model reading mechanisms in the context of source code comprehension: the backtracking mechanism and multi-line reading.

Backtracking Mechanism

The backtracking mechanism is a cognitive process that allows readers to revisit previously read text to clarify or reinforce their understanding. This mechanism is essential for reading comprehension, as it enables readers to correct misunderstandings, fill in gaps in their understanding, and make connections between different parts of the text.

In the context of source code comprehension, the backtracking mechanism is crucial for understanding complex code structures, identifying bugs, and debugging code. Programmers often need to go back and forth between different parts of the code to understand the overall logic, trace the flow of data, and identify the root cause of errors.

Multi-Line Reading

Multi-line reading is a reading mechanism that allows readers to see and process multiple lines of text simultaneously. This mechanism is essential for understanding the overall structure and logic of a text, as it enables readers to make connections between different parts of the text and identify patterns and relationships that span multiple lines.

In this Work

The objective of this study is to enhance the existing ACT-R model of source code comprehension by incorporating a mechanism for backward navigation. While multi-line reading is an important aspect of source code comprehension, this study focuses on the backtracking mechanism as it is a key cognitive process that enables programmers to understand complex code structures and identify errors. Understanding when and why programmers go back to previous code lines is crucial for developing a more accurate and realistic model of source code comprehension. Extending this work to include multi-line reading would be an interesting extension for future research, but it is beyond the scope of this thesis.

3

Part I Methodology: Replication Study

This chapter describes the methodology of our replication study. It consists of two parts: a simulation evaluation and an empirical evaluation. It is based on the study by Closheim.

We will start by presenting the research questions that will be answered in this study. This will provide us an overview of the aims of this part of the thesis. Afterward, we will describe the study materials and study design of the simulation evaluation and the empirical evaluation. Finally, the data analysis and evaluation of the two parts will be presented.

3.1 Research Questions

Our goal is to replicate the study by Closheim to strengthen the results of Closheim's original study. Nevertheless, the original model had some weaknesses that we wanted to address. While the distances between declarations and re-declarations/usages were great enough to be noticeable, we would like to explore how the model behaves when the distances are further increased. This leads us to the following research question:

> **RQ1**: How do increased distances between declarations and re-declarations/usages influence the accuracy of the ACT-R model in terms of processing time and error rate?

In addition, the model was only able to make one type of error, the inability to retrieve a particular value from memory. As humans are prone to making errors in other ways, we would like to explore how the model behaves when it is able to make more types of errors. Therefore, we added partial matching to the model to allow for similarity errors. This leads us to our second research question:

> **RQ2**: Can we increase the accuracy of the ACT-R model in terms of processing time and error rate by implementing similarity errors between variable names?

These two research questions enabled us to explore the model's behavior in more detail and to strengthen the results of Closheim's original study.

3.2 Study Design

In this section, we will describe the design of the study. We performed two types of evaluation: a simulation evaluation and an empirical evaluation. The empirical evaluation

was a questionnaire where participants were presented with code snippets and asked to solve them.

The simulation evaluation consisted of extending an existing cognitive model with the ability to make similarity errors and training it with the results of the empirical evaluation. The aim was to enable us to use the cognitive model to predict the behavior of programmers on a larger scale. The prediction accuracy of the cognitive model was evaluated to understand how well the model is able to predict the behavior of the participants.

3.2.1 Study Materials

We replicated the study by Closheim [10] with some changes to the code snippets that were presented to the participants.

The participants were presented with eight tasks, each consisting of a code snippet. The code snippet was either a treatment or a control code snippet. After each code snippet, the participants were be asked to work on an intermediate task to perform a reset of the working memory of the participants to prevent sequence effects.

We have also asked the participants to fill out a socio-demographic questionnaire at the end of the study, as has been done in Closheim's original study.

While the code snippets were different to Closheim's original study, the intermediate tasks and socio-demographic questions were the same.

We create a questionnaire with the help of SoSci Survey[1], as it allows the free creation and execution of questionnaires for academic purposes. We used LabJS[2] to present the code snippets to the participants, as it allowed us to record the times and key presses of the participants, which was necessary for the analysis of the results.

3.2.1.1 *Code Snippets*

The code snippets were the same for both studies. They were shown to the participants one line at a time. To continue to the next line, the participants had to press the *Space* key. At the end of the code snippet, the participants were asked to enter the value of the variable that is declared in the print statement and press *Space* or a button on the screen to submit their answer. The snippets consist of simple arithmetic operations, such as addition, subtraction, multiplication and division. Variables can be declared multiple times, also without being used in between. Multiple operations can be performed on the same code line, but point-before-dash calculations were not taken into account, as we assume a simple left-to-right reading of the code. Additionally, it is impossible to return to a previous code line.

The code snippets have the following differences to Closheim's original study:

- The distances between declarations and re-declarations/usages were increased
- The variable names were made more similar to each other

1 https://www.soscisurvey.de/
2 https://lab.js.org/

3.2.1.2 *Training Code Snippets*

We first let the participants practice with training code snippets. These code snippets are similar to the code snippets used in the study. However, they are easier to solve and do not contain any of the changes made to the experimental code snippets. This allowed the participants to familiarize themselves with the interface and the tasks they will be presented with. As the ACT-R model does not need training code snippets, they were not used in the simulation evaluation.

3.2.1.3 *Experimental Code Snippets*

An example of changed code snippets can be found in Listing 3.1. Code snippets were either changed by changing the variables names to be more similar to each other as in (b) or by increasing the length of the code snippet as in (c) by adding filler lines of code.

```
i = 4
k = 8
h = k * 2
y = i * 4 - h
print(y)
```

(a) Original Study

```
p = 4
q = 8
d = q * 2
b = p * 4 - d
print(b)
```

(b) Similar variable names

```
i = 4
k = 8
// test
assert(k == 8)
// continue
h = k * 2
y = i * 4 - h
print(y)
```

(c) Increased Distances

Listing 3.1: Code Snippets Changes

The code snippets that were used can be found in the appendix A.4.

We used the same treatments of code snippets as in Closheim's original study, which are the following:

- CD: We observe the effect of the distance between declaration and usage of a variable.
  - Difference between Treatment and Control: We have a longer distance between declaration and usage of a variable in the treatment code snippet.
- CR: We observe the effect of a double declaration at the beginning of code snippet and subsequent usage of the variable.
  - Difference between Treatment and Control: The control code snippet has no double declaration.
- DR: We observe the effect of a double declaration and the distance between the last redeclaration of the variable and subsequent usage of the variable.
  - Difference between Treatment and Control: The control code snippet has a double declaration with the redeclaration of the variable being just before the usage of the variable, contrary to the treatment code snippet, where the double declaration is at the beginning of the code snippet.

- RP: We observe the effect of a larger distance between a double declaration and subsequent usage of the variable.
  - Difference between Treatment and Control: The control code snippet has a double declaration that is near the usage of the variable, contrary to the treatment code snippet, where the double declaration is at the beginning of the code snippet.

We had two types of code snippets, namely treatment and control code snippets, as in Closheim's original study. The treatment code snippets contained the changes mentioned above, while the control code snippets were used as a baseline to compare the results of the treatment code snippets. Therefore, each treatment code snippet had a corresponding control code snippet.

We reused the treatment code snippets from Closheim's original study and split them into two groups of equal size of four snippets: one group was changed to implement the increased distances between declarations and re-declarations/usages, while the other group was changed to implement the similarity between variable names. This allowed us to compare the results of our study to Closheim's original study. The control code snippets were adjusted accordingly.

To validate the effectiveness of our code snippets, we conducted a pilot study with three participants. This enabled us to see if the code snippets were solvable and if the changes had the desired effect on the participants.

3.2.1.4 *Intermediate Tasks*

The intermediate tasks was a ten-second video clip of fish in an aquarium, which was the same as in Closheim's original study. Video clips without letters or numbers were used to properly reset the working memory of the participants. Additionally, using such videos does not accelerate the mental fatigue of the participants, as they are not required to perform any tasks during the video.

3.2.1.5 *Socio-Demographic Questionnaire*

The socio-demographic questionnaire was the same as in Closheim's original study. This will enable researchers to use both datasets together, if the need arises in future studies, even though we did not analyse any of the socio-demographic data in the present study.

The questions that were asked had the following themes:

- Age
- Country
- Education
- Job Title
- Programming Experience
- Experience with Programming Paradigms

The entire questionnaire can be found in the appendix A.1.

3.2.2 Simulation Evaluation

In this section, we will describe the cognitive model that was used in the study.

We built upon the existing model by Closheim, created with the cognitive architecture ACT-R, and extended it with the ability to make similarity errors. Additionally, the model is now able to handle *assert* keywords and comments prepended with "//".

As ACT-R is able to model human input and output, we used it to model the behavior of participants in the study. By utilizing the vision interface of ACT-R in the same way as in the existing model, we were able to model the visual input of the participants, namely the code snippets they will be presented with. This allowed us to copy the environmental conditions as in the empirical evaluation, enabling us to use the results of the empirical evaluation to train the ACT-R model.

The model is able to perform the following tasks:

- Read one line at a time from left to right
- Store variable names and values in declarative memory
- Retrieve variable names and values from declarative memory
- Perform arithmetic operations, namely addition, subtraction, multiplication and division, on integer numbers from -20 to 20
- Read comments
- Read and verify assert statements
- Answer a print statement with the expected output of the model by entering the guessed number and pressing *Space*.

The model aims to mimic human behavior as closely as possible by using the same input and output as the participants in the empirical evaluation. The model reads the unparsed line of code and processes it in the same way as the participants. When the model is finished with a line, it will also press the *Space* key to continue to the next line. The same is true for the print statement, where the model will enter the guessed number and press *Space* to submit the answer.

Everytime a variable is declared, the model stores the variable name and value in declarative memory. This is also the case if the variable was already declared before. When a variable is used, the model tries to retrieve the variable name and value from declarative memory. As the model returns chunks based on their current activation value, the model is able to retrieve the wrong value of a variable.

The parameters that were tuned to train the model can be found in Table 3.1. Even though the model has more parameters that can be tuned, we focused on these four parameters as they are the most relevant for our study. As more parameters would lead to a more complex model, we kept the number of parameters to a minimum. Additionally, the parameters rt, ans and lf have been used in Closheim's original study and have been shown to have an impact on the model's behavior.

Parameter	Name	Default
rt	retrieval threshold	0
ans	activation noise s	0.2-0.8
lf	latency factor	1
mp	mismatch penalty	0

Table 3.1: Default values for parameters [18].

3.2.3 Empirical Evaluation

The study was conducted to gather data on how humans behave when presented with the previously mentioned code snippets. The participants were first presented with the training code snippets, followed by the experimental code snippets. Afterward, they were presented with the socio-demographic questionnaire. We used the same procedure as in Closheim's original study, except for the changes to the code snippets and the recruitment of participants.

We used a crossover design, as in Closheim's original study, to control as many confounding variables as possible. A crossover design is a combination of a within-subject and between-subject design.

The within-subject design part was implemented by presenting each participant with code snippets for each group of code snippet. The between-subject design part was implemented by assigning each participant to a group that received either the treatment or control code snippets.

Each participants was assigned to one of the four groups. The difference between group 1–2 and 3–4 was that the participants had either the treatment or control variant of a code snippet presented to them. The difference between group 1–3 and 2–4 was that the participants had a different order of the code snippets. These groups were created to reduce learning effects and to ensure that the participants were not biased by the order of the code snippets.

We prepared an application of the study to be reviewed by the ethical review board of the Faculty of Mathematics and Computer Science at the Saarland University. The ethical review board reviewed and accepted the study without the need of additional modifications.

3.2.3.1 *Recruitment*

For such studies that require a larger sample size, Russo [30] named three limitations:

- Ensuring the representatives of the selected population
- Finding enough data points for a statistically significant result
- Correct operationalization of the variables

By using crowdsourcing platforms, the first two limitations can be addressed, as they allow for a large number of participants to be recruited in a short amount of time while ensuring the representativeness of the sample.

Therefore, we chose to use a crowdsourcing platform to recruit participants for the study. This enabled us to reach a large number of participants in a short amount of time. We used the *Connect* platform of the website CloudResearch[3] to recruit participants for the study. CloudResearch is a platform that works similarly to Amazon Mechanical Turk, but offers additional features that are useful for academic studies.

To further ensure the representativeness of the sample, we chose to use a pre-screening process for the participants to ensure that they have programming experience. Russo provided a list of demographic information that should be used to pre-screen participants to increase the probability of receiving participants with programming experience. This list includes the following information:

- have knowledge of software development techniques
- have computer programming skills
- use technology at work (e.g., software) at least once a day
- approval rate of at least 95% on their crowdsourcing platform

Notably, this list is associated to the crowdsourcing platform Prolific[4]. Additionally, Russo recommended to not filter by industry or to balance the gender, as those would falsify the representativeness of the sample. In our case, we only added a filter to ensure that all the participants had programming experience.

After the participants were pre-screened, we conducted a pilot study to ensure that the participants have, in fact, programming experience. This enabled us to filter out participants that do not meet the requirements of the study. We used a multiple-choice screening test to ensure that the participants have programming experience. The questions were based on Danilova et al. [12] screening questions, that they developed to ensure that the participants have programming experience. The used questions can be found in the appendix A.2.

We were able to recruit 124 participants for this part of the study. The participants were rewarded with $5.00 for their participation in the study, which took around 10–20 min to complete.

3.2.3.2 *Exclusion Criteria*

We used similar exclusion criteria as in **CognModelChris's** original study. We excluded any participants that did not complete the questionnaire entirely. We also excluded participants that completed the study too fast or too slow, i.e less or more than 1.5 times the interquartile range below the first quartile or over the third quartile. This was done to ensure that the participants were not rushing through the study or taking too long to complete it. After applying the exclusion criteria for participants of the crowdsourcing platform, we were left with 96 participants.

3 https://www.cloudresearch.com/
4 https://www.prolific.co/

3.3 Data Mining

In this section, we will present how we analyzed this part of the thesis. We analyzed the results of the empirical evaluation. These results were used to train the parameters of the cognitive model, which could then be evaluated.

3.3.1 Empirical Data Mining

We used the same data mining as in Closheim's original study. This allowed us to ensure comparability between the results of Closheim's original study and our study. Closheim provided a detailed description of the data mining and an executable script to analyze the data, which we used and expanded to account for our changes.

The data mining consisted of gathering the following:

- Group number of the participant to determine which type of code snippet was used (treatment or control)
- Time taken by the participant for each line
- Total time for the task
- Which code snippet was used
- Given answer
- Correctness of the answer as a true or false value
- Answers to the socio-demographic questionnaire

These values were then be used in the next step to train our cognitive model.

3.3.2 ACT-R Simulation

Closheim provided an analysis script written in Lisp to create and analyze the data. It takes the number of simulation runs and different parameters as input and outputs the results of the simulation in the same form as the empirical data. We converted this script into the programming language Python for easier readability and maintainability. Additionally, the ability to run the script in parallel was added to speed up the model simulation, as the training of the model would take a long time. We only focused on the difference between control and treatment snippets. Groups were of different sizes to better fit the data.

Simulations were done by setting a new random seed for each simulation run to reflect human behavior as closely as possible. The results of the simulations runs were saved in the same format as the empirical data.

We used a Bayes Optimization algorithm to find the parameters that fit the data the best for the model. We prepared a Python script that takes parameters, the group size and the

code snippet as input and outputs the results of the simulation in the same format as the empirical data. The results of the simulation were then used to calculate a value, which depended on which aspect of the model was to be optimized. The Bayes Optimization algorithm then tried to maximize the value of the results of the simulation.

3.4 Evaluation

The evaluation was performed in three steps:

1. Analyze the results of the empirical evaluation and compare it to Closheim's original study
2. Train the parameters of the cognitive model
3. Evaluate the cognitive model

3.4.1 Empirical Evaluation

The results of the empirical evaluation were analyzed in terms of time and error rate. This allowed us to infer which effect the changes to the code snippets had on the participants. We used the same statistical tests as in Closheim's original study, namely the t-test and Cohen's d for the time and χ^2 test and ϕ test for the error rate.

3.4.2 Treatment Effects

The empirical data was analyzed to determine the treatment effects of the code snippets. We specifically looked at time and error effects of the participants.

3.4.2.1 *Time Effects*

We used the same method as in Closheim's original study to analyze the time effects of the code snippets, namely the t-test and Cohen's d.

For the t-test, we divided the times of the participants into two groups: one group that received the treatment code snippets and one group that received the control code snippets and performed an independent t-test.

To be able to perform an independent t-test, we need data independence, normality and homogeneity of variance. We checked the data independence by ensuring that each data point is independent of each other. We calculated the Shapiro-Wilk test to check the normality of the data, but as the t-test is robust to violations of normality [28, 39], we did not use its results for the data analysis. We used the Levene's test to check the homogeneity of variance. If the data is not normally distributed, we used the Welch's test instead as it calculates its test statistic with both variances.

To be able to compare the effect of the treatments with the data from the simulation evaluation, we calculated the Cohen's d.

3.4.2.2 *Error Rate Effects*

As already mentioned, we used the χ^2 test to analyze the error rate of the participants to compare whether there was a significant difference between the error rate of the treatment and the control code snippets.

To be able to perform a χ^2 test, we need a nominal scale, independent measurements and a sample size of at least five. As already explained in the previous section, we ensured that the data is independent. We also have a nominal scale, as the error rate is either true or false. If a category has less than five data points, we used Fisher's exact test instead of the χ^2 test, as it has no such requirement.

We calculated the ϕ test to be able to compare the effect of the treatments with the data from the simulation evaluation, since it is a measure of association between two binary variables.

3.4.2.3 *Significance Level Correction*

We used the Bonferroni-Holm correction to correct the significance level for multiple comparisons to account for alpha error accumulation. We did not use the Bonferroni correction, as it is too conservative and can lead to a high risk of Type II errors [8].

Since we performed two significance tests per code snippet and we had eight code snippets to analyze, we had a total of 16 significance tests. We used a significance level of $\alpha = 0.05$ as a base value and divided it by the number of tests to get the corrected significance level for the first test. This leads to a corrected significance level of $\alpha = 0.05/16 = 0.003125$ for the first test, with the following tests having a significance level of $\alpha = 0.05/(16 - i)$, where i is the number of the test, with the tests are ordered by their p-value.

3.4.3 Model Training

The training of the cognitive model was done in the same way as in Closheim's original study except for the difference of splitting the data into two sets: one training data set, which was used to train the model, and one test data set, which was used to evaluate the model [13]. We used 80% of the data for training and 20% for testing, as this is a common practice in machine learning. This enabled us to evaluate the model on data that it has not seen before and thus provide a lower risk of overfitting and a higher generalizability of the model.

3.4.3.1 *Distance Metric*

The model was optimized for time and error rate with the help of the KS statistic and the χ^2 test. Additionally, we also performed an optimization of the combined time and error data to find a set of parameters that fit the data best.

Time Data

We used the KS statistic to compare the empirical data with the simulation data, as it is a non-parametric test that does not assume any distribution of the data. Therefore, it is a good choice for our data, as we do not know the distribution of the data. Additionally, it always gave us a significance level at which the two distributions are different, which was used for the combined optimization of time and error data.

Error Rate Data

We used the χ^2 test to compare the empirical data with the simulation data, as we only needed to check if the two distributions are different. As our error rate data is binary, we could not use the KS statistic, as it is not suitable for binary data. Additionally, the χ^2 test also provides a significance level at which the two distributions are different, which was used for the combined optimization of time and error data.

It should be noted that in certain marginal instances, specifically when the column sums were found to be zero, the χ^2 test was bypassed and a value of 0 and a significance of 1 were assumed directly. This is a necessary step because the χ^2 test is not defined for 0 sums. The underlying reason for the efficacy of this particular shortcut is attributable to the constant row sums, which are consistent with the number of simulations and the empirical data. It can thus be concluded that, in the event of the row sums being equal and one of the column sums being 0, the distribution in the other column must be identical.

Combined Data

We used the significance levels of KS statistic and the χ^2 test to compare the empirical data with the simulation data, as we wanted to check if the two distributions are different. We needed to use the significances levels, as the chi^2 test and the KS statistic are not comparable, as they have different ranges.

To avoid the accumulation of distance values, the individual summands were squared. This increased the weight of individual large deviations while keeping smaller deviations relatively small.

3.4.3.2 *Searching Strategy*

To find the best set of parameters, we used a Bayes Optimization algorithm, as it has proven to be more efficient for training ACT-R models compared to a grid search [22]. We only varied the four parameters mentioned above, as the number of parameters should be kept to a minimum to avoid overfitting and reducing its generalizability [33], while also still being able to capture the behavior of the participants due to being a well-constructed model [34]. We also started runs with different amounts of varied parameters for better coverage of the parameter space.

3.4.4 Simulation Evaluation

To evaluate the resulting model, we evaluated the time and error rate data of the model compared to the empirical data. Additionally, we examined if the model was able to replicate observed effects of the code snippets.

Time Data

We used the KS statistic to evaluate the time data of the trained model. By reporting both the F-statistic and the p-value of each group and code snippet, we were able to determine if the model was able to replicate the time data of the empirical evaluation and for each combination of group and code snippet.

Error Rate Data

We used the χ^2 test to evaluate the error rate data of the trained model. By reporting both the χ^2 statistic and the p-value of each group and code snippet, we were able to determine if the model was able to replicate the error rate data of the empirical evaluation and for each combination of group and code snippet.

Treatment Effects

Finally, we also evaluated the effect of the treatments on the code comprehension of the code snippets. We analyzed the treatment effects of the model in the same way as in the empirical evaluation. Afterwards, we compared the results of the model with the results of the empirical evaluation to see if the model was able to replicate the treatment effects.

We also checked if the model predicted effects that were not present in the empirical evaluation and discussed the implications of these effects.

4

Part I Evaluation: Replication Study

In this chapter, we will present and discuss the evaluation of the results collected from the empirical evaluation and the simulation evaluation.

4.1 Results

First, we will present the data collection and data preparation. We will then give a short overview over the socio-demographic data of the participants. Afterward, we will present the results of the empirical evaluation. Finally, we will focus on the fitting of the ACT-R model to the empirical data and the simulation evaluation.

4.1.1 Data Collection and Preparation

The data collection was conducted using CloudResearch. The questionnaire was created using the SoSci[1] platform and was distributed via CloudResearch[2]. We started the questionnaire on 2025-02-10 and received the last results on 2025-02-18. During this time, the questionnaire was closed for four days to ensure the data quality of the first participants.

In total, 682 participants started the questionnaire, of which 145 participants completed the questionnaire. Of these 145 participants, 124 participants completed the questionnaire entirely and were included in the analysis. The high number of participants that started the questionnaire but did not complete it is due to the fact that we used a crowdsourcing platform and performed a screening at the start of the questionnaire, that filtered out those participants who did not meet the requirements. The questionnaire received a rating of 4.3\5 with 73 votes on CloudResearch, which implies that the participants were satisfied with the quality of the questionnaire.

Of the 124 participants, we excluded 28 participants due to completing the study too fast or too slow, i.e less/more than 1.5 times the interquartile range below the first quartile/above the third quartile. Therefore, 96 participants were included in the analysis and evaluation. The participants were randomly assigned to four groups, whose distribution is shown in Table 4.1.

As a reminder, we created four groups to reduce learning effects by changing the order of the code snippets. As groups 1 and 3, and groups 2 and 4, differed solely in the order of their snippet presentation, groups 1 and 3 were combined into group 1, and groups 2 and

1 https://www.soscisurvey.de/
2 https://www.cloudresearch.com/

Group	Count
1	24
2	24
3	23
4	25

Table 4.1: Number of participants randomly assigned to four groups

4 were combined into group 2 for the subsequent evaluations. The two remaining groups differentiate in the type of the code snippet, i.e. treatment or control group, which is used to analyze the effect of the treatment. This resulted in a total of 47 participants for Group 1 and 49 for Group 2. All simulations of the ACT-R model utilised in the subsequent analyses were calculated with these two group sizes.

4.1.2 Socio-Demographic Data

The following section will provide an overview of the participants and their socio-demographic data.

The average participant was 34.8 years old, with a standard deviation of 10.1 years. The youngest participant was 18 years old, while the oldest participant was 64 years old.

The distribution of the age groups can be seen in Figure 4.1. Most people were around 20–40 years old, even though older participants are still present.

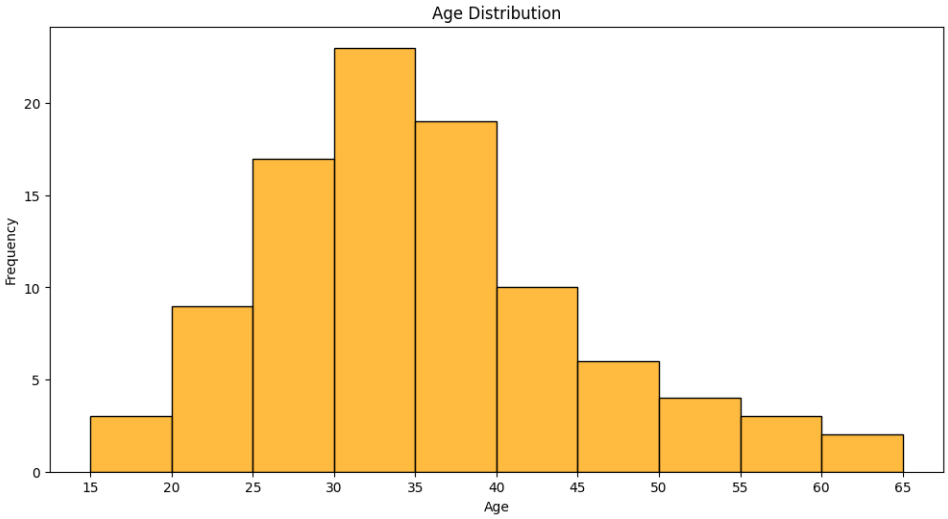


Figure 4.1: Age Groups of Participants

The country distribution of the participants is shown in Figure 4.2. We can clearly see that the majority of participants were from the USA, followed by Canada and the United Kingdom. This is probably due to the fact that the questionnaire was distributed via CloudResearch, which is a crowdsourcing platform that is mainly used in the USA and

other english-speaking countries. Additionally, CloudResearch is not available in some countries, for example in Germany.

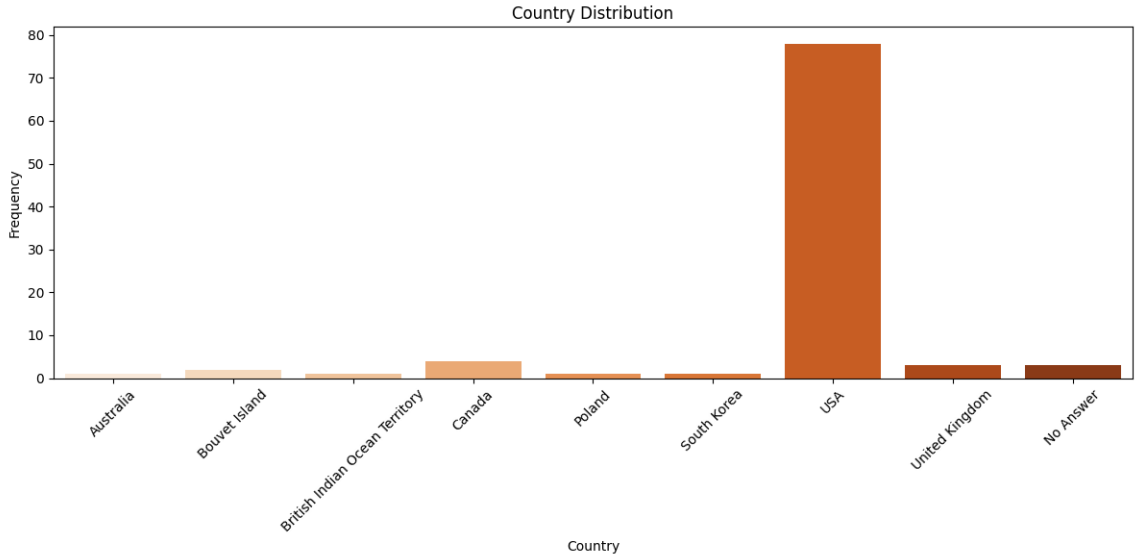


Figure 4.2: Country Distribution of Participants

The participants were asked to indicate their highest level of education. The distribution of the education levels is shown in Figure 4.3 The majority of participants had a bachelor's degree with 64.8%, followed by a master's degree with 18.8% and a high school diploma with 15.6%.

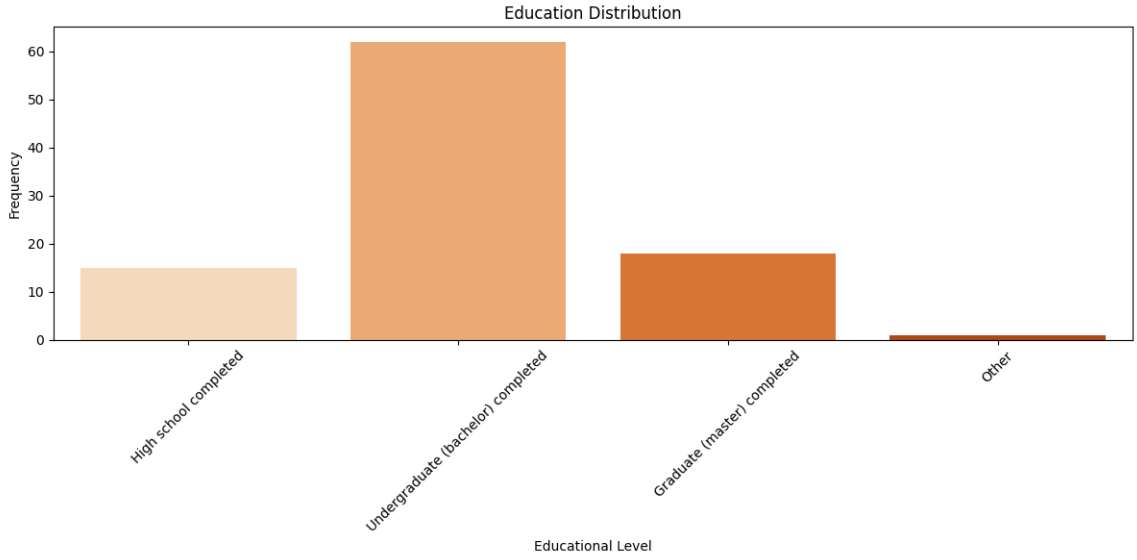


Figure 4.3: Education Level of Participants

The participants were also asked to indicate their gender identity. The most common gender of the participants with around 78% was male. Woman followed with around 18%, while the remaining 4% of participants identified as non-binary.

4.1.3 Empirical Evaluation

The following chapter will present the results of the empirical evaluation. We will present the data of both code snippets for each group, due to having different changes (i.e. similarity and distance changes) and the results of the statistical analysis. Each type of treatment will be discussed separately to allow for a better understanding of the results.

For each type of treatment, we will analyse the results of the empirical evaluation with regard to the time data and the error data.

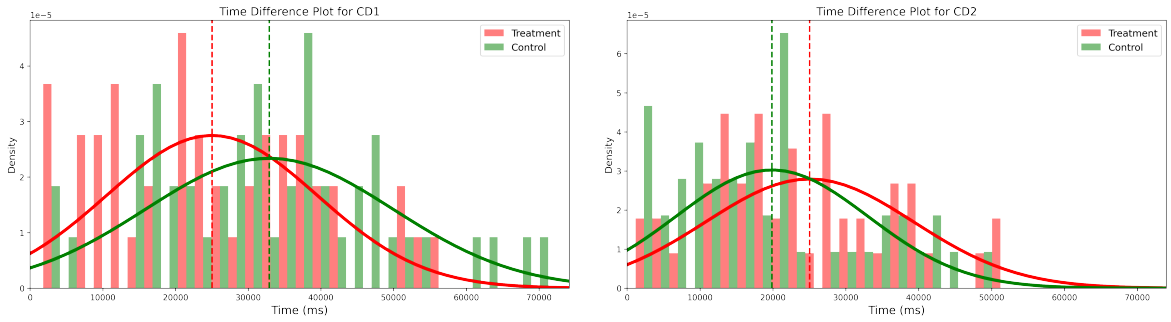
We used the t-test to compare the means of the two groups. A value of $p < 0.05$ was considered statistically significant, but was later adjusted with the Bonferroni correction in Section 4.1.3.5 to account for multiple comparisons. While our group sizes are great enough to assume normality, we also checked the normality of the data with the Shapiro-Wilk test. The results of the Shapiro-Wilk test are shown in the tables for each code snippet, but were not used. Additionally, we tested for homogeneity of variance with the Levene test, which is also reported in the tables for each code snippet. In the following, if the Levene test says that the data is not normally distributed, we used the Welch's t-test instead of the t-test without reporting it again. The values are reported in the tables in the same column as the t-test.

4.1.3.1 *Code Distance*

The code distance treatment was designed to investigate the influence of the distance between the declaration of a variable and its usage at a later point in the code snippet.

Time Data

The results of the time data of code distance treatment are shown in Figure 4.4.



(a) (b)

Figure 4.4: Time Data for Code Distance Snippet CD1 (a) and CD2 (b)

We can see that for code snippet CD1, the treatment group was faster by around 8 seconds on average than the control group. This was unexpected, as the treatment group should have been slower due to the distance between the declaration and usage of the variable. The

change introduced for CD1 was a higher distance between the declaration and usage of the variable.

For code snippet CD2, the treatment group was slower by around 5 seconds on average than the control group. The change introduced for CD2 was similarity between variable names. This result was expected, as the treatment group had to deal with similar variable names.

The results of the statistical analysis can be seen in Table 4.2.

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
CD1	0.958	0.089	0.978	0.502	0.978	0.502	2.435	**0.017**	0.497
CD2	0.934	**0.010**	0.957	0.074	0.139	0.710	1.840	0.069	0.376

Table 4.2: Statistical Analysis for CD1 and CD2

For CD1, there is a significant difference between groups, supported by a medium effect size. For CD2, the difference is not statistically significant, but a small to medium effect size is observed. The assumptions for the t-test (normality and homogeneity of variance) are mostly met, except for the control group of CD2, which deviates from normality.

Error Data

The values of the error data analysis can be seen in Table 4.3.

The analysis of the CD1 and CD2 error rates shows slight differences in correctness between the treatment and control groups. For CD1, the control group achieved a higher correctness rate (65.30%) compared to the treatment group (53.19%), but the difference was not statistically significant ($p = 0.317$) and showed a small effect size ($\phi = 0.102$). For CD2, the correctness rates of the treatment (63.83%) and control (65.31%) groups were nearly identical, with no statistical significance ($p = 1.0$) and no effect size ($\phi = 0.0$). These results suggest that while there are observable differences in CD1, they are not strong enough to be statistically significant. For CD2, the performance of both groups is nearly identical.

