Master's Thesis

Cognitive Modeling in Code Comprehension: An Empirical Study of Short-Term Memory Retrievals

Christian Closheim

August 27, 2024

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 Linguistics

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

UNIVERSITÄT
DES
SAARLANDES

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

There are many theoretical models of how code comprehension cognition works. But these are all more or less abstract and difficult to apply for someone who has not dealt with them intensively. For this reason, the idea of a simulatable model in which code can be entered and then analyzed for complexity came up in the academic literature. Such a model could help, for example, to either reduce the complexity of the code or offer help to comprehend it better where necessary. But although there were many calls and announcements for such a cognitive model, there still exists no concrete implementation of one.

In this work, we addressed this task by developing an ACT-R model capable of predicting line processing time and the probability of comprehension errors.

We evaluated this model using eight different pairs of code snippets, for which we expected a difference in time or error rate, to test if the model is able to replicate these differences. To obtain data to test the model against and to evaluate possible differences in comprehension of differently formatted code, the code snippets were completed by 63 participants in an online study using a randomized cross-subject design. Based on the empirical data, we found no significant differences between the respective formatting of the code snippets, neither in terms of processing time nor error rate.

Despite the lack of significant differences, we were able to evaluate the model using the empirical data. Using Bayesian Optimization, the model's hyperparameters were optimized. Through successive expansion of the hyperparameter space, we ultimately identified three hyperparameters that influence the model. The model with optimized hyperparameters was able to largely replicate the empirical results, performing better in predicting time than in predicting errors.

This work aims to serve as a foundation for advancing the topic of a simulative model for code comprehension. We have demonstrated that it is feasible to represent simple tasks in ACT-R. The next steps should involve extending the model. Potential improvements include expanding the range of errors the model can produce or integrating a model for eye movement. This work highlights much potential for improvements in both the model itself and the evaluation methodology.

Contents

List of Figures

List of Tables

Listings

Acronyms

ACT-R Adaptive Control of Thought-Rational

ans activation noise s

BH Bonferoni-Holm

BOLD Blood-oxygenation-level dependent

BO Bayes Optimization

CCM Cognitive Complexity Metric

CD Code Distance

CR Repeated Code

DR Declaration Redeclaration Distance

KS Kolmogorov-Smirnov

lf latency factor
LLM Large Language Model
RP Repeated Distance
rt retrieval threshold
UI User Interface

1 Introduction

1.1 Goal of this Thesis

A common introduction to the topic of code comprehension is to emphasize that this practice makes up a large part of developers' daily routines and should therefore be the focus of research. But let us take a step back and look at the developer as a production unit that produces code. As with all production processes, there are quality requirements, specified standards and resources. It is not economical to waste unnecessary resources and time. If resources are scarce, quality and quantity are usually the first to suffer. Therefore they should be used efficiently and purposefully.

A fundamental resource that every developer brings with them is their mental ability. It is therefore important not to burden their mental ability with unnecessary cognitive work and to use it effectively. But the question remains what exactly burdens cognitive processes the most and what could be changed most easily. The topic of mental models deals with this question.

Since near the beginning of researching code comprehension, the idea of mental models goes along with this topic [6]. While most mental models use a more theoretical approach to support think-aloud experiments [7, 37, 44, 46], at the end of the 80s and beginning 90s the idea came up to build a calculable mental model [8, 50]. The advantage of such a calculable model is that not only the sequence of different mental processes could be simulated, but also the time they take, which would create a measurable variable. Depending on its complexity, such a model would allow various mental processes of the programmer to be simulated in advance and the mental load to be measured. Although this would not replace experiments on human subjects, research resources could be used more efficiently and the model could be continuously improved and evaluated at the same time.

Unfortunately, there have only been several announcements about developing a simulatable mental model until now. The most concrete proposal was made by Hansen et al. [28] in 2012, in which they proposed an implementation in Adaptive Control of Thought-Rational (ACT-R) based on the Cognitive Complexity Metric (CCM) [8] and showed how the various parameters of the CCM could be implemented by the modules in ACT-R. However, a concrete model has not been published since then.

This situation with many theories and ideas for a cognitive model of code comprehension forms the basis of our work. A fundamental problem common to all proposed models is their overly general approach, attempting to cover the entire complex area of code comprehension and produce a generalist model. Instead of developing an entire model, which would go beyond the scope of this thesis in every respect, the aim is to evaluate whether the mental processes during a simple bottom-up code comprehension can be simulated by ACT-R and whether it might prove worthwhile to pursue this approach further. Therefore, the focus of this work lies on the mental execution of code.

1.2 Overview

In Chapter 2, we explain and elucidate the key terms and concepts necessary to understand this thesis. Chapter 3 follows with a description of the entire workflow of this thesis, encompassing the development of the model, the execution of experiments, and the evaluation of results.

These results are presented, contextualized, and discussed in Chapter 4. Chapter 5 provides an overview of the most significant related works that address similar topics. Finally, Chapter 6 offers a summary of all findings and provides an outlook on how future research can build upon this work.

2

Background

In this section, all relevant core information that is needed to understand this thesis will be explained. First, there is an introduction to the cognitive architecture ACT-R. This is followed by an explanation of what Source Code Comprehension is.

ACT-R

ACT-R is a cognitive architecture built on top of LISP [26]. The theory and later the executable architecture were developed in the 1990s with a large contribution by John R. Anderson. A cognitive architecture is an abstraction of human cognition from the real-world low-level details, like neuronal activation, to higher-level functions of the mind [2]. In contrast to neural networks, a cognitive architecture is based on a collection of defined if-then rules [28], which makes interpretation easier compared to neural networks, which are based on complicated activation functions between the individual layers and neurons. A cognitive architecture is also capable of learning [26], but the path from an input to the result always remains traceable, which makes this form particularly interesting for the analysis of behaviour if not only the results, but also the reasoning behind them is of interest.

ACT-R is a widely used architecture developed by cognitive scientists [26]. It has some additional benefits over other cognitive architectures. For example, it contains perception and motor modules that can be used to simulate interaction with the environment, which makes the setup more realistic. Blood-oxygenation-level dependent (BOLD) measurements can also be simulated, which makes this architecture particularly interesting for cognitive research in the context of fMRI studies.

The ACT-R architecture is divided into eight modules, visualized in Figure 2.1 [28]. Each module represents a specific cognitive function, which are also physically separated in the brain. The outer modules cannot communicate directly with each other but only via the procedural module.

The information unit for communicating between the individual modules is a chunk. A chunk essentially consists of a collection of variables with values called slots. Chunks can be declared in advance and structured hierarchically. Each module contains a buffer that can store a chunk, regardless of the complexity of the chunk. In addition, the buffer itself can be in different states, such as failure, empty or full. The current state of the overall system is

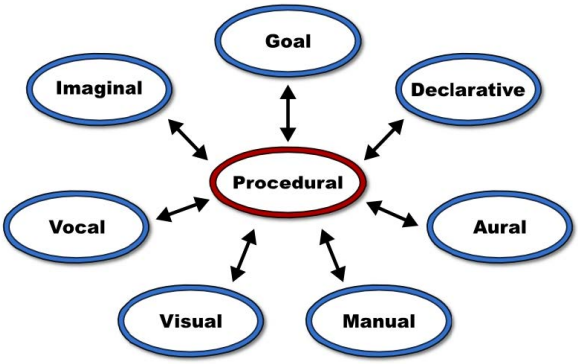


Figure 2.1: ACT-R 6.0 modules [28].

described by the current states and chunk assignment of all buffers.

This is an implemented analogy to human cognition. In human memory, we essentially distinguish between short-term and long-term memory, with a fluid transition between them, and between declarative memory and procedural memory. Declarative memory represents all the information we have about the world, while procedural memory is a collection of procedures, both physical and mental, for interacting with the environment [54].

The concept of a chunk as a unit of information is also derived from this analogy, as our units of knowledge are not always uniformly structured and can become more complex with practice. For instance, chess masters can quickly memorize the positions of all pieces in a game if they follow a logical flow of play, whereas beginners cannot. However, both groups perform equally poorly with random arrangements. This illustrates that chunks can process a lot of information in the right context and very little in others [4].
The same applies to procedures. A skilled juggler no longer focuses on the position of their hands or each ball individually; the movement flows smoothly. A beginner, however, must concentrate on many processes simultaneously before these movements can be integrated into a higher-level overall concept.

These characteristics—the separation of short-term and long-term memory, the distinction between declarative and procedural knowledge, and the hierarchical organization of knowledge into chunks—are reflected in ACT-R [1, 3]. The following will contain a closer look at the individual modules that are relevant for the following work.

The current state of the overall system is changed via productions held in the procedural module. These are pattern-matching rules that check the current state of the system and fire if matching. As declared in the production rule, new chunks are written into the buffers and the state is thus changed. If multiple productions match, the one with the highest utility is chosen. If there are still multiple matchings, the selection is random. The utility is predefined and can be changed if one enables utility learning. However, since the execution logic in the cognitive model used for this proposal itself is deterministic, there is no need for this. Therefore, there are no tie-breaks between multiple productions. This process runs

either until no more production rules match and the simulation is terminated or until a timeout expires.

The visual module includes both the control of the visual focus and the processing of the content. In the created cognitive model, the visual module is used for the control of the visual focus to read a line element by element and to process the read characters. At this point, it should be noted that the implementation automatically bundles characters into words and numbers so that a number can be read as a whole.

The manual module includes the output of keystrokes on the keyboard or movements with the mouse. These are transmitted as a command and executed by the manual module. The time required for certain movements is also taken into account here. In the created cognitive model, only keyboard actions are incorporated.