Overall, the findings indicate no strong association between group type and correctness for either CD1 or CD2.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	22	25	53.19	1.0	0.317	0.102
Control	17	32	65.30			

Table 4.3: CD1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	17	32	63.83	0.0	1.0	0.0
Control	17	30	65.31			

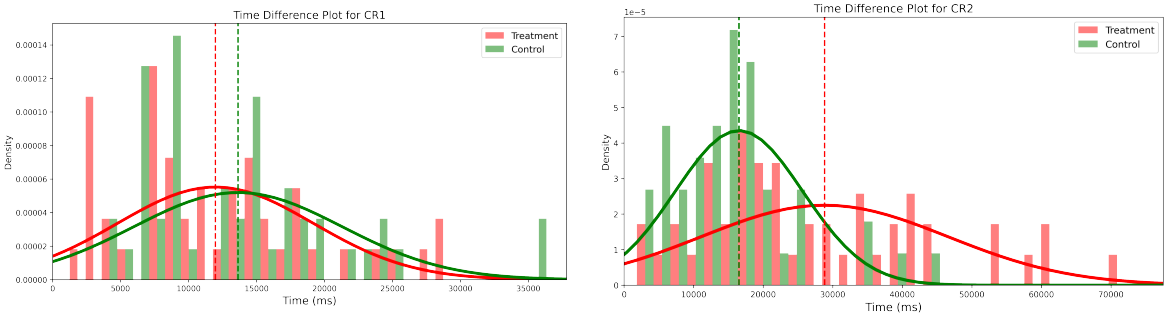
Table 4.4: CD2 Error Rate

4.1.3.2 *Repeated Code*

The repeated code treatment was designed to investigate the influence of having multiple declarations of variables in the same code snippet.

Time Data

The results of the time data of repeated code treatment are shown in Figure 4.5.



(a) (b)

Figure 4.5: Time Data for RepeatedCode Snippet CR1 (a) and CR2 (b)

We can see that for code snippet CR1, the treatment group was slightly faster by around 1.5 seconds on average than the control group. The change introduced for CR1 was similarity between variable names.

For code snippet CR2, the treatment group was slower by around 12 seconds on average than the control group. The change introduced for CR2 was a higher distance between the declaration and usage of the variable. This result was expected, as the treatment group had to deal with a higher distance between the declaration and usage of the variable.

The results of the statistical analysis can be seen in Table 4.5.

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
CR1	0.924	**0.004**	0.931	**0.008**	0.039	0.845	1.090	0.278	0.223
CR2	0.927	**0.006**	0.942	**0.017**	12.707	**0.001**	4.242	**<0.001**	0.866

Table 4.5: Statistical Analysis for CR1 and CR2

For CR1, there was no significant difference between groups, with a small effect size. For CR2, there was a significant difference with a large effect size, but the variances were unequal.

Error Data

The analysis of the CR1 and CR2 error rates shows differences in correctness between the treatment and control groups. For CR1, the control group achieved a higher correctness rate (93.88%) compared to the treatment group (78.72%), with a small to medium effect size ($\phi = 0.191$), but the difference was not statistically significant ($p = 0.061$). For CR2, the treatment group performed slightly better (91.84%) than the control group (85.11%), but the difference was also not statistically significant ($p = 0.475$) and showed a negligible effect size ($\phi = 0.073$). These results suggest that while there are observable differences

in correctness rates, they lack statistical significance. The effect sizes indicate weak or no association between group type and correctness.

Overall, the findings highlight no strong evidence of a significant advantage for either group in these conditions.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	10	37	78.72	3.500	0.061	0.191
Control	3	46	93.88			

Table 4.6: CR1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	4	45	91.84	0.510	0.475	0.073
Control	7	40	85.11			

Table 4.7: CR2 Error Rate

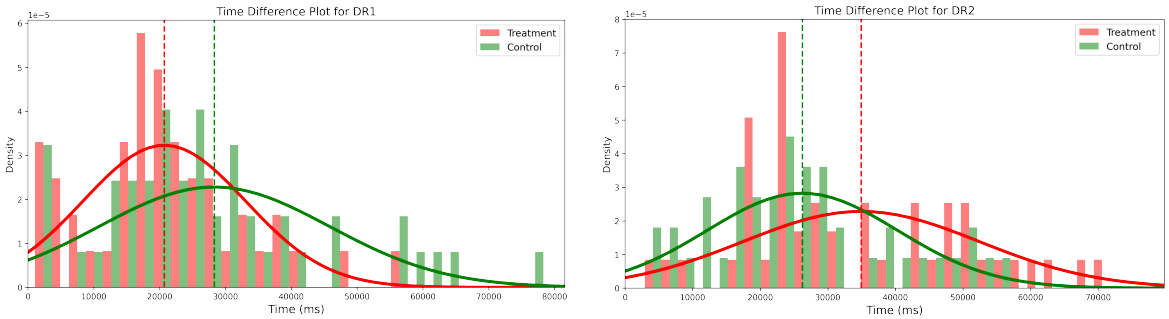
4.1.3.3 *Declaration Redeclaration Distance*

The declaration redeclaration distance treatment was designed to investigate the interaction between the CD and CR treatments. Specifically, the treatment consists of having a double declaration of a variable at the beginning of the code snippet (CR treatment) and a usage of the variable at the end of the code snippet (CD treatment).

The control code snippets have their second declaration of the double declaration in the line before the usage of the variable.

Time Data

The results of the time data of repeated code treatment are shown in Figure 4.6.



(a) (b)

Figure 4.6: Time Data for Declaration Redeclaration Snippet DR1 (a) and DR2 (b)

We can see that for code snippet DR1, the treatment group was faster by around 8 seconds on average than the control group. The change introduced for DR1 was similarity between variable names.

For code snippet DR2, the treatment group was slower by around 9 seconds on average than the control group. The change introduced for DR2 was a higher distance between the declaration and usage of the variable. This result was expected, as the treatment group had to deal with a higher distance between the declaration and usage of the variable.

The results of the statistical analysis can be seen in Table 4.8.

For both DR1 and DR2, a significant difference is observed between the treatment and control groups, with medium to large effect sizes. The assumptions for the t-test are largely

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
DR1	0.939	**0.013**	0.954	0.063	3.080	0.083	2.445	**0.016**	0.499
DR2	0.959	0.095	0.947	**0.029**	2.599	0.110	2.677	**0.009**	0.546

Table 4.8: Statistical Analysis for DR1 and DR2

met, with the exception of the control group in DR1, which exhibits a minor deviation from normality.

Error Data

The analysis of the DR1 and DR2 error rates reveals distinct patterns in performance between the treatment and control groups. For DR1, the control group achieved a higher correctness rate (67.35%) compared to the treatment group (57.45%), but the difference was not statistically significant ($p = 0.429$) and showed a negligible effect size ($\phi = 0.081$). In DR2, the treatment group performed better (65.31%) than the control group (46.81%), but again, the difference was not statistically significant ($p = 0.105$) with a small effect size ($\phi = 0.165$). These results suggest that while there are observable differences in correctness rates, they lack statistical significance and are associated with small or negligible effect sizes.

Overall, the findings indicate no strong association between group type and correctness for either DR1 or DR2.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	20	27	57.45	0.625	0.429	0.081
Control	16	33	67.35			

Table 4.9: DR1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	17	32	65.31	2.626	0.105	0.165
Control	25	22	46.81			

Table 4.10: DR2 Error Rate

4.1.3.4 *Repeated Distance*

The repeated distance treatment manipulated the position of filler lines relative to a double declaration and its usage. In the treatment group, some filler lines were inserted between the double declaration and its subsequent usage. In contrast, in the control group, the same filler lines appeared before the double declaration.

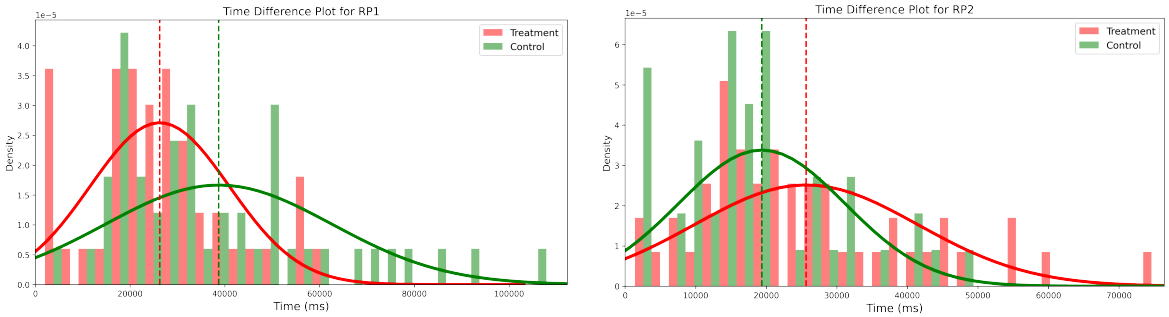
Time Data

The results of the time data of repeated code treatment are shown in Figure 4.7.

We can see that for code snippet RP1, the treatment group was faster by around 12.5 seconds on average than the control group. The change introduced for RP1 was a higher distance between the declaration and usage of the variable.

For code snippet RP2, the treatment group was slower by around 6 seconds on average than the control group. The change introduced for RP2 was similarity between variable names.

The results of the statistical analysis can be seen in Table 4.11.



(a) (b)

Figure 4.7: Time Data for Repeated Distance Snippet RP1 (a) and RP2 (b)

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
RP1	0.930	**0.006**	0.954	0.060	6.537	**0.012**	3.056	**0.003**	0.624
RP2	0.943	**0.023**	0.931	**0.007**	2.526	0.115	2.195	**0.031**	0.448

Table 4.11: Statistical Analysis for RP1 and RP2

For both conditions (RP1 and RP2), the t-tests demonstrate significant differences between the groups with large effect sizes. The assumption of normality is violated for some groups, which has the potential to influence the interpretation of the results. It is evident that the variance homogeneity is violated for RP1, which is why we used the Welch t-test. In RP2, however, the variances are homogeneous.

Error Data

The statistical analysis highlights significant differences between treatment and control groups in several conditions. For RP1, the control group outperformed the treatment group with a large effect size ($\phi = 0.442$), while RP2 showed no significant difference between groups ($\phi = 0.0$). The t-tests for time data revealed significant differences for both RP1 and RP2, with medium to large effect sizes favoring the control group. Normality assumptions were violated in some cases, particularly for RP1, which may affect the interpretation of results. The error rate analysis for RP1 showed a strong association between group type and correctness, while RP2 demonstrated no such association.

Overall, the control group consistently performed better in RP1, where the differences were more pronounced. These findings suggest that the control group had an advantage in correctness due to the nature of the code snippets and the treatment applied.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	30	17	36.17	18.714	**<0.001**	0.442
Control	9	40	81.63			

Table 4.12: RP1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	17	32	65.31	0.0	1.0	0.0
Control	16	31	65.96			

Table 4.13: RP2 Error Rate

4.1.3.5 *Significance Level Correction*

We checked the significance level correction for the t-tests and χ^2 tests of all eight code snippets. Of those 16 values, seven values were significant at the 0.05 level. The significance level correction can be seen in Table 4.14.

	CR2 t-test	RP1 χ^2	CD1 t-test	CD2 t-test	DR1 t-test	DR2 t-test	RP2 t-test
p-value	<0.001	<0.001	0.003	0.009	0.016	0.017	0.031
Bound	0.003125	0.003333	0.003571	0.003846	0.004167	0.004545	0.005
Significant	Yes	Yes	Yes	No	No	No	No

Table 4.14: Significance Level Correction

The values of the t-tests for CD1, CD2, DR1, and DR2 were not significant after the correction and may have been due to chance. The values of the t-tests for CR2, RP1, and RP2 were significant after the correction and are not due to chance.

4.1.4 Simulation Evaluation

In this section, we will analyze the results of the fitting of the model to the data of the simulation evaluation. Afterwards, we will analyze the results of the fitted model.

4.1.4.1 *Model Fitting*

We performed different types of optimizations with the data of the empirical evaluation to train the parameters of the ACT-R model. Specifically, we trained either three (ans, lf, rt) or four parameters (ans, lf, rt, mp) of the model. We used the following ranges for the parameters:

- ans: [0.01, 2.0]
- lf: [0.1, 5.0]
- rt: [-10.0, 10.0]
- mp: [0.2, 2.0]

The parameter mp was set to `1.0` if not varied, as it seemed like a good value during the tests that were run prior to starting the optimization. The Bayesian Optimization tried to maximize either the time data with the help of the KS statistic or the error data with the help of the χ^2 statistic. Additionally, we also tried to maximize the significance of the combined time and error data.

Due to time constrains, we did not find an maximum for the optimization of the model parameters. The experiments were run for 12 days, but due to the increased model complexity, the run time of a single optimization took too long to run more steps.

In the following, we will describe the best parameters in regard to time data, error data and combined data for the optimizations with three and four parameters.

Time Data

The optimization was run for 640 steps. There was no convergence of the optimization, as we needed to stop the optimization due to time constraints. The best possible reachable sum of squared KS statistics was -1.239 and found at the 621st step of the optimization. The parameters for this run are described in Table 4.15 (a).

Parameter	Value
ans	0.418
lf	4.253
rt	-0.204
mp	1.860

(a)

Parameter	Value
ans	0.010
lf	5.000
rt	-0.056
mp	0.200

(b)

Parameter	Value
ans	0.014
lf	4.066
rt	-8.100
mp	1.0

(c)

Table 4.15: Best Parameters for Time (a), Error (b) and Combined Data (c)

The evolution of the Bayesian Optimization can be seen in Figure 4.8.

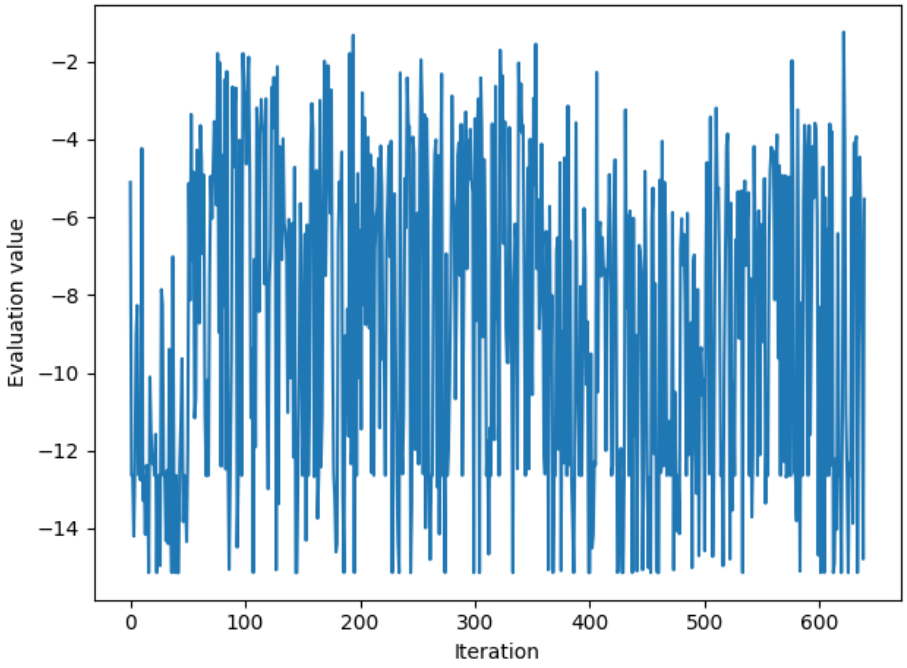
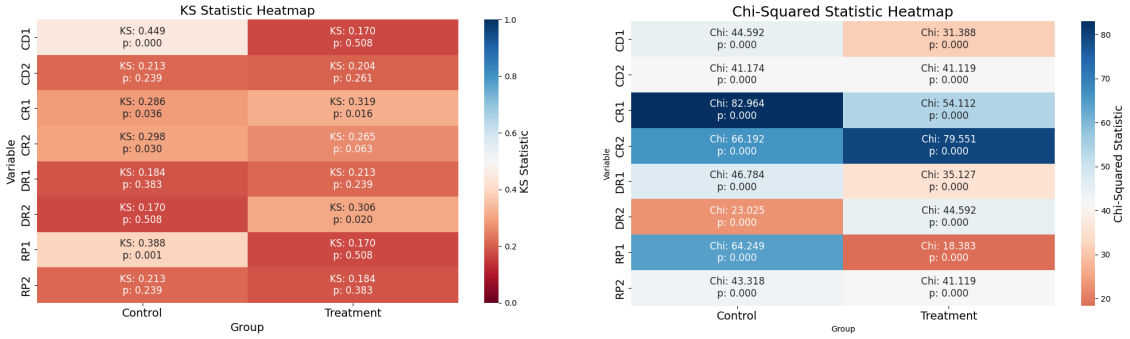


Figure 4.8: Evolution of the BO algorithm with 50 start points for tuning ans, lf, rt, mp for time data

The accuracy in terms of time data and error rate of this approximation can be seen in Figure 4.9.

If we consider Figure 4.9 (b), we can see that all values are very high with all significance values being 0.0. This means that the model was not able at all to predict the responses of the participants. This was to be expected, as we only optimized in regards to time for this optimization. If we have a look at the error data, we can see that the model did not return

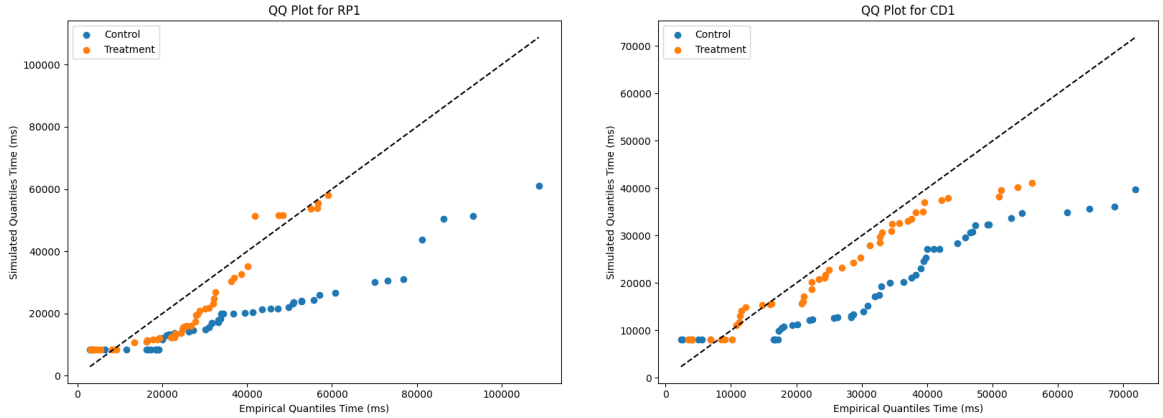


(a) (b)

Figure 4.9: Heatmaps of KS statistic (a) and χ^2 statistic (b) for tuning ans, lf, rt, mp for time data

an answer for nearly every code snippet and thus had a false result. This optimization is therefore not realistic and does not help in providing a useful model, even if the time values are quite good.

As can be seen in Figure 4.9 (a), the worst value is 0.449. Over half of the code snippets have a significance value of over 0.05, which means that the probability that the simulated and empirical data have the same distribution is correspondingly high.



(a) (b)

Figure 4.10: QQ Plots for RP1 (a) and CD1 (b) tuning ans, lf, rt, mp for time data

We used QQ plots to visualize the difference between the empirical and simulation data. As we can see in Figure 4.10, the overall direction seems to be correct, but the values are shifted to the bottom for those code snippets that are not significant. Mostly, the simulation is faster than the empirical data, leading to this shift to the bottom.

Error Data

The optimization was run for 628 steps. There was no convergence of the optimization, as we needed to stop the optimization due to time constraints. The best possible reachable sum for the χ^2 statistics was -964.353 found at the 104th step of the optimization. The parameters

for this run are described in Table 4.15 (b). The evolution of the Bayesian Optimization can be seen in Figure 4.11.

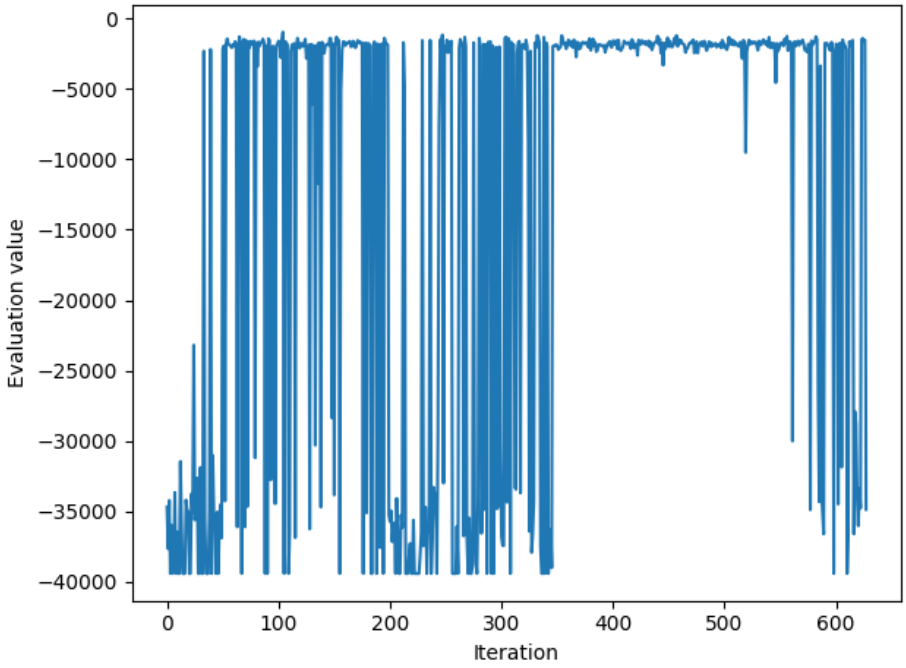
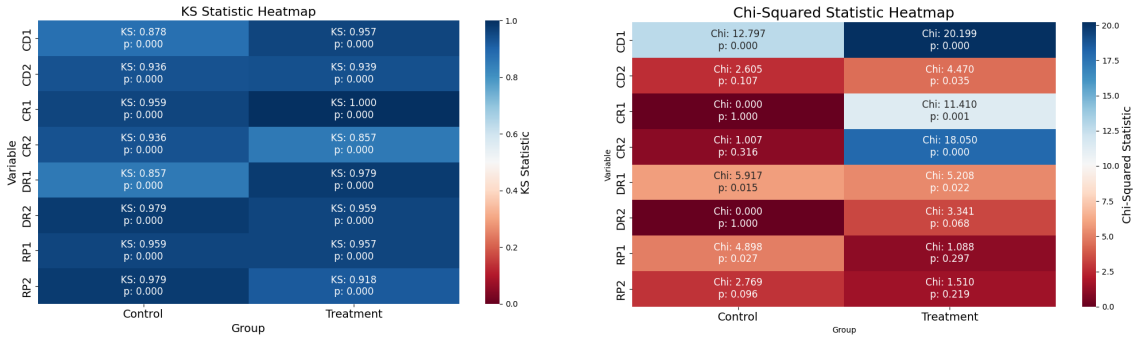


Figure 4.11: Evolution of the BO algorithm with 50 start points for tuning ans, lf, rt, mp for error data

The accuracy in terms of time data and error rate of this approximation can be seen in Figure 4.12.



(a) (b)

Figure 4.12: Heatmaps of KS statistic (a) and χ^2 statistic (b) for tuning ans, lf, rt, mp for error data

As we can see in Figure 4.12 (a), the simulated time does not fit the empirical data at all. This result could be expected as we only optimized in regards to correctness of the answers. Nevertheless, the distribution of the time data is vastly different due to the simulated time data not having nearly as much variance as the empirical data. The simulated times are concentrated at around 60 seconds, whereas the empirical data has time values between 3–70 seconds for the code snippet CD1.

	False	True	Correctness (%)
Control	2 (17)	47 (32)	96% (65%)
Treatment	2 (22)	45 (25)	96% (53%)

Table 4.16: CD1 Error Rate for simulated data with empirical data in parentheses

	False	True	Correctness (%)
Control	25 (16)	22 (31)	47% (66%)
Treatment	24 (17)	25 (32)	51% (65%)

Table 4.17: RP2 Error Rate for simulated data with empirical data in parentheses

The correctness of two code snippets are shown in Table 4.16 and Table 4.17. The values in the brackets describes the values of the empirical data. Half of the code snippets have a significance value of over 0.05, which means that the probability that the simulated and empirical data have similar distributions is correspondingly high. Nevertheless, some χ^2 values are very high, which means that the model was not able at all to predict the responses of the participants. This is especially true for the code snippets CD1, which can be seen in Table 4.16.

Combined Data

The optimization was run for 635 steps. There was no convergence of the optimization, as we needed to stop the optimization due to time constraints. The best possible reachable sum for the combined significance was 5.084 found at the 302th step of the optimization. The parameters for this run are described in Table 4.15 (c).

The evolution of the Bayesian Optimization can be seen in Figure 4.13.

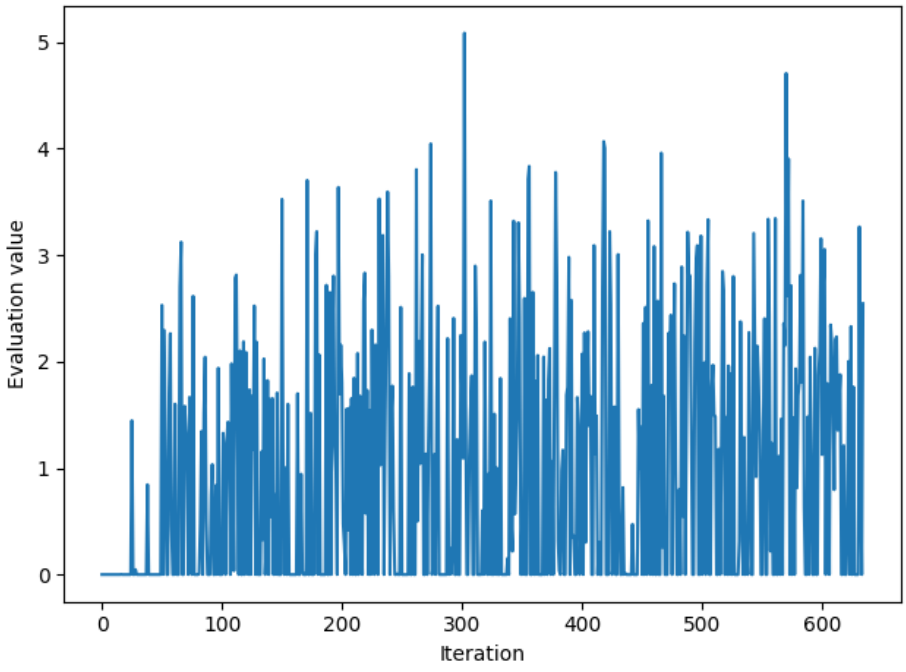
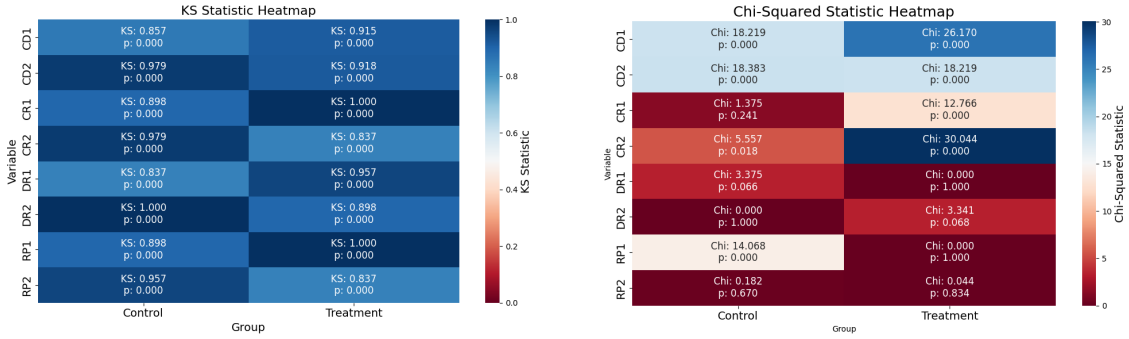


Figure 4.13: Evolution of the BO algorithm with 50 start points for tuning ans, lf, rt for combined data

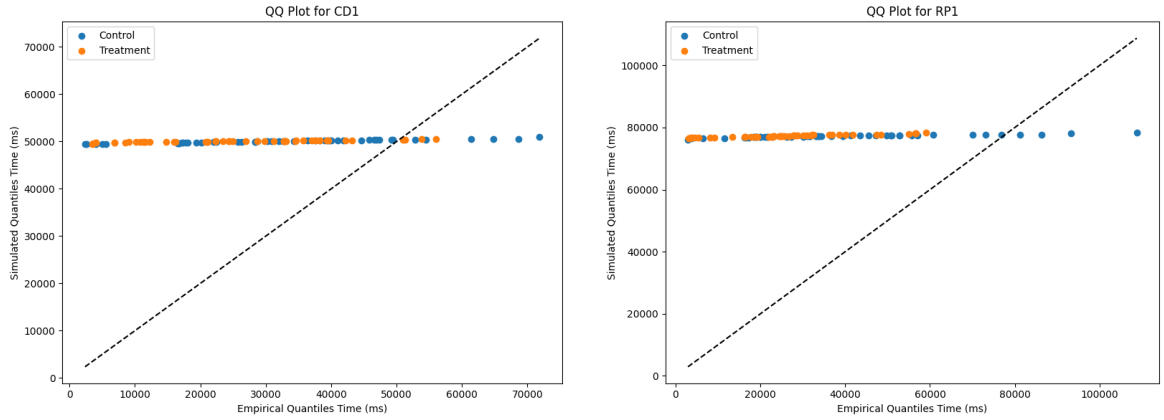
The accuracy in terms of time data and error rate of this approximation can be seen in Figure 4.14.



(a) (b)

Figure 4.14: Heatmaps of KS (a) statistic and χ^2 statistic (b) for tuning ans, lf, rt, mp for combined data

As we can see in Figure 4.14 (a), the simulated time does not fit the empirical data at all, similar to the error data. Here, we did expect to have some compromise between the time and error data, but the model was not able to correctly predict the time needed to solve the code snippets. The distribution of the time data is vastly different due to the simulated time data not having nearly as much variance as the empirical data, which can be seen in the QQ plots in Figure 4.15.



(a) (b)

Figure 4.15: QQ Plots for CD1 (a) and RP1 (b) tuning ans, lf, rt for time data

4.1.4.2 *Model Evaluation*

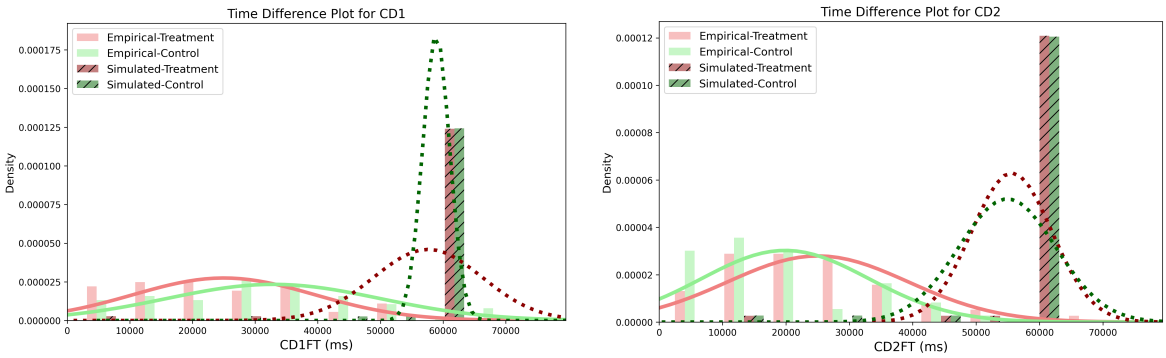
Finally, we will evaluate the model with the help of the KS statistic and the χ^2 statistic, as we did for the empirical evaluation. For this evaluation, we used the results of the simulation with the parameters that were found in Section 4.1.4.1. We chose the error data optimization, as it was the most accurate. As all different optimizations, except for the time optimization, were not able to predict the time data, we chose the model with the best error data. The time data was not chosen, as it was not a faithful representation of the empirical data, due to mostly not returning any answers for the code snippets.

4.1.4.3 *Code Distance*

The code distance treatment was designed to investigate the influence of the distance between the declaration of a variable and its usage at a later point in the code snippet.

Time Data

The results of the time data of code distance treatment are shown in Figure 4.16.



(a) (b)

Figure 4.16: Time Data for Code Distance Snippet CD1 (a) and CD2 (b)

We can see that for both code snippets, the simulation time data is much higher and less distributed than the empirical time data. The simulation time data is concentrated at around 60 seconds, whereas the empirical data has time values between 3–70 seconds for the code snippet CD1. Therefore, the time data of the simulation is not a good representation of the empirical data.

The results of the statistical analysis can be seen in Table 4.18.

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
CD1	0.921 (0.953)	**0.004 (0.058)**	0.949 (0.986)	**0.034** (0.817)	0.001 (0.619)	0.971 (0.433)	2.567 (0.455)	**0.012** (0.650)	0.524 (0.093)
CD2	0.943 (0.939)	**0.020 (0.013)**	0.981 (0.944)	0.636 (0.024)	0.019 (0.005)	0.891 (0.941)	-0.929 (-0.151)	0.355 (0.880)	-0.190 (-0.031)

Table 4.18: Statistical Analysis for CD1 and CD2 with empirical data in brackets

The analysis for CD1 shows that the simulation data does not deviate from normality ($p = 0.358$) and exhibits significant variance differences ($p = 0.002$), unlike the empirical data. The simulation indicates no significant treatment effect ($p = 0.728$) with a negligible effect size ($d = -0.071$), whereas the empirical data shows a significant difference ($p = 0.017$) with a medium effect size ($d = 0.497$). For CD2, both the simulation and empirical data align

more closely, showing no significant differences ($p = 0.588$ and $p = 0.069$) and small effect sizes ($d = -0.111$ and $d = 0.376$).

These results suggest that the simulation underestimates the treatment effect for CD1 but accurately reflects the minimal impact of the treatment for CD2. The discrepancies in variance and effect size for CD1 highlight areas where the model requires refinement to better replicate human behavior.

Error Data

The values of the error data analysis can be seen in Table 6.23.

The error rate analysis for CD1 and CD2 reveals significant differences between the simulation and empirical data. For CD1, the simulation data shows near-perfect correctness for both the treatment and control groups, which significantly overestimates the empirical results. Similarly, for CD2, the simulation data indicates higher correctness rates for both groups compared to the empirical data, particularly for the treatment group.

The (χ^2) tests for both CD1 and CD2 show no significant differences ($p = 1.0$ and $p = 0.715$) in the simulation data, which aligns with the empirical results. However, the consistently higher correctness rates in the simulation suggest that the model underestimates the cognitive challenges posed by the treatment, increased code distance and variable similarity.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	2 (22)	45 (25)	95.74 (53.19)	0.0	1.0	0.0
Control	2 (17)	47 (32)	95.92 (65.30)	(1.0)	(0.317)	(0.102)

Table 4.19: CD1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	7 (17)	42 (32)	85.71 (63.83)	0.133	0.715	0.037
Control	9 (17)	38 (30)	80.85 (65.31)	(0.0)	(1.0)	(0.0)

Table 4.20: CD2 Error Rate

4.1.4.4 *Repeated Code*

The repeated code treatment was designed to investigate the influence of having multiple declarations of variables in the same code snippet.

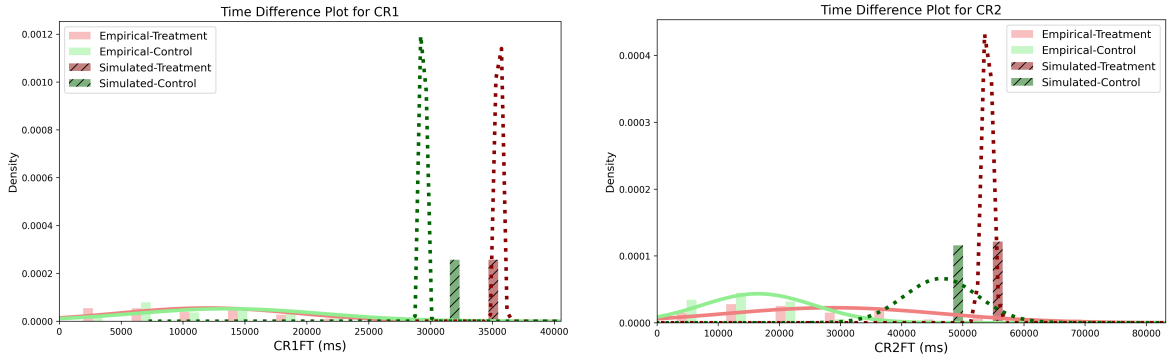
Time Data

The results of the time data of repeated code treatment are shown in Figure 4.17.

Here again, we can see that for both code snippets, the simulation time data is much higher and less distributed than the empirical time data. The simulation time data is concentrated at around 30–35 seconds, whereas the empirical data has time values between 2–25 seconds for the code snippet CR1. Therefore, the time data of the simulation is not a good representation of the empirical data.

The results of the statistical analysis can be seen in Table 4.21.

For code snippet CR1, the simulation data shows a significant difference between the treatment and control groups ($p < 0.001$), with a very large effect size ($d = 26.716$). This contrasts with the empirical data, where no significant difference was observed ($p = 0.278$) and the effect size was small ($d = 0.223$). The treatment groups of both the simulation and empirical data are similar, but the control group in the empirical data performed faster and



(a) (b)

Figure 4.17: Time Data for Repeated Code Snippet CR1 (a) and CR2 (b)

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	**p**	**Value**	**p**	**Value**	**p**	**Value**	**p**	
CR1	0.979 (0.924)	0.476 **(0.004)**	0.979 (0.931)	0.513 **(0.008)**	0.631 (0.039)	0.429 (0.845)	130.853 (1.090)	**<0.001** (0.278)	26.716 (0.223)
CR2	0.291 (0.927)	**<0.001** (0.006)	0.396 (0.942)	**<0.001** **(0.017)**	2.298 (12.707)	0.133 **(0.001)**	-8.781 (4.242)	**<0.001** **(<0.001)**	-1.793 (0.866)

Table 4.21: Statistical Analysis for CR1 and CR2 with empirical data in brackets

exhibited more variability compared to the simulation data.

For code snippet CR2, the simulation data also shows a significant difference between the treatment and control groups ($p < 0.001$), with a large negative effect size ($d = -1.793$). This is in contrast to the empirical data, which also shows a significant difference ($p < 0.001$) but with a still large but smaller positive effect size ($d = 0.866$). The simulation data indicates a much larger discrepancy between the groups in the other direction compared to the empirical data, suggesting that the model overestimates the treatment effect.

The results of the statistical analysis highlight that the simulation overestimates the differences between the treatment and control groups for both CR1 and CR2, particularly in terms of effect sizes.

Error Data

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	27 (10)	20 (37)	42.55 (78.72)	24.443	**<0.001**	0.505
Control	4 (3)	45 (46)	91.84 (93.88)	(3.500)	(0.061)	(0.191)

Table 4.22: CR1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	24 (4)	25 (45)	51.02 (91.84)	19.477	**<0.001**	0.450
Control	3 (7)	44 (40)	93.62 (85.11)	(0.510)	(0.475)	(0.073)

Table 4.23: CR2 Error Rate

For CR1, the simulation data shows a correctness rate of 42.55% for the treatment group, which is significantly lower than the empirical rate of 78.72%. Similarly, the control group in the simulation achieved a correctness rate of 91.84%, slightly lower than the empirical rate of 93.88%. The χ^2 test for CR1 indicates a significant difference ($p < 0.001$) in the simulation data, with a large effect size ($\phi = 0.505$), contrasting with the empirical data, which showed no significant difference ($p = 0.061$) and a small effect size ($\phi = 0.191$). Additionally, the answer distribution has a small peak for an error that is due to the treatment, which is also present in the empirical data, but not as pronounced as in the simulation data.

For CR2, the simulation data shows a correctness rate of 51.02% for the treatment group, which is significantly lower than the empirical rate of 91.84%. The control group in the simulation achieved a correctness rate of 93.62%, slightly higher than the empirical rate of 85.11%. The χ^2 test for CR2 also indicates a significant difference ($p < 0.001$) in the simulation data, with a large effect size ($\phi = 0.450$), whereas the empirical data showed no significant difference ($p = 0.475$) and a negligible effect size ($\phi = 0.073$).

Overall, the findings suggest that the simulation overestimates the differences in correctness rates for both CR1 and CR2. While the empirical data shows no significant differences between the treatment and control groups, the simulation predicts significant differences with large effect sizes.

4.1.4.5 *Declaration Redeclaration Distance*

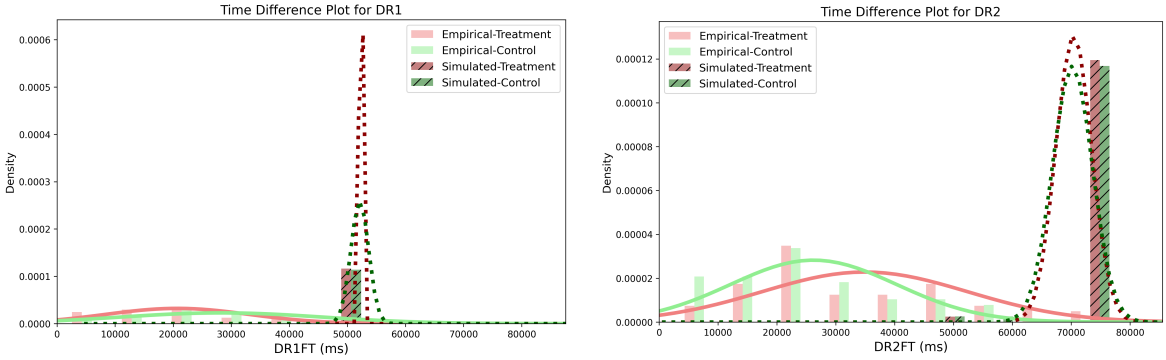
The declaration redeclaration distance treatment was designed to investigate the interaction between the CD and CR treatments. Specifically, the treatment consists of having a double

declaration of a variable at the beginning of the code snippet (CR treatment) and a usage of the variable at the end of the code snippet (CD treatment).

The control code snippets have their second declaration of the double declaration in the line before the usage of the variable.

Time Data

The results of the time data of repeated code treatment are shown in Figure 4.18.



(a) (b)

Figure 4.18: Time Data for Declaration Redeclaration Snippet DR1 (a) and DR2 (b)

Again we can see that for both code snippets, the simulation time data is much higher and less distributed than the empirical time data. The simulation time data is concentrated at around 75 seconds, whereas the empirical data has time values between 5–75 seconds for the code snippet DR2. Therefore, the time data of the simulation is not a good representation of the empirical data.

The results of the statistical analysis can be seen in Table 4.24.

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
DR1	0.268 (0.939)	**<0.001 (0.013)**	0.984 (0.954)	0.744 (0.063)	1.143 (3.080)	0.288 (0.083)	0.925 (2.445)	0.357 **(0.016)**	0.189 (0.499)
DR2	0.258 (0.959)	**<0.001** (0.095)	0.196 (0.947)	**<0.001 (0.029)**	0.090 (2.599)	0.765 (0.110)	-0.344 (2.677)	0.732 **(0.009)**	-0.070 (0.546)

Table 4.24: Statistical Analysis for DR1 and DR2 with empirical data in brackets

For DR1, the simulation data shows no significant difference between the control and treatment groups ($p = 0.357$), with a small effect size ($d = 0.189$). This contrasts with the empirical data, which indicates a significant difference ($p = 0.016$) and a medium effect size ($d = 0.499$). The simulation underestimates the treatment effect compared to the empirical data.

For DR2, the simulation data also shows no significant difference between the groups ($p = 0.732$), with a negligible effect size ($d = -0.070$). This contrasts with the empirical data, which reports a significant difference ($p = 0.009$) and a medium effect size ($d = 0.546$). The simulation again underestimates the treatment effect observed in the empirical data.

Overall, the findings suggest that the simulation underestimates the treatment effects for both DR1 and DR2, particularly in terms of effect sizes and statistical significance.

Error Data

The analysis of the error rates for DR1 and DR2 reveals notable differences between the simulation and the empirical data. For DR1, the treatment group in the simulation achieved a correctness rate of 31.91%, which is significantly lower than the empirical rate of 57.45%. Similarly, the control group in the simulation had a correctness rate of 40.82%, also lower than the empirical rate of 67.35%. The χ^2 test for DR1 shows no significant difference ($p = 0.488$) in the simulation data, in agreement with the empirical results ($p = 0.429$).

For DR2, the treatment group in the simulation achieved a correctness rate of 44.90%, which is lower than the empirical rate of 65.31%. The control group in the simulation achieved a correctness rate of 44.68%, which is similar to the empirical rate of 46.81%. The χ^2 test for DR2 shows no significant difference ($p = 1.000$) in the simulation data, which contrasts with the empirical results ($p = 0.105$) that showed a small effect size.

Overall, the simulation underestimates the correctness rates for both DR1 and DR2, especially for the treatment groups. While the simulation aligns with the empirical results in terms of statistical significance for DR1, it fails to capture the differences observed in the empirical data for DR2.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	32 (20)	15 (27)	31.91 (57.45)	0.481	0.488	0.071
Control	29 (16)	20 (33)	40.82 (67.35)	(0.625)	(0.429)	(0.081)

Table 4.25: DR1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	27 (17)	22 (32)	44.90 (65.31)	0.0	1.0	0.0
Control	26 (25)	21 (22)	44.68 (46.81)	(2.626)	(0.105)	(0.165)

Table 4.26: DR2 Error Rate

4.1.4.6 *Repeated Distance*

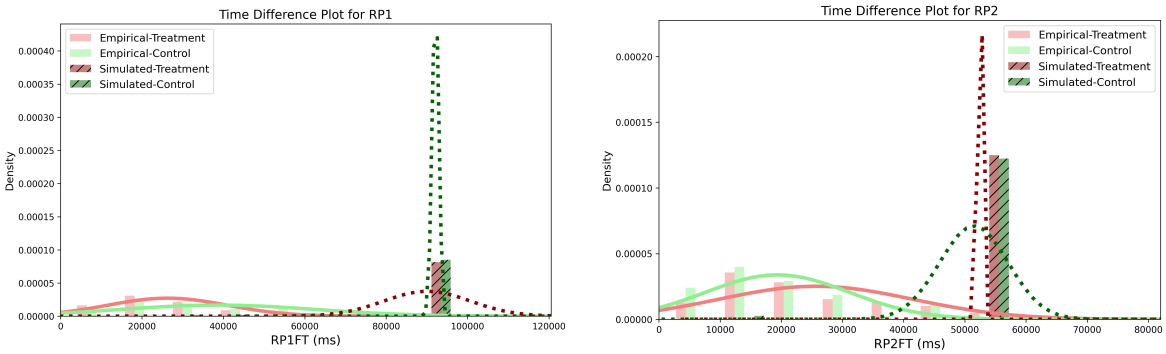
The repeated distance treatment manipulated the position of filler lines relative to a double declaration and its usage. In the treatment group, some filler lines were inserted between the double declaration and its subsequent usage. In contrast, in the control group, the same filler lines appeared before the double declaration.

Time Data

The results of the time data of repeated code treatment are shown in Figure 4.19.

The last type of treatment also shows that the simulation time data is much higher and less distributed than the empirical time data. The simulation time data is concentrated at around 52 seconds, whereas the empirical data has time values between 3–50 seconds for the code snippet RP1. Therefore, the time data of the simulation is not a good representation of the empirical data.

The results of the statistical analysis can be seen in Table 4.27.



(a) (b)

Figure 4.19: Time Data for Repeated Distance Snippet RP1 (a) and RP2 (b)

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
RP1	0.578 (0.930)	**<0.001** **(0.006)**	0.235 (0.954)	**<0.001** (0.060)	1.777 (6.537)	0.186 **(0.012)**	-1.350 (3.056)	0.180 **(0.003)**	-0.276 (0.624)
RP2	0.162 (0.943)	**<0.001** **(0.023)**	0.987 (0.931)	0.846 **(0.007)**	1.135 (2.526)	0.289 (0.115)	-1.120 (2.195)	0.266 **(0.031)**	-0.229 (0.448)

Table 4.27: Statistical Analysis for RP1 and RP2 with empirical data in brackets

For RP1, the simulation data shows no significant difference between the control and treatment groups ($p = 0.180$), with a small effect size ($d = -0.276$). This contrasts with the empirical data, which indicates a significant difference ($p = 0.003$) and a medium effect size ($d = 0.624$). The simulation seems to underestimate the influence of the treatment effect compared to the empirical data.

For RP2, the simulation data also shows no significant difference between the groups ($p = 0.266$), with a small effect size ($d = -0.229$). Similarly, the empirical data shows a significant difference ($p = 0.031$) with a medium effect size ($d = 0.448$). The simulation underestimates the treatment effect observed in the empirical data.

Overall, the findings suggest that the simulation underestimates the treatment effects for both RP1 and RP2, which can be observed in the effect sizes and statistical significance.

Error Data

The error rate analysis for RP1 and RP2 highlights differences between the treatment and control groups. For RP1, the treatment group achieved a correctness rate of 48.94%, which is lower than the control group's 59.18%. However, the difference was not statistically significant ($p = 0.422$) and showed a small effect size ($\phi = 0.082$). In contrast, the empirical data shows a much larger difference, with the treatment group achieving a correctness rate of 36.17% and the control group achieving 81.63%, which was statistically significant ($p < 0.001$) with a large effect size ($\phi = 0.442$).

For RP2, the treatment group achieved a correctness rate of 51.02%, which is slightly higher than the control group's 46.81%. The difference was not statistically significant ($p = 0.835$) and exhibited a negligible effect size ($\phi = 0.021$). Similarly, the empirical data shows no significant difference, with the treatment group achieving a correctness rate of 65.31% and the control group achieving 65.96% ($p = 1.0$, $\phi = 0.0$).

Overall, the findings suggest that the simulation underestimates the differences observed in the empirical data for RP1, where the control group significantly outperformed the treatment group. For RP2, the simulation aligns more closely with the empirical data, showing minimal differences between the treatment and control groups.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	24 (30)	23 (17)	48.94 (36.17)	0.644	0.422	0.082
Control	20 (9)	29 (40)	59.18 (81.63)	(18.714)	**(<0.001)**	(0.442)

Table 4.28: RP1 Error Rate

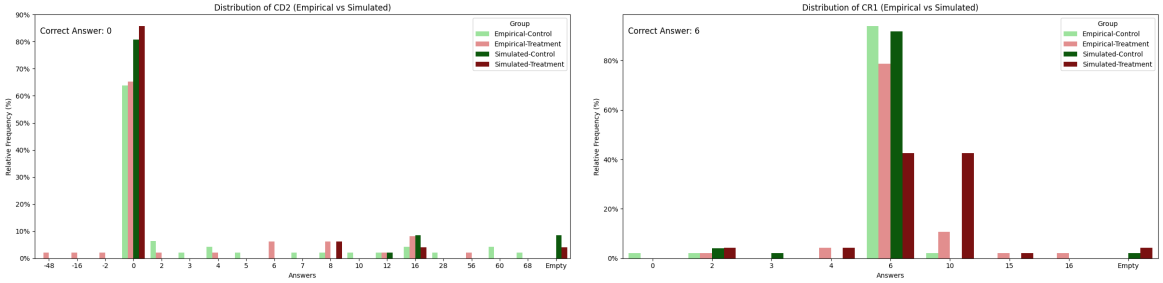
Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	24 (17)	25 (32)	51.02 (65.31)	0.043	0.835	0.021
Control	25 (16)	22 (31)	46.81 (65.96)	(0.0)	(1.0)	(0.0)

Table 4.29: RP2 Error Rate

4.1.4.7 *Similarity Errors*

In this section, we will analyze the errors that were made by the participants in the empirical evaluation and the simulation evaluation. The code snippets that were adjusted to facilitate similarity errors were CD2, CR1, DR1 and RP2.

For CD2, we can see that both groups in the empirical and simulation data made a few more pronounced errors, namely the value `8` instead of the value `0`. This result may be achieved by involuntarily exchanging the values of the variables `d` and `b` in the last line in the code snippet. Another possible error is to use the value `16`. This result can be achieved

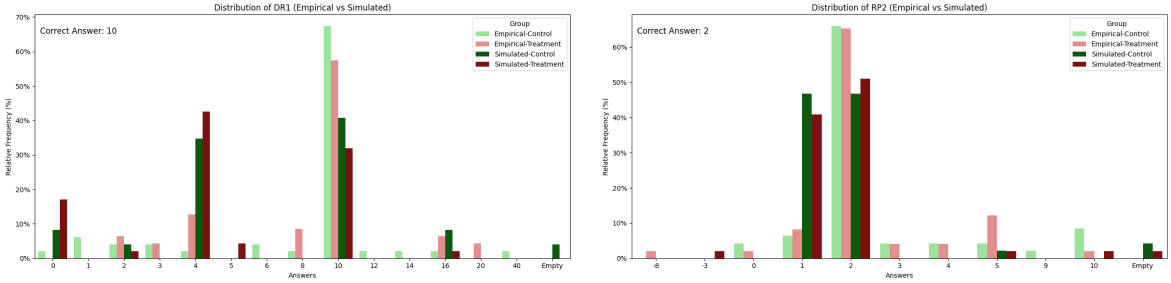


(a) (b)

Figure 4.20: Answer Distribution of Code Snippet CD2 (a) and CR1 (b)

by involuntarily exchanging the values of the variables `p` and `q` in the second last line in the code snippet.

For CR1, we can see that both groups in the empirical and simulation data made a rare similarity error, namely the value `2` instead of the value `6`. This may be due to a confusion of the values of the variables `b` and `d` in the second last line in the code snippet.



(a) (b)

Figure 4.21: Answer Distribution of Code Snippet DR1 (a) and RP2 (b)

For DR1, we can see that both groups in the empirical and simulation data made a rare error, namely the value `2` instead of the value `10`, which could be due to confusing the values of the variables `h` and `n` in the last line in the code snippet.

Finally, for RP2, we can see that both groups in the empirical and simulation data made a more pronounced error, namely the value `5` instead of the value `2`, which could be due to confusing the values of the variables `p` and `q` in the second last line of the code snippet. This error was more common in the empirical data than in the simulation data.

4.1.4.8 *Significance Level Correction*

We checked the significance level correction for the t-tests and χ^2 tests of all eight code snippets. Of those 16 values, five values were significant at the 0.05 level. The significance level correction can be seen in Table 4.30.

	CR1 t-test	CR2 t-test	CR1 χ^2	CR2 χ^2	CD1 t-test
p-value	<0.001	<0.001	<0.001	<0.001	0.012
Bound	0.003125	0.003333	0.003571	0.003846	0.004167
Significant	Yes	Yes	Yes	Yes	No

Table 4.30: Significance Level Correction of the Model Results

The value of the t-test for CD1 was not significant after the correction and may have been due to chance. All other values were still significant after the correction.

Comparison with Test Dataset

We ran the model with the same parameters for the same amount of iterations as there are participants in the test dataset. We then evaluated the model with the same statistical tests as we did for the empirical evaluation. The error value of the test dataset was -701.721, which is much better than the error value of the training dataset, which was -964.353. The time data and the error data had a better fit to the test dataset than to the training dataset, which is a good sign that the model is able to generalize to other datasets.

4.2 Discussion

In this section, we will discuss the results of the empirical evaluation and the simulation evaluation.

4.2.1 Empirical Evaluation

We evaluated the results of the effects of the treatments on the time data and the error data. We will not discuss the results of the socio-demographic data, as those are out of scope of this work.

Code Distance

For code snippet CD2, the participants in the treatment group were slower than the participants in the control group, which was expected. The code snippet change was to change the variable names to be similar to each other. The similarly named variables probably led to a higher cognitive load for the participants, as they had to connect the name of the variable with its value while reading the code snippet for a longer time. Thus they needed more time to process the code snippet. The control group had a lower distance between the variables that needed to be remembered, which may have lead to a lower cognitive load and therefore a faster processing time. While the effect size was medium, the difference was not statistically significant.

The error data did not show any signicant difference between the groups, therefore in this case, the treatment had apparently no effect on the error rate.

For code snippet CD1, the treatment group was unexpectedly faster than the control group. The code snippet was changed to have longer distances between declarations and usages of variable. If we follow the logic of the CD2 code snippet, we would expect the treatment group to be slower than the control group, especially as the distance between the declaration and usage of the variable was greater. We are not sure why the treatment group was faster, as the treatment group needed to remember two variables from the start of the code snippet instead of one. A possible explanantion could be that the treatment group finished the code snippet faster, but that was due to not remembering the variable and thus directly finishing the code snippet, which led to a lower correctness rate. There was no strong association between group type and correctness, as the effect size was small and the difference was not statistically significant, but it could still be a hint to support the previous explanation.

Repeated Code

For code snippet CR1, the treatment group was only slightly faster than the control group. The analysis also revealed that there was no significant difference between the groups in terms of time and correctness. In the original work of Closheim, the author also did not find a significant difference in the acquired data. Therefore, it seems as if the treatment group was not affected by the similarity of the variable names.

Contrarily, for code snippet CR2, the treatment group was much slower than the control group. We found a significant difference between the groups, with a large effect size, for the time data. The code snippet had a longer distance between the declaration and usage of the variable. This leads us to the possibility that having a double declaration of a variable in the code snippet led to a higher cognitive load for the treatment group, as they had to remember the correct value of a variable for a longer time. It may also have lead to uncertainty about the correct value and thus to a longer processing time. The error data did not show any significant difference between the groups, therefore this treatment did not have a strong effect on the correctness of the participants.

Declaration Redeclaration Distance

For code snippet DR1, the treatment group was also significantly faster than the control group. The difference is significant with a nearly a large size effect. The code snippet was changed to have variable names that are similar to each other. Such a result was also unexpected, as the distance between the last declaraction of the variable and its usage was higher for the treatment group. Nevertheless, some other factors may have been at play here.

As we can see in the control code snippet 4.1, the declaration of the variable *h* is one line farther away than in the treatment code snippet 4.1 (a). These order effects may interact with each other to make the treatment code snippet easier than the control code snippet as the value of the variable *h* may be more difficult to remember than the value of the variable

```
l = 2
l = 5
i = 8
h = i / 4
n = h * l
print(n)
```

(a)

```
l = 2
i = 8
h = i / 4
l = 5
n = h * l
print(n)
```

(b)

Listing 4.1: DR1 Treatment (a) and Control (b) Snippets

l and thus also lead to a longer processing time. The error in comparison did not show any significant difference between the groups.

For code snippet DR2, the treatment group was also significantly slower than the control group. The difference is significant with a nearly large size effect. Due to having a larger distance between the double declaration and the subsequent usage, the effect of the treatment appears to be stronger than in the similarity variant of the code snippet. The longer distance between the declaration and the retrieval of the value of the variable may have led to a higher cognitive load for the treatment group, as they had to remember the value of the variable for a longer time. While the error data did not show any significant difference between the groups, the treatment group performed nearly 20% better than the control group. As the result is not significant, it should be taken with caution, but it may indicate that the treatment group was able to remember the value of the variable better than the control group due to taking more time to process the code snippet.

Repeated Distance

For code snippet RP1, we again had the case that the treatment group was significantly faster than the control group. Nevertheless, these results were not expected, as the treatment code snippet was designed to be easier to solve than the control code snippet. In the control code snippet, the participant only needed to remember the last 2 lines before the print statement, whereas in the treatment code snippet, the participant had to remember up to seven lines before the print statement. The standard deviation of the time data is rather high with 14.7 seconds for the treatment group and 24 seconds for the control group, which may cause these unexpected results. Additionally, there were more participants in the treatment group that took less than 5 seconds for the code snippet, which may have led to a bias in the results. Moreover, participants in the treatment group may have been giving up earlier than the participants in the control group, as they may have been sure that they do not remember the variable correctly.

Regarding the error data, the control group performed significantly better than the treatment group with a nearly large effect size. This supports the previous argument that the treatment group may have given up earlier than the control group, as they were not able to remember the variable correctly. We can conclude that the RP treatment correlates with the

error rate for this code snippet.

For code snippet RP2, the treatment group was slower than the control group. This code snippet was changed to have similar variable names. The t-test also showed a significant difference between the groups with a nearly large effect size. As RP1 had a much faster processing time for the treatment group, it may be that the similarity between the variable names contributed to the slower processing time of the treatment group in this code snippet. While this may be a satisfactory explanation, we cannot be sure that this is the only reason for the slower processing time, as other factors as specific characteristics of the participants or code snippets may have played a role in the results.

The error data did show nearly no difference between the groups with nearly identical correctness rates. It seems as if the participants were able to remember the variable correctly, even if the variable names were similar to each other, but needed more time to retrieve the correct value of the variable.

Overall Evaluation

Overall, we can extract the following conclusions from the empirical evaluation:

Time Effects

We had several unexpected results in regard to the time data, namely for the code snippets CD1, CR1, DR1 and RP1. For all four of them, the treatment group was faster than the control group, which was not expected. While this may all be explained by some facts that are specific to the code snippet or the participants, we cannot be sure what the exact reason for these results are. From a logical standpoint, we would expect the treatment group to be slower than the control group, as they had to deal with a higher cognitive load due to the treatment. A treatment may prove as ineffective if the code snippet is altered in a specific way, but it should not reduce the time needed to process the code snippet. As the affected code snippet were comprised of two code snippets that had their length increased and two code snippets that had their variables renamed, the probability of these changes leading to a faster processing time is rather low, considering that for the other code snippets, the treatment group was slower than the control group. It would be necessary to conduct further studies to investigate these results and to find out if they are reproducible. Other possible explanations for these results could be that the treatment group was not able to remember the variable correctly and thus directly finished the code snippet, which led to a lower correctness rate, but a faster processing time. Additionally, order effects could have played a role in the results.

For the other code snippets with the expected behavior, we can see that the distance between the declaration and usage of the variable may have a signicant effect on the processing time of the code snippet. We had a signicant and strong effect for the code snippets CR2 and DR2, where the treatment group was slower than the control group. Additionally, the similarity of the variable names may also have an effect on the processing time, as we saw in the code snippets CD2 and RP2, where the treatment group was slower than the control group. We can also see that double declarations of variables may increase the effectiveness of similarity errors.

Error Effects

We observed several interesting results in regard to the error data. First of all, most code snippets had no significant difference between the groups. Additionally, the effect sizes were rather small, which indicates that the treatments did not have a strong effect on the correctness of the participants. Nevertheless, the code snippet RP1 had a significant difference between the groups, with a nearly large effect size, and DR2, while not significant, still had a rather large difference in correctness between the groups. This may be due having to having double declarations of variables in combination with a higher distance between the declaration and usage of the variable. As the effect is less pronounced in DR2 than RP1, we can conclude that the distance between the declaration and usage of the variable may have a stronger effect on the correctness than the double declaration of the variable. Additionally, greater distances are more likely to lead to errors than smaller distances, as we saw in the code snippets RP2 and DR1. Finally, there may be some order effects that lead to a higher error rate, as for example the control code snippet DR2 had a higher error rate than the treatment code snippet, even though the control code snippet should have been easier to solve.

4.2.2 Simulation Evaluation

In this section, we will evaluate the results of the model fitting and the model evaluation of the simulation evaluation. First, we will discuss the model fitting process. Afterwards, we will evaluate the results of the simulation in comparison to the empirical data.

4.2.2.1 *Model Fitting*

We trained the model in the same way as Closheim did in their study. Unfortunately, the model fitting process was not as successful as in the original study. The time data of our empirical evaluation was much more spread out than the time data of the original study, which led to problems with fitting the time data of the simulation evaluation. Additionally, we were also unable to fit the error data of the simulation evaluation as well as in the original study.

Due to time constraints, we were unable to wait for the optimization process to converge, which may have led to a less optimal model than in the original study. While it may be possible that the model would have converged to a better solution if we had waited longer, we do not think that further optimizations would have led to a much better model, as the model in its current state is probably not able to fit our collected data properly.

Optimizating the model in regards to the time data always led to the model trying to create a retrieval error, which leads to a rather large wide distribution of the time data, which was needed to fit the time data, which in turn led to no responses for the error data. Such a behavior needs to be avoided, as the model should not create a retrieval error for the time data, but rather try to fit the time data with a normal distribution. This behavior needs to be fixed in the model itself, by providing the model with a better ability to create wide

distributions for the time data without creating retrieval errors.

The usage of the significance level to optimize the model was not as successful as in the original study, as it provided no better results than the error data optimization. This is due to the fact that the model was not able to fit the time data properly. If we had a model that was able to fit the time data properly, we probably would have been able to use this optimization method to find the best model parameters.

As a final note, while the test set provided an interesting insight into the model's performance, it is important to acknowledge that depending on the seed that was used the test set may not be representative of the model's performance. A better approach would have been to use a different method, such as cross-validation [6], to ensure that the model is able to generalize well to unseen data. This would have required more time and resources than were available for this thesis, but it would have provided a more robust evaluation of the model's performance.

4.2.2.2 *Model Evaluation*

We evaluated the results of the effects of the treatments on the time data and the error data in comparison to the empirical data.

Code Distance

We need to keep in mind that the results of CD1 were unexpected, as the treatment group was faster than the control group. Therefore we should be careful with the interpretation of the comparison of the simulation and empirical data.

The results of the code distance treatment provide insights into the differences between the simulation and empirical data. For CD1, the simulation time data suggest no significant treatment effect with a negligible effect size, in contrast to the empirical time data, which show a significant difference with a medium effect size. The simulation time data is concentrated around a few values, unlike the empirical data, which is more distributed. These findings suggest that the simulation underestimates the variability in the time needed to process code snippets. Short pauses to process some variables are not implemented in the model, which may lead to a more concentrated distribution of the time data.

For CD2, the simulation and empirical time data analysis align more closely, showing no significant differences and small effect sizes. This consistency suggests that the simulation accurately reflects the minimal impact of similar variable names on task performance. However, the simulation data also exhibits a more concentrated distribution compared to the empirical data, probably due to the lack of some cognitive processes in the model.

The error rate analysis reveals further limitations of the model. For CD1, the simulation data shows near-perfect correctness for both the treatment and control groups, significantly overestimating the empirical results. Similarly, for CD2, the simulation data indicates higher correctness rates for both groups compared to the empirical data, particularly for the

treatment group. While the χ^2 tests for both CD1 and CD2 shows no significant differences in the simulation data, consistent with the empirical results, the consistently higher correctness rates in the simulation suggest that the model underestimates the cognitive challenges posed by the treatment, increased code distance and variable similarity.

If we take a look at the error distribution of CD2, we can see that the model was able to replicate some errors of the empirical data, especially those that may be due to similarity errors. This indicates that the model is able to replicate some cognitive processes that lead to smilarity errors.

Overall, while the model aligns well with the empirical data for CD2 in terms of time and error rate, it overestimates the impact of the treatment in CD1 in terms of time and underestimates the cognitive challenges in terms of error rates. Additionally, the model is not able to vary the time data properly, which leads to a more concentrated distribution of the time data than in the empirical data.

Repeated Code

For CR1, the simulation time data suggests a significant treatment effect with a very large effect size, in contrast to the empirical time data, which shows no significant difference and a small effect size. The simulation time data is again concentrated around a few values, indicating that the model is not able to replicate the variability observed in the empirical data. These findings suggest that the simulation underestimates the variability in the time needed to process code snippets.

For CR2, the simulation data also shows a significant treatment effect with a large effect size, which aligns with the empirical data. As CR1 had an overestimation of the treatment effect, whereas CR2 had a similar effect, we can conclude that the treatment effect is enhanced by the distance between the declaration and usage of the variable, as CR1 had a smaller distance than CR2 and instead used similar variable names. This leads us to believe that the model may overestimate the treatment effect for smaller distances and/or may also overestimate the effect of similarity of the variable names.

The error rate analysis reveals additional limitations of the model. For CR1, the simulation data shows a correctness rate of 42.55% for the treatment group, which is significantly lower than the empirical rate of 78.72%. The control group in the simulation achieved a correctness rate of 91.84%, slightly lower than the empirical rate of 93.88%. The significant difference observed in the simulation data, with a large effect size, contrasts with the empirical data, which showed no significant difference and a small effect size. This suggests that the model overestimates the impact of repeated declarations on correctness rates.

If we take a look at the error distribution of CR1, we can see that the model was able to replicate one error that may be due to similarity of the empirical data errors. This indicates that the model may be able to replicate some cognitive processes that lead to smilarity errors.

For CR2, the simulation data also shows a significant difference between the treatment and control groups, with a large effect size. However, the treatment group in the simulation performed significantly worse than the empirical data, achieving a correctness rate of 51.02%

compared to 91.84%. The control group in the simulation slightly overperformed compared to the empirical data. These findings suggest that the simulation overestimates the treatment effect on correctness rates, particularly for the treatment group.

Overall, while the simulation captures some general trends, it overestimates the treatment effects for both CR1 and CR2 in terms of time and error rates. The differences in time data suggest that the model may also overestimate the effect of similarity of variable names. The model also fails to replicate the variability observed in the empirical time data, likely due to missing cognitive processes, such as pauses to resolve ambiguities or variability in processing repeated declarations.