The declarative module is the access to long-term memory of the cognitive architecture. It works via the retrieval buffer, which can request chunks from the declarative memory using pattern matching of requested slots. This is where the subsymbolic level comes into play: Several chunks from the declarative memory can correspond to the requested pattern, but only one can be loaded.

$$A_i = B_i + \epsilon_i \tag{2.1}$$

To decide which one to load, the activation of each chunk is first calculated as shown in Equation 2.1. This consists of the base level activation B_i and noise ϵ_i. There is an additional context component, but since it is not used, it is not needed to understand the following outlines.

$$B_i = ln\left(\frac{n}{1-d}\right) - d * ln(L) \tag{2.2}$$

The base level of each chunk is calculated as shown in Equation 2.2. This takes into account how often the chunk has already been presented (n), how long the chunk has already been in memory (L) and what the decay parameter is (d).
The noise is calculated as a logistic distribution and has an additional parameter (s) of its own.

$$p_i = 1 + e^{\frac{1}{\frac{\tau - A_i}{s}}} \tag{2.3}$$

$$T_i = Fe^{-A} \tag{2.4}$$

This activation can now be used to calculate both the probability p_i and the time T_i that the declarative module needs to provide a certain chunk. These calculations are shown in Equation 2.3 and 2.4 and use the retrieval threshold τ. This value is a limit that the activation must at least exceed to have a chance to be retrieved. The latency factor F serves as the basis for calculating the time.

$$T_f = Fe^{-\tau} \tag{2.5}$$

One should choose the retrieval threshold carefully because if no chunk fulfils the required conditions, the time the declarative module needs to report back is calculated from the threshold hyperparameter, as shown in Equation 2.5. If the threshold is set too low, the module also needs a very long time to report failed accesses.

The imaginal module in turn is used to create new chunks, store them in the working memory and, when you have finished using them, store them in the declarative memory. All chunks that are stored in the imaginal buffer are automatically written to the declarative memory when the buffer is cleared.

The goal module is used to control the work process by using chunks to determine which work step is to be carried out and defining and controling sequences of certain actions.

Source Code Comprehension

Wyrich [56] defines:

> "Source code comprehension describes a person's intentional act and degree of accomplishment in inferring the meaning of source code." [56]

It should be noted that the degree of accomplishment is a continuous spectrum, so one cannot simply define if someone has understood code with a yes or no. In addition, the degree of accomplishment is always dependent on the use case and the underlying goals of why the source code should be understood. It differs depending on whether one is debugging, trying to improve, or aiming to expand the code.

The meaning of the source code can be divided into three dimensions, namely the functional level, what the code does exactly, the specification level, what the code should do, and the context level, what the idea for the code was originally. These three dimensions may overlap, but might also be very different from each other, precisely because code development is a dynamic process. In this thesis we utilized time and error rate as a measurement for source code comprehension.

3

Methodology

This chapter describes the methodology of the thesis core evaluation.

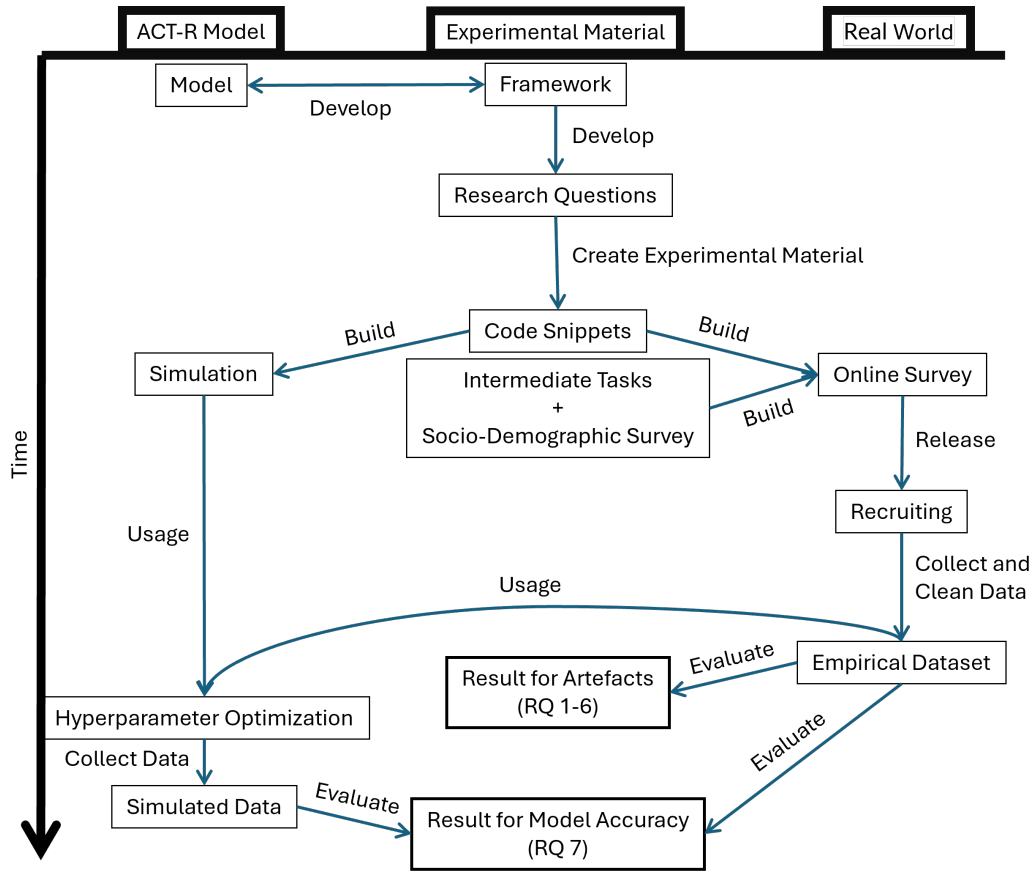


Figure 3.1: Schematic representation of the research methology.

In Figure 3.1 the methodological workflow of this study is visualized. We began with the parallel development of our model and a corresponding framework for conducting our experiments. Special attention was given to ensuring that both the model and human participants could use the framework similarly.

Next, we identified intriguing effects that could be investigated using the available framework. These effects led to the formulation of the seven research questions addressed in this

study.

With these research questions we developed the research materials. We created a set of eight code snippet pairs, each with a treatment and a control format. These pairs were used to set up both the online survey for participants and the simulation. Additionally, human participants received intermediate tasks (as their memory cannot be reset between tasks like a cognitive model) and completed a socio-demographic questionnaire. The development phase concluded with test runs of the simulation and online questionnaire to identify and correct any errors.

During the experimentation phase, we released the online survey and recruited participants through various channels. Data were collected and subsequently cleaned, resulting in the empirical dataset for further analysis.
This dataset was used to tune the model's hyperparameters using Bayes Optimization (BO). Multiple datasets were simulated and evaluated, and the hyperparameters were iteratively refined, yielding a final simulated dataset for the final evaluation.

In the evaluation phase, we first examined the empirical dataset for significant differences in processing time and error rate between the treatment and control groups of the code snippets. We then assessed how well the simulated dataset matched the empirical dataset, in terms of both time and error rate, as well as the observed effects and their replication with similar effect sizes.

All these steps are detailed in the following sections, following the temporal order described here. The chapter concludes by positioning this study design within the conceptual framework of Wyrich [56].

3.1 Cognitive Model for Code Comprehension

In this section, we first explain the experimental framework with which the ACT-R model can operate. Subsequently, we describe how the model functions and was developed.

3.1.1 Framework

The interface used for this study was the ACT-R User Interface (UI) framework, facilitating graphic output display and recognizing diverse inputs, including presenting text and recognizing key pressures. This allowed us to use almost the same framework for the model simulation as well as for the participants. The code snippets were presented line by line, navigated with the space bar, and mimicked a line-by-line execution to enforce a simplified reading behaviour. While this framework deviates from real-world programming practices, it allows for a detailed analysis of time spent per line. At the end, a print statement was shown and the result had to be typed in and the space bar had to be pressed finally.

The code snippets comprised only variables and simple arithmetic operations, which means adding, subtracting, multiplying and dividing on integers in the range from minus 20 to plus 20 for all operands and results. The result of arithmetic operations could also be applied to variables.

Variables could be assigned new values several times. Several operations were also possible in one line, although point-before-dash calculations were not taken into account, since a simple left-to-right reading behaviour was used.

This framework focused on the purely mechanical execution of code and did not correspond to the real behaviour of a developer outside this setting, but it allowed us to control and track the reading order and speed. In addition, the test subjects were forced to remember the values of the variables because jumping back to a previous line was impossible for them.

Having to remember the variables could cost the participants some time. In a real-world setting, they might tend to look up the variables in previous lines instead, but this would also cost them time. Quick recall from memory is therefore more advantageous. This framework was a strong simplification of reality, but observable effects should also be relevant outside this setting.

3.1.2 ACT-R Model

The model created for this thesis is inspired by Danker and Anderson [16] and Lebiere [35], who studied simple arithmetic performance. Similar to their model, mathematical facts were stored directly in declarative memory. The range was set at minus 20 to plus 20. Included were the mathematical facts for all four basic arithmetic operations, for which all operands and results lie within this range. This allowed us to focus on the code execution process and we did not have to deal with all the arithmetic mental processes. Since the aim of this work is to investigate code comprehension and not mental arithmetic speed, this is a justifiable simplification. As Marewski and Mehlhorn [40] noted, every model is wrong in its own way because it is a simplification of reality.

In addition, only integers were taken into account for division and decimal signs were cut of. Where this model went beyond other models is that longer terms could also be calculated here, with the restriction that no point before dash calculation was taken into account. The formula was only evaluated from left to right. In addition, the model could also handle variables, calculate with them, and assign and overwrite values. Variables were stored as entries in the declarative memory.