Declaration Redeclaration Distance

We need to keep in mind that the results of DR1 were unexpected, as the treatment group was faster than the control group. Therefore we should be careful with the interpretation of the comparison of the simulation and empirical data.

For DR1, the simulation time data suggests no significant treatment effect with a small effect size, in contrast to the empirical time data, which shows a significant difference with a medium effect size. This result suggests that the simulation overestimates the treatment effect, compared to the empirical data. The simulation time data is concentrated around 75 seconds, whereas the empirical data is more distributed, with time values ranging from 5 to 75 seconds. The lack of variability in the simulation data may be due to missing cognitive processes, such as pauses to resolve ambiguities caused by the double declaration.

For DR2, the simulation data also shows no significant treatment effect with a negligible effect size, whereas the empirical data shows a significant difference with a medium effect size. This discrepancy indicates that the simulation underestimates the treatment effect for DR2 as well. The concentrated distribution of the simulation time data further highlights the limitations of the model in replicating the variability observed in human behavior.

As DR2 had an underestimation of the code snippet effect on the time needed to process it, we can conclude that the model may underestimate the treatment effect itself, as one code snippet had similar variable names and the other code snippet had a larger distance between the declaration and usage of the variable.

The error rate analysis reveals additional limitations of the model. For DR1, the simulation data shows a correctness rate of 31.91% for the treatment group, which is significantly lower than the empirical rate of 57.45%. Similarly, the control group in the simulation achieved a correctness rate of 40.82%, also lower than the empirical rate of 67.35%. The lack of a significant difference in both the simulation and empirical data suggests that the model underestimates the correctness rates but aligns with the statistical significance observed in the empirical data.

If we take a look at the error distribution of DR2, we can see that the model was able to replicate some errors of the empirical data, especially those that may be due to similarity errors. This indicates that the model is able to replicate some cognitive processes that lead

to smilarity errors.

For DR2, the simulation data shows a correctness rate of 44.90% for the treatment group, which is lower than the empirical rate of 65.31%. The control group in the simulation achieved a correctness rate of 44.68%, which is similar to the empirical rate of 46.81%. While the simulation aligns with the empirical results in terms of no significant difference, it underestimates the correctness rates, particularly for the treatment group. This suggests that the model fails to capture the cognitive challenges posed by the combination of increased code distance and repeated declarations.

Overall, the simulation underestimates the treatment effects for both DR1 and DR2 in terms of time and error rates. The differences in time data suggest that the model may fail to account for the interaction between repeated declarations and subsequent usage of the variable. Additionally, the model's inability to replicate the variability observed in the empirical data highlights the need for further refinement, such as incorporating pauses to resolve ambiguities or variability in processing double declarations.

Repeated Distance

We need to keep in mind that the results of RP1 were unexpected, as the treatment group was faster than the control group. Therefore we should be careful with the interpretation of the comparison of the simulation and empirical data.

For RP1, the simulation time data suggests no significant treatment effect with a small effect size, in contrast to the empirical time data, which shows a significant difference with a medium effect size. The simulation time data is concentrated around 52 seconds, whereas the empirical data is more distributed, with time values ranging from 3 to 50 seconds. These findings suggest that the simulation underestimates the treatment effect and fails to replicate the variability observed in the empirical data. The lack of variability in the simulation data may be due to missing cognitive processes, such as pauses to resolve ambiguities caused by the code snippet.

The error rate analysis reveals additional limitations of the model. For RP1, the simulation data shows a correctness rate of 48.94% for the treatment group, which is higher than the empirical rate of 36.17%. Similarly, the control group in the simulation achieved a correctness rate of 59.18%, which is significantly lower than the empirical rate of 81.63%. The lack of a significant difference in the simulation data contrasts with the empirical data, which showed a significant difference with a large effect size. This suggests that the model underestimates the impact of repeated distance in combination with a larger distance between the last declaration of the variable and its usage on correctness rates, particularly for the control group. The model does not capture the ease of the task for the control group, which may be due to some confusion about the value of the variable due to the double declaration of the variable at the beginning of the code snippet. Therefore, the model may overestimate the effect of a double declaration of a variable on the correctness rate.

For RP2, the simulation data shows no significant difference between the treatment and control groups, with correctness rates of 51.02% and 46.81%, respectively. This aligns with the empirical data, which also shows no significant difference, with correctness rates of 65.31% and 65.96%. However, the simulation underestimates the correctness rates for both groups compared to the empirical data, particularly for the treatment group. These findings suggest that the simulation fails to capture the cognitive challenges posed by the repeated distance treatment, similarly to RP1.

If we take a look at the error distribution of RP2, we can see that the model was able to replicate some errors of the empirical data, especially those that may be due to similarity errors. This indicates that the model is able to replicate some cognitive processes that lead to smilarity errors.

Overall, the simulation underestimates the treatment effects for RP2 and overestimates them for RP1 in terms of time and error rates. The differences in time data suggest that the model may fail to account for the interaction between repeated distance and cognitive load. Additionally, the model's inability to replicate the variability observed in the empirical data highlights the need for further refinement, such as incorporating pauses to resolve ambiguities or variability in processing repeated distances. Finally, the error data also suggests that the model may not accurately capture the cognitive challenges posed by the treatment, as it consistently underestimates the correctness rates for both RP1 and RP2, especially for the treatment group in RP1.

Overall Evaluation

The overall evaluation of the simulation model reveals both strengths and weaknesses in its ability to replicate human behavior in programming tasks. We will summarize the findings of the simulation evaluation in the following sections.

Time Effects

The model aligns well with the empirical data for some scenarios, such as CD2 and CR2, accurately capturing the minimal or significant effects of the treatments. However, for other scenarios, such as CD1, CR1, DR1, and RP1, the model either overestimates or underestimates the treatment effects. For CD1 and RP1, the model underestimates the treatment effects, while for CR1, it overestimates the impact of repeated declarations. Additionally, the simulation consistently fails to replicate the variability observed in the empirical time data, with concentrated distributions that suggest missing cognitive processes, such as pauses to resolve ambiguities or variability in processing repeated declarations and distances. Finally, the model also consequently overestimates the time needed to process the code snippets, which is due to the model not being able to replicate the variability in the time data properly.

Error Effects

The model generally underestimates the cognitive challenges posed by the treatments, leading to overestimated correctness rates for scenarios such as CD1, CR1, and RP2. For CD1,

the model predicts near-perfect correctness rates, significantly overestimating the empirical results. For CR1 and RP1, the model underestimates the differences between the treatment and control groups, failing to capture the significant effects observed in the empirical data. While the model aligns more closely with the empirical data for CD2, CR2, and DR2, it still underestimates the correctness rates for treatment groups and fails to replicate the distribution of errors, particularly for scenarios involving increased code distance or repeated declarations.

If we look at the comparison of similarity errors, we can see that the model was able to replicate some expected similarity errors of the empirical data. This indicates that the model is able to replicate some cognitive processes that lead to similarity errors, but it is not able to replicate all of them.

Model Improvements

We propose the following improvements to the cognitive model based on the evaluation of the simulation results to increase the accuracy of the model and to better align it with the empirical data. We will focus on addressing the variability and cognitive challenges observed in the empirical data.

First, the model should incorporate mechanisms to better replicate the variability in time data, such as introducing pauses to resolve ambiguities or variability in processing repeated declarations and distances. This could involve adding probabilistic elements to simulate the range of human behavior observed in the empirical data.

Second, the model should refine its handling of error rates by better capturing the cognitive challenges posed by treatments such as increased code distance, repeated declarations, and variable similarity. This could include implementing additional cognitive processes, such as memory decay or confusion caused by double declarations, to better reflect the observed error patterns.

Finally, a longer training process and parameter optimization would be beneficial to improve the model's fit to the empirical data. As the current optimization process was limited, extending it could help the model better capture the complexities of human behavior in programming tasks.

In conclusion, while the cognitive model provides a valuable framework for understanding cognitive processes in programming tasks, it requires further refinement to accurately replicate the variability and cognitive challenges observed in the empirical data. These improvements would enhance the model's ability to simulate human behavior and provide more accurate predictions for programming tasks.

4.2.3 Comparison of Closheim's Study and Part I

In this section, we will compare the results of the empirical evaluation and the simulation evaluation with the results of the study of Closheim.

Empirical Evaluation

The time needed to process code snippets was significantly higher and spread out for our study than for the study of Closheim, leading also to problems with fitting the time data of the simulation evaluation. Such a difference was not expected, as the code snippets were designed to be similar to the code snippets of Closheim. This may be due to the fact that they did not use a crowdsourcing platform as we did, which may have led to a different sample of participants that were less distracted and more motivated to solve the tasks.

While the empirical evaluation of Closheim did not find any significant difference between the groups, we found several significant differences between the groups in our study for both time and error data, which supported the findings of Closheim. Additionally, we also found significant differences between the groups when using code snippets with similar variable names, which was a factor that was proposed by Closheim as a possible direction for future work.

Nevertheless, we also found several unexpected results in our study, which did not match the results of Closheim, namely for the code snippets CD1, CR1, DR1 and RP1. For these code snippets, the treatment group was faster than the control group, which was not expected.

Overall, we can conclude that most of the results of our study are in line with the findings of Closheim, but there is the need of further studies to investigate the unexpected results and to find out if they are reproducible.

Simulation Evaluation

Closheim was able to fit the time data of their simulation evaluation much better than we were able to do in our study. This is in part due to the fact that the time data of their empirical evaluation was much less spread out than the time data of our empirical evaluation, which led to problems with fitting the time data of the simulation evaluation, as the model in its current state is not able to replicate the variability in the time data properly.

We also had more problems to fit the error data of our simulation evaluation than Closheim had in their study. The problems, namely the overestimation or underestimation of the treatment effects, were similar to the problems that of Closheim, but more pronounced, especially for the code snippet CD1.

The increased distance between declarations and re-declarations had a negative impact on the accuracy of the ACT-R model compared to Closheim's original study. The inclusion of similarity-based errors between variable names improved the ability of the ACT-R model to reproduce human-like error patterns, but the model still fails to capture the full range of such errors and their impact on processing time and needs further refinements.

Overall, we can conclude that the simulation evaluation of our study is not able to reproduce the favourable results of Closheim's original study and that the model needs

further refinements to be able to replicate the variability in the time data and to better capture the cognitive challenges posed by the treatments.

4.2.4 Answering the Research Questions

The aim of this part of the thesis was to evaluate the cognitive model that was created in Part I and to evaluate the accuracy in terms of processing time and the error rate. First, we wanted to answer the following research question:

> **RQ1**: How do increased distances between declarations and re-declarations/usages influence the accuracy of the ACT-R model in terms of processing time and error rate?

The evaluation shows that the ACT-R model partially captures the effects of increased distance between declarations and re-declarations, but has limitations in both time and error predictions. In terms of processing time, the model often fails to reflect empirical variability and tends to produce more concentrated distributions, probably due to missing cognitive mechanisms such as pausing or ambiguity resolution. For example, while it correctly predicts minimal distance effects in some scenarios, it either overestimates or underestimates treatment effects in others. Some treatment effects are more pronounced in the empirical data than in the simulation, suggesting that the model may not fully account for the cognitive load associated with increased distances.

In terms of error rates, the model generally underestimates the cognitive difficulty of increased distances. It predicts overly optimistic correct rates and fails to reflect the significant group differences observed in the empirical data. This suggests that while the model structurally represents increased distance, it does not fully simulate the cognitive load it imposes. Overall, increased distance has a negative impact on the accuracy of the ACT-R model, primarily due to oversimplified time variability and limited error modelling.

Our second research question was:

> **RQ2**: Can we increase the accuracy of the ACT-R model in terms of processing time and error rate by implementing similarity errors between variable names?

The inclusion of similarity-based errors between variable names appears to improve the ability of the ACT-R model to reproduce human-like error patterns. The evaluation shows that in several snippets the model was able to reproduce specific similarity errors that also occurred in the empirical data. This suggests that the implementation of a similarity-based retrieval mechanism allows the model to simulate confusion due to similar variable names, more accurately reflecting real-world cognitive challenges.

However, the model still fails to capture the full range of such errors and their impact on processing time. While it benefits from the inclusion of similarity in retrieval, the effect sizes and variability in both time and error data remain under- or overestimated in several scenarios. Therefore, the implementation of similarity errors is a promising step towards

improving model accuracy, but further refinements, such as probabilistic retrieval, better modelling of activation noise and memory confusion dynamics, are needed to fully capture the cognitive processes involved in programming tasks.

4.3 Threats to Validity

In the following sections, we will discuss the threats to internal and external validity of the study.

4.3.1 External Validity

The use of a crowdsourcing platform such as Amazon Mechanical Turk may threaten the external validity of the study due to, for example, self-misrepresentation, self-selection bias and high attrition rates of participants [1]. To reduce these threats, we used CloudResearch, as it provides better data quality from participants [20].

Due to the nature of this study, we were only able to generalize the results to a certain extent. Especially the code snippets were be a simplification of real-world code due to their line-by-line reading order, having only letters as variables, using only simple arithmetic operations and no other keywords or function calls except for "print()". Nevertheless, the aim of our study is not to provide a generalization to all code comprehension tasks, but to provide a better understanding of the cognitive processes of programmers when reading code snippets.

The external validity could be threatened by an unrepresentative sample. We tried to reduce this threat by using CloudResearch, as it provides participants that are more representative of the general population than other crowdsourcing platform such as Amazon Mechanical Turk. Nevertheless, we encountered unexpected results in our study, which may be due to the sample of participants that we used.

4.3.2 Internal Validity

Crowdsourcing platforms may lead to a biased sample due to, for example, inattention, high attrition rates, inconsistent english language skills and non-naivete of participants [1]. We tried to reduce these threats by using CloudResearch instead of Amazon Mechanical Turk, as it provides better quality participants [20].

Feitelson [14] provided a list of threats to the validity for code comprehension studies. Feitelson described the pitfalls "Misleading Code", where participants may make errors due to being unintentionally misled, and "Recognized Code", where code is easily recognized. We tried to avoid these pitfalls by paying attention to these details when creating the code

snippets.

Another threat to the validity of this study was that cognitive architectures are not perfect models of human behavior. They may create models that are able to replicate specific behaviors while not aligning with the underlying cognitive processes. These models function as a simplification of the human brain and are used to understand cognitive processes better. Specifically, issues such as learning effects, are missing from such models. Therefore, we cannot expect the model to be simulate the exact inner workings of the brain.

Due to the scope of this thesis and in accordance to the original study, we only varied four parameters of the model. We may encounter better results by varying more parameters, but this would lead to a more complex model and being unable to compare the studies.

As this part of the thesis was based on the thesis of [10], we had to rely on the correctness of the theoretical foundation of the model from the original study. If the model of the original study was not correct, we may have produce a model and results that are also not correct. Nonetheless, our aim was to replicate this thesis as closely as possible to provide a more robust model for future studies. Therefore, even a model that is not correct may have provide valuable insights.

5 Part II Methodology: Extension Study

This chapter describes the methodology of our extension study that was conducted. For this part, the ACT-R model was extended to include a backtracking mechanism. It consists of two parts: a simulation evaluation and an empirical evaluation. First, we will present the research question that will be answered in this study. This will provide us an overview of the aims of this part of the thesis. We then describe the study materials and the study design of the simulation evaluation and the empirical evaluation. Finally, the data analysis and evaluation of the two parts will be presented.

5.1 Research Questions

While Closheim's ACT-R model is able to simulate the behavior of developers during code comprehension tasks, the simplifications made in the model lead to a lower external validity, as experimental conditions are not comparable to real-world conditions. Thus, there is a need to extend the model to be more in line with real-world conditions. Implementing a backtracking mechanism would allow the participants to return to a previous code line, which is a common strategy used by developers. This leads us to the following research question:

> **RQ3**: How does the possibility to return to a previous code line influences the accuracy and predictions of the ACT-R model?

5.2 Study Design

In this section we describe the design of the study. We conducted two types of study: a simulation evaluation and an empirical evaluation. The empirical evaluation consisted of two parts. One was a questionnaire combined with a think-aloud protocol and short interview at the end. For the second one, we performed the empirical evaluation of Part I by having some participants do the same tasks as in Part I, but with the ability to go back to previous code lines.

The simulation evaluation consisted of extending an existing cognitive model with the backtracking mechanism and training it with the results of the empirical evaluation. The implementation of the backtracking mechanism was built upon the results of the think-aloud protocol and the semi-structured interview. We used the results of the crowdsourcing study to train the parameters of the cognitive model. This allowed us to use the cognitive model

to predict programmer behaviour on a larger scale. The prediction accuracy of the cognitive model was evaluated to understand how well the model is able to predict the behavior of the participants.

5.2.1 Study Materials

We used the same materials as we used in Part I of this thesis. This enabled us to compare the results of both parts of this thesis. The materials consisted of the following:

- Training code snippets
- Experimental code snippets (control and treatment)
- Intermediate tasks
- Socio-demographic questionnaire

With the *Left Arrow* key, the participants were able to return to a previous code line. The *Left Arrow* key was chosen as the *Backspace* key is already used to delete characters during the last screen where the result should be entered.

We also create a questionnaire with the help of SoSci Survey[1], as it allows the creation and execution of questionnaires for academic purposes for free. We used LabJS[2] to show the users the code snippets, as it allowed us to record the times and key presses of the participants.

5.2.2 Simulation Evaluation

In this section, we will describe the cognitive model that was used in the study.

We created a new version of the ACT-R model, which included a backtracking mechanism. We extended our model from PartI to include the possibility to return to a previous code line. This enabled us to compare the results of the model with the results of Part I.

As ACT-R is able to model human input and output, we used it to model the behavior of participants in the study. By utilizing the vision interface of ACT-R in the same way as in the existing model, we were able to model the visual input of the participants, namely the code snippets they were presented with. This allowed us to copy the environmental conditions as in the empirical evaluation, enabling us to use the results of the empirical evaluation to train the ACT-R model.

The model was able to perform the following tasks:

- Read one line at a time from left to right
- Store variable names and values in declarative memory
- Retrieve variable names and values from declarative memory

1 https://www.soscisurvey.de/
2 https://lab.js.org/

- Perform arithmetic operations, namely addition, subtraction, multiplication and division, on integer numbers from -20 to 20
- Read comments
- Read and verify assert statements
- Return to a previous code line by pressing the *Left Arrow* key and advancing to the next line by pressing the *Right Arrow* key
- Answer a print statement with the expected output of the model by entering the guessed number and pressing *Space*.

As already explained in Part I, the model aims to mimic human behavior as closely as possible by using the same input and output as the participants in the empirical evaluation and reads the unparsed line of code and processes it in the same way as the participants. When the model will be finished with a line, it will also press the *Right Arrow* key to continue to the next line. After the print statement, the model will enter the guessed number and press *Space* to submit the answer.

5.2.2.1 *Backtracking Mechanism*

Compared to Part I, the model is now able to return to a previous code line. To return to a previous code line, the model presses the *Left Arrow* key. To simulate the decision-making of the participants, the model uses the retrieval threshold parameter of the ACT-R model.

Everytime the value of a variable is needed, the model retrieves the value of the variable from declarative memory. The retrieval process is based on the activation value of the chunks in declarative memory. If the calculated activation value of the chunk is higher than the retrieval threshold, the model will retrieve the value of the variable, otherwise the model will output a retrieval error. If the retrieval fails, the model goes back to previous code lines till it finds the last value of the variable.

The model will simulate going back to a previous code line by showing the previous code line to itself. The model will then go back to the previous code line and scan the line to see if the searched variable is declared. If it is not declared, the model will do this step for every previous code line until it finds the declaration of the variable. Once the value of the variable has been found, the model will continue where it left off by skipping the lines it has already read and the part of the current line it has already processed.

We also implemented nested backtrackings, which means that the model can go back to a forgotten variable, and if in this calculation another variable value is forgotten, the model returns to previous code lines to search for the newly forgotten variable. After finding the value of the variable, the model will resume with the line where the previous variable was forgotten.

Additionally, the model uses assert statements as a way to check the value of the variable when going back. When going back to look for the value of a variable, if an assert statement is found, the model will look at the variable in it. If the variable is the variable that was searched, the model will evaluate the value of the other equality in the assert statement. This behavior was implemented as we observed this behavior during the manual analysis of the data received from CloudResearch and during the think-aloud study.

We implemented the following stopping conditions:

- The model reached the last line and was able to run it
- The model returned more than three times to the same variable at once, i. e.,it was not able to use the found value of the variable three times in a row
- The model looped through the code lines more than 500 times

By implementing these stopping criteria, we were able to prevent the model from going into an infinite loop and prevent simulation of unrealistic behavior.

The implementation of this backtracking mechanism was mostly done in the form of ACT-R productions. By trying to implement most functionality in ACT-R, we tried to keep the model as cognitively faithful as possible and avoid introducing unintended functionality that could contradict established psychological principles.

The processing of which code line and the order of which variable the model is currently looking for when going back was implemented in Python due to the complexity of this feature and the need to use a more complex data structure than the ACT-R production system allows. Furthermore, a Python implementation promotes better maintainability and readability of the code.

5.2.3 Empirical Evaluation

This section describes the empirical evaluation that was conducted.

The think-aloud study was conducted as a pilot study to understand the participants' thoughts in more detail. Understanding when and why participants choose to go back to a previous line of code would allow us to build a more accurate model.

The participants were introduced to the process of the study and presented with the training code snippets. Subsequently, the participants were shown the experimental code snippets. Lastly, they completed the socio-demographic questionnaire.

We used the think-aloud protocol to capture the thoughts of the participants. The think-aloud protocol is a method where participants verbalize their thoughts during a task and is used to turn implicit mental models into understandable insights [37]. Each session was audio recorded for later transcription and analysis.

During the last two code snippets, participants were asked questions during the code snippet to gather more information on why participants decide to backtrack to a previous line of code.

Additionally, we conducted a short semi-structured interview at the end of the study to gather more information about the participants' thoughts. This enabled us to understand the participants' thoughts in more detail and to build a more accurate model. The interview was semi-structured, meaning that we had a set of questions to ask the participants, but we were also open to follow-up questions and discussions. Some questions were specifically tailored to the participants' behavior during the study. This helped address a limitation of the think-aloud protocol, as participants may not be able to verbalize all of their thoughts [9].

We prepared an application of the study to be reviewed by the ethical review board of the Faculty of Mathematics and Computer Science at the Saarland University. The ethical review board reviewed and accepted the study without the need of additional modifications.

We gathered additional data points by having some of the participants of the empirical evaluation of Part I do the same tasks as in Part I, but with the possibility to go back to a previous code line. This allowed us to perform a statistical analysis on the results of Part II, as only three data points would not have been enough. The study procedure was the same as for Part I.

5.2.3.1 *Recruitment*

We recruited three participants for the think-aloud study. As we only needed a low number of participants, we used convenience sampling. While convenience sampling is not representative of the population [16], it is a good method to use for a pilot study, as it requires little effort and enough data can be gathered to make a preliminary analysis.

The recruitment of the participants for the additional data points was done in the same way as in Part I, with the help of the *Connect* platform on CloudResearch. We were able to recruit 123 participants for this part of the study. The participants were rewarded with $5.00 for their participation in the study, which took around 10–20 min to complete.

5.2.3.2 *Exclusion Criteria*

For our pilot study, we aimed to exclude any participants that did not complete the questionnaire entirely. There were no exclusions necessary for the pilot study.

The exclusion criteria for participants from CloudResearch was the same as in Part I. After applying the exclusion criteria for participants of the crowdsourcing platform, we were left with 96 participants.

5.3 Data Mining

In this section, we will present how we analyzed the second part of the thesis. First, we analyzed the results of the empirical studies. Then, these results were used to extend and train the parameters of the cognitive model, which was then evaluated.

5.3.1 Empirical Data Mining

We used the same data mining procedure as in Part I with some additional data. This allowed us to ensure comparability between the results of Part I and Part II.

The data mining consisted of gathering the following:

- Type of code snippet was used (treatment or control)

- Time taken by the participant for each line
- Total time for the task
- Which code snippet was used
- Given answer
- Correctness of the answer as a true or false value
- Answers to the socio-demographic questionnaire

Additionally to the previously mentioned data points, for our three participants from the pilot study, we gathered the following data:

- Think-aloud protocol
- Interview answers

5.3.2 ACT-R Simulation

We used a slightly changed version of the analysis script of Part I, which was changed to reflect the changes in the cognitive model due to the backtracking mechanism.

Simulations were done by setting a new random seed for each simulation run to reflect human behavior as closely as possible. The results of the simulations runs were saved in the same format as the empirical data except for the think-aloud protocol.

We used a Bayes Optimization algorithm to find the parameters that fit the data the best for the model. We prepared a Python script that takes parameters, the group size and the code snippet as input and outputs the results of the simulation in the same format as the empirical data. The results of the simulation were then used to calculate a value, which dependent on which aspect of the model was to be optimized. The Bayes Optimization algorithm then tried to maximize the value of the results of the simulation.

5.4 Evaluation

The evaluation was performed in four steps:

1. Analyze the results of the think-aloud study
2. Analyze the results of the empirical evaluation
3. Train the parameters of the cognitive model
4. Evaluate the cognitive model

5.4.1 Empirical Evaluation

First, we analyzed the results of the think-aloud study. The results of the additional data points from the *Connect* platform were analyzed in the same way as in Part I, which is why we do not explain the analysis in detail here. Additionally, we also analyzed the backtracking behavior of the participants in detail.

5.4.1.1 *Think-Aloud Study*

The think-aloud study was first transcribed and then analyzed. Two think-aloud studies were conducted in German, due to the participants being German, while the remaining one was conducted in English. The interview questions can be seen in the Appendix A.3. The think-aloud protocol and the semi-structured interview were analyzed with the software *MAXQDA*[3]. The think-aloud protocol was transcribed, analyzed and finally coded with the help of the software.

We classified the data using thematic analysis [11] and coded the data with an inductive approach, as we did not want to erroneously 'force' a preconceived result. We first extracted the relevant passage as a quote. From this quote, we created labels that are describing as accurately as possible the quote. Finally, categories were built from these labels. These categories were then used to understand the thought processes of the participants.

By analyzing the data of the think-aloud protocol and the interview, we were able to understand the behavior of the participants in more detail. We performed this step before implementing the changes to the cognitive model, as they provided valuable insights into code comprehension behavior. We tried to replicate these behaviors in the cognitive model.

5.4.1.2 *Backtracking Behavior*

We analyzed different aspects of the backtracking behavior of the participants. This was done by analyzing how many participants chose to go back, how many times they went back, and to which lines they went back. As this analysis was explorative, we did not have any specific hypotheses about the behavior of the participants.

5.4.2 Simulation Evaluation

The simulation evaluation was done in the same way as in Part I. Additionally, we also analyzed the backtracking behavior of the model, but not as detailed as in the empirical evaluation. We only analyzed how many times the model went back to a previous code line during each code snippet. This enabled us to compare and evaluate the behavior of the model with the behavior of the participants in a more general way. If we would have wanted to analyze the backtracking behavior of the model in more detail, we would have needed to save additional data points during the run, which would have increased the simulation

3 https://www.maxqda.com/

time significantly. As the simulation time was already quite high, we decided to not save additional data points.

6 Part II Evaluation: Extension Study

In this chapter, we will present and discuss the evaluation of the results collected from the think-aloud pilot study, the empirical evaluation and the simulation evaluation of a cognitive model able to use backtracking.

6.1 Results

First, we will present the data collection and data preparation. We will then give a short overview over the socio-demographic data of the participants. Afterward, we will present the results of the empirical evaluation. Finally, we will focus on the fitting of the ACT-R model to the empirical data and the simulation evaluation.

6.1.1 Data Collection and Preparation

In this section, we will present the data collection and preparation of the data acquired from the think-aloud pilot study and the empirical evaluation.

Think-Aloud Study

We used convenience sampling to recruit participants for the think-aloud pilot study. We contacted three contacts and asked them to participate in the study, which they agreed to. The participants were recruited from the local university and were all students. One participant was a bachelor student in cybersecurity, one was a master student in computer science and one was a master student in embedded systems.

Their ages were 23, 26 and 27 years. All three participants identified as male.

The data was collected using an audio recorder. The participants were asked to think aloud while solving the task. Additionally, the participants were asked to answer a few questions after the task.

We did not need to remove any participants from the analysis.

The data was transcribed and analyzed using the qualitative data analysis software MAXQDA[1].

1 https://www.maxqda.com/

Empirical Evaluation

The data collection was conducted using CloudResearch. The questionnaire was created using the SoSci[2] platform and was distributed via CloudResearch[3]. We started the questionnaire on 2025-02-10 and received the last results on 2025-02-18. During this time, the questionnaire was closed for four days to ensure the data quality of the first participants.

In total, 656 participants started the questionnaire, of which 139 participants completed the questionnaire. Of these 139 participants, 123 participants completed the questionnaire entirely and were included in the analysis. The high number of participants that started the questionnaire but did not complete it is due to the fact that we used a crowdsourcing platform and performed a screening at the start of the questionnaire, that filtered out those participants who did not meet the requirements. The questionnaire received a rating of 4.6\5 with 64 votes on CloudResearch, which implies that the participants were satisfied with the quality of the questionnaire.

Of the 123 participants, we excluded 27 participants due to completing the study too fast, i.e less than 1.5 times the interquartile range below the first quartile. Therefore, 96 participants were included in the analysis and evaluation. The participants were randomly assigned to four groups, whose distribution is shown in Table 6.1.

Group	Count
1	24
2	25
3	25
4	22

Table 6.1: Number of participants randomly assigned to four groups

As a reminder, we created four groups to reduce learning effects by changing the order of the code snippets. As groups 1 and 3, and groups 2 and 4, differed solely in the order of their snippet presentation, groups 1 and 3 were combined into group 1, and groups 2 and 4 were combined into group 2 for the subsequent evaluations. The two remaining groups differentiate in the type of the code snippet, i.e. treatment or control group, which is used to analyze the effect of the treatment. This resulted in a total of 49 participants for Group 1 and 47 for Group 2. All simulations of the ACT-R model utilised in the subsequent analyses were calculated with these two group sizes.

6.1.2 Socio-Demographic Data

The following section will provide an overview of the participants and their socio-demographic data for the empirical evaluation and the think-aloud pilot study.

2 https://www.soscisurvey.de/
3 https://www.cloudresearch.com/

Empirical Evaluation

The average participant was 37.0 years old, with a standard deviation of 12.0 years. The youngest participant was 19 years old, while the oldest participant was 82 years old.

The distribution of the age groups can be seen in Figure 6.1. Most people were around 25–40 years old, even though older participants are still present.

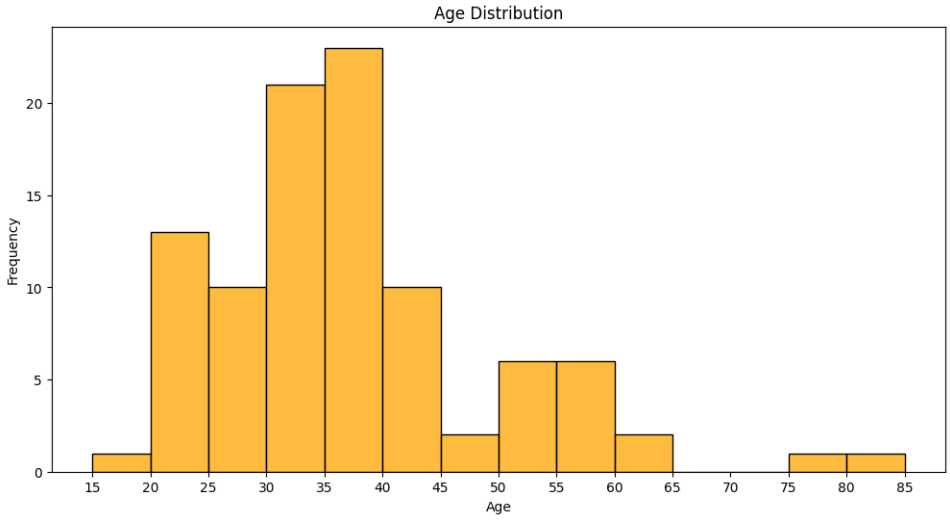


Figure 6.1: Age Groups of Participants

The country distribution of the participants is shown in Figure 6.2. We can clearly see that the majority of participants were from the USA, followed by the United Kingdom and Canada. This is probably due to the fact that the questionnaire was distributed via CloudResearch, which is a crowdsourcing platform that is mainly used in the USA and other english-speaking countries. Additionally, CloudResearch is not available in some countries, for example in Germany.

The participants were asked to indicate their highest level of education. The distribution of the education levels is shown in Figure 6.3 The majority of participants had a bachelor's degree with 57.3%, followed by a master's degree with 22.9% and a high school diploma with 14.6%.

The participants were also asked to indicate their gender identity. The most common gender of the participants with 75% was male. Woman followed with around 22.9%, while one remaining participant had their gender not listed and the other preferred not to answer.