Differing from models that only focus on arithmetic tasks, this model also integrated the reading of a line of code and the typing of results on the keyboard to closely mimic human behaviour. The model processed the string independently without any external parser and pressed the space bar at the end of each line, whereupon a new line could be presented. If

Parameter	Name	Default
rt	retrieval threshold	0
ans	activation noise s	0.2-0.8
lf	latency factor	1
le	latency exponent	1
bll	base level learning	0.5

Table 3.1: Default values for hyperparameters [26].

the model recognized a print command, it evaluated the expression in the print command to a value and pressed the corresponding keys for the result followed by the space bar. The key sequence for all numbers was stored in the declarative memory. For the sake of simplicity, it was assumed that the sequence of keys for the numbers between minus 20 and plus 20 was known as a single fact and did not need to be broken down into individual components.

If values of variables were overwritten, a new entry was created in the memory, between which a distinction must be made when retrieving. At the same time, people also need different amounts of time to remember the value of variables. To account for this, a feature was activated that returns chunks from the declarative memory based on activation, which is a stochastic procedure. Therefore, in experiments, several runs were necessary to obtain stable mean values.

The model had several hyperparameters, listed in Table 3.1, with which it can be fine-tuned to the results found from the experiments with human subjects. These hyperparameters included retrieval threshold (rt), activation noise s (ans) and latency factor (lf) which were relevant for calculating the recall probability in equation 2.3 and the recall time in equation 2.4.

3.2 Testing Tasks

To test the accuracy of the cognitive model, it required stimulus material that could be processed by both the cognitive model and the participants. Using only time as a criterium for evaluating the cognitive model falls short. For example, Gopstein et al. [25] were able to show that different syntax forms, although they were semantically identical, can lead to errors in interpretation by humans. They called the list of these code snippets that significantly lead to a higher error rate "atoms of confusion". Unfortunately, the 15 "atoms of confusion" which demonstrated a statistically significant effect, are not compatible with our reduced framework. But we were inspired by the atoms of confusion and constructed four forms of artefacts that also take up effects from linguistics and deal more specifically with the cognitive processes and can be described with ACT-R. This allowed the two dimensions of time and error rate to be examined simultaneously, forming a better overall picture. These

four artefacts are explained below regarding their origin and possible influence on code understanding.

3.2.1 Repeated Code

Gopstein et al. [25] analysed repeated code as a candidate for atoms of confusion, focusing on scenarios where recently declared variables are reassigned with new values.

```
a = 1
a = 3
a = 2
print(a)
```

(a) Original.

```
a = 1
a = 2
print(a)
```

(b) Refactoring.

Listing 3.1: Example for repeated code adapted from Gopstein et al. [25].

The original study did exclude the effects of repeated, unreachable, and dead code in the list of "atoms of confusion" with a significance level of 0.059. This effect should be re-examined since the original paper only deals with error rate, but not the time required for comprehension or cognitive load introduced by repeated code. An example of how a repeated code example could look is shown in Listing 3.1. This is an original example from the study by Gopstein et al. [25].

In his report, Dörzapf [18] examined 31 GitHub repositories for the use of double declarations without a usage in between. He came up with a total number of over 60 thousand cases with an average distance of over six lines. While it should be noted that this also includes various conditionals and loops, this number nevertheless illustrates the frequency of these code patterns. A simple example is a first declaration of a variable with null, which later gets overwritten with other assignments. These examples once again illustrate the relevance of this topic in practice.

3.2.2 Declaration near Usage

So far, no one in literature has analysed the concept of declaring variables near their first usage in the context of source code comprehension. While refactoring tools for different languages include this refactoring technique [17], there are discussions about the potential benefits and risks of this refactoring, particularly in some edge cases [32]. But while the risks of refactoring can be well described, such as the changed code behaviour, there exists no study of its benefits, making it difficult to weigh up the pros and cons.

Dörzapf [18] also examined the distance between declaration and first usage in the 31 GitHub repositories. For a total number of over one million declarations, the average distance from declaration to usage was 3.7 lines, with a median of one. It should be noted that the number of lines is not the same as the number of statements, as longer statements can also span several lines. However, the report also lists some examples where it would

have been possible to change the order so that the declaration would be closer to the first usage. These examples show that the topic of declaring close to the usage also has practical relevance.

In research on language comprehension, there is a long-known related effect: The speed at which pronouns can be assigned depends on the distance between the pronoun and its antecedent, which is the word the pronoun refers to [11, 15, 20, 23]. In addition, Li et al. [39] have recently investigated this effect, using an ACT-R model among other things, which describes the strength of the bond between a pronoun and its antecedent through the temporal distance in spoken language. It could therefore be possible that this effect is also transferable to the pairing of value and variable names. Therefore, this will be exploratory investigated in this thesis.

```
a = 1
b = 2
c = b + 2
d = c + a
print(d)
```

(a) Original.

```
b = 2
c = b + 2
a = 1
d = c + a
print(d)
```

(b) Refactoring.

Listing 3.2: Example for declaration near usage.

Listing 3.2 shows examples of what code snippets could look like for this effect specifically. Given the existing knowledge gap regarding whether this refactoring introduces more risk or provides added value for code comprehension, the investigation was aimed at addressing this uncertainty.

3.2.3 Interactions

Interactions may arise between the effects of repeated code and declaration near usage, presenting additional considerations. As the literature on the main effects is already very limited, this situation is naturally exacerbated when it comes to interactions. A search on interactions for code understanding of repeated code and declaration near usage was not successful.

The possible combinations of interactions might be bigger, but at this point, the focus will be limited to two possible interactions that could be relevant.

3.2.3.1 *Declaration Redeclaration Distance*

The first type of interaction is the variation of the distance between a declaration and its redeclaration. A possible code snippet is visualized in Listing 3.3. While increasing the distance, it must be noted that there are two possibilities concerning the first usage. If you keep the distance to the first usage immediately after the redeclaration, the first declaration must be moved further up. If, on the other hand, you want to keep the total length the same,

```
a = 1
a = 3
b = 2
c = b + 2
d = c + a
print(d)
```

(a) Original.

```
a = 1
b = 2
c = b + 2
a = 3
d = c + a
print(d)
```

(b) Refactoring.

Listing 3.3: Example for the interaction of distance between the declaration and redeclaration.

you will inevitably have to reduce the distance between the redeclaration and the first usage. For our very tightly controlled framework, we have opted for the second variant, so that the code snippets are no longer than absolutely necessary.

3.2.3.2 *Repeated Distance*

```
a = 1
a = 3
b = 2
c = b + 2
d = c + a
print(d)
```

(a) Original.

```
b = 2
c = b + 2
a = 1
a = 3
d = c + a
print(d)
```

(b) Refactoring.

Listing 3.4: Example for the interaction of repeated code near usage.

The second type of interaction we have dealt with is the distance between a double declaration and the first usage. This is shown in Listing 3.4. The position of the double declaration is of interest as to whether the effect of repeated code is greater or smaller due to the closeness to the first usage.

3.3 Research Questions

This thesis aims to answer two groups of research questions. The first group deals with the question of whether effects can be found with the artefacts for human participants. The second group is concerned with whether the data collected can be explained by the cognitive model. Both groups are discussed below.

3.3.1 Artefacts Effects

As we could only find one single artefact compatible with our framework in the literature, we constructed others ourselves. First, we must therefore ask what effects these artefacts have and whether they are significant. In doing so, we must distinguish between the two

dimensions of time and error rate. The following questions therefore arise for the two main effects:

RQ1: Does Repeated Code increase the time needed to understand the code?

RQ2: Does Repeated Code lead to a higher probability of misunderstanding the code?

RQ3: Does an increased distance between declaration and usage result in a longer time required for understanding the code?

RQ4: Does an increased distance between declaration and usage lead to a higher probability of misunderstanding the code?

In addition, we want to look at the interactions to see whether they have a moderating effect on the two main effects, both in terms of time and the probabilities of mistakes. Hence the two additional questions:

RQ5: Does an increased distance between repeated code and usage amplify the effect of repeated code?

RQ6: Does an increased distance between declaration and redeclaration mitigate the effect of repeated code?

3.3.2 Cognitive Model Explanation

The main goal of this thesis is to develop and evaluate a cognitive model for code comprehension. Therefore, all previous research questions serve mostly as a tool to answer this last research question, ideally with more methodological means:

RQ7: How accurately does the ACT-R model reflect subject's behaviour in terms of processing time and error rate?

The effects found can be used to analyze whether they are simulated by the cognitive model. But even if no significant effects can be found, which may also be due to reasons such as insufficient test power, the model should be able to generate no significant difference for the same number of simulated data, but still produce data that is as similar as possible.

3.4 Study Material

To answer the research questions, we developed research material that could be processed equally by the cognitive model and human subjects. For this purpose, eight pairs of code snippets were constructed, ensuring compatibility with both ACT-R and human processing.

Each participant was required to complete eight tasks, with each task consisting of a code snippet. Participants received either a treatment or control snippet depending on their

assigned group. Since the memory of human subjects cannot be reset like in ACT-R, an intermediate task was included between each primary task to mitigate the influence of the previous task on the subsequent one. The creation of the individual tasks is explained in detail below.

Additionally, to identify individual differences in the strength of the effects among different groups of people, further metadata was collected from the human subjects. A questionnaire was constructed to capture the most relevant socio-demographic data that could influence code comprehension ability.

To facilitate access to the tasks, we opted for an online questionnaire. The questionnaire was generated using SoSci Survey [36] and made available to participants via www.soscisurvey.de.