6.1.3 Think-Aloud Study

We used the software MAXQDA[4] to analyze the data of the think-aloud pilot study. The recordings were transcribed and important parts were marked and coded, which resulted in the following results:

It is evident that uncertainty or not knowing the value of a variable are the most important aspect of reasons to use backtracking. The most prominent label with 24 mentions is that

4 https://www.maxqda.com/

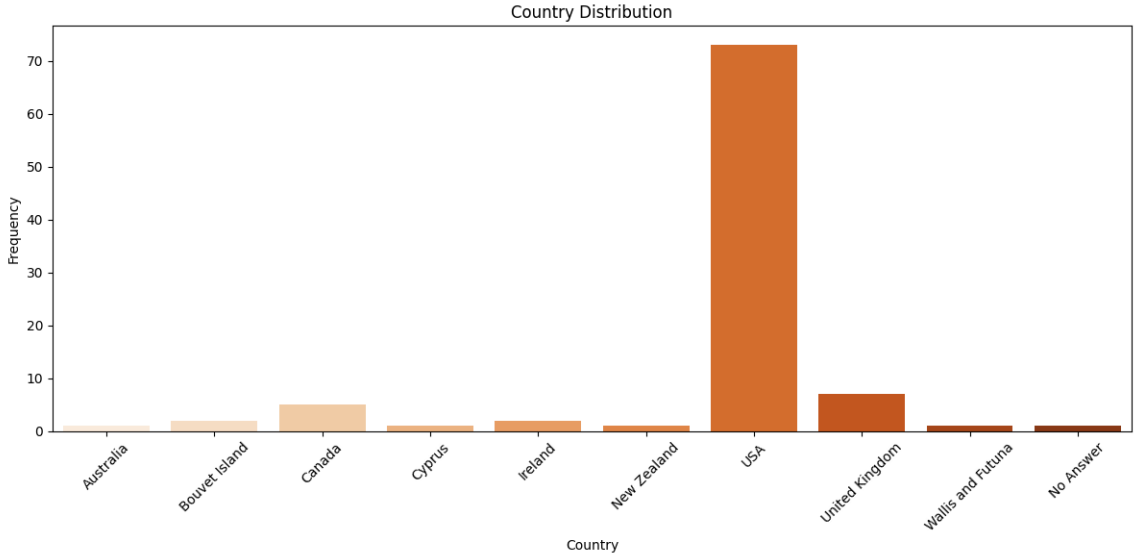


Figure 6.2: Country Distribution of Participants

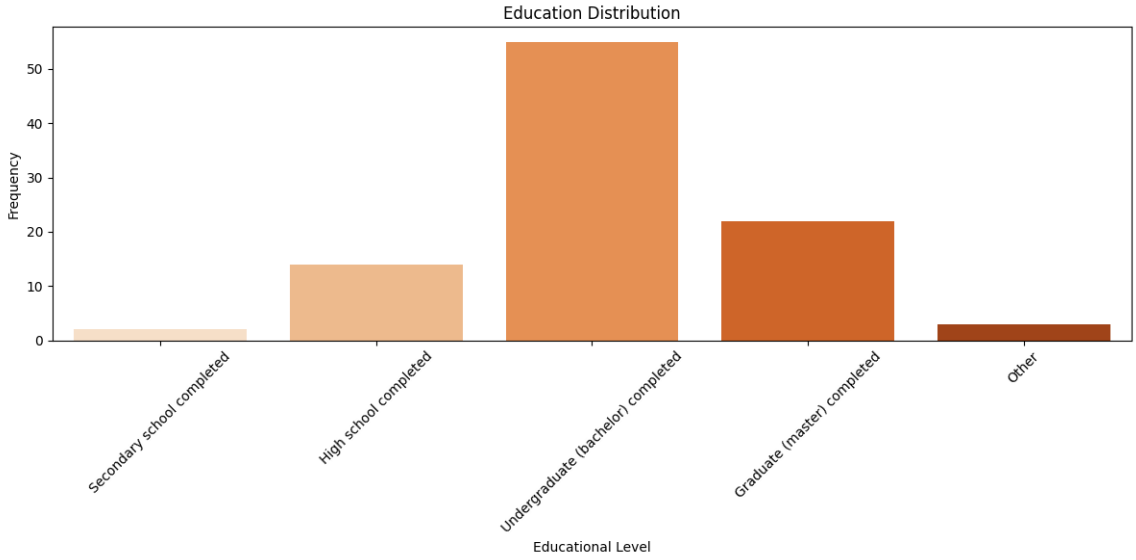


Figure 6.3: Education Level of Participants

Table 6.2: Backtracking Reasoning Categories

Categories	# of Mentions	# of Labels	Description
Uncertainty	40	7	Mentions of being unsure about/forgetting the value
Cognitive Load	6	3	Mentions of amount/complexity of information
Strategy	5	4	Mentions of minimizing effort or maximizing memory use

participants forgot the value of the variable and thus wanted to go back to remember it. A few participants mentioned that they were unsure about the value of the variable due to different reasons and wanted to go back to check it.

Cognitive load also played an important role in the backtracking behavior of the participants. Most participants mentioned that the similar variable names made it difficult to remember the values of the variables. Additionally, there were mentions of having too much input and thus difficulty to remember the value of a specific variable, necessitating backtracking.

Participants also used different strategies during their backtracking behavior. Some participants went back for multiple variables during one backtracking action. Others used *assert* keywords as anchors to remember the value of a variable, thus making further backtracking to the wanted variable unnecessary. One participant also mentioned being unsure about a value but deciding not to backtrack due to being too lazy.

6.1.4 Empirical Evaluation

The following chapter will present the results of the empirical evaluation. We will present the data of both code snippets for each group, due to having different changes (i.e. similarity and distance changes) and the results of the statistical analysis. Each type of treatment will be discussed separately to allow for a better understanding of the results.

For each type of treatment, we will analyze the results of the empirical evaluation with regard to the time data and the error data.

We used the t-test to compare the means of the two groups. A value of $p < 0.05$ was considered statistically significant, but was later adjusted with the Bonferroni correction in Section 6.1.4.6 to account for multiple comparisons. While our group sizes are great enough to assume normality, we also checked the normality of the data with the Shapiro-Wilk test. The results of the Shapiro-Wilk test are shown in the tables for each code snippet, but were not used. Additionally, we tested for homogeneity of variance with the Levene test, which is also reported in the tables for each code snippet. In the following, if the Levene test says that the data is not normally distributed, we used the Welch's t-test instead of the t-test without reporting it again. The values are reported in the tables in the same column as the t-test.

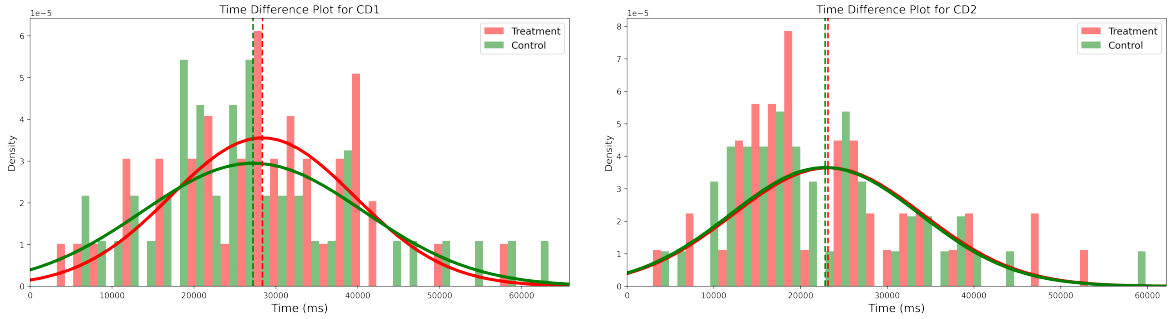
Finally, we will also have a detailed look at the usage and impact of backtracking on the results.

6.1.4.1 *Code Distance*

The code distance treatment was designed to investigate the influence of the distance between the declaration of a variable and its usage at a later point in the code snippet.

Time Data

The results of the time data of code distance treatment are shown in Figure 6.4.



(a) (b)

Figure 6.4: Time Data for Code Distance Snippet CD1 (a) and CD2 (b)

We can see that for code snippet CD1, the control group was faster by around 1 second on average than the treatment group. The change introduced for CD1 was a higher distance between the declaration and usage of the variable. This aligns with the expectation that the treatment group would be slower due to the higher distance between the declaration and usage of the variable.

For code snippet CD2, the treatment group was slower by around 0.3 seconds on average than the control group. The change introduced for CD2 was similarity between variable names.

The results of the statistical analysis can be seen in Table 6.3.

	Shapiro Control		**Shapiro Treatment**		**Levene**		**t-Test**		**Cohen's d**
	Value	**p**	**Value**	**p**	**Value**	**p**	**Value**	**p**	
CD1	0.953	0.058	0.986	0.817	0.619	0.433	0.455	0.650	0.093
CD2	0.939	**0.013**	0.944	**0.024**	0.005	0.941	-0.151	0.880	-0.031

Table 6.3: Statistical Analysis for CD1 and CD2

The results suggest that for CD1, the data is normally distributed, variances are equal, and there is no significant difference between control and treatment groups, with a negligible effect size. For CD2, the data deviates from normality, but variances are equal, and there is no significant difference between groups, with a similarly negligible effect size.

Error Data

The values of the error data analysis can be seen in Table 6.4.

For CD1, the treatment group achieved a higher correctness rate (87.76%) compared to the control group (76.60%). However, the difference was not statistically significant ($p = 0.244$) and had a small effect size ($\phi = 0.119$). For CD2, the treatment group had a correctness rate of 80.85%, slightly lower than the control group's 87.76%. Again, the difference was not statistically significant ($p = 0.516$) and showed a negligible effect size ($\phi = 0.066$). These

results suggest that while there are observable differences in correctness rates, they lack statistical significance and have minimal practical impact.

Overall, the findings indicate no strong association between group type and correctness for either CD1 or CD2.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	6	43	87.76	1.356	0.244	0.119
Control	11	36	76.60			

Table 6.4: CD1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	9	38	80.85	0.423	0.516	0.066
Control	6	43	87.76			

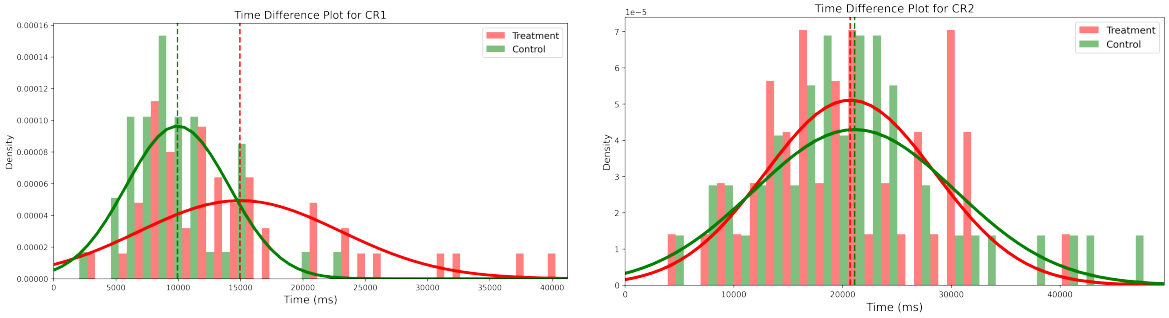
Table 6.5: CD2 Error Rate

6.1.4.2 *Repeated Code*

The repeated code treatment was designed to investigate the influence of having multiple declarations of variables in the same code snippet.

Time Data

The results of the time data of repeated code treatment are shown in Figure 6.5.



(a) (b)

Figure 6.5: Time Data for Code Snippet CR1 (a) and CR2 (b)

We can see that for code snippet CR1, the control group was faster by around 5 seconds on average than the treatment group. The change introduced for CR1 was similarity between variable names.

For code snippet CR2, the treatment group was slightly faster by around 0.5 seconds on average than the control group. The change introduced for CR2 was a higher distance between the declaration and usage of the variable.

The results of the statistical analysis can be seen in Table 6.6.

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
CR1	0.946	**0.029**	0.870	**<0.001**	7.399	**0.008**	3.793	**<0.001**	0.774
CR2	0.946	**0.026**	0.979	0.551	0.101	0.752	0.236	0.814	0.048

Table 6.6: Statistical Analysis for CR1 and CR2

The CR1 analysis reveals substantial disparities in normality, variance, and means between the control and treatment groups, exhibiting a large effect size. The t-test confirmed a significant difference in means ($p < 0.001$). The CR2 demonstrates no significant disparities in variance or means, and the effect size is negligible, indicating minimal practical differences between the groups.

Error Data

For CR1, both the treatment and control groups achieved similar levels of accuracy, with the treatment group attaining an accuracy rate of 87.76% and the control group attaining an accuracy rate of 87.23%. The observed difference was not statistically significant ($p = 1.0$) and had no effect size ($\phi = 0.0$). Similarly, the treatment group exhibited a marginally elevated rate of correctness (93.62%) in comparison to the control group (91.84%). However, this difference was not statistically significant ($p = 1.0$) and showed no effect size ($\phi = 0.0$). The findings indicate that there is no statistically significant discrepancy in the accuracy of responses between the treatment and control groups for either CR1 or CR2. The findings indicate an absence of a strong association between group type and correctness in the Repeated Code treatment.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	6	43	87.76	0.0	1.0	0.0
Control	6	41	87.23			

Table 6.7: CR1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	3	44	93.62	0.0	1.0	0.0
Control	4	45	91.84			

Table 6.8: CR2 Error Rate

6.1.4.3 *Declaration Redeclaration Distance*

The declaration redeclaration distance treatment was designed to investigate the interaction between the CD and CR treatments. Specifically, the treatment consists of having a double declaration of a variable at the beginning of the code snippet (CR treatment) and a usage of the variable at the end of the code snippet (CD treatment).

The control code snippets have their second declaration of the double declaration in the line before the usage of the variable.

Time Data

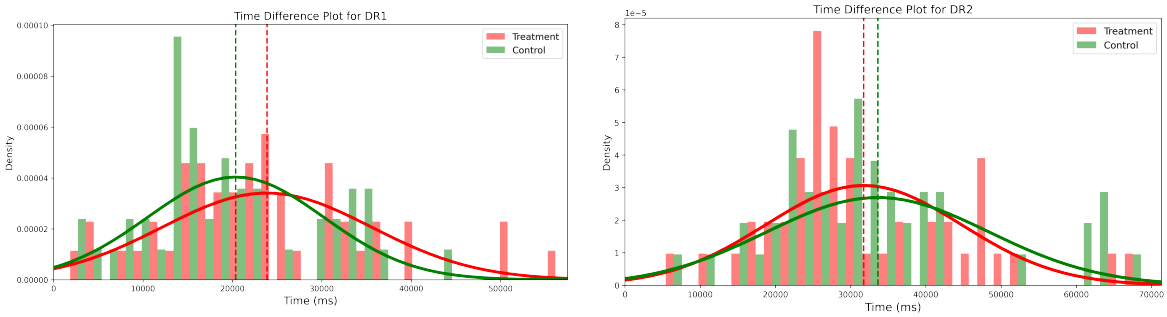
The results of the time data of repeated code treatment are shown in Figure 6.6.

We can see that for code snippet DR1, the treatment group was slower by around 3.5 seconds on average than the control group. The change introduced for DR1 was similarity between variable names.

For code snippet DR2, the treatment group was slighty faster by around 2 seconds on average than the control group. The change introduced for DR2 was a higher distance between the declaration and usage of the variable.

The results of the statistical analysis can be seen in Table 6.9.

The results suggest that for DR1, the data is approximately normally distributed, variances are equal, and there is no significant difference between control and treatment groups, with



(a) (b)

Figure 6.6: Time Data for Code Snippet DR1 (a) and DR2 (b)

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
DR1	0.956	0.077	0.962	0.110	0.458	0.500	1.576	0.118	0.322
DR2	0.933	**0.008**	0.942	**0.022**	0.321	0.572	0.662	0.510	0.135

Table 6.9: Statistical Analysis for DR1 and DR2

a medium effect size. For DR2, the data deviates from normality, but variances are equal, and there is no significant difference between groups, with a small effect size.

Overall, the findings suggest no strong association between group type and speed for the DR treatment.

Error Data

The analysis of the DR1 and DR2 error rates highlights minor differences in correctness between the treatment and control groups. For DR1, the treatment group achieved a higher correctness rate (83.67%) compared to the control group (76.60%), but the difference was not statistically significant ($p = 0.539$) and had a negligible effect size ($\phi = 0.063$). Similarly, for DR2, the treatment group exhibited a slightly lower correctness rate (74.47%) compared to the control group (77.55%), but this difference was also not statistically significant ($p = 0.909$) and showed a negligible effect size ($\phi = 0.012$). These results suggest that while there are observable differences in correctness rates, they lack statistical significance and practical impact.

Overall, the findings indicate no strong association between group type and correctness for either DR1 or DR2.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	8	41	83.67	0.377	0.539	0.063
Control	11	36	76.60			

Table 6.10: DR1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	12	35	74.47	0.013	0.909	0.012
Control	11	38	77.55			

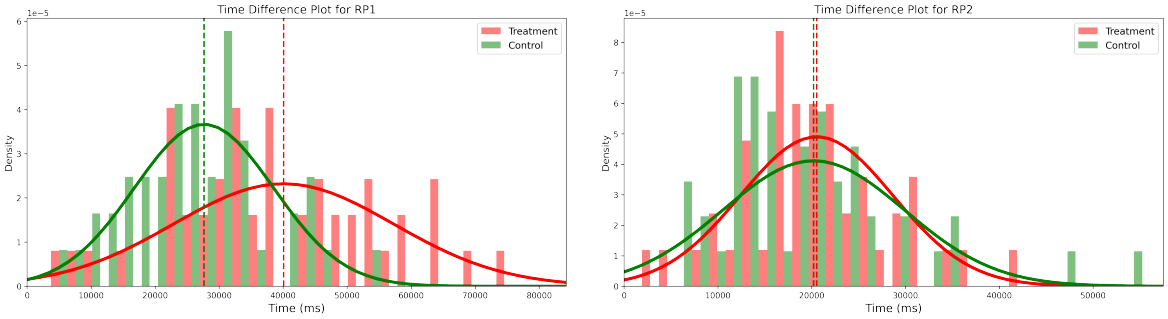
Table 6.11: DR2 Error Rate

6.1.4.4 *Repeated Distance*

The repeated distance treatment manipulated the position of filler lines relative to a double declaration and its usage. In the treatment group, some filler lines were inserted between the double declaration and its subsequent usage. In contrast, in the control group, the same filler lines appeared before the double declaration.

Time Data

The results of the time data of repeated code treatment are shown in Figure 6.7.



(a) (b)

Figure 6.7: Time Data for Code Snippet RP1 (a) and RP2 (b)

We can see that for code snippet RP1, the treatment group was slower by around 12.5 seconds on average than the control group. The change introduced for RP1 was a higher distance between the declaration and usage of the variable, with less filler lines like comments.

For code snippet RP2, the treatment group was slighter slower by around 0.3 seconds on average than the control group. The change introduced for RP2 was similarity between variable names.

The results of the statistical analysis can be seen in Table 6.12.

	Shapiro Control		**Shapiro Treatment**		**Levene**		**t-Test**		**Cohen's d**
	Value	**p**	**Value**	**p**	**Value**	**p**	**Value**	**p**	
RP1	0.986	0.836	0.987	0.878	8.272	**0.005**	4.210	**<0.001**	0.860
RP2	0.880	**<0.001**	0.987	0.870	0.610	0.437	-0.176	0.861	-0.036

Table 6.12: Statistical Analysis for RP1 and RP2

The findings indicate that for RP1, the data appears to be approximately normally distributed for both groups. However, it is evident that the variances are not equal. The application of the t-test reveals a statistically significant difference between the control and treatment groups, with a substantial effect size. For RP2, the data exhibits deviation from normality for the control group. Nevertheless, it is observed that the variances are equal, and the t-test demonstrates an absence of significant difference between the groups, exhibiting a negligible effect size.

The findings indicate a strong association between group type and speed for RP1, while no meaningful association is observed for RP2.

Error Data

The analysis of RP1 and RP2 error rates can be seen in Table 6.32. For RP1, the control group achieved a higher correctness rate (85.11%) compared to the treatment group (71.43%), but the difference was not statistically significant ($p = 0.170$) and showed a small effect size ($\phi = 0.140$). For RP2, the control group also performed slightly better (85.71%) than the treatment group (80.85%), but this difference was not statistically significant ($p = 0.715$) and had a negligible effect size ($\phi = 0.037$). These results suggest that while the control group consistently outperformed the treatment group in correctness, the differences lack statistical significance and practical impact.

Overall, the findings indicate no strong association between group type and correctness for either RP1 or RP2.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	14	35	71.43	1.887	0.16957	0.140
Control	7	40	85.11			

Table 6.13: RP1 Error Rate

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	9	38	80.85	0.133	0.71494	0.037
Control	7	42	85.71			

Table 6.14: RP2 Error Rate

6.1.4.5 *Backtracking*

In this section, we will analyze the backtracking behavior of the participants.

Of all participants, 69.8% of the participants backtracked at least once during the study. The backtracking behavior of the participants is shown in Figure 6.8.

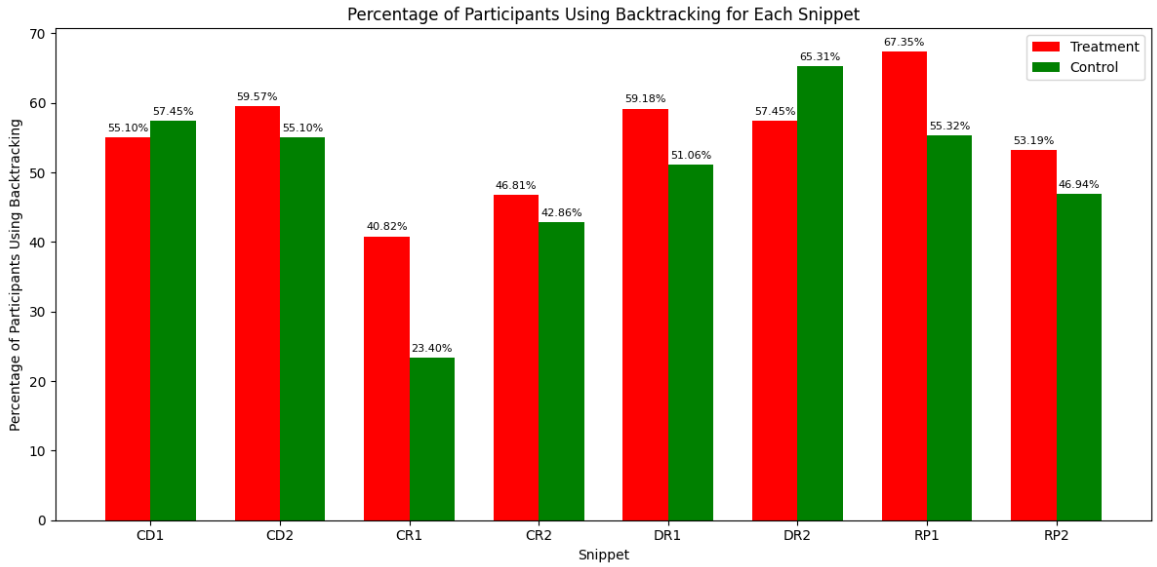


Figure 6.8: Backtracking Behavior of Participants

For easier code snippets, for example, CR1, we can see that less people backtracked than for harder code snippets, for example, RP1. For most code snippets, more participants went back to a previous line if the code snippet was a treatment variant instead of the control variant, ecept for DR2, where around 8 % more participants backtracked in the control group than in the treatment group.

The amount of average lines of code a participant backtracked is shown in Figure 6.9.

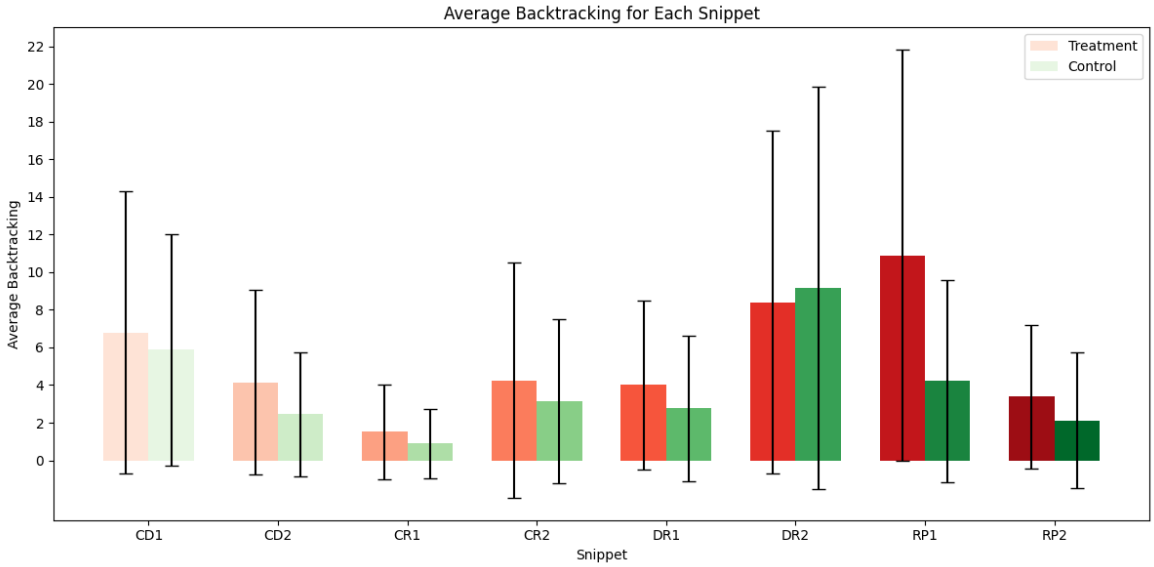


Figure 6.9: Backtracking Usage of Participants

The height of the bars indicates the average number of lines a participant backtracked. The error bars indicate the standard deviation of the backtracking behavior.

The results show that the participants backtracked more in the treatment group than in the control group for all code snippets, except for DR2, where the participants backtracked less in the treatment group than in the control group. For easier code snippets as CR1 or CD2, the participants backtracked less lines of code.

The greatest difference of backtracking behavior in between code snippets was for RP1, where the treatment group backtracked around 2.5 lines of code more than the control group. For the other code snippets, the difference was not as high, with around 1-2 lines of code more for the treatment group than for the control group.

In Figure 6.10, we can see the backtracking behavior of the participants on the print line of the code snippets. The print line is the line where the participants had to remember the value of the variable in the print statement. The bars represent the percentage of participants that backtracked at the print line, while the number on top of the bars represents the average number of lines the participants backtracked at the print line.

For six of eight code snippets, the treatment group backtracked more than the control group at the print line. As CR1 was an easy code snippet, especially for the control group, this code snippet had the lowest backtracking behavior at the print line. The greatest amount and also the highest difference between treatment and control code snippets of

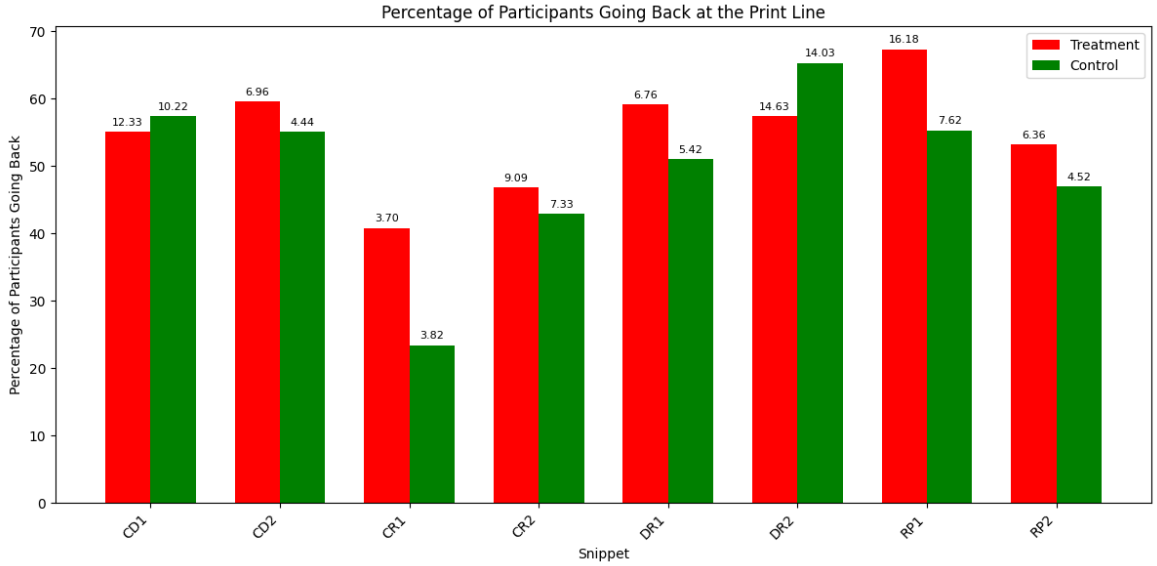


Figure 6.10: Backtracking Behavior of Participants

lines backtracked at the print line was for RP1 with around 16.2 lines of code compared to 7.6 lines of code for the control group. Code snippets that had their distance increased instead of using similar variable names had a higher backtracking behavior at the print line than code snippets that had their variable names changed, which is to be expected due to the higher distance between the declaration and usage of the variable.

CR1 and CR2 Backtracking Behavior

We analyzed the backtracking behavior of the participants for CR1 and CR2 in more detail, as these code snippets were easily comparable due to the treatment code snippet only having one line of code more at the beginning of the code snippet and due to the differences in the backtracking behavior of the participants for these code snippets seen in the previous section.

CR1 Backtracking Behavior The x-axis shows the code line that was shown to the participants in the order of the code snippet. The y-axis shows the number of times participants started to backtrack at this code line. The number on top of the bars shows the average number of lines the participants backtracked from this line of code at once. The second line has no backtracking behavior for the control snippet as it was the first line of code in the code snippet.

We can see that we have a much higher amount of lines backtracked for the treatment group than for the control group in the line of code $p = b * d$. Interestingly, the treatment group backtracked less times at the print line $print(p)$ than the control group.

CR2 Backtracking Behavior The information in Figure 6.12 is presented in the same way as in Figure 6.11.

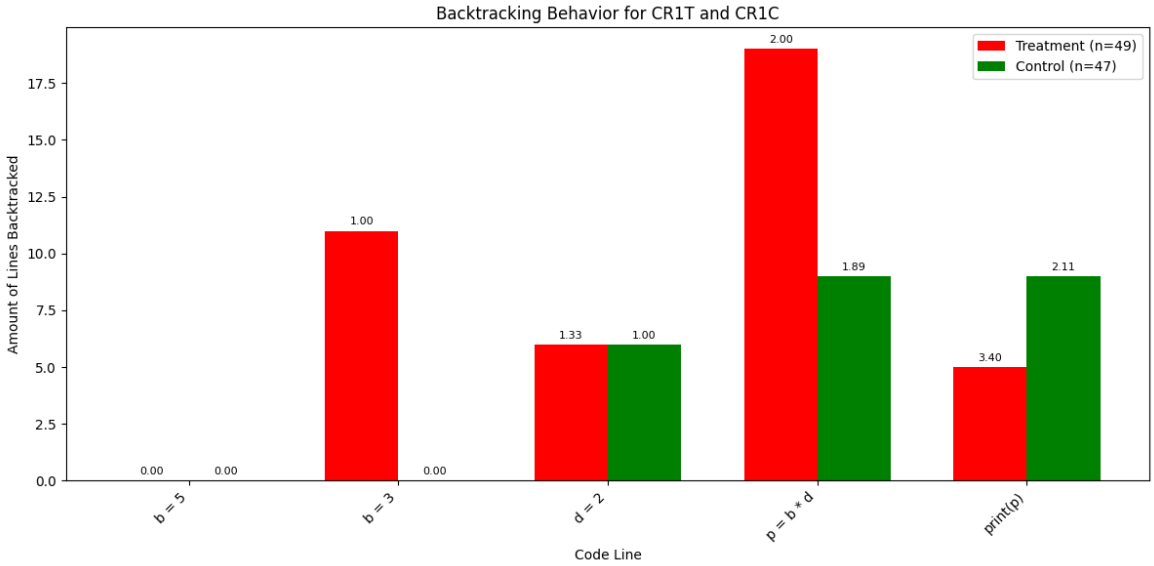


Figure 6.11: Backtracking Behavior of Participants for CR1

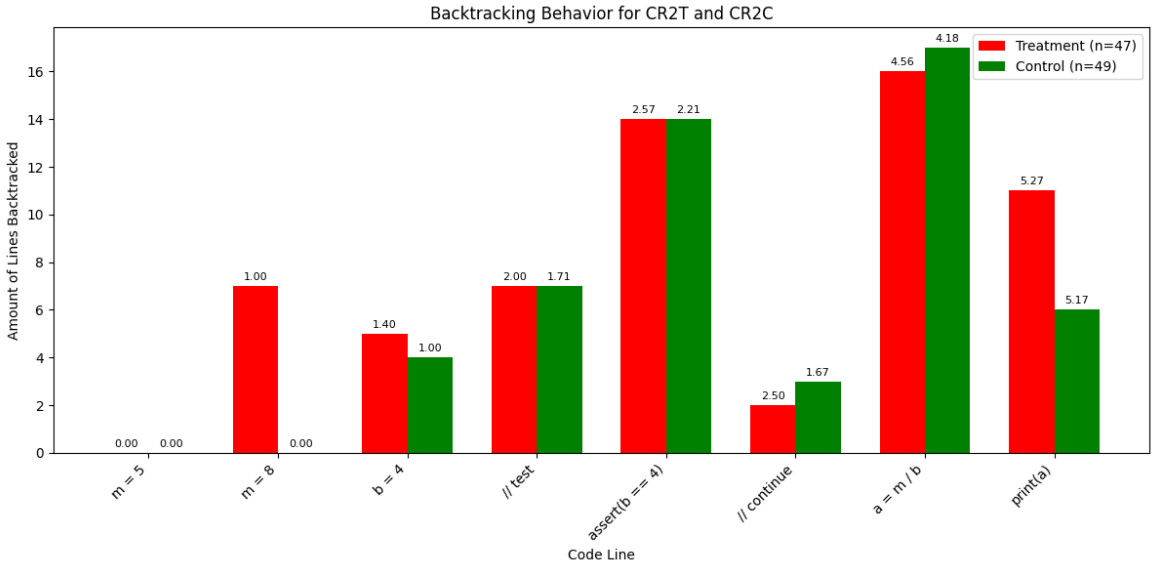


Figure 6.12: Backtracking Behavior of Participants for CR2

We can see that we have similar backtracking behaviors for the treatment and control group in all lines of code except for the print line $print(p)$, where the treatment group backtracked nearly double as often as the control group. The treatment group backtracked farther back for most lines of code than the control group, but we need to keep in mind that the treatment code snippet was one line longer than the control code snippet.

6.1.4.6 *Significance Level Correction*

We checked the significance level correction for the t-tests and χ^2 tests of all eight code snippets. Of those 16 values, two values were significant at the 0.05 level. The significance level correction can be seen in Table 6.15.

	RP1 t-test	**CR1 χ^2**
p-value	**<0.001**	**<0.001**
Bound	0.003125	0.003333
Significant	Yes	Yes

Table 6.15: Significance Level Correction

Even with the correction, the t-test for RP1 and the χ^2 test for CR1 are still significant.

6.1.5 Simulation Evaluation

In this section, we will analyze the results of the fitting of the model to the data of the simulation evaluation. Afterwards, we will analyze the results of the fitted model.

6.1.5.1 *Model Fitting*

We performed different types of optimizations with the data of the empirical evaluation to train the parameters of the ACT-R model. Specifically, we trained either three (ans, lf, rt) or four parameters (ans, lf, rt, mp) of the model. We used the following ranges for the parameters:

- ans: [0.01, 2.0]
- lf: [0.1, 5.0]
- rt: [-10.0, 10.0]
- mp: [0.2, 2.0]

The parameter mp was set to `1.5` if not varied, as it seemed like a good value during the tests that were run prior to starting the optimization. The Bayesian Optimization tried to maximize either the time data with the help of the KS statistic or the error data with the help of the χ^2 statistic. Additionally, we also tried to maximize the significance of the combined time and error data.