3.4.1 Code Snippets

The research questions proposed above were examined using code snippets. We tried to reuse some from previous studies or collected from GitHub, but the framework restrictions were to tight to use real-world code snippets. Therefore, we created the code snippets artificially. We set ourselves several goals and requirements for the creation of the code snippets. The snippets should not have more than four different variables in order not to be too difficult but also not too easy [13, 14]. The variables themselves should have random names to avoid any sequence effects [21]. The code snippets were created with variables, which were later replaced by random letters, thus ensuring purely random naming. Only whole numbers should occur during the calculation. If someone made a mistake and a decimal number appeared, they were instructed to cut off the decimal places. The code snippets were created by hand according to the criteria of the artefacts, which are explained below. To generate control snippets, the treatment snippets were refactored such that they will have the same result while using different code.

The number of code snippets used for the experiment depends on the execution time. An average execution time of 15 minutes was planned for the test subjects. The time should be distributed equally for all effects. After some pilot experiments, we ended up with three training and eight experiment code snippets with an expected execution time of 15 up to 20 minutes for the whole experiment, including intermediate tasks and the socio-demographic survey.

For the ACT-R simulation we used the integrated UI to present the code line by line inside the virtual window and register the key pressures for the print statements. We had to variate the task a bit for the online survey and used Lab.js by Henninger et al. [30] for it, since there is a predefined plugin to use it inside SoSci. The main difference was the integration of an input field after the last line to enter the result of the print statement of the page before. This change was neccessary due to difficulties in reacting directly to keyboard input

within Lab.js. However, as nothing had changed in terms of the relevant times on the lines beforehand, we have accepted this change.

3.4.1.1 *Training Tasks*

```
z = 5
j = 3
a = z * j
print(a)
```

(a) Training 1.

```
q = 6
e = 3
e = 2
m = q / e
u = m + e
print(u)
```

(b) Training 2.

```
q = 8
r = 1
b = 5
d = q - b + r
print(d)
```

(c) Training 3.

Listing 3.5: Training code snippets.

The code snippets for the training tasks are visualized in Listing 3.5. They were presented in this order for each subject. Listing 3.5a was deliberately constructed easier to facilitate a quick understanding of the concept of the tasks. After the first test runs, a double declaration was added to Listing 3.5b, as the double declaration used in the experimental task sometimes caused irritation because it was not part of the training tasks. After each training task, the test subjects received feedback on whether their answer was correct and were presented with the entire code snippet including the expected answer.

3.4.1.2 *Experiment Tasks*

The experimental tasks were divided in four blocks, one per artefact. This resulted in two tasks per block. Each tasks consisted of one treatment code snippet and one control code snippet.

The test subjects were divided into two groups with a mixed subject design. One group performed the treatment code snippet for one task and the control code for the other. The other group had to perform the reverse of this.

Since the number of tasks in succession could lead to both training effects and fatigue effects, the order would actually have to be controlled by randomization. However, since there are $8! = 40320$ possible sequences for the eight tasks and we would not even have covered half a percent of the possible combinations with our expected number of participants of less than 100 subjects, so we decided to proceed differently and determined one random order. We then split the two groups in half and had them perform these sequences either in the forward direction or in the backward direction. This way we expect that fatigue effects and training effects should balance each other out.

In total, this experimental setup comprised four groups of test subjects, divided into treatment and control tasks and forward or backward direction. The exact order of the experimental tasks is visualized in Table 3.2. In the following, the task related code snippets are explained.

Table 3.2: Order of the tasks for the groups.

Order	1T/2C	1C/2T
CD2, RP2, CD1, DR1, DR2, RP1, CR2, CR1	1	2
CR1, CR2, RP1, DR2, DR1, CD1, RP2, CD2	3	4

Annotation: 1T: First Snippet as Treatment, 1C: First Snippet as Control, 2T: Second Snippet as Treatment, 2C: Second Snippet as Control

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

(a) Treatment.

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

(b) Control.

Listing 3.6: Code snippet CD1.

```
x = 1
v = 5
c = 6
i = x * 4
print(i)
```

(a) Treatment.

```
v = 5
c = 6
x = 1
i = x * 4
print(i)
```

(b) Control.

Listing 3.7: Code snippet CD2.

Code Distance For the Code Distance (CD) snippets we used the following schema: Declaration of a variable, two variable declarations in between, a variable declaration that included the first variable followed by the output of the last variable declaration. The control group was created by swapping the first variable with the two other variables. The two code snippets are shown in Listing 3.6 and Listing 3.7.
Since the CD effect might be reduced by the inclusion of several variables, a special difference between the two groups was introduced. As Listing 3.7 shows, we have omitted the use of the variables v and c in the declaration of i.

```
y = 5
y = 3
t = 2
k = y * t
print(k)
```

(a) Treatment.

```
y = 3
t = 2
k = y * t
print(k)
```

(b) Control.

Listing 3.8: Code snippet CR1.

```
m = 5
m = 8
b = 4
a = m / b
print(a)
```

(a) Treatment.

```
m = 8
b = 4
a = m / b
print(a)
```

(b) Control.

Listing 3.9: Code snippet CR2.

Repeated Code When creating the Repeated Code (CR) snippets, we used the following pattern: A double declaration at the beginning was followed by another variable declaration and then both variables were calculated in a third variable which was then output. The control snippet was then created by omitting the first declaration. The results can be seen in Listing 3.8 and Listing 3.9.

```
c = 2
c = 5
q = 8
w = q / 4
k = w * c
print(k)
```

(a) Treatment.

```
c = 2
q = 8
w = q / 4
c = 5
k = w * c
print(k)
```

(b) Control.

Listing 3.10: Code snippet DR1.

```
f = 6
f = 2
q = 4
u = 5
x = u - q + f
print(x)
```

(a) Treatment.

```
f = 6
q = 4
u = 5
f = 2
x = u - q + f
print(x)
```

(b) Control.

Listing 3.11: Code snippet DR2.

Declaration Redeclaration Distance The Declaration Redeclaration Distance (DR) snippets were created using this schema: Double declaration, two other variable declarations and a fourth declaration using the first variable declared. The control was created by changing the order and bringing the redeclaration of the first variable between the declarations of the two other variables and the usage. The used code snippets are visualized in Listing 3.10 and Listing 3.11.

Repeated Distance For the Repeated Distance (RP) snippets we followed the same schema as for DR: Double declaration, two other variable declarations and a fourth declaration using the first variable declared. The difference laid in the control snippet, which was created by changing the order, the declaration, and the redeclaration of the first variable between the

```
t = 5
t = 3
q = 2
h = q * 4
y = h - t
print(y)
```

(a) Treatment.

```
q = 2
h = q * 4
t = 5
t = 3
y = h - t
print(y)
```

(b) Control.

Listing 3.12: Code snippet RP1.

```
q = 1
q = 2
w = 5
y = w - 4
x = y * q
print(x)
```

(a) Treatment.

```
w = 5
y = w - 4
q = 1
q = 2
x = y * q
print(x)
```

(b) Control.

Listing 3.13: Code snippet RP2.

declarations of the two other variables and the usage. The used code snippets are visualized in Listing 3.12 and Listing 3.13.

3.4.2 Intermediate Tasks

To reduce the influence of the tasks on the subsequent tasks, we played ten-second video clips of fish in an aquarium as an intermediate task. This approach was informed by Gee et al. [24], who demonstrated that observing fish has a relaxing effect. The order of the video clips was the same for all participants and it was impossible to skip them. Several forms of intermediate tasks were considered, ranging from attention tests to tic-tac-toe. We decided to use video clips because we did not want to prime the subjects on letters or numbers, and we did not want to further challenge them cognitively to avoid accelerating fatigue. Videos from Pexels [22], which are freely available for non-commercial use, were utilized in our experiments.

3.4.3 Socio-Demographic Survey

For the socio-demographic data, we asked the following questions in the survey after the code comprehension tasks:

3.4.3.1 *Distraction*

To assess the value of the answers given, the subjects were asked whether they could complete the tasks in one go without distractions. This is relevant concerning the cognitively demanding task because distractions could significantly distort the result [19].

3.4.3.2 *Gender Identification*

The primary purpose of the gender identification question was to report the distribution in the data set to evaluate whether the distribution is representative. The question was based on the proposed guidelines for gender-inclusive language in research questions from Northeastern University [51]. In addition to five fixed answer options (man, woman, transman, transwoman, non-binary), there was also an open answer option and a no-answer field.

3.4.3.3 *Age*

We asked for the age in an open question to report the distribution. In addition, age could influence reaction times and processing speeds, making documentation useful for possible posthoc analyses.

3.4.3.4 *Country*

We asked for the current country for demographic data, to be able to report the distribution over different countries.

3.4.3.5 *Education*

The educational level can be a hint and correlate of intelligence, which influences the code comprehension skills [53]. Therefore, we asked for the highest educational level in a multiple choice format, starting from still in school to university degrees.

3.4.3.6 *Job Title*

The effects of the different artefacts could vary between professionals and novices. Therefore we asked for the current job title in an open question.

3.4.3.7 *Programming Experience*

The experience of the subjects can influence the performance of a subject. Therefore, based on the work of Siegmund et al. [45], we surveyed the programming experience. A key facet is the subject's own estimation of their own experience compared to classmates or colleagues. This was assessed using a five-point Likert scale ranging from very inexperienced to very experienced.

3.4.3.8 *Experience with Programming Paradigms*

Another aspect that emerges from the work of Siegmund et al. [45] is the self-assessment of personal experience with logical programming. This is particularly relevant because the code snippets used also belong to this paradigm. In order to not consider the individual paradigm in isolation, the Functional Programming and Object-Oriented Programming paradigms were also surveyed, although these do not discriminate well between the groups of experienced and inexperienced users according to the results of Siegmund et al. [45].

We have nevertheless included these paradigms so as to not to give the impression of incompleteness to the participants by only asking questions about one programming paradigm. These questions were also asked using a five-point Likert scale.

3.5 Design and Testing Phase

During the development of the online test, several reviews were conducted in accordance with the 4-eyes principle to identify potential errors. In a second phase, three testers were asked to complete the questionnaire and identify any comprehension issues. Subsequently, the instructions were revised once more prior to the commencement of the experiments, and the training tasks were modified to incorporate the use of variable redeclaration. Finally, the complete test was subjected to a comprehensive review by the Ethics Council of the Faculty of Mathematics and Computer Science at Saarland University. As a modification, it was recommended that all questions in the socio-demographic questionnaire should permit an explicit choice to refuse to answer. We added this option to all questions. Subsequently, the online test was approved.

3.6 Data Mining

To answer the research questions we collected data from human participants as well as from simulations with our ACT-R model. In the following the data collection process, the data preparation, and the simulation process is described in detail.

3.6.1 Collection

For sampling participants for the online study, we used a referral-chain sampling strategy [5]. We acknowledge that this method does not lead to a representative sample and thus limits the generalizability of the results. Nonetheless, we chose this method for its simplicity and to quickly gather a sufficient number of participants, as the primary focus of this work lay on the feasibility study rather than the generalizability of the results. To obtain a diverse group, we used multiple recruitment channels.
Participants were recruited through personal connections, online forums like Arduino-Forum, Code-Forum, Python-Forum, Java-Forum, using survey exchange platforms like survey swap and pollpool, and finally recruiting students on the campus of Saarland University.

The data to answer the first six research questions was collected using SoSci Survey [36]. The link to the online survey was distributed using the channels described above. In order to control as many independent variables as possible, the aim was to use a mixed-subject design. This is a combination of an inter-subject design and a between-subject design.
As explained in Section 3.4.1, the code snippets appeared in pairs. Each group received either the treatment or the control snippet for the between-subjects part. This process guar-

anteed that both groups processed all effects without having directly replicated code, which prevents a learning effect.
The inter-subject part was implemented by looking at multiple code snippets for each effect, so that each subject saw a treatment and a control snippet for each artefact, but not from the same snippet. This enables us to better recognize individual differences and reduces the probability that the observed effects only occurred due to group membership.

How the experimental tasks were assigned to the groups was explained in detail in Section 3.4.1.2 with a detailed overview in Table 3.2. To ensure that all four groups were approximately the same size, SoSci was pre-set to randomize but balance the assignment based on the completed questionnaires so that the groups were as equal in size as possible for evaluation.

3.6.2 Exclusion Criteria

All participants who did not complete the questionnaire fully or who answered "no" to the question of whether they completed the questionnaire with full concentration were excluded from the analysis. Additionally, participants who completed the questionnaire very quickly were excluded, as this suggests inadequate engagement. For this, we used the definition of an outlier as 1.5 times the interquartile range below the first quartile. All participants faster than this threshold were excluded. We did not exclude those who took longer, as different strategies might result in varying completion times.

3.6.3 Preparation

The data collected with SoSci must be prepared for analysis. Due to the way the questions are created in SoSci, the results of the treatment and control snippets were stored in different variables, while the given answers were stored in the same variable. Since group membership was sufficient as a marker to distinguish whether the collected times belong to the treatment or the control snippet, the two variables are combined.

The results of the Lab.js experiments were saved as a JSON file that stored various metadata, including the time per frame, in our case per line. Since the relevant time was the time spent on the target line, which was the penultimate line in all of our selected code snippets, this time was extracted from the JSON file and saved as a pure numeric variable. Additionally, we also collected the total time per snippet.

The answers given by the respondents were evaluated as correct or incorrect, and this boolean result was stored as a separate new variable. We ended up with a dataset that contained the group number, for each task the time on the target line, the time for the total snippet, the given answer and whether that answer is correct or incorrect, and the answers for the socio-demographic survey.

3.6.4 Simulation

To answer the seventh research question, a simulation process using python and lisp was used. For the simulation an experiment script was created in lisp, which takes hyperparameters and the number of simulation runs as input and produces a dataset in the same style as the prepared data set, explained in Section 3.6.3. To have the same dataset, we also split the code snippets in two blocks. Since the order did not matter for the ACT-R model, we only distinguished between two groups and only focused on the difference between the treatment and the control tasks. To better fit the data, both groups could have different sizes.

A simulation started by setting a new random seed. Then each task was executed and the results were stored. This reflects the same behaviour human participants would use to create a dataset. In the stored data we collected in a line a group number for each simulation run, and for each task the time on the target line, the given result for the print statement and a boolean if the given answer is right or wrong. This format is nearly the same as for the human participants.

3.7 Evaluation

The evaluation was done in three steps. The first step was to examine the empirical data for the artefacts. In the second step, a strategy was developed for fitting the cognitive model to reproduce the empirical data as well as possible. Then, a final assessment was made of how well the cognitive model reproduced the individual effects found. Each step is described in more detail below.

3.7.1 Artefacts

The empirical data was examined to answer the first six research questions. First, the data were prepared. This was followed by an evaluation of the time or error rate effects for each artefact. The procedure described below is identical for all artefacts considered, and we always considered both tasks for the same artefact with treatment and control snippets.

3.7.1.1 *Preparation*

First, we had to create a variable from the group variable that allowed us to differentiate between treatment and control. Which code snippets were processed by which groups is shown in Table 3.2. Therefore, the treatment and control distinction variable was created as follows: For groups 1 and 3, the variable is set to 1 for the first tasks, which were marked with 1, and to 2 for the second tasks, which were marked with 2. The labelling for groups 2 and 4 was exactly the opposite. Here, the distinguishing variable was set to 2 for the first tasks and to 1 for the second tasks. This meant that all snippets marked with 1 were

treatment snippets and those marked with 2 were control snippets.

To combine the data of both tasks, the time data was normalized per task. To prevent a possible effect the normalization was calculated for the combined mean and deviation of treatment and control snippet. If we had normalized treatment and control individually, we would have had no effect because the means of both groups would have been zero.

After normalizing the time data, we were able to combine both tasks. This resulted in three data columns for each artefact. These were the treatment control discrimination variable, the normalized time data, and whether the given response is True or False.

3.7.1.2 *Time Effects*

A t-test was used to test whether the artefact had a time effect. The t-test was chosen because it allows us to examine whether there is a difference and how large it is. For this purpose, the time variables were divided into two groups, the treatment and control tasks, and compared with an independent t-test.

The prerequisites for computing an independent t-test are data independence, at least one interval scale level of the data, normal distribution within the groups, and homogeneity of variance between the two groups.
We ensured the data's independence by ensuring that each data point came from either a different participant or different snippets of code.
The interval scale level is natural for time data.
We tested for normal distribution using the Shapiro-Wilk test. The t-test is quite robust to a violation of this assumption [43, 55]. Therefore, the results of the Shapiro-Wilk test are reported, but the judgment has no effect on the way the results are calculated.
We tested for homogeneity of variance with the Levene test. If this assumption was violated, the Welch test was used as an alternative. The Welch test explicitly calculates its test statistic with both variances.

Since the investigation of artefacts was of a more exploratory nature, and we also wanted to investigate whether the time data found in the simulation show the same effect with the same magnitude. For this, we needed a metric to measure the size of the effect. In the case of the time data, we use Cohen's d [12].

3.7.1.3 *Error Rate Effects*

The summarized data set from both tasks was also used to evaluate the error rate. For this purpose, it was assessed whether the results of the test subjects were correct or incorrect. This could then be used in a χ^2 test to compare whether there was a significant difference between the target and the control task. The dimensions of the contingency table were treatment and control snippet respectively correct and incorrect results.

The prerequisites for the χ^2 test are a nominal scale level, independent measurements and at least five observations per category.

As explained for time effects, the independence of the data is ensured by the study design. The nominal scale level is the lowest level the data can reach and is ensured because we have clear categories.
Whether there will be at least five entries per group could not be guaranteed in advance, because it might also happen that a task is too easy or too difficult, contrary to expectations, so that there are no right or wrong answers. Alternatively, Fisher's exact test was used, which has no requirement for the number of observations.

The effect of the artifacts on the error rate also required a measure of the effect size. Since this is a two-by-two crosstabulation, we could use ϕ as such a measure.

3.7.1.4 *Correcting Significance Levels*

Since we applied this procedure a total of four times for all artefacts together, we performed a total of eight tests. Since we were doing the analysis post hoc and had not formulated any specific hypotheses in advance, we had to correct the significance level for our eight tests, otherwise we would have had alpha error accumulation.
As a measure for correcting the significance level of the tests, we used the Bonferoni-Holm (BH) correction, which is slightly more liberal than the pure Bonferoni correction [29]. This gave us bounds of $\frac{0.05}{8}, \frac{0.05}{7}, \ldots, \frac{0.05}{2}, \frac{0.05}{1}$ for the significance levels in ascending order of size.

3.7.2 Cognitive Model Fitting

To answer the seventh research question, we needed a method to tune the cognitive model based on the observed empirical data. This method had two main components. The first component was a distance metric that allowed us to assess how different a simulated data set is from the empirical data set. The second component was a strategy for generating the combinations of hyperparameters to be tested, in order to search for the set of hyperparameters that minimizes the distance to the empirical data set. Both components will be described below.

3.7.2.1 *Distance Metric*

To measure the distance between two data sets, we needed to encounter the time data and the error rate. Both had to be integrated into the metric, since our goal was to simulate the time and error rate as closely as possible. The goal was to have a metric that starts at zero for identical data and increases as the data sets become more dissimilar. This allowed us to search for a minimum or by negating it to search for a maximum. To do this, we first had to consider the two cases separately.