Due to time constrains, we did not find an maximum for the optimization of the model parameters. The experiments were run for 12 days, but due to the increased model complexity, the run time of a single optimization took too long to run more steps.

Consequently, an attempt was made to manually identify parameters that would align with the data from the empirical evaluation which lead to a further optimization of the model parameters. The results of this attempt will be presented in the final section.

In the following, we will describe the best parameters in regard to time data, error data and combined data for the optimizations with three and four parameters.

Time Data

The optimization was run for 309 steps. There was no convergence of the optimization, as we needed to stop the optimization due to time constraints. The best possible reachable sum for the sum of squared KS statistics was -1.576 found at the 287th step of the optimization. The parameters for this run are described in Table 6.16 (a).

Parameter	Value
ans	0.850
lf	1.331
rt	1.835
mp	1.5

(a)

Parameter	Value
ans	0.147
lf	0.957
rt	-8.906
mp	1.974

(b)

Parameter	Value
ans	0.193
lf	0.502
rt	0.146
mp	1.389

(c)

Table 6.16: Best Parameters for Time (a), Error (b) and Combined Data (c)

The evolution of the Bayesian Optimization can be seen in Figure 6.13.

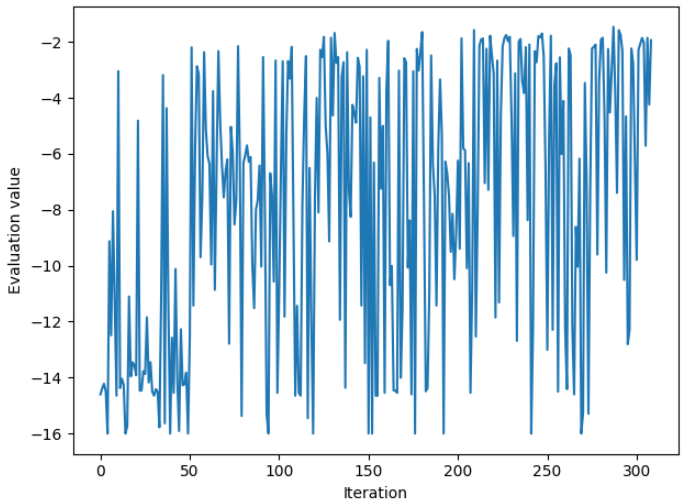
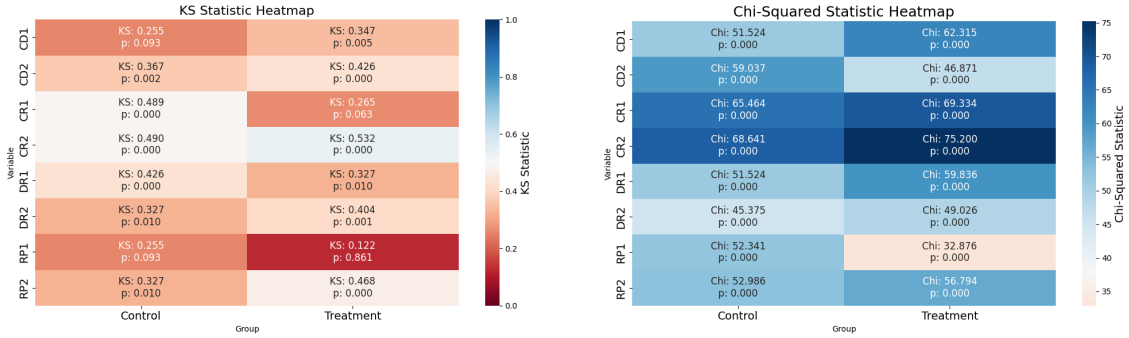


Figure 6.13: Evolution of the BO algorithm with 50 start points for tuning ans, lf, rt for time data

The accuracy in terms of time data and error rate of this approximation can be seen in Figure 6.14.

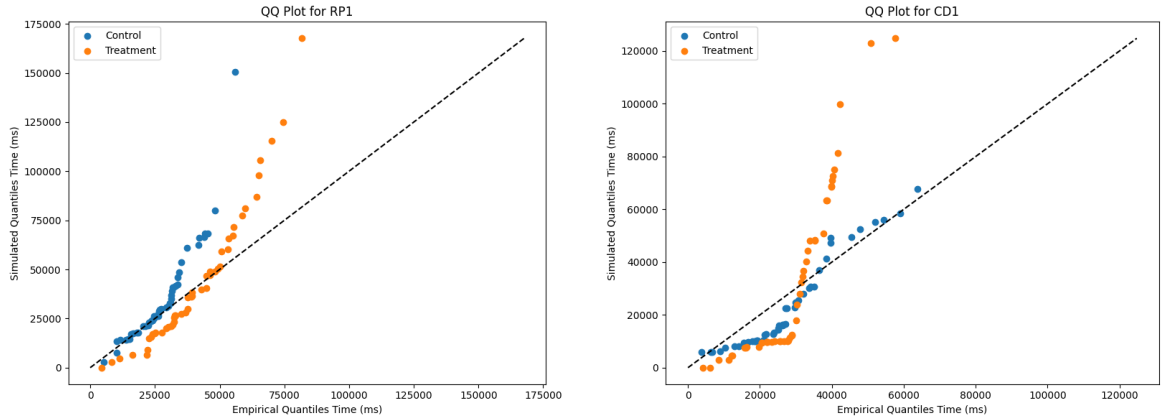


(a) (b)

Figure 6.14: Heatmaps of KS statistic (a) and χ^2 statistic (b) for tuning ans, lf, rt for time data

If we consider Figure 6.14 (a), we can see that all values are very high with all significance values being 0.0. This means that the model was not able at all to predict the responses of the participants. This was to be expected, as we only optimized in regards to time for this optimization. If we have a look at the error data, we can see that the model did not return an answer for nearly every code snippet and thus had a false result.. This optimization is therefore not realistic and does not help in providing a useful model, even if the time values are quite good.

As can be seen in Figure 6.14 (b), the worst value is 0.426. Nearly half of the code snippets have a significance value of over 0.05, which means that the probability that the simulated and empirical data have the same distribution is correspondingly high.



(a) (b)

Figure 6.15: QQ Plots for RP1 (a) and CD1 (b) tuning ans, lf, rt for time data

We used QQ plots to visualize the difference between the empirical and simulation data. As we can see in Figure 6.15, for lower time values, the simulated data fits pretty well, whereas if we advance to longer times, the model will overstimate the time needed to complete the code snippet.

Error Data

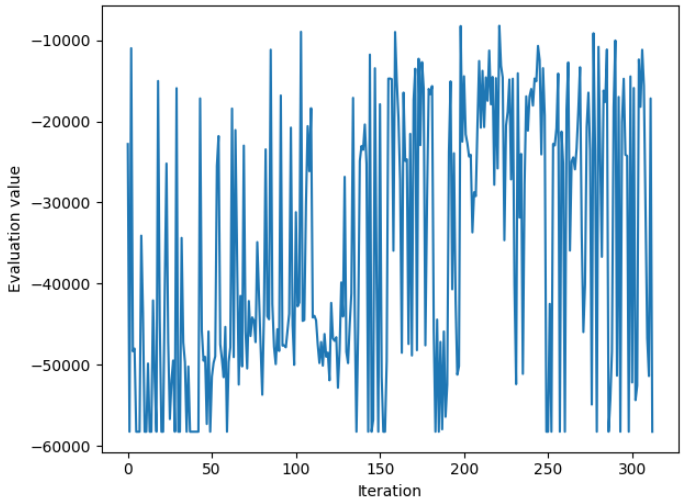
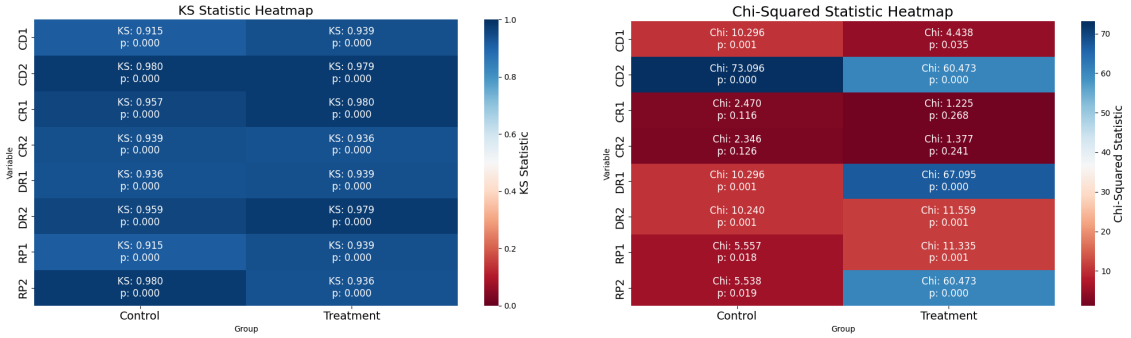


Figure 6.16: Evolution of the BO algorithm with 50 start points for tuning ans, lf, rt, mp for time data

The optimization was run for 315 steps. There was no convergence of the optimization, as we needed to stop the optimization due to time constraints. The best possible reachable sum for the χ^2 statistics was -8215.705 found at the 221th step of the optimization. The parameters for this run are described in Table 6.16 (b). The evolution of the Bayesian Optimization can be seen in Figure 6.16.



(a) (b)

Figure 6.17: Heatmaps of KS statistic (a) and χ^2 statistic (b) for tuning ans, lf, rt, mp for error data

The accuracy in terms of time data and error rate of this approximation can be seen in Figure 6.17.

As we can see in Figure 6.17 (a), the simulated time does not fit the empirical data at all. This result could be expected as we only optimized in regards to correctness of the answers. Nevertheless, the distribution of the time data is vastly different due to the simulated time data not having nearly as much variance as the empirical data. The simulated times are concentrated at around 11-12 seconds, whereas the empirical data has time values between 5–65 seconds, as can be seen in the QQ plot in Figure 6.18.

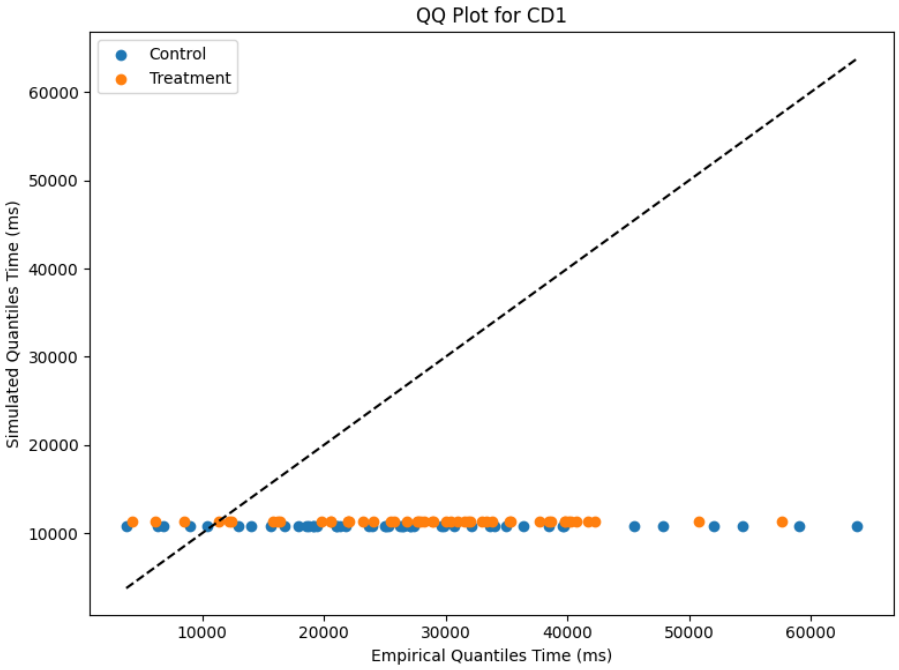


Figure 6.18: QQ Plots for CD1 for error data

Group	False	True	Correctness (%)
Control	49 (6)	0 (43)	0% (88%)
Treatment	47 (9)	0 (38)	0% (81%)

(a)

Group	False	True	Correctness (%)
Control	0 (4)	49 (45)	100% (92%)
Treatment	0 (3)	47 (44)	100% (94%)

(b)

Table 6.17: Error Rates for CD2 (a) and CR2 (b) with empirical data in brackets

The correctness of two code snippets are shown in Table 6.17 (a) and (b). The values in the brackets describes the values of the empirical data. While the χ^2 statistics already look more promising than those of the time data, we still have a lot differences to the results of the empirical data, as can be seen in Table 6.17 (a). Otherwise, the data in Table 6.17 (b) looks very similar to the empirical data.

Combined Data

The optimization was run for 155 steps. There was no convergence of the optimization, as we needed to stop the optimization due to time constraints. The best possible reachable sum for the combined significance was 6.962 found at the 119th step of the optimization. The parameters for this run are described in Table 6.16 (c). We reached this result by changing the range of the rt parameter to [-1.0, 1.0], as these were the values that needed to be used so that the model also uses the backtracking mechanism. For lower rt values, the model may just not use the backtracking mechanism at all, as it will encounter no retrieval errors.

The evolution of the Bayesian Optimization can be seen in Figure 6.19.

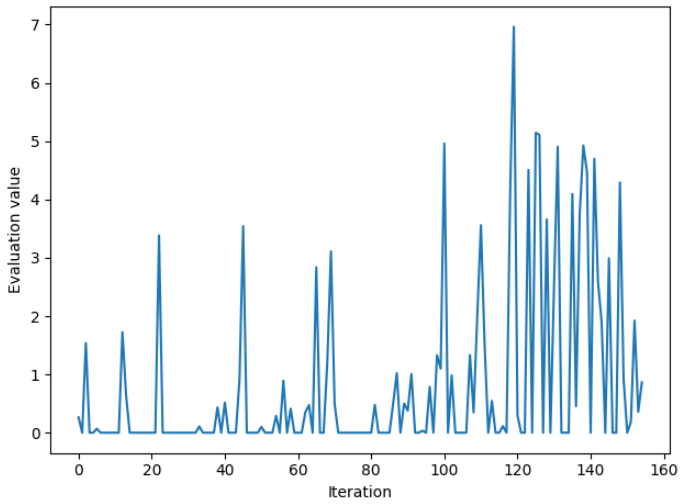


Figure 6.19: Evolution of the BO algorithm with 50 start points for tuning ans, lf, rt, mp for time data

The accuracy in terms of time data and error rate of this approximation can be seen in Figure 6.20.

Group	False	True	Correctness (%)
Control	10 (11)	37 (36)	79% (77%)
Treatment	12 (6)	37 (43)	76% (88%)

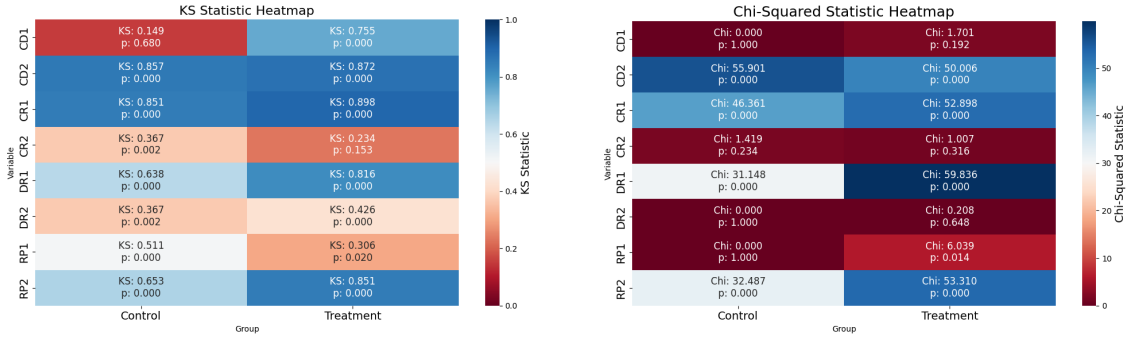
(a)

Group	False	True	Correctness (%)
Control	44 (6)	5 (43)	10% (88%)
Treatment	44 (9)	3 (38)	6% (81%)

(b)

Table 6.18: Error Rates for CD1 (a) and CD2 (b) with empirical data in brackets

The results of the χ^2 statistic are mixed, with some code snippets being a very good fit while others do not fit at all. If we look at Table 6.18 (a), we can see that the model was able

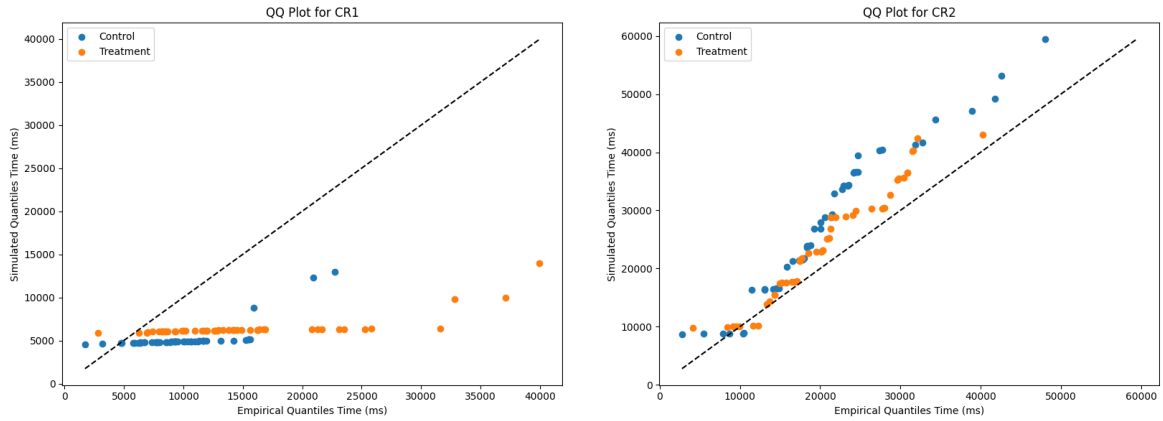


(a) (b)

Figure 6.20: Heatmaps of KS statistic (a) and χ^2 statistic (b) for tuning ans, lf, rt, mp for combined data

to predict the results of the empirical data very well. Contrary to that, Table 6.18 (b) shows that the model was not able to predict the results of the empirical data at all.

The time data in Figure 6.20 (a) also showed mixed results, as some code snippets seemed to have similar distributions. Nevertheless, most code snippets do not have a significance value of over 0.05 and possess therefore a different time distribution. We can see the differences in Figure 6.21, where the CR2 code snippet has a very similar distribution to the empirical data, while the CR1 code snippet has a very different distribution, as the time values of the simulation are concentrated in small value range.



(a) (b)

Figure 6.21: QQ Plots for CR1 (a) and CR2 (b) tuning ans, lf, rt, mp for time data

Combined Data with Parameter bll

As these results did not provide satisfactory findings, we also tried to manually fit the model to the data of the empirical evaluation with parameter values according to our understanding of the model. During this testing, we found out that while the strongly recommended value of the parameter bll was 0.5, it caused our model to forget the value of its variables too fast, causing repeated backtracking for information that was just seen.

Parameter	Value
ans	0.308
lf	1.803
rt	-0.040
mp	7.483
bll	0.220

Table 6.19: Best Parameters for Combined Data with bll

By decreasing its value, the value of a variable can be retrieved more easily. This caused another effect, as now variables that had been seen often due to backtracking caused errors due to being retrieved for another variable as their activation was too high. By increasing the parameter mp to a higher value, we increased the penalty of having different variables and therefore increased the chance of retrieving the correct variable.

We then started a new optimization with the following parameter ranges:

- ans: [0.01, 2.0]
- lf: [0.1, 5.0]
- rt: [-1.0, 1.0]
- mp: [5.0, 10.0]
- bll: [0.2, 0.3]

These values were chosen after experimenting with the model and trying to find good values for the parameters.

The optimization was run for 73 steps. There was no convergence of the optimization, as we needed to stop the optimization due to time constraints. The best possible reachable sum for the combined significance was 7.424. The parameters for this run are described in Table 6.19.

The evolution of the Bayesian Optimization can be seen in Figure 6.22.

The accuracy in terms of time data and error rate of this approximation can be seen in Figure 6.23.

The time data in Figure 6.23 (a) showed promising results, as most code snippets have a significance value of over 0.05 and therefore have a similar distribution.

If we look at the QQ plot in Figure 6.24 (a), we can see that the simulated data fits the empirical data very well, as the values are very close to the diagonal line. In Figure 6.24 (b), we can see that while the control group has no similar distrubtion, the overall positions of the points are shifted to the left, which means that the model is overrestimating the time needed to complete the code snippet. For both code snippets, we can see that higher times do not fit well on the diagonal line.

The results of the χ^2 statistic are still mixed, with some code snippets being a very good fit while others do not fit at all. Nevertheless, this model is able to predict the results of the

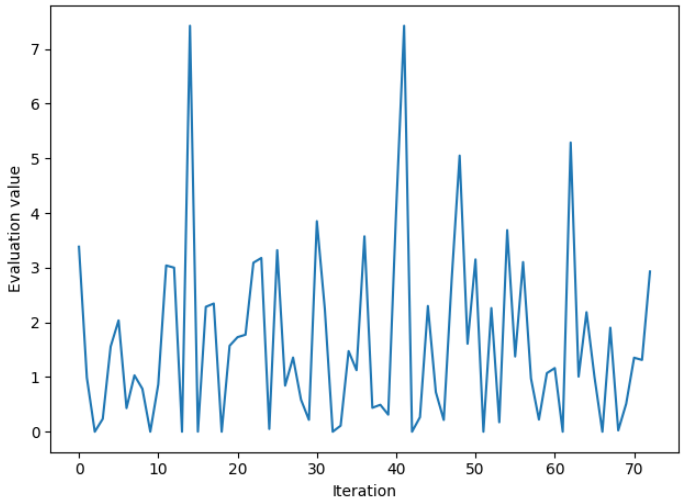
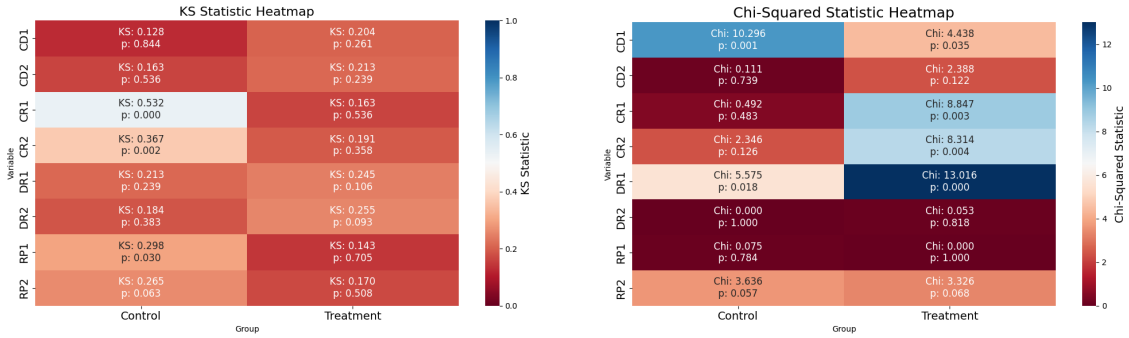
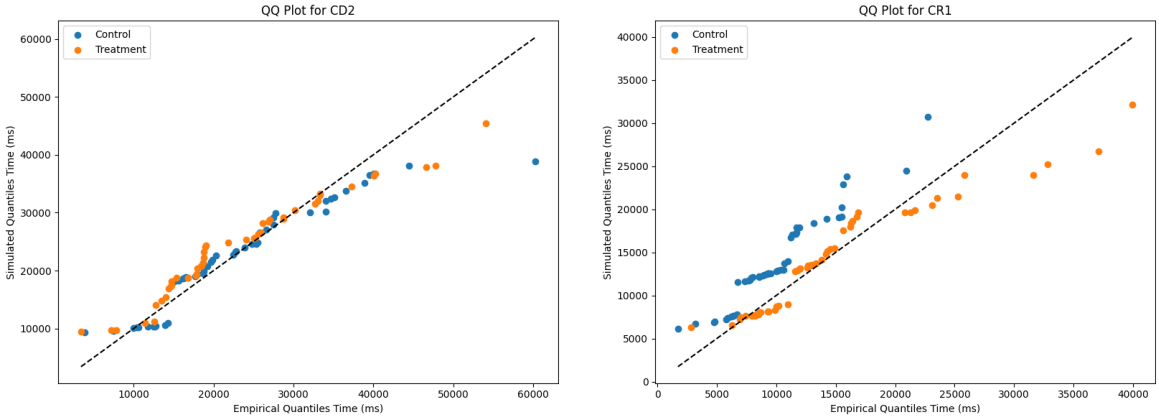


Figure 6.22: Evolution of the BO algorithm with 50 start points for tuning ans, lf, rt, mp, bll for time data



(a) (b)

Figure 6.23: Heatmaps of KS statistic (a) and χ^2 statistic (b) for tuning ans, lf, rt, mp, bll for combined data



(a) (b)

Figure 6.24: QQ Plots for CD2 (a) and CR1 (b) tuning ans, lf, rt, mp, bll for combined data

empirical data better than the previous model with ten code snippets having a significance value of over 0.05. For code snippets CD1 and DR1, the model was not able to predict the results of the empirical data very well for the treatment and control groups. Additionally, the treatment groups of CR1 and CR2 also did not fit well. If we take a look at the error rates in Table 6.20 (a), we can see that the model was always able to respond with the correct value, which the real participants were unable to do. For more difficult code snippets such as RP1, the model was able to predict the results of the empirical data very well, as can be seen in Table 6.20 (b).

Group	False	True	Correctness (%)
Control	0 (11)	47 (36)	100% (77%)
Treatment	0 (6)	49 (43)	100% (88%)

(a)

Group	False	True	Correctness (%)
Control	9 (7)	38 (40)	81% (85%)
Treatment	13 (14)	36 (35)	73% (71%)

(b)

Table 6.20: Error Rates for CD1 (a) and RP1 (b) with empirical data in brackets

6.1.5.2 *Model Evaluation*

Finally, we will evaluate the model with the help of the KS statistic and the χ^2 statistic, as we did for the empirical evaluation. For this evaluation, we used the results of the simulation with the parameters that were found in Section 6.1.5.1.

For all following time graphs, the dashed lines and striped bars represent the simulation data, while the solid lines and solid bars represent the empirical data. All following statistics will show the results of the simulation data first, followed by the empirical data in brackets.

6.1.5.3 *Code Distance*

The code distance treatment was designed to investigate the influence of the distance between the declaration of a variable and its usage at a later point in the code snippet.

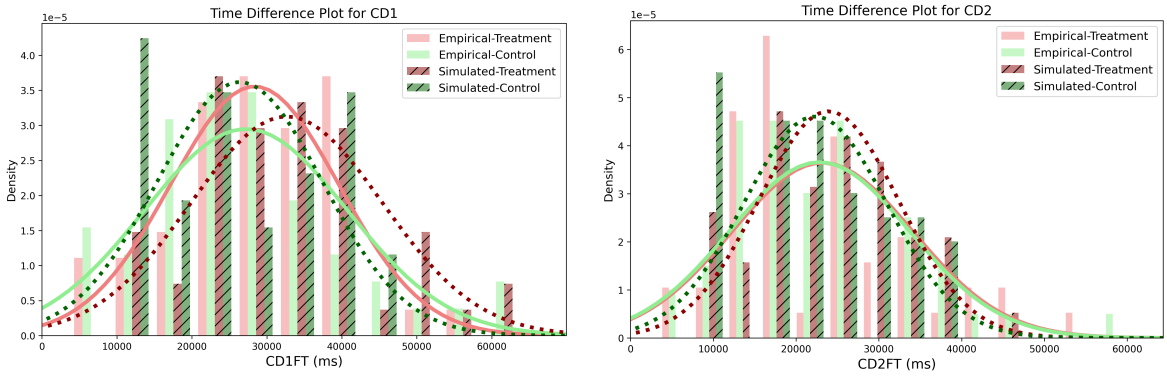
Time Data

The results of the time data of code distance treatment are shown in Figure 6.25.

We can see that for code snippet CD1, the treatment group was slower by around 6 seconds on average than the control group. The effect of the treatment was stronger in the simulation than in the empirical data, as the treatment group of the simulation was around 4 seconds slower than the treatment group of the empirical data. The control group times were very similar, with the simulation data having a more pronounced peak.

For code snippet CD2, the treatment group distributions of the simulation and empirical data were nearly identical. The control group of the simulation data had a more pronounced peak than the empirical data, which was more evenly distributed.

The results of the statistical analysis can be seen in Table 6.21.



(a) (b)

Figure 6.25: Time Data for Code Distance Snippet CD1 (a) and CD2 (b)

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
CD1	0.921 (0.953)	**0.004** (0.058)	0.949 (0.986)	0.034 (0.817)	0.001 (0.619)	0.971 (0.433)	2.567 (0.455)	**0.012** (0.650)	0.524 (0.093)
CD2	0.943 (0.939)	**0.020** **(0.013)**	0.981 (0.944)	0.636 **(0.024)**	0.019 (0.005)	0.891 (0.941)	-0.929 (-0.151)	0.355 (0.880)	-0.190 (-0.031)

Table 6.21: Statistical Analysis for CD1 and CD2 with empirical data in brackets

The analysis for CD1 shows that the simulation data deviates from normality and exhibits significant variance differences, unlike the empirical data. The simulation also indicates a significant treatment effect with a medium effect size ($d = 0.524$), whereas the empirical data shows no significant difference ($p = 0.650$) and a negligible effect size ($d = 0.093$). For CD2, both simulation and empirical data align, showing no significant differences in time ($p = 0.355$ and $p = 0.880$) and negligible effect sizes ($d = -0.190$ and $d = -0.031$).

These results suggest that the simulation overestimates the impact of increased code distance in CD1 but accurately reflects the minimal effect of similar variable names in CD2. Discrepancies in normality, variance, and effect size for CD1 highlight areas for model refinement to better align with human behavior.

Error Data

The values of the error data analysis can be seen in Table 6.23.

The error rate analysis for CD1 and CD2 reveals notable differences between the simulation and empirical data. For CD1, the simulation data shows perfect correctness for both the treatment and control groups, which significantly overestimates the empirical results. Similarly, for CD2, the simulation data indicates higher correctness rates compared to the empirical data.

The χ^2 tests for both CD1 and CD2 show no significant differences ($p = 1.0$) in the simulation data, which aligns with the empirical results. However, the simulation consistently predicts higher correctness rates, suggesting that the model may underestimate the cognitive challenges posed by the treatment, increased code distance or variable similarity.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	0 (6)	49 (43)	100.0 (87.76)	0.0	1.0	0.0
Control	0 (11)	47 (36)	100.0 (76.60)	(1.356)	(0.244)	(0.119)

Table 6.22: CD1 Error Rate with empirical data in brackets

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	3 (9)	44 (38)	93.62 (80.85)	0.0	1.0	0.0
Control	4 (6)	45 (43)	91.84 (87.76)	(0.423)	(0.516)	(0.066)

Table 6.23: CD2 Error Rate with empirical data in brackets

6.1.5.4 *Repeated Code*

The repeated code treatment was designed to investigate the influence of having multiple declarations of variables in the same code snippet.

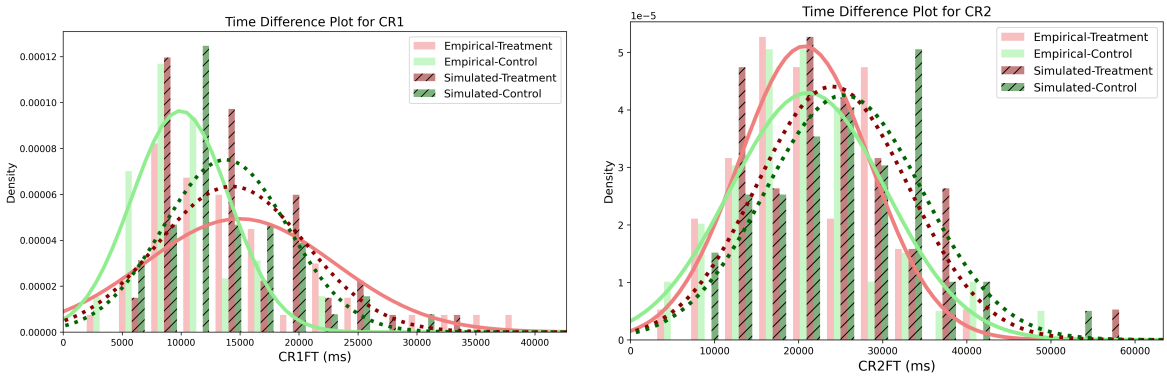
Time Data

The results of the time data of repeated code treatment are shown in Figure 6.26.

For code snippet CR1, the difference between the treatment and control groups is very small with 0.5 seconds on average, contrary to the empirical data, where a greater difference can be seen. The treatment groups of both the simulation and empirical data are nearly identical, while the control group of the empirical data has a more pronounced peak and was faster than the simulation data.

Code snippet CR2 shows a similar distribution for both groups, while being around 4 seconds slower on average than the empirical data.

The results of the statistical analysis can be seen in Table 6.24.



(a) (b)

Figure 6.26: Time Data for Repeated Code Snippet CR1 (a) and CR2 (b)

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
CR1	0.915 (0.946)	**0.002** **(0.029)**	0.919 (0.870)	**0.002** **(0.001)**	2.765 (7.399)	0.100 **(0.008)**	0.440 (3.793)	0.661 **(<0.001)**	0.090 (0.774)
CR2	0.966 (0.946)	0.174 **(0.026)**	0.906 (0.979)	**0.001** (0.551)	0.411 (0.101)	0.523 (0.752)	0.734 (0.236)	0.465 (0.814)	0.150 (0.048)

Table 6.24: Statistical Analysis for CR1 and CR2 with empirical data in brackets

The statistical analysis for CR1 and CR2 highlights differences between the simulation and empirical data. For CR1, the simulation data shows no significant difference between the control and treatment groups ($p = 0.661$), with a negligible effect size ($d = 0.090$). This contrasts with the empirical data, where a significant difference was observed ($p < 0.001$) and a large effect size ($d = 0.774$) was reported.

For CR2, both the simulation and empirical data show no significant differences between the groups ($p = 0.465$ and $p = 0.814$, respectively), with small effect sizes ($d = 0.150$ and $d = 0.048$). This alignment indicates that the model accurately reflects the minimal impact of increased distance between variable declaration and usage in this scenario.

Error Data

The error rate analysis for CR1 and CR2 highlights notable differences between the simulation and empirical data. For CR1, the simulation data shows a correctness rate of 59.18% for the treatment group, which is significantly lower than the empirical rate of 87.76%. Similarly, the control group in the simulation achieved a correctness rate of 93.61%, slightly higher than the empirical rate of 87.23%. The χ^2 test for CR1 indicates a significant difference ($p < 0.001$) in the simulation data, with a medium effect size ($\phi = 0.379$), contrasting with the empirical data, which showed no significant difference ($p = 1.0$) and no effect size ($\phi = 0.0$).

For CR2, the simulation data shows a correctness rate of 68.09% for the treatment group, which is significantly lower than the empirical rate of 93.62%. The control group in the simulation achieved a correctness rate of 100.00%, higher than the empirical rate of 91.84%. The χ^2 test for CR2 also indicates a significant difference ($p < 0.001$) in the simulation data, with a medium effect size ($\phi = 0.411$), whereas the empirical data showed no significant difference ($p = 1.0$) and no effect size ($\phi = 0.0$).

Overall, the findings suggest that the simulation overestimates the differences in correctness rates for both CR1 and CR2. While the empirical data shows no significant differences between the treatment and control groups, the simulation predicts significant differences with medium effect sizes, highlighting discrepancies that may require further refinement of the simulation model.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	20 (6)	29 (43)	59.18 (87.76)	13.780	**<0.001**	0.379
Control	3 (6)	44 (41)	93.61 (87.23)	(0.0)	(1.0)	(0.0)

Table 6.25: CR1 Error Rate with empirical data in brackets

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	15 (4)	32 (45)	68.09 (91.84)	16.193	**<0.001**	0.411
Control	0 (3)	49 (44)	100.00 (93.62)	(0.0)	(1.0)	(0.0)

Table 6.26: CR2 Error Rate with empirical data in brackets

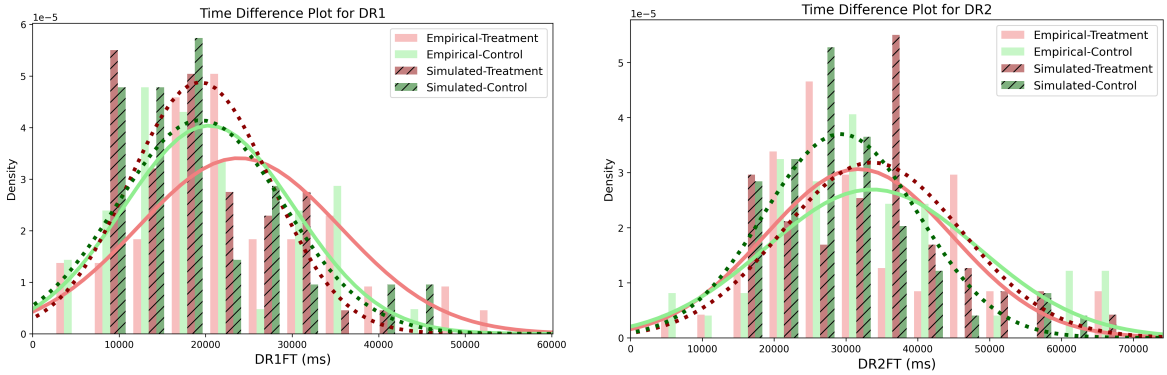
6.1.5.5 *Declaration Redeclaration Distance*

The declaration redeclaration distance treatment was designed to investigate the interaction between the CD and CR treatments. Specifically, the treatment consists of having a double declaration of a variable at the beginning of the code snippet (CR treatment) and a usage of the variable at the end of the code snippet (CD treatment).

The control code snippets have their second declaration of the double declaration in the line before the usage of the variable.

Time Data

The results of the time data of repeated code treatment are shown in Figure 6.27.



(a) (b)

Figure 6.27: Time Data for Declaration Redeclaration Snippet DR1 (a) and DR2 (b)

We can see that for code snippet DR1, the treatment group was as fast as the control group, with a more pronounced peak. Contrary to the simulation data, the treatment group of the empirical data was slower than the control group.

For code snippet DR2, the control group was faster on average than the treatment group. The empirical data does not reflect this, as the treatment group was even a bit faster than the control group.

The results of the statistical analysis can be seen in Table 6.27.

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
DR1	0.844 (0.956)	**<0.001** (0.077)	0.917 (0.962)	**0.002** (0.110)	0.110 (0.458)	0.741 (0.500)	-0.030 (1.576)	0.976 (0.118)	-0.006 (0.322)
DR2	0.922 (0.933)	**0.003** **(0.008)**	0.954 (0.942)	0.063 **(0.022)**	1.164 (0.321)	0.283 (0.572)	-1.805 (0.662)	0.074 (0.510)	-0.368 (0.135)

Table 6.27: Statistical Analysis for DR1 and DR2 with empirical data in brackets

For DR1, the simulation data shows no significant difference between the control and treatment groups (p = 0.976), with a negligible effect size (d = -0.006). This contrasts with the empirical data, where a medium effect size (d = 0.322) was observed, though the difference was not statistically significant (p = 0.118).

For DR2, the simulation data also shows no significant difference between the groups (p = 0.074), with a small effect size (d = -0.368). This aligns with the empirical data, which similarly reported no significant difference (p = 0.510) and a small effect size (= 0.135).

Overall, the findings suggest that the simulation underestimates the potential effect size for DR1 but aligns more closely with the empirical results for DR2.

Error Data

The error rate analysis for DR1 and DR2 highlights notable differences between the simulation and empirical data. For DR1, the treatment group in the simulation achieved a correctness rate of 46.94%, significantly lower than the empirical rate of 83.67%. Similarly, the control group in the simulation had a correctness rate of 51.06%, also lower than the empirical rate of 76.60%. The χ^2 test for DR1 indicates no significant difference (p = 0.841) in the simulation data, aligning with the empirical results (p = 0.539).

For DR2, the treatment group in the simulation achieved a correctness rate of 70.21%, slightly lower than the empirical rate of 74.47%. The control group in the simulation matched the empirical correctness rate of 77.55%. The χ^2 test for DR2 also shows no significant difference (p = 0.558) in the simulation data, consistent with the empirical results (p = 0.909).

Overall, the simulation underestimates the correctness rates for DR1 but aligns more closely with the empirical results for DR2.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	26 (8)	23 (41)	46.94 (83.67)	0.040	0.842	0.020
Control	23 (11)	24 (36)	51.06 (76.60)	(0.377)	(0.539)	(0.063)

Table 6.28: DR1 Error Rate with empirical data in brackets

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	14 (12)	33 (35)	70.21 (74.47)	0.344	0.558	0.060
Control	11 (11)	38 (38)	77.55 (77.55)	(0.013)	(0.909)	(0.012)

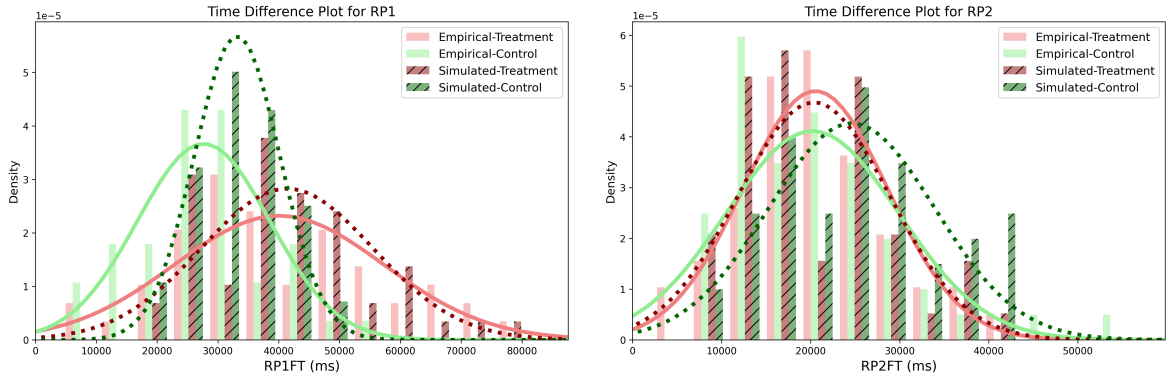
Table 6.29: DR2 Error Rate with empirical data in brackets

6.1.5.6 *Repeated Distance*

The repeated distance treatment manipulated the position of filler lines relative to a double declaration and its usage. In the treatment group, some filler lines were inserted between the double declaration and its subsequent usage. In contrast, in the control group, the same filler lines appeared before the double declaration.

Time Data

The results of the time data of repeated code treatment are shown in Figure 6.28.



(a) (b)

Figure 6.28: Time Data for Repeated Distance Snippet RP (a) and RP (b)

We can see that for code snippet RP1, the treatment group of the simulation data is very similar to the treatment group of the empirical data. The control group of the simulation data is more concentrated on specific times than its counterpart in the empirical data, while also being slower on average.

For code snippet RP2, the treatment groups of both the simulation and empirical data are very similar, while the control group of the simulation data is slower than its counterpart in the empirical data and both treatment groups.

The results of the statistical analysis can be seen in Table 6.30.

	Shapiro Control		Shapiro Treatment		Levene		t-Test		Cohen's d
	Value	p	Value	p	Value	p	Value	p	
RP1	0.979 (0.986)	0.542 (0.836)	0.956 (0.987)	0.067 (0.878)	12.088 (8.272)	**<0.001** **(0.005)**	3.590 (4.210)	**<0.001** **(<0.001)**	0.733 (0.860)
RP2	0.955 (0.880)	0.058 (<0.001)	0.919 (0.987)	**0.003** (0.870)	0.601 (0.610)	0.440 (0.437)	2.321 (-0.176)	**0.022** (0.861)	0.474 (-0.036)

Table 6.30: Statistical Analysis for RP1 and RP2 with empirical data in brackets

For RP1, the simulation data shows a significant difference between the control and treatment groups ($p < 0.001$), with a large effect size ($d = 0.733$). This aligns with the empirical data, which also reported a significant difference ($p < 0.001$) and an even larger effect size ($d = 0.860$). However, the simulation slightly underestimates the magnitude of the effect compared to the empirical data.

For RP2, the simulation data shows a significant difference between the groups ($p = 0.022$), with a medium effect size ($d = 0.474$). This contrasts with the empirical data, which reported no significant difference ($p = 0.861$) and a negligible effect size ($d = -0.036$). The simulation overestimates the impact of the treatment for RP2.

Overall, the findings suggest that the simulation aligns well with the empirical results for RP1, capturing the significant effect of the treatment, albeit with a slightly smaller effect size. However, for RP2, the simulation overestimates the treatment's impact, indicating areas where the model requires refinement to better reflect the empirical observations.

Error Data

The error rate analysis for RP1 and RP2 highlights differences between the treatment and control groups. For RP1, the treatment group achieved a correctness rate of 73.47%, which is lower than the control group's 80.85%. However, the difference was not statistically significant ($p = 0.537$) and showed a small effect size ($\phi = 0.063$). This suggests that while the control group performed slightly better, the difference lacks practical significance. The simulation data is nearly identical to the empirical data.

For RP2, the treatment group achieved a correctness rate of 61.70%, which is also lower than the control group's 67.35%. Similar to RP1, the difference was not statistically significant ($p = 0.715$) and exhibited a negligible effect size ($\phi = 0.037$). This indicates that the treatment had minimal impact on the correctness rates for RP2. Specifically, the treatment group performed differently in the empirical data, achieving a correctness rate of 85.71%. Nevertheless, there is significant difference between the treatment and control groups in either the simulation or empirical data.

Overall, the findings suggest that while the control group consistently outperformed the treatment group in correctness for both RP1 and RP2, the differences are neither statistically significant nor practically meaningful. These results highlight the limited effect of the treatment on error rates in these scenarios.

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	13 (14)	36 (35)	73.47 (71.43)	0.381	0.537	0.063
Control	9 (7)	38 (40)	80.85 (85.11)	(1.887)	(0.170)	(0.140)

Table 6.31: RP1 Error Rate with empirical data in brackets

Group	False	True	Correctness (%)	χ^2	p	ϕ
Treatment	18 (7)	29 (42)	61.70 (85.71)	0.133	0.715	0.037
Control	16 (9)	33 (38)	67.35 (80.85)	(0.133)	(0.715)	(0.037)

Table 6.32: RP2 Error Rate with empirical data in brackets

6.1.5.7 *Backtracking*

In this section, we will analyze the backtracking behavior of the model.

The model backtracked at least once during every simulation run. The backtracking behavior of the model for the different code snippets is shown in Figure 6.29.

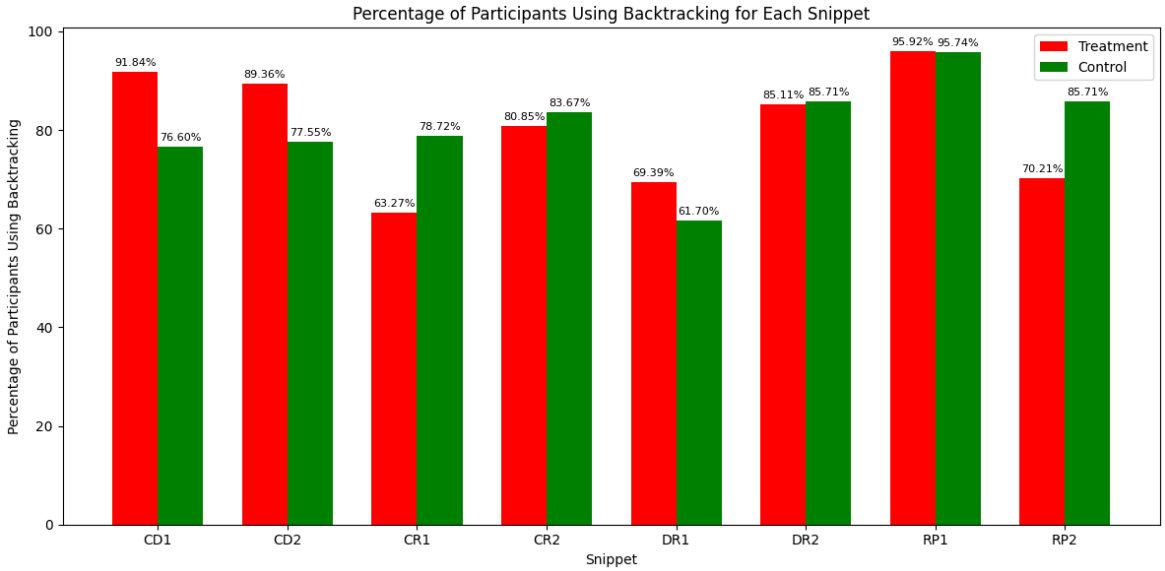


Figure 6.29: Backtracking Behavior of Participants

We can see that the model backtracked more often for difficult code snippets such as RP1 and less for easier code snippets such as CR1, similarly to the empirical data. Interestingly, the model backtracked the least for DR1. This is likely due to the rather low correctness

rate of the model for this code snippet, as the model confuses the values of the double declaration, and thus does not feel the need to backtrack to verify. The code snippet is on the easier side but due to similarity errors, the model does not backtrack as often as for the other code snippets.

Half of the code snippets have a higher backtracking rate for the treatment group than for the control group. This does not reflect the empirical data, as the treatment group had a higher backtracking rate for all except one code snippet. Especially the code snippets CR1 is interesting, as the model backtracked more often for the control group than for the treatment group, even though the code snippets were the same except for the treatment group having one additional line, a double declaration, in the first line. This results seems to be due to the same effect as for DR1, where the model confuses the values of the double declaration and thus does not feel the need to backtrack to verify, which can be seen in the distribution of the answer data.

The amount of average lines of code a participant backtracked is shown in Figure 6.30.

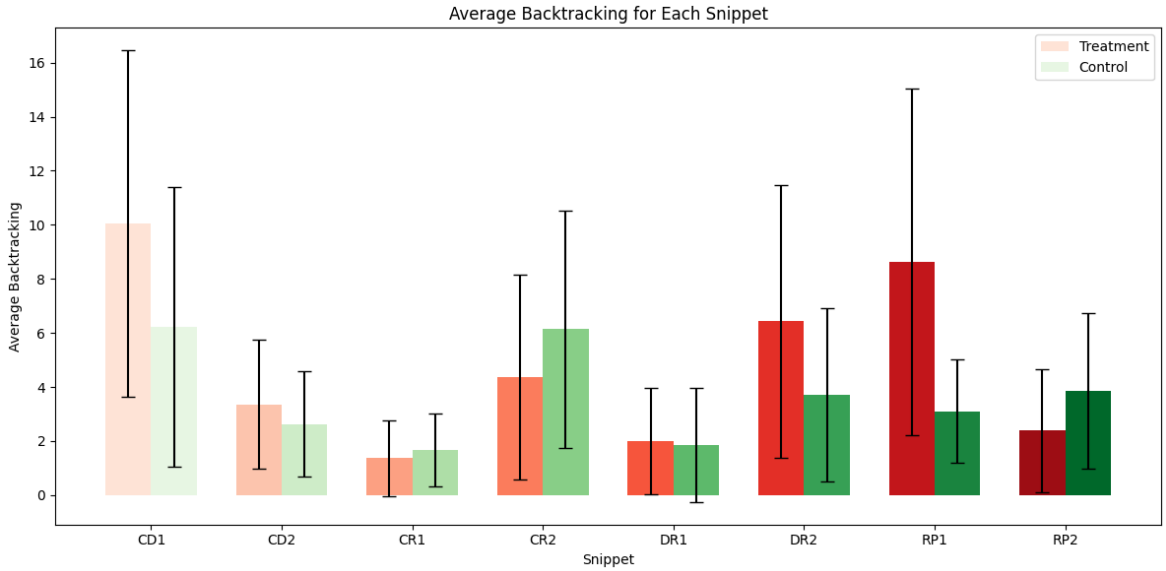


Figure 6.30: Backtracking Usage of Participants

The height of the bars indicates the average number of lines a participant backtracked. The error bars indicate the standard deviation of the backtracking behavior.

We can see five of the eight code snippets have a higher amount of lines backtracked for the treatment group than for the control group. This is similar to the empirical data except for CR1, CR2 and RP2, where the treatment group backtracked less than the control group. CR1 and CR2 are probably due to the same effect as described previously, where the model confuses the values of the double declaration and thus does not feel the need to backtrack to verify. RP2 is probably due to ordering effects, as one line of code where a calculation is done was moved to the beginning of the code snippet, increasing the distance between the declaration and usage of the variable and thus also the amount of lines backtracked.

Otherwise, the amount of lines backtracked is similar to the empirical data, with the empirical data having at most 11 lines on average backtracked, while the simulation data has at most 10 lines on average backtracked.

CD1 has a much higher backtracking usage in the simulation data than in the empirical data. This is probably due to the fact that the model is only able to backtrack to one variable at a time, as the treatment code snippet has two variables that are needed in one line of code one next to each other, whereas the control snippet only has one variable that is far away from the usage.

6.1.5.8 *Significance Level Correction*

We checked the significance level correction for the t-tests and χ^2 tests of all eight code snippets. Of those 16 values, five values were significant at the 0.05 level. The significance level correction can be seen in Table 6.33.

	CR1 χ^2	**CR2 χ^2**	**RP1 t-test**	**CD1 t-test**	**RP2 t-test**
p-value	<0.001	<0.001	<0.001	0.012	0.022
Bound	0.003125	0.003333	0.003571	0.003846	0.004167
Significant	Yes	Yes	Yes	No	No

Table 6.33: Significance Level Correction of the Model Results

The values of the t-test for CD1 and RP2 were not significant after the correction and may have been due to chance. All other values were still significant after the correction.

Comparison with Test Dataset

We ran the model with the same parameters for the same amount of iterations as there are participants in the test dataset. We then evaluated the model with the same statistical tests as we did for the empirical evaluation. The significance value of the combined was 14.153, which is nearly double the value of the evaluation with the training dataset. The time data and the error data had a better fit to the test dataset than to the training dataset, which is a good sign that the model is able to generalize to other datasets.

6.2 Discussion

In this section, we will discuss the results of the empirical evaluation, which encompasses the think-aloud study and the data from CloudResearch, and the simulation evaluation.

6.2.1 Think-Aloud Study

The most important finding is the confirmation that most backtracking behavior is due to uncertainty or forgetting the value of a variable. This aligned with our understanding of the

code comprehension behavior that was expected. According to this finding, we chose to implement the backtracking mechanism in a way that mimics uncertainty, which was done with the help of the retrieval threshold.

Another interesting observation was the mention of cognitive loads being reasons for using backtracking. Having code snippets that are difficult to understand with lots of variables may necessitate to backtrack to previous lines. This supports the previous hypothesis that implementing backtracking into ACT-R is an important step into creating a universal cognitive model for code comprehension.

The different strategies that were used by the participants give us a glimpse into the possible strategies a person can use with backtracking. As we only have a small sample size, more strategies are possible. All these strategies would have to be implemented into ACT-R to simulate the human mind. As returning for multiple variables in one go would break the scope of this work, we did not add this to our model. Nevertheless, it is an important finding that should be implemented in subsequent iterations. The strategy to use *assert* keywords to remember the value of a variable was implemented without too much additional work, as this was also something that could be seen from the data of the CloudResearch study.

All in all, the think-aloud study gave us a lot of insights into the backtracking behavior of participants and validated some of our hypotheses of the backtracking behavior. The findings were used to implement the backtracking behavior into the cognitive model in a way that resembles the result of this think-aloud study.

6.2.2 Empirical Evaluation

We evaluated the results of the effects of the treatments on the time data and the error data. We will not discuss the results of the socio-demographic data, as those are out of scope of this work.

Code Distance

Both code snippets have no signicant difference in the time data and the error data. For CD2, the treatment and control group has nearly the same time plot. Therefore, one may hypothesise that the addition of backtracking reduces the effects of the code distance treatment on the time data. We have a slight edge for the control group in CD1, compared to CD2. As CD1 is a code snippet that was changed to be longer, it supports our previous argument, as apparently the treatment has an effect, just not as strong as we expected.

While the error rates were slightly different, they were not statistically significant. As once the control group had a higher error rate than the treatment group and once the treatment group had a higher error rate than the control group, we can guess that the addition of backtracking reduces the effects of the code distance treatment in regards to the error rates as well.

The cause of these results could also be that the effect of the treatment is not strong enough to be detected in the time data and the error data or even not existing at all. Closheim also did not find a significant difference in the time data and the error data for the code distance treatment in their study, which would support the previous argument.

Repeated Code

The code snippet CR2 had no significant difference in the time data and the error data. As this code snippet had an increased distance between the declaration and usage of the variable, we can assume that the treatment does not have a strong effect on the time data and the error data.

Contrarily, while the code snippet CR1 did not have a signicant difference in the error rate, the time data varied significantly between the treatment and control group. As the change introduced for CR1 was similarity between variable names, we can assume that introducing similarity between variable names when a variable is declared twice may have a stronger effect on the time data. Double declarations with confusing variable names may lead to a higher cognitive load for the participants, as they have to remember the variable names and their values without mixing them up. Additionally, participants that were confused by the variable names may have backtracked more often to check the value of the variable, which would lead to a higher time data for the treatment group than for the control group.

Declaration Redeclaration Distance

For the time data, both code snippets DR1 and DR2 had a slower control group than the treatment group. While the statistical analysis showed no significant difference for both code snippets, DR1 was nearly signicant and also had a medium effect size. As DR1 also had similarity between variable names and a double declaration of a variable, we see the same effect as in CR1, where the treatment group was slower than the control group, just not as pronounced.

For both code snippets, the error rates were not significantly different. Interestingly, both code snippets had a higher correctness rate for the treatment group than for the control group, which is the opposite of what we expected. While the difference is not significant, this could be a hint that the addition of backtracking reduces the effects of the declaration redeclaration distance treatment on the error rates. If a participant is not sure about the value of a variable, which is rather the case in the treatment group, they may backtrack to the declaration of the variable and check the value of the variable. This could lead to a higher correctness rate for the treatment group than for the control group, as those in the control group may not backtrack as often and therefore not check the value of the variable, as they are sure about its value.

Repeated Distance

RP1 shows a significant difference in the time data between the treatment and control group, with a large effect size, with the treatment group being slower than the control group. Such

a result was expected, as the treatment group had a higher distance between the declaration and usage of the variable. The longer times can be explained by most participants not remembering the value of a variable or being unsure of its value and then needing to return to the declaration of the variable to check its value. Especially the second-last line, which was needed to calculate the value of the print statement, contained a variable that was declared in the first and second line. These long distances probably added to the time needed to process the code snippet. Remembering a value for longer and more lines of code is harder than remembering a value for shorter and less lines of code, which may have also led to participants backtracking more often than once to check the value of the variable.

While the error rates were not significantly different, the treatment group had a lower correctness rate than the control group. Even though both groups were able to go back to previous lines of code, the treatment group made more errors than the control group. It may be that while they had the ability to backtrack, the treatment group was not able to remember the value of the variable as well as the control group due to the higher distances between the declaration and usage of the variable. Therefore, even with all the information available, the RP treatment may still cause more errors than without the treatment.

Contrary to that, RP2 had no significant difference in the time data and the error data. The average time was nearly identical for both groups. This is probably due to the lower distance between the declaration and usage of the variable, which was similar to the control group. As the code snippet was changed to have similar variable names, the results contradict the observations from before for code snippet CR1 and DR1, where the addition of similarity may have led to a higher cognitive load for the participants and thus a higher time requirement. It could be that having a variable declared twice in the control code snippet lead to a higher cognitive load for the participants and thus also higher processing times. Therefore, double declarations with confusing variable names may lead to a higher cognitive load for the participants than just one of those two effects alone.

Backtracking

The analysis of the backtracking behavior of participants revealed several interesting insights.

Firstly, it is evident that backtracking is a prevalent behaviour among participants and, by extension, programmers, given that 70% of the participants exhibited at least one instance of backtracking during the course of the study. While we do not have evidence of why participants backtracked, we can assume that they were unsure about the value of a variable and wanted to check its value. This supports the notion that code comprehension is an iterative and frequently non-linear process.

An important observation is that the treatment group consistently backtracked more than the control group across most code snippets. This trend suggests that the introduction of our specific treatments causes an increase in complexity, leading participants to seek clarification by revisiting previous lines of code. Depending on the treatment it may be more or less pronounced. The treatment is especially pronounced in code snippet CR1, where the treatment group backtracked nearly twice as often as the control group. This demonstrates that the presence of a double declaration in code can potentially lead to confusion for readers, necessitating the review of prior lines of code to ascertain the variable's value. This,

in turn, results in an increase in cognitive load, as participants must navigate back through the code to clarify their understanding.

Interestingly, the code snippet DR2 had a lower backtracking behavior in the treatment group than in the control group, which is contrary to the other code snippets. This could be due to order effects in the code snippet, as the difference between the treatment and control group was only the order of the lines of code.

If we consider the amount of lines of code backtracked, code snippet RP1 stands out with a significantly higher backtracking behavior in the treatment group than in the control group. This finding highlights that an increased distance between declaration and usage significantly impairs comprehension, reinforcing theories that suggest spatial locality in code plays a crucial role in mental modeling during reading reference(s).

Additionally, treatment effects seem to not only lead to more backtracking but also a higher usage of lines of code backtracked. Therefore, this finding supports the hypothesis that our treatments have an effect on the cognitive load of the participants.

If we consider the backtracking behavior at the print line, we can see that most participants backtracked to verify the value of the printed variable, with most treatment groups backtracking more often and further back than the control groups. This finding suggests that when key information is separated from its usage by large blocks of filler code, participants experience greater difficulty in retrieving it, and consequently rely on backtracking to compensate.

Finally, the detailed analysis of the backtracking behavior for CR1 and CR2 revealed valuable insights into the participants' cognitive processes. While the treatment group in CR1 exhibited a pronounced backtracking behavior, particularly at the line before the print steatement, the print statement itself saw a lower backtracking rate. This suggests that participants try to understand the value of each variable in each line, even given the possibility to backtrack to check its value. As we had a double declaration in the first two lines of code for the treatment group and no double declaration for the control group, the double declaration may have lead to confusion about its value, leading to a higher backtracking behavior and thus a higher cognitive load.

This effect was not observed in the CR2 experiment, wherein the treatment group exhibited a greater degree of backtracking at the print line in comparison to the control group, rather than at the line preceding the print statement. This leads to the assumption that the distance has a stronger effect than the double declaration on the backtracking behavior of the participants. Having a longer distance between the declaration of a variable and its usage may lead to a higher amount of backtracking behavior. As the treatment group backtracked around twice as often in the print line than the control group, it seems as the double declaration treatment does have an effect on the backtracking behavior of the participants, but it shifted to the print line, as participants were probably confused about the value of the variable and thus backtracked more often to check its value.

Overall Evaluation

Overall, we can extract the following conclusions from the empirical evaluation:

Time Effects

The time data revealed that similar variable names (e. g.,CR1 and DR1) can lead to significantly longer processing times compared to control versions, suggesting increased cognitive load from similarity between variable names. Increased distance between declaration and usage (e. g.,RP1) also caused significantly longer times, especially when combined with variable similarity. Interestingly, in snippets where only distance was altered (e. g.,CD2), no significant time effects were found, indicating that backtracking can mitigate the cost of spatial code complexity. Overall, semantic confusion appears to impact time more consistently than structural changes do, unless both are combined.

Error Effects

The study revealed no statistically significant differences across treatments, with only subtle trends observed. In several cases (e. g.,DR1 and DR2), the treatment groups demonstrated slight superiority, which may be attributed to careful reading and increased backtracking induced by the confusing nature of the code. This finding indicates that errors are less susceptible to structural manipulations when programmers have access to the full code and can reread as required. This may be a significant implication for the design of programming environments or educational tools.

Backtracking Effects

Taken together, these findings suggest that minor structural changes in code, such as increasing the distance between variable declaration and use, can significantly affect comprehension strategies and impose additional cognitive load. The results emphasize the importance of code clarity and locality, especially in educational contexts or environments where quick understanding is essential.

6.2.3 Simulation Evaluation

In this section, we will evaluate the simulation results of the cognitive model. First, we will discuss the model fitting process. Afterwards, we will evaluate the results of the simulation in comparison to the empirical data.

6.2.3.1 *Model Fitting*

Training the cognitive model was a challenging process, as the previously used parameters and parameter ranges were not suitable for the new model. In addition, the increased complexity of the model increased the time for completing one run by a factor of three, which made the training process very time-consuming.

Due to the time constraints of this thesis, we were not able to optimize the model in regards to time and error data with the additional parameter variation of bll. It would have been interesting to see if the results of these optimizations would have led to a better fit than the current results. Additionally, a longer training process would have been beneficial

to find better parameters for the model. As we were only able to optimize the parameters for 76 iterations, a longer optimization process might be able to fit the model better to the empirical data.

The model provides some loop holes, as lower retrieval thresholds lead to no backtracking behavior at all. Optimization of only either time or error data leads to a model that is not able to fit the other data well. This is probably due to the backtracking mechanism being able to be misused to fit the time data of the model, while not being able to fit the error data well, as some of the defined stopping conditions may interfere with the computed result.

The use of the significance level as a combined measure warrants critical evaluation, as the resulting interpretations can differ significantly. For example, a p-value of 1 is almost never observed for the KS statistic, whereas it is often observed for the χ^2 statistic related to the error rate. Consequently, an improved score does not necessarily indicate a better fit between the model and the data. Nevertheless, it proved to be the best measure for the model fitting process, as it was able to fit the model to the empirical data better than the other measures.

As a final note, while the test set provided an interesting insight into the model's performance, it is important to acknowledge that depending on the seed that was used the test set may not be representative of the model's performance. A better approach would have been to use a different method, such as cross-validation [6], to ensure that the model is able to generalize well to unseen data. This would have required more time and resources than were available for this thesis, but it would have provided a more robust evaluation of the model's performance.

6.2.3.2 *Model Evaluation*

We evaluated the results of the effects of the treatments on the time data and the error data in comparison to the empirical data.

Code Distance

The results of the code distance treatment provide interesting insights into the differences between the simulation and the empirical data. For CD1, the simulation time data suggest a significant treatment effect with a medium effect size, in contrast to the empirical time data which show no significant difference and a negligible effect size. This discrepancy suggests that the simulation overestimates the effect of increased code distance on task performance. In addition, the simulation time data deviate from normality and show significant variance differences that are not observed in the empirical time data. These findings suggest that the simulation model may need to be refined to better capture the cognitive processes involved in dealing with increased code distance.

For CD2, the simulation and empirical time data are more in agreement, showing no significant differences over time and negligible effect sizes. This consistency suggests that the simulation accurately reflects the minimal impact of similar variable names on task

performance. However, the more pronounced peak in the control group time data of the simulation data compared to the empirical data highlights potential differences in variability that could be investigated further.

Some further limitations of the simulation error data are evident in the error rate analysis. For CD1, the simulation error data significantly overestimate the empirical results. Similarly, for CD2, the simulation error data shows higher accuracy rates than the empirical error data. While the χ^2 tests for both CD1 and CD2 show no significant differences in the simulation data, consistent with the empirical results, the consistently higher correct rates in the simulation suggest that the model may underestimate the cognitive challenges. As this effect is less pronounced for CD2, we can assume that this is especially true for the treatment effects, as CD1 did not have similar variable names, while CD2 did have similar variable names.

Overall, while the model aligns well with the empirical data for CD2, it overestimates the impact of the treatment in CD1 in terms of time and underestimes the cognitive challenges in terms of error rates. These discrepancies suggest areas for improvement in the simulation model to better reflect human behavior and cognitive challenges in these scenarios.

Repeated Code

The error rate analysis for CR1 and CR2 reveals significant discrepancies between the simulation and empirical time data. For CR1, the empirical time data indicates a significant difference between the treatment and control groups, with a medium effect size. In contrast, the simulation time data show no significant difference and no effect size, suggesting that the simulation underestimates the effect of the treatment, namely only the double declaration of one variable. Such a double declaration may lead to confusion for readers, necessitating the review of prior lines of code to ascertain the variable's value and thus increasing the time needed to process the code snippet. The model may need to have a lower retrieval threshold to better reflect the cognitive processes involved in dealing with increased code distance.

For CR2, the simulation and empirical time data show no significant differences and only small effect sizes. The model underestimates the time needed to process the code snippet, which is probably due to some parameters needing more optimization.

Regarding the error rate, the simulation data CR1 indicates a significant difference between the treatment and control groups, with a medium effect size. The treatment group in the simulation performed significantly worse than the empirical data. Nearly all wrong answers had the value `10`, which is no value that can be achieved by the code snippet, even through similarity errors. This suggests that the model needs adjustments of the parameters to better reflect the cognitive challenges associated with double declarations.

For CR2, the simulation data also shows a significant difference between the treatment and control groups, with a medium effect size. The control group in the simulation achieved perfect correctness, which slightly overestimates the empirical results, while the treatment group in the simulation underperformed compared to the empirical data. This suggests that the model overestimates the effect of the double declaration on the correctness rates, as the

treatment group in the simulation performed significantly worse than the empirical data, while only making errors due to the treatment.