Time Data The times on the target line may have had different means as well as different distributions. Both should be taken into account in a distance metric. A statistic that considers both and meets our requirements for a distance metric is the Kolmogorov-

Smirnov (KS) statistic, which gave us both a distance measure and optionally a significance level at which the two distributions differ [33].

Error Rate Data For the answers, we were only interested in whether the probabilities of correct and incorrect answers are identical. Since this measure only has a nominal scale, we could not use the KS statistic at this point, but we could use the χ^2 test to check whether the distributions of correct and incorrect answers in both groups were similar or not. Again, χ^2 gave us a measure that we could use as a distance metric and also a p-value that we could use to assess the fit of our data [9].

Combination The initial plan was to optimize both time and error rate simultaneously. However, following the initial experiments, it became evident that the KS statistic is constrained to a range between zero and one, whereas the χ^2 distribution could take on any positive value. Consequently, the optimization of error rate has become the dominant factor, eclipsing that of time. We used the trick of using the significant values to combine the two metrics, since the significant values for both are in the same range. However, since there were still issues with interpretation, as the χ^2 test is less likely to become significant compared to the KS test, we decided to perform both combined optimization as well as separate optimizations for time and error rate.
To prevent the accumulation of distance values, the individual summands were squared. This increased the weight of individual large deviations while keeping smaller deviations relatively small. This helped to find a hyperparameter set that generalizes better.

3.7.2.2 *Searching Strategies*

To find the set of hyperparameters that best describes the data, we needed a search strategy. From testing, we knew that three parameters have the most influence on the result. These were the rt, ans and lf.

The majority of parameters possessed a fixed default value, which is listed in Table 3.1. As Taatgen and Rijn [49] emphasize in their work, the default values serve as a suitable starting point for the model, as they are based on decades of experience. With regard to the rt parameter, we deviated from the default value of 0 and instead employed a value of -10, derived from a example project presented in the unit five code description in the ACT-R tutorials provided by the ACT-R Research Group [27]. This approach ensured that a value was always returned. Consequently, an initial search space was established within which we could fine-tune the ans parameter.

To optimize the search process, we employed the BO algorithm, which has demonstrated superior efficiency compared to a basic Grid Search when tuning ACT-R models [31]. Following the initial optimization, we expanded the degrees of freedom and the search space to identify the optimal parameter combination. We aimed to minimize the number of parameters that deviate from their default values, because a well-constructed model should have a minimal number of parameters and be defined by its underlying rules [47]. Additionally, an

excessive number of parameters and degrees of freedom can lead to overfitting, increasing the noise in the model and ultimately reducing its generalizability [48].

3.7.3 Model Evaluation

Finally, to answer the seventh research question, it was necessary to look at each task individually. Since we used an aggregated measure to optimize the hyperparameters, it is possible that the model makes better or worse predictions in some tasks than in others. Since we already simulated a dataset with the same number of individuals as in our empirical dataset in the last search cycle, we could now look at the tasks separately.

3.7.3.1 *Time Data*

To evaluate the time data, we considered the KS statistic described in the Section 3.7.2.1 for the task and all groups, which allowed us to consider whether the time data are significantly different. We reported both the F-value and the significance level provided by the KS test for all tasks an groups.
We could then analyze whether the time data were less consistent for certain groups or tasks than for others.

3.7.3.2 *Error Rate Data*

For the error rate data, we followed a procedure analogue to the one described before in Section 3.7.2.1. We used the χ^2 measure and reported both the χ^2 value and the significance level provided by the χ^2 test. After this, we could analyse how well the simulated data fits the empirical data.

3.7.3.3 *Artefacts Effects*

Finally, we wanted to examine whether the simulated data replicated the observed effects of the artefacts. For this we followed the same procedure as described in Section 3.7.1.
Finally, we could compare whether the prediction of the model fits the empirical data well. To do this, we compared whether the effects that were present in the empirical data were also present in the simulation data. To do this, we could compare whether the differences in time or error rate were significant and whether the effect sizes were of the same order of magnitude.

In addition, we needed to consider whether effects that the model might have predicted do not occur in the empirical data. This would then be the starting point for further discussion of why empirical effects may not be predicted or why predicted effects are not found empirically.

3.8 Integration in the Conceptual Model

To make this approach comparable with other experiments, the following describes how it fits into the conceptual model proposed by Wyrich [56]. The model is visualized in Figure 3.2.

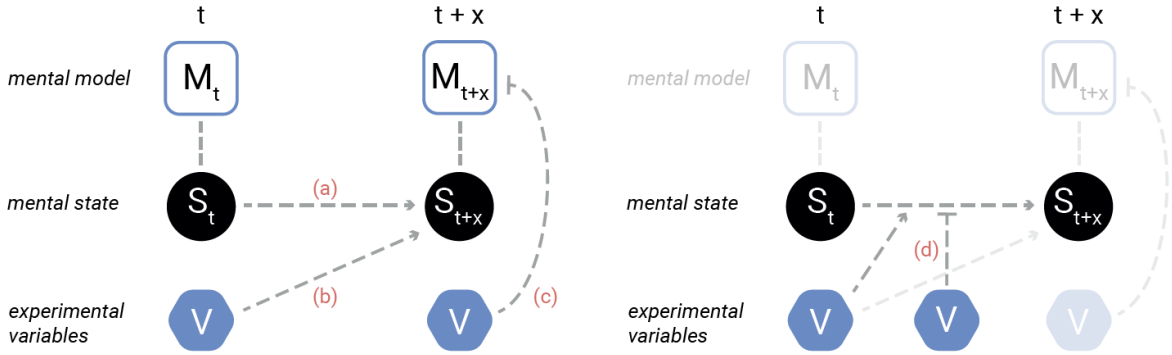


Figure 3.2: Conceptual model for code comprehension experiments [56].

Link (b) contains all variables that probably influence the comprehension task. In our case, these are the different code snippets and refactorings that differ between the test subjects. As a result, the mental states are expected to develop differently. However, since the test subjects have different mental states at the beginning at time t, an attempt is made to control this effect by random assignment to the test groups.

In the model, link (c) is used to get an impression of the unobservable mental model. In this case, the correctness of the answer to the question about what the code snippet prints at the end is used. Although this only gives a limited impression of the final mental model, it is simple and can be simulated.

Link (d) depicts the observation of cognitive processes, transforming the mental state into the next. In our operationalization, this is the time spent per line. This is only a correlate of the cognitive processes, but a simple method that can easily be applied to many subjects. The granularity in this case is row-wise.

Link (a) is a special case in this approach. This link describes the transition from the mental state at the beginning to the one at the end of the experiment. This process can potentially have many mental states in between. So far, however, there is no adequate description of this process [56].

With the ACT-R model, the mental model, mental state and the transition process are imitated. However, this is a simulation based on ACT-R that provides the same observable variables as the test subjects. By tracking the error rate of the answers as well as the time per line, i.e. observing the result and the process, it can be determined how accurately the ACT-R model simulates the real observable behaviour.

4 Evaluation

In this chapter, we will present, contextualize, and critically reflect on the results of the previously described methodology.

4.1 Results

For the results, we first address data collection and data preparation. We then focus on the socio-demographic metadata of the participants before discussing the analysis of the artefacts and the model evaluation.

4.1.1 Data Collection

Data collection occurred in three phases. First came the design and testing phase of the questionnaire, described in Section 3.5. This was followed by the actual data collection phase. Subsequently, the data needed to be cleaned and processed. Each of these phases is explained in detail.

4.1.1.1 *Collection*

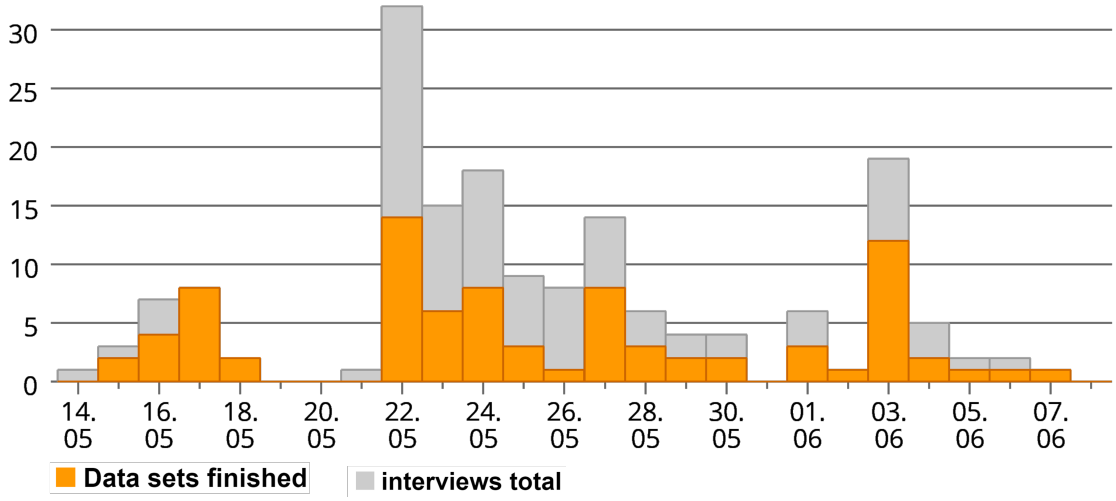


Figure 4.1: Distribution of participant numbers during the survey phase.

The survey period ran from 05/14/2024 to 06/08/2024. During this time, the link to the online test was clicked a total of 480 times. Of these, 168 questionnaires were started. Of these, 84 were completed. The distribution of the number of started and completed questionnaires is shown in Figure 4.1.

4.1.1.2 *Data Cleaning Phase*

Of the 84 participants who completed the questionnaire, 21 indicated that they did not complete it without interruption. These participants were therefore excluded from further analysis. No additional participants were excluded due to excessively fast completion times. Therefore, 63 participants remained in the data set and will be used for all further analyses and evaluations. The distribution among the 4 groups is shown in the following Table 4.1.

Table 4.1: Final number of participants per group.

Group	Count
1	18
2	13
3	16
4	16

Since groups 1 and 3 and groups 2 and 4 differed only 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 following evaluations. This resulted in a number of 34 participants for Group 1 and 29 for Group 2. All simulations of the ACT-R model used in the following were executed with these two group sizes.

4.1.2 Participant Characteristics

The following section takes a closer look at the remaining 63 participants in terms of their socio-demographic characteristics and programming experience.

4.1.2.1 *Socio-Demographic Characteristics*

The 63 participants were on average 31.8 years old with a standard deviation of 11.8 years. The oldest participant was 67 years old, while the youngest participant was 18 years old. A distribution of age groups can be seen in Figure 4.2. It is clear that the majority of participants were in their 20s and 30s, but there are a few older participants, indicating a broader range of subjects in the data set.

Germany clearly dominated the distribution of countries of origin. However, there were a few participants from other countries, mainly from English-speaking countries. A detailed overview of the number of participants from all countries can be seen in Figure 4.3.

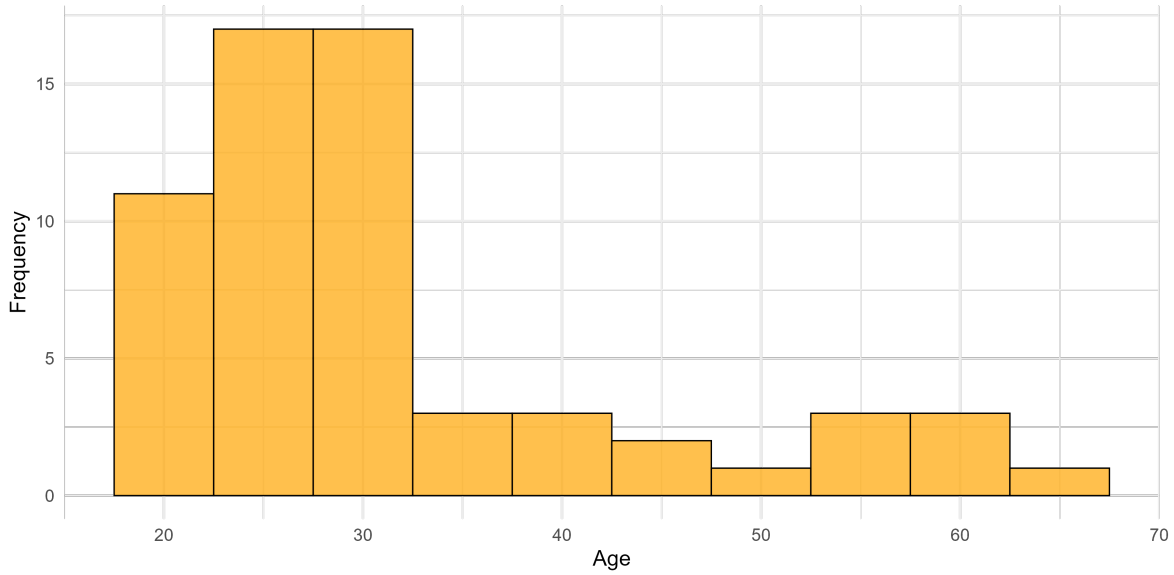


Figure 4.2: Distribution of the age of the participants.

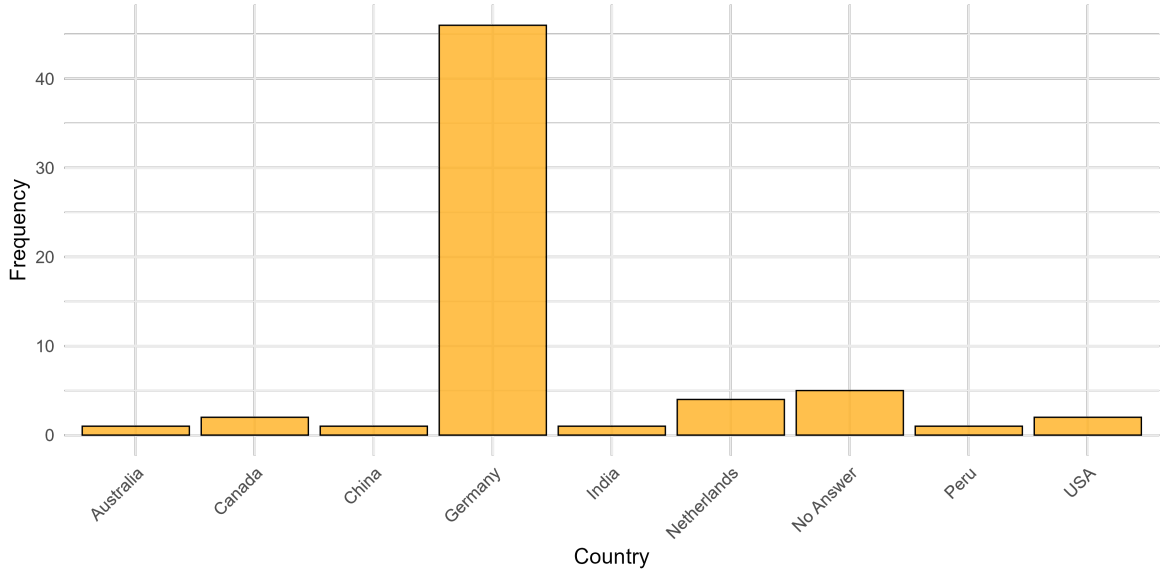


Figure 4.3: Distribution of the countries of origin of the participants.

The educational background of the respondents is clearly university based. 84.1 percent of all respondents have at least a bachelor's degree and 71.4 percent have a higher university degree. There are a few participants with other types of school diplomas and even some respondents without a high school diploma. The distribution of highest educational attainment is visualized in the Figure 4.4.

In terms of the distribution of gender identity, respondents identifying as male were by far the most common, with a share of 71.4 percent. Far fewer identified as women, with 22.2 percent. The rest either identified as a gender not included in our selection or preferred not

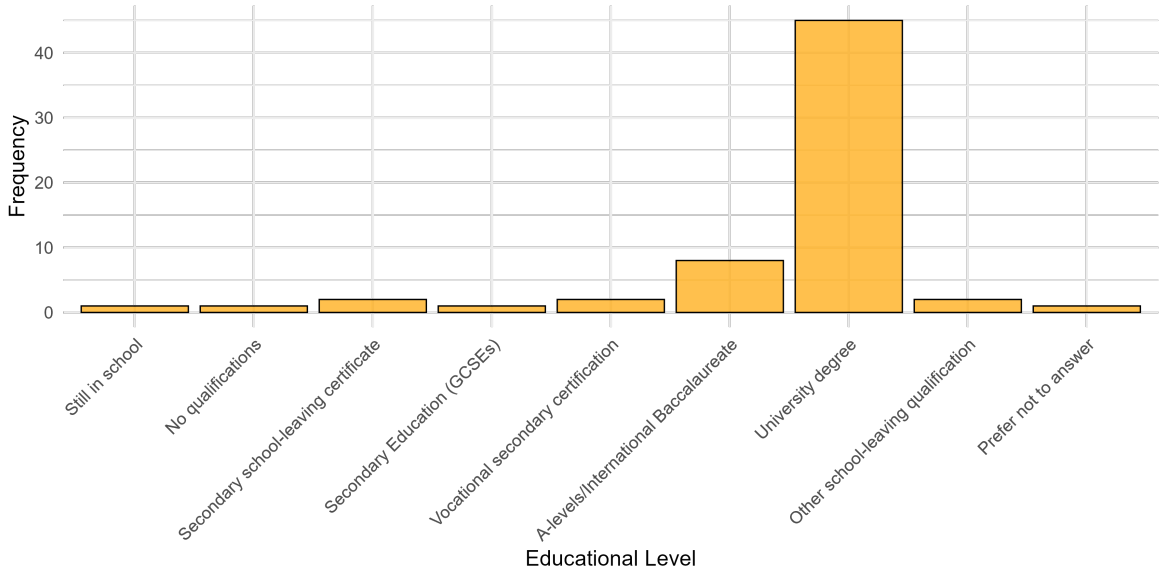


Figure 4.4: Distribution of educational attainment of the participants.