Overall, the findings highlight discrepancies between the simulation and empirical time data for CR1 and CR2, suggesting areas where the simulation model requires refinement. Specifically the effect of the CR treatment on the time data is underestimated, while the effect of the CR treatment on the error data is overestimated.

Additionally, the simulation model appears to under- and overestimate the impact of the treatment on correctness rates for CR1 and CR2, indicating that the model may need to be adjusted to better reflect the cognitive challenges associated with double declarations.

Declaration Redeclaration Distance

The results of the time and error rate analyses for DR1 and DR2 reveal notable differences between the simulation and empirical time data. For DR1, the simulation time data shows no significant difference in time between the treatment and control groups, with a negligible effect size. This contrasts with the empirical data, which indicates a medium effect size, though the difference is not statistically significant. These discrepancies suggest that the simulation model does not fully capture the cognitive challenges associated with declaration-redeclaration distance, particularly for DR1. It may also be that other order effects are present in the empirical data, which have not been captured in the simulation due to missing productions of the ACT-R model.

For DR2, the simulation and empirical time data align more closely, showing no significant differences between the treatment and control groups and small effect sizes. Nevertheless, the simulation expects an increase in time needed for the treatment group, which is not observed in the empirical data. This may also be due to some order effects that are not yet implemented in the simulation model.

For DR1, the simulation also underestimates the correctness rates for both the treatment and control groups compared to the empirical data. As this code snippet implemented similarity between variable names, we can see that the model overestimated the effect of this similarity on the correctness rates, as 25% of the errors that were made were due to similarity errors. The other 25% of the errors were due the treatment, namely the double declaration, which aligns with the empirical data. Similarly, the error rate analysis for DR2 shows consistent results between the simulation and empirical data, with no significant differences and comparable correctness rates. However, the simulation slightly underestimates the treatment group's correctness rate while accurately reflecting the control group's performance. Nevertheless, the errors are not as dsitributed as in the empirical data. This suggests that the model still needs to be adjusted to create errors of wrongly remembered values, as the model is not able to create errors of wrongly remembered values, but only similarity errors.

Overall, the findings suggest that the simulation model performs better for DR2, mostly accurately capturing the minimal impact of declaration-redeclaration distance on both time and correctness rates. However, for DR1, the simulation underestimates the effect size and

correctness rates, highlighting areas where the model requires refinement to better align with observed human behavior. These results emphasize the need for further adjustments to the simulation parameters to improve its accuracy. Additionally, the model may need further refinements to better capture the cognitive processes, such as order effects or more error types, that influence task performance in these scenarios.

Repeated Distance

The results of the time and error rate analyses for RP1 and RP2 show both agreement and disagreement between the simulation and empirical data. For RP1, the simulation time data show a significant difference between the treatment and control groups, with a large effect size. This is consistent with the empirical data, which also shows a significant difference, but with an even larger effect size. However, the simulation slightly underestimates the size of the effect compared to the empirical data. The treatment groups in both datasets are very similar, but the control group in the simulation is slower and more concentrated at certain times than its empirical counterpart, suggesting that the simulation may not fully capture the variability in the performance of the control group.

For RP2, the simulation data shows a significant difference between the treatment and control groups, with a medium effect size. This contrasts with the empirical data, which reports no significant difference and a negligible effect size. While the treatment groups in both datasets are very similar, the control group in the simulation is slower than its empirical counterpart and both treatment groups. This suggests that the order effects that are present are overestimated in the simulation, leading to a higher time data for the treatment group than for the control group. The code snippet has a variable that needs to be calculated at the beginning of the code snippet, which is at a lower position in the treatment. This may lead to the control group being slower than the treatment group, as they have to backtrack more often to check the value of the variable.

The error rate analysis further supports these findings. For RP1, the control group consistently outperformed the treatment group in both the simulation and empirical data, but the differences were not statistically significant and showed small effect sizes. The simulation data closely mirrors the empirical data in terms of correctness rates, suggesting that the model accurately captures the limited effect of the treatment on error rates for RP1. For RP2, the treatment group in the empirical data performed slightly worse than the control group, similar to the simulation data. Nevertheless, in both cases, the differences were not statistically significant and exhibited negligible effect sizes, indicating minimal practical impact of the treatment on correctness rates. However, the model had a lower overall correctness rate for both groups compared to the empirical data. The model overestimates the effect of the double declaration on the correctness rates, as >30% of the errors that were made were due to similarity errors.

Overall, the findings suggest that the simulation aligns well with the empirical results for RP1, capturing the significant effect of the treatment. For RP2, the simulation overestimates the order effects impact on time and produces an overall lower correctness rate than the empirical data due to an overestimation of the treatment effect. These discrepancies highlight

the need for further refinement of the simulation model to better align with observed human behavior, particularly for scenarios like RP2 where order effects and cognitive load play a significant role in task performance.

Backtracking

The analysis of the backtracking behavior reveals both similarities and discrepancies between the simulation and empirical data. The model consistently backtracked at least once during every simulation run, with the frequency of backtracking varying across code snippets, which does not reflect the behavior of the participants in the empirical data. Such a behavior may be remedied by implementing productions that do not use backtracking with certain probabilities or by finding better suited parameters for the model.

We observed the effect of double declarations on backtracking behavior in the simulation data, where the model confused the values of the double declaration and therefore did not need to backtrack as often as the participants in the empirical data. If there would have been only one value, the model may have backtracked more often to check the value of the variable, as the retrieval threshold would have been lower due to only one declaration of the variable. This effect is also present in the amount of lines backtracked. These discrepancies highlight limitations in the model's ability to replicate human backtracking behavior, particularly in scenarios involving double declarations.

Order effects may also play a role in the backtracking behavior of the participants, as the simulation data shows a higher backtracking usage for RP1 compared to the empirical data.
The simulation data also shows a higher backtracking usage for CD1 compared to the empirical data. This discrepancy is likely due to the model's limitation of backtracking to only one variable at a time. In the treatment code snippet for CD1, two variables are needed in one line of code, which increases the backtracking usage compared to the control snippet where only one variable is far from its usage. This highlights a potential area for improvement in the model's backtracking mechanism to better handle scenarios involving multiple variables.

Additionally, there is a need to implement the different strategies that participants may use, such as those observed in the think-aloud study but not yet implemented in the model, to properly simulate the backtracking behavior of participants.

Overall, the findings suggest that while the simulation captures some general trends in backtracking behavior, it exhibits some deviations from the empirical data in specific scenarios. These discrepancies are particularly evident in cases involving double declarations and ordering effects, indicating areas where the model requires refinement to better replicate human backtracking behavior.

Overall Evaluation

The overall evaluation of the simulation model reveals both strengths and weaknesses in its ability to replicate human behavior in programming tasks. We will summarize the findings of the simulation evaluation in the following sections.

Time Effects

The model aligns well with the empirical data for some scenarios, such as CD2 and RP1, accurately capturing the minimal or significant effects of the treatments. However, for other scenarios, such as CD1, CR1, and RP2, the model either overestimates or underestimates the treatment effects. For CD1 and RP2, the model overestimates the impact of the treatment, while for CR1, it underestimates the effect of double declarations. Additionally, differences in variability and order effects, particularly for RP2, highlight areas where the model fails to fully replicate the empirical time data.

Error Effects

The model generally overestimates the treatment effects on error rates, particularly for CD1, CR1, and RP2. For CD1, the model predicts higher correctness rates than observed in the empirical data, underestimating the cognitive challenges. For CR1 and RP2, the model overestimates the impact of double declarations and similarity errors, leading to lower correctness rates than the empirical data. While the model aligns more closely with the empirical data for CD2, CR2, and DR2, it still exhibits some discrepancies, such as underestimating the treatment group's correctness rates or failing to replicate the distribution of errors.

Backtracking Effects

The model captures some general trends in backtracking behavior, such as higher backtracking rates for more difficult code snippets like RP1. However, it deviates from the empirical data in several scenarios. For example, the model backtracked less for DR1 and CR1 due to an increased activation caused by double declarations, which reduced the need for backtracking. Additionally, the model overestimates backtracking usage for CD1 due to its limitation of backtracking to only one variable at a time. The model also fails to replicate the variability in backtracking strategies observed in the empirical data, such as differences in backtracking rates between treatment and control groups.

Model Improvements

We propose the following improvements to the cognitive model based on the evaluation of the simulation results to increase the accuracy of the model and to better align it with the empirical data. We will focus on the backtracking aspect, as other improvements have already been discussed in Part I.

First of all, a longer training process would be beneficial to find better parameters for the model. As we were only able to optimize the parameters for 76 iterations, a longer optimization process might be able to fit the model better to the empirical data.

Additionally, a possible improvement would be to implement the different strategies that participants may use, such as handling backtracking scenarios involving multiple variables. This could be achieved by using the `:ppm` (procedural partial matching) parameter of the ACT-R model to implement different strategies for backtracking, such as using a probabilistic production to simulate scenarios where participants do not backtrack.

One could also implement a backtracking threshold for triggering backtracking based on task complexity, code snippet difficulty, or the number of variables involved. This would allow the model to adapt its backtracking behavior to better reflect human variability.

In conclusion, this cognitive model provides a valuable framework for understanding cognitive processes in programming tasks, but it requires further refinement to accurately capture the complexities of human behavior.

6.2.4 Comparison of Part I and Part II

In this section, we will compare the results of the empirical evaluation and the simulation evaluation with the results of Part I.

Empirical Evaluation

The time needed to process code snippets was similarly high and spread out for both parts of the thesis. The impact of similar variable names on processing time was stronger in Part II than in Part I, where the effect was negligible. Such a behavior may be due to the fact that the ability to backtrack may encourage participants to go back to be sure of the value of a variable, increasing the overall processing time. The empirical data of Part II also produced less significant effects than in Part I, which may be due to the fact that the ability to backtrack reduces the treatment effects.

Contrary to Part I and in line with the findings of Closheim's thesis, we had no unexpected behavior of the time data for the code snippets CD1, CR1, DR1 and RP1. This finding suggests that the unexpected behavior of the time data in Part I was some random noise in the data and not a systematic effect.

Simulation Evaluation

The addition of the backtracking mechanism to the cognitive model in Part II led to a better fit of the time data to the empirical data than in Part I. As the model was now able to vary the time needed more easily by backtracking, it could fit the time data better than in Part I.

Additionally, we were also able to fit the error data better than in Part I. The backtracking mechanism probably allows to fit the error data better, as it allows users to remember the values of variables better and thus leads to a lower error rate, which can be better simulated by the model.

Overall, the addition of backtracking to the cognitive model in Part II improved the fit of the time and error data to the empirical data significantly compared to Part I while also improving the model's generalizability.

6.2.5 Answering the Research Question

The aim of this part of the thesis was to evaluate the cognitive model that was created in Part II and to evaluate the accuracy in terms of processing time and the error rate. Namely, we wanted to answer the following research question:

> **RQ3**: How does the possibility to return to a previous code line influences the accuracy and predictions of the ACT-R model?

The addition of backtracking to the ACT-R model improves its ability to simulate human-like retrieval in code comprehension, especially in tasks requiring re-evaluation of variables. However, the current implementation overuses backtracking, unlike the empirical data where participants only occasionally backtracked. This rigid behavior reduces predictive accuracy by ignoring strategic variability in when and why humans choose to revisit previous lines.

Backtracking greatly improved the fitting of the processing time by allowing variability in the time needed to process a code snippet compared to Part I, while only improving the fitting of the error data slightly. The model is a valuable step towards understanding cognitive processes in programming tasks, but requires further refinement to accurately capture the complexities of human behavior.

6.3 Threats to Validity

In this section, we will discuss the threats to validity of this part of the thesis.

6.3.1 External Validity

The same threats to external validity as in Part I apply to this part of the thesis.

One threat to validity for the model creation is the low number of participants. As we only aim to recruit around three participants, the results of the study will not generalizable to the population. However, as this is a pilot study, our aim of the study is not to generalize the results of the think-aloud study, but to create hypotheses for the functionality of the backtracking component of cognitive model. Additionally, we recruited 123 participants for the empirical study, which is a sufficient number of participants to generalize the results of the study.

Another threat to validity is the use of convenience sampling for the pilot study. As convenience sampling is not representative of the population, the results of the study may not be generalizable to the population. Here again, the aim of the study is not to generalize the results, but to create hypotheses for the cognitive model from the empirical data.

6.3.2 Internal Validity

The same threats to internal validity as in Part I apply to this part of the thesis.

A threat to the internal validity of the study is the use of the think-aloud protocol. There exist several limitations of the think-aloud protocol, such as the participants not being able to verbalize all their thoughts [9]. We tried to reduce this threat by conducting a short semi-structured interview at the end of the study to gather more information about the participants' thoughts.

Another threat to the internal validity of the study is the implementation of the backtracking mechanism in the cognitive model. As we needed to base the functionality of our model on the results of the think-aloud study, we may have missed some important aspects of the backtracking mechanism that were not mentioned by the participants. We tried to mitigate this threat by conducting discussion with the advisor and other persons to rule out any grave errors in the implementation.

7

Related Work

This thesis aims to replicate and extend the ACT-R model of [10]. To achieve this, we enhanced the model by incorporating a backtracking mechanism and a similarity error mechanism. Consequently, we conducted a comprehensive literature review on cognitive architectures, letter similarity, and reading patterns.

Hansen et al. [19] provided an overview of cognitive architectures and discussed the theoretical implementation of a cognitive complexity metric. They provided a theoretical background on cognitive architectures and how they can be used to model human behavior. Furthermore, they highlight that ACT-R offers the advantage of readily available perception and motor modules, as well as the ability to simulate the blood-oxygenation-level dependent effect using ACT-R, which broadens the scope of potential experiments. The potential of cognitive architectures to model human behavior in a variety of tasks, including code comprehension, is recognized. They provided a theoretical implementation of a cognitive complexity metric, which could be used to measure the cognitive complexity of a task. However, they did not provide a concrete implementation of the metric. This is in part due to implementation difficulties, interpretation issues, and the possibility of social rejection of the results of such a model. Therefore, we will attempt to avoid overcomplicating the model and keep it as simple as possible.

In their study, Simpson et al. [32] developed a visual-similarity matrix for Latin-based alphabets. Such visual similarity matrices may be employed to ascertain the degree of resemblance between two letters. Research in this area has been conducted for over a century and a half, with findings applied across a range of disciplines, including psychology, cognitive science, and computer science Mueller and Weidemann [25]. The latest results in this field are presented by Simpson et al., who also provides a visual-similarity matrix for Latin-based alphabets, in contrast to other studies that have only considered the English alphabet. Moreover, the matrix was constructed under controlled reading conditions, which is crucial for ensuring the validity of the results. The matrix was provided for use by other researchers, and we will employ it to ascertain the similarity between variable names in the ACT-R model.

Busjahn et al. [7] investigated the linearity of code reading for novices and experts. They found that code reading is less linear than natural language reading. Additionally, they found that novices read code line-by-line, while experts read top-down and skip lines. These results imply that developers do not read code in a linear fashion, which is important for the implementation of the backtracking mechanism in the ACT-R model. Especially expert

developers may go back to previous code lines to verify their hypothesis of the code.

Jbara and Feitelson [21] explored how programmers read code with the help of eye tracking data. The scanpaths of programmers were analyzed during the reading and comprehension of code snippets, and the patterns observed were classified into categories that may be typical when reading code. The findings revealed that the majority of time and effort spent on reading and understanding code is concentrated in the initial few lines. Subsequently, the lines of code are scanned with less frequency and for shorter periods of time. Moreover, the researchers discovered that the act of reading general code is not a linear process, as their identified patterns suggest.

They also identified a pattern, named *Look back*, which describes the behavior of programmers who return to a previous code line before continuing their reading. This pattern is particularly relevant for incorporating a backtracking mechanism in the ACT-R model, as it aligns with the behaviors observed in programmers when they encounter a variable with a forgotten value or when they need to verify a hypothesis.

This work is built upon the master thesis of [10]. In their thesis, Closheim developed an ACT-R model to simulate the behavior of developers during code comprehension tasks. They conducted a study with 63 participants on different types of code snippets. They developed an ACT-R model that could realize a mental model of developers during code comprehension tasks and adjusted the parameters of the model to fit the data of the study. The study consisted of four different types of code snippets, which aimed induce a higher error rate in the participants. The types of code snippets were the same as the ones we used in our study in Part I.

The higher error rate was inspired by the study of [17]. Gopstein et al. found that the error rate of developers increased when they had to work with code snippets that contained certain syntax forms. They named these code snippets *atoms of confusion*.

While the results of the model were promising, the model was not able to predict the behavior of developers accurately in all cases. Especially the prediction of the error rate was not accurate as the model was not able to simulate the different types of errors developers made during the study. Furthermore, the model's validation would have been stronger if the study had included a larger number of participants. Finally, the model was not able to allow the participants to go back to a previous code line. This leads us to our main aims of possible improvement of Closheim work, namely a stronger validation of an improved version of the model and the implementation of a backtracking mechanism.

Table 7.1 provides a summary of the previously discussed related publications and their influence on this thesis.

Table 7.1: Reference overview

Reference	**What they did**	**How we build on it**
Hansen et al. [19]	Overview on cognitive architectures and theoretical implementation of cognitive complexity metric	Theoretical background
Simpson et al. [32]	Visual-similarity matrix of letters	Using results for partial matching
Busjahn et al. [7]	Investigated the linearity of code reading for novices and experts	Theoretical background on code reading
Jbara and Feitelson [21]	Explored how programmers read code	Theoretical background on backtracking mechanism
Closheim [10]	Developed an ACT-R model to simulate the behavior of developers during code comprehension	Main building block of extended model

8

Conclusion

With so much time spent reading and understanding code, code comprehension is a critical task in software development. As a result, it is an active area of research in software engineering. Nevertheless, the cognitive processes involved in code comprehension are complex and not yet fully understood.

In this thesis, we aimed to replicate the results of a previously developed cognitive model for code comprehension tasks by Closheim [10]. Additionally, we extended this model with a backtracking mechanism and similarity errors to improve its generalizability and provide a better foundation for subsequent studies in this area.

We conducted two studies: a replication study and an extension study. The replication study was unable to successfuly replicate the results of Closheim's study. Especially the processing time of Closheim's model was significantly higher and had not enough variability to be able to replicate the results. Closheim's model was not able to fit the empirical data well, which may also be due to the different sample of participants. By adding similarity errors and increasing the length of the code snippets, we were able to show more significant effects than in the original study, leading to a better understanding of the cognitive processes involved in code comprehension.

The extension study was able to successfully replicate the results of its empirical results. Our extended model was able to fit the empirical data well and was able to show significant effects, if not as strong as in Part I. The backtracking mechanism enabled new possibilities for our extended model to simulate the cognitive processes of developers during code comprehension tasks and provided a better generalization of our extended model.

In conclusion, we were able to show that the model of Closheim is a good starting point for simulating the cognitive processes of developers during code comprehension tasks and that the addition of a backtracking mechanism and similarity errors can improve the generalizability of the model. With further improvements and refinements, this model could be used to gain insights into the cognitive processes of developers and to develop tools that support developers in their everyday work through more efficient code comprehension.

Future Work

As this work is the second step in a long-term research project, there are many possibilities for future work. In the following, we will discuss some of the possibilities for future work.

Multi-Line Reading

As we already mentioned in the background chapter, our current model is limited to reading one line of code at a time. This is a significant limitation, as developers often need to go back and forth between different lines of code to understand the overall structure and flow of the code. Backtracking was a first step in this direction, but it is still not a perfect model of the behavior of developers.

As ACT-R is posseses a built-in mechanism for simulating eye movements, it would be possible to implement a model that is able to read multiple lines of code at a time. A study with eye-tracking data could be used to train the model and to evaluate its performance. This would also need further analysis of the eye-tracking data to understand the different strategies that participants use to read code, especially in the case of forgetting or being unsure about the value of a variable.

If such a model could be developed, it would be possible to simulate the cognitive processes of developers during code comprehension tasks in a more realistic way, enabling the development of tools that support developers in their everyday work through more efficient code comprehension.

Variability of Processing Time

A second possibility for future work would be to add variability of the processing time in the model. During our studies, we observed that the processing time of the participants had a high variability, which could not be replicated by the model. This is a significant limitation, as the model was not able to fit the empirical data well.

Implementing pauses in the model could be a first step in this direction. These pauses could be implemented by adding a production rule that would randomly be triggered, causing the model to pause for a certain amount of time. This would allow the model to simulate the variability of processing time that we observed in our studies.

Broader Code Understanding

A third possibility for future work would be to expand the syntax of the pseudo code language used in our studies. Specifically, one could add more keywords and operators to the language to make it more similar to real programming languages. This would allow for a more realistic simulation of the cognitive processes of developers during code comprehen-

sion tasks.

Such a extension could be realized by adding more production rules to the model that would allow it to understand the new keywords and operators. Currently, the model has its current capabilities each saved in a different file. Additional operators and keywords could be added to the model by adding new files with the corresponding production rules. This would allow modularity and flexibility in the model, enabling the addition of new features and capabilities without having to modify the existing code.

A

Appendix

A.1 Socio-Demographic Questionnaire

1. Which gender do you identify with?
2. How old are you?
3. Which country do you currently live in?
4. What is the highest educational qualification you have?
5. What is your current profession?
6. How would you estimate your programming experience compared to your classmates or colleagues?
7. How would you estimate your programming experience for these programming paradigms? (logical, functional, and object-oriented programming)

A.2 Screening Questions

1. Choose the answer that best fits the description of a compiler's function.
   a) Refactoring code
   b) Connecting to the network
   c) Aggregating user data
   d) I don't know
   e) Translating code into executable instructions
   f) Collecting user data
2. Choose the answer that best fits the definition of a recursive function.
   a) I don't know
   b) A function that runs for an infinite time
   c) A function that does not have a return value
   d) A function that can be called from other functions

e) A function that calls itself

f) A function that does not require any inputs

3. Which of these values would be the most fitting for a Boolean?

a) Small

b) I don't know

c) Solid

d) Quadratic

e) Red

f) True

4. What is the parameter of the function?

```
main {
 print(func("hello world"))
}

String func(String in) {
 int x = len(in)
 String out = ""
 for (int i = x - 1; i >= 0; i--){
  out.append(in[i])
 }
 return out
}
```

a) String out

b) String in

c) I don't know

d) int i = x-1;i >= 0; i–

e) Outputting a string

f) int x = len(in)

5. What would this function output?

```
main {
 print(func("hello world"))
}

String func(String in) {
 int x = len(in)
 String out = ""
```

```
 for (int i = x - 1; i >= 0; i--){
  out.append(in[i])
 }
 return out
}
```

a) hello world
b) hello world 10
c) dlrow olleh
d) word hello
e) HELLO WORLD
f) I don't know
g) hellow world hello world hello world hello world

A.3 Interview Questions

General Questions

1. What were you looking for when you went back?
2. How did going back help you understand the code?
3. Did you go back to a previous line without being unsure about/forgetting a variable?
   - If yes, why did you do it?
4. Were there moments where you considered going back but chose not to? Why?

Specific Questions

These questions were asked if the specific situation arose during the interview:

1. If a participant went back multiple times to the same line:
   - What made you return to this specific line multiple times?
2. If a participant took a long time before going back:
   - What triggered your decision to go back at that moment?
3. If a participant hesitated before answering but didn't go back:
   - Were you debating whether to check a previous line? If yes, why?
4. If a person went back for a variable which was not seen in the current line:
   - Why did you go back in this line even though this variable as not seen?

a.4 Code Snippets

CD1 Treatment

```
h = 4
k = 6
// variables
m = 3
q = 5
a = 1
i = a * 4 - h
print(i)
```

CD1 Control

```
a = 1
h = 4
k = 6
// variables
m = 3
q = 5
i = a * 4 - h
print(i)
```

CD2 Treatment

```
p = 4
q = 8
d = q * 2
b = p * 4 - d
print(b)
```

CD2 Control

```
q = 8
d = q * 2
p = 4
b = p * 4 - d
print(b)
```

CR1 Treatment

```
b = 5
b = 3
d = 2
p = b * d
print(p)
```

CR1 Control

```
b = 3
d = 2
p = b * d
print(p)
```

CR2 Treatment

```
m = 5
m = 8
b = 4
// test
assert(b == 4)
// continue
a = m / b
print(a)
```

CR2 Control

```
m = 8
b = 4
// test
assert(b == 4)
// continue
a = m / b
print(a)
```

DR1 Treatment

```
l = 2
l = 5
i = 8
h = i / 4
n = h * l
print(n)
```

DR1 Control

```
l = 2
i = 8
h = i / 4
l = 5
n = h * l
print(n)
```

DR2 Treatment

```
f = 6
f = 2
q = 4
// test
assert(q == 4)
// continue
u = 5
x = u - q + f
print(x)
```

DR2 Control

```
f = 6
q = 4
// test
assert(q == 4)
// continue
u = 5
f = 2
x = u - q + f
print(x)
```

RP1 Treatment

```
a = 5
a = 3
t = 2
d = 2
j = t * d
h = j * 4
m = h / d
y = a * 2
print(y)
```

RP1 Control

```
t = 2
d = 2
j = t * d
h = j * 4
m = h / d
a = 5
a = 3
y = a * 2
print(y)
```

RP2 Treatment

```
q = 1
q = 2
p = 5
b = p - 4
d = b * q
print(d)
```

RP2 Control

```
p = 5
b = p - 4
q = 1
q = 2
d = b * q
print(d)
```

Figure A.1: Code Snippets Used in the Study

Statement on the Usage of Generative Digital Assistants

For this thesis, the following generative digital assistant has been used: We have used DEEP-L WRITE[1] for text rewriting. Additionally, we used ChatGPT 4o[2] to generate code for the analysis of the data. We are aware of the potential dangers of using these tools and have used them sensibly with caution and with critical thinking.

We used Deep-L Write to rewrite parts of the text to improve the readability and to ensure that the text is free of grammatical errors. We only used Deep-L Write for rewriting and did not use it to generate new content. It was used especially in the background section to improve the readability and accuracy of the text.

For ChatGPT, we used it to generate code snippets for the analysis of the data. We used the generated code snippets as a starting point and modified them to fit our needs. We did not blindly trust the output of the generative digital assistants and always checked the output for correctness and reliability.

1 https://www.deepl.com/de/write
2 https://openai.com/chatgpt

Bibliography

[1] Herman Aguinis, Isabel Villamor, and Ravi S Ramani. "MTurk research: Review and recommendations." In: *Journal of Management* 47.4 (2021), pp. 823–837.

[2] Mohammad Alshayeb. "Empirical investigation of refactoring effect on software quality." In: *Information and software technology* 51.9 (2009), pp. 1319–1326.

[3] John R Anderson. "ACT: A simple theory of complex cognition." In: *American psychologist* 51.4 (1996), p. 355.

[4] John R Anderson, Daniel Bothell, Michael D Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. "An integrated theory of the mind." In: *Psychological review* 111.4 (2004), p. 1036.

[5] John Robert Anderson. *How can the human mind occur in the physical universe?* Vol. 3. Oxford University Press, 2007.

[6] Michael W Browne. "Cross-validation methods." In: *Journal of mathematical psychology* 44.1 (2000), pp. 108–132.

[7] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. "Eye movements in code reading: Relaxing the linear order." In: *2015 ieee 23rd international conference on program comprehension*. IEEE. 2015, pp. 255–265.

[8] Robert J Cabin and Randall J Mitchell. "To Bonferroni or not to Bonferroni: when and how are the questions." In: *Bulletin of the ecological society of America* 81.3 (2000), pp. 246–248.

[9] Elizabeth Charters. "The use of think-aloud methods in qualitative research an introduction to think-aloud methods." In: *Brock Education Journal* 12.2 (2003).

[10] Christian Closheim. "Cognitive Modeling in Code Comprehension: An Empirical Study of Short-Term Memory Retrievals." Masterthesis. Germany: Saarland University, 2024.

[11] Daniela S Cruzes and Tore Dyba. "Recommended steps for thematic synthesis in software engineering." In: *2011 international symposium on empirical software engineering and measurement*. IEEE. 2011, pp. 275–284.

[12] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. "Do you really code? designing and evaluating screening questions for online surveys with programmers." In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 537–548.

[13] Pedro Domingos. "A few useful things to know about machine learning." In: *Communications of the ACM* 55.10 (2012), pp. 78–87.

[14] Dror G. Feitelson. "Considerations and Pitfalls in Controlled Experiments on Code Comprehension." In: *CoRR* abs/2103.08769 (2021). arXiv: 2103.08769. URL: https://arxiv.org/abs/2103.08769.

[15] Anett Fekete and Zoltán Porkoláb. "A comprehensive review on software comprehension models." In: *Annales Mathematicae et Informaticae*. Vol. 51. Liceum University Press. 2020, pp. 103–111.

[16] Jawad Golzar, Shagofah Noor, and Omid Tajik. "Convenience sampling." In: *International Journal of Education & Language Studies* 1.2 (2022), pp. 72–77.

[17] Dan Gopstein, Jake Iannacone, Yu Yan, Lois DeLong, Yanyan Zhuang, Martin K.-C. Yeh, and Justin Cappos. "Understanding misunderstandings in source code." In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2017. Paderborn, Germany: Association for Computing Machinery, 2017, pp. 129–139. ISBN: 9781450351058. DOI: 10.1145/3106237.3106264. URL: https://doi.org/10.1145/3106237.3106264.

[18] ACT-R Research Group. *ACT-R 7.26+ Reference Manual*. Reference Manual. 2021. URL: http://act-r.psy.cmu.edu/actr7.x/reference-manual.pdf.

[19] Michael E Hansen, Andrew Lumsdaine, and Robert L Goldstone. "Cognitive architectures: A way forward for the psychology of programming." In: *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*. 2012, pp. 27–38.

[20] David J Hauser, Aaron J Moss, Cheskie Rosenzweig, Shalom N Jaffe, Jonathan Robinson, and Leib Litman. "Evaluating CloudResearch's Approved Group as a solution for problematic data quality on MTurk." In: *Behavior Research Methods* 55.8 (2023), pp. 3953–3964.

[21] Ahmad Jbara and Dror G Feitelson. "How programmers read regular code: a controlled experiment using eye tracking." In: *Empirical software engineering* 22 (2017), pp. 1440–1477.

[22] Antti Kangasrääsiö, Jussi PP Jokinen, Antti Oulasvirta, Andrew Howes, and Samuel Kaski. "Parameter inference for computational cognitive models with approximate Bayesian computation." In: *Cognitive science* 43.6 (2019), e12738.

[23] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. "An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks." In: *Proceedings of the 2006 ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA '06. Portland, Maine, USA: Association for Computing Machinery, 2006, pp. 137–148. ISBN: 1595933751. DOI: 10.1145/1134285.1134328. URL: https://doi.org/10.1145/1134285.1134328.

[24] Roberto Minelli, Andrea Mocci, and Michele Lanza. "I know what you did last summer-an investigation of how developers spend their time." In: *2015 IEEE 23rd international conference on program comprehension*. IEEE. 2015, pp. 25–35.

[25] Shane T. Mueller and Christoph T. Weidemann. "Alphabetic letter identification: Effects of perceivability, similarity, and bias." In: *Acta Psychologica* 139.1 (2012), pp. 19–37. ISSN: 0001-6918. DOI: https://doi.org/10.1016/j.actpsy.2011.09.014. URL: https://www.sciencedirect.com/science/article/pii/S0001691811001843.

[26] Marvin Muñoz Barón, Marvin Wyrich, and Stefan Wagner. "An empirical validation of cognitive complexity as a measure of source code understandability." In: *Proceedings of the 14th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*. 2020, pp. 1–12.

[27] Mayra Nilson, Vard Antinyan, and Lucas Gren. "Do internal software quality tools measure validated metrics?" In: *Product-Focused Software Process Improvement: 20th International Conference, PROFES 2019, Barcelona, Spain, November 27–29, 2019, Proceedings 20*. Springer. 2019, pp. 637–648.

[28] Dieter Rasch and Volker Guiard. "The robustness of parametric statistical methods." In: *Psychology Science* 46 (2004), pp. 175–208.

[29] Keith Rayner. "Eye movements in reading and information processing: 20 years of research." In: *Psychological Bulletin* 124.3 (1998), pp. 372–422.

[30] Daniel Russo. *Recruiting Software Engineers on Prolific*. 2022. arXiv: 2203.14695 [cs.SE]. URL: https://arxiv.org/abs/2203.14695.

[31] Simone Scalabrino, Gabriele Bavota, Christopher Vendome, Mario Linares-Vasquez, Denys Poshyvanyk, and Rocco Oliveto. "Automatically assessing code understandability." In: *IEEE Transactions on Software Engineering* 47.3 (2019), pp. 595–613.

[32] Ian C Simpson, Petroula Mousikou, Juan Manuel Montoya, and Sylvia Defior. "A letter visual-similarity matrix for Latin-based alphabets." In: *Behavior research methods* 45 (2013), pp. 431–439.

[33] Niels Taatgen and Hedderik van Rijn. "Nice graphs, good R2, but still a poor fit? how to be more sure your model explains your data." In: *Proceedings of the 2010 International Conference on Cognitive Modeling*. Citeseer. 2010, pp. 247–252.

[34] Niels A Taatgen and John R Anderson. "Constraints in cognitive architectures." In: *Cambridge handbook of computational psychology* (2008), pp. 170–185.

[35] Rebecca Treiman, Brett Kessler, and Tatiana Cury Pollo. "Learning about the letter name subset of the vocabulary: Evidence from US and Brazilian preschoolers." In: *Applied Psycholinguistics* 27.2 (2006), pp. 211–227.

[36] Anne M Treisman and Garry Gelade. "A feature-integration theory of attention." In: *Cognitive psychology* 12.1 (1980), pp. 97–136.

[37] Maarten Van Someren, Yvonne F Barnard, and J Sandberg. "The think aloud method: a practical approach to modelling cognitive." In: *London: AcademicPress* 11.6 (1994).

[38] Anneliese Von Mayrhauser and A Marie Vans. "Program comprehension during software maintenance and evolution." In: *Computer* 28.8 (1995), pp. 44–55.

[39] Rand R Wilcox. *Introduction to robust estimation and hypothesis testing*. Academic press, 2011.

[40] Marvin Wyrich. *Source Code Comprehension: A Contemporary Definition and Conceptual Model for Empirical Investigation*. 2023. arXiv: 2310.11301 [cs.SE]. URL: https://arxiv.org/abs/2310.11301.

[41] Marvin Wyrich, Justus Bogner, and Stefan Wagner. "40 years of designing code comprehension experiments: A systematic mapping study." In: *ACM Computing Surveys* 56.4 (2023), pp. 1–42.

[42] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E Hassan, and Shanping Li. "Measuring program comprehension: A large-scale field study with professionals." In: *IEEE Transactions on Software Engineering* 44.10 (2017), pp. 951–976.