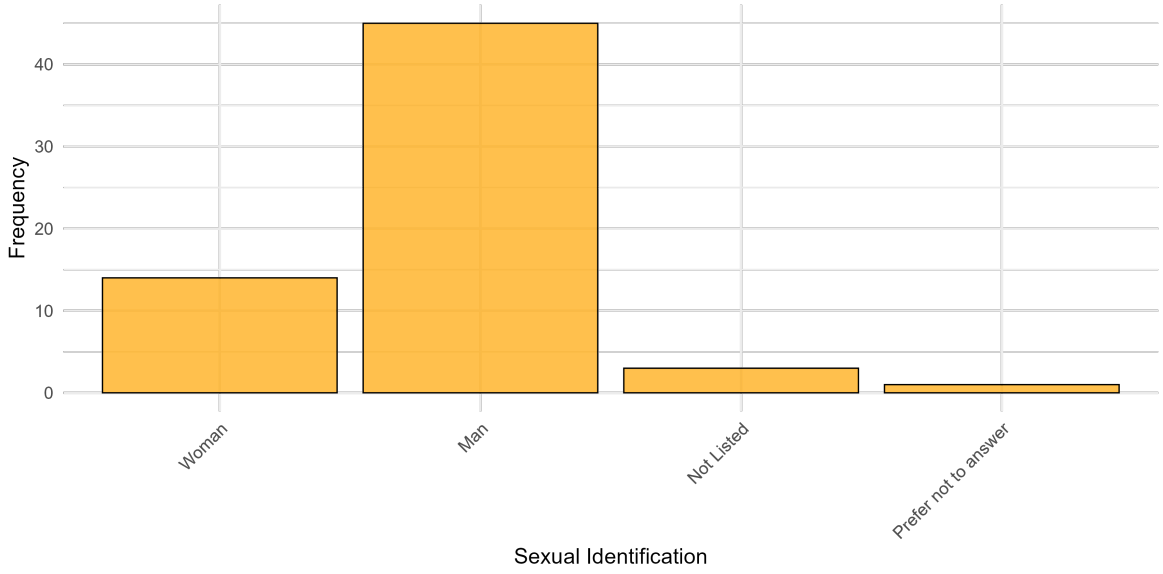


Figure 4.5: Distribution of gender identification of the participants.

to answer this question. The distribution is visualized in the Figure 4.5.

The current work situation of the participants was queried in an open question. A total of 23 participants indicated that they are currently engaged in academic studies. The remaining participants were distributed as follows: seven work in research, ten work with software, and a further ten work in business as analysts or in management. The remaining participants represent a variety of other fields or did not provide any information.

4.1.2.2 *Coding Skills*

The four questions on programming ability are evaluated in more detail below. All questions are presented on a 5-point Likert scale, which is assumed to be an interval scale in the following. It ranged from very inexperienced to very experienced.

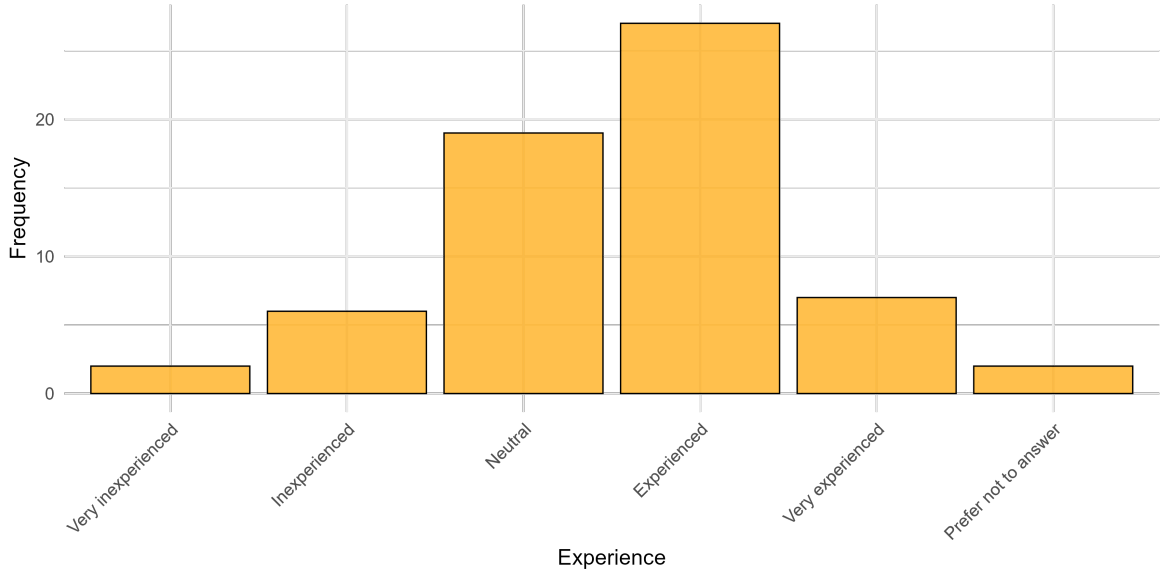


Figure 4.6: Distribution of self-assessment of programming experience compared to classmates or colleagues of the participants.

On average, participants considered themselves to be just as experienced or slightly more experienced than their colleagues or classmates, with a mean of 3.365 and a standard deviation of 1.222. The distribution of the exact answers is shown in Figure 4.6.

In the case of logical programming, the distribution is split. Here the participants tended to rate their experience as neutral, although there is a larger spread with one group that tends to be more experienced and another that tends to be less experienced. On average, participants rated their experience with logic programming as 2.841 with a standard deviation of 1.358.

The pattern for functional programming is similar to the comparison with colleagues and classmates. Again, participants rated themselves as more experienced with a mean of 3.19 and a standard deviation of 1.148.

Object-oriented programming is one of the most common programming paradigms, and this is reflected in the participants' answers [34]. Here the participants tended to rate themselves from experienced to very experienced with a mean of 3.65 and a standard deviation of 1.259.

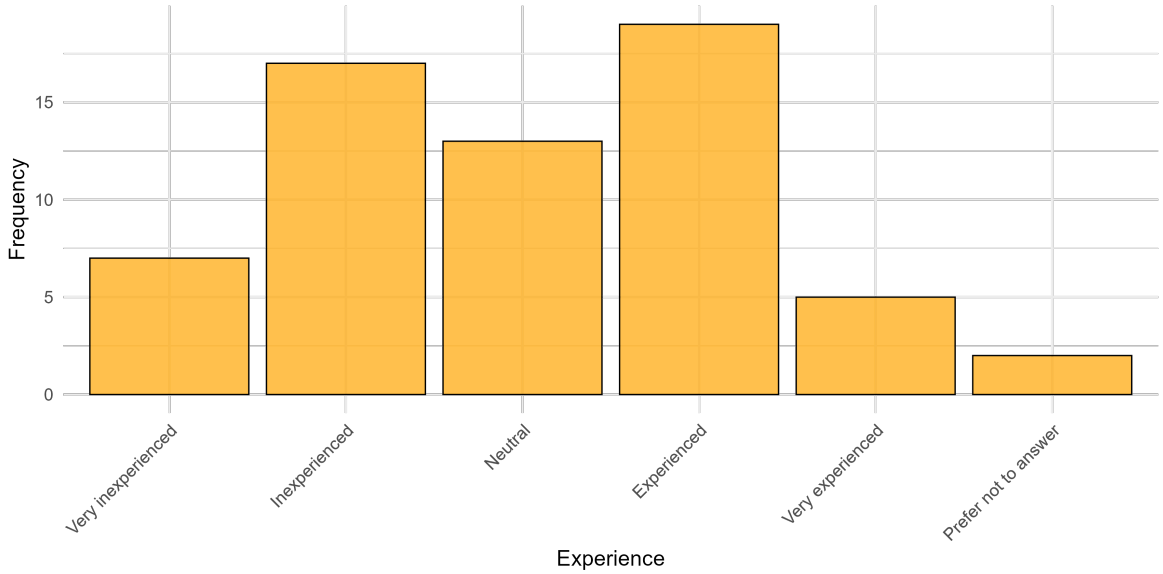


Figure 4.7: Distribution of self-assessment of programming experience with logical programming of the participants.

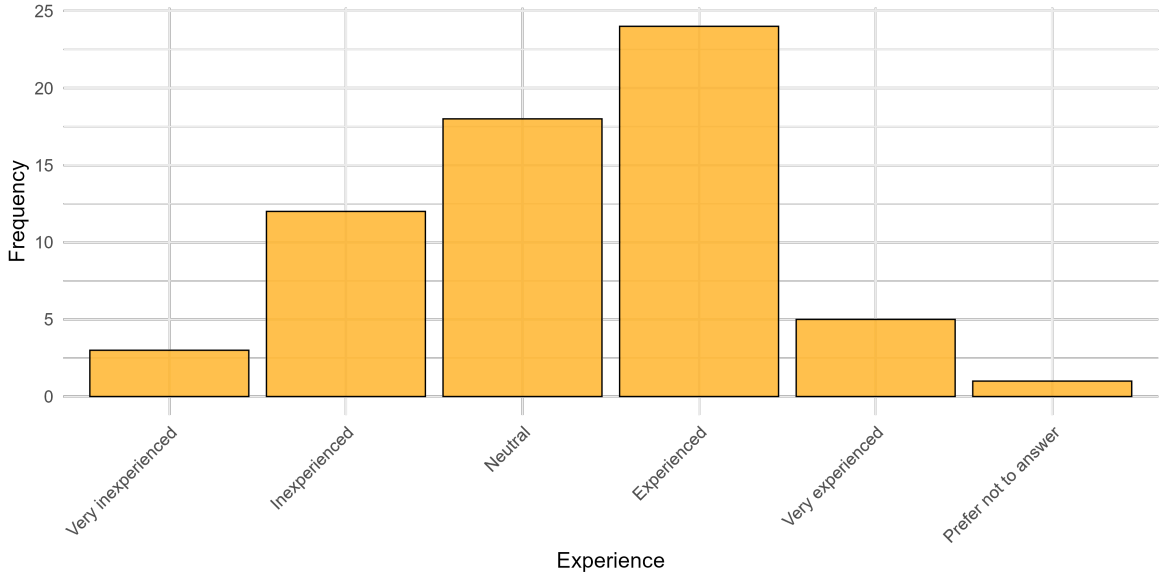


Figure 4.8: Distribution of self-assessment of programming experience with functional programming of the participants.

The detailed distribution of the self-assessments for all three programming paradigms is shown in the Figures 4.7 to 4.9. The participants were more familiar with object-oriented programming than with logical programming. However, the code snippets were purly examples of logic programming. This must be taken into account in the further analysis.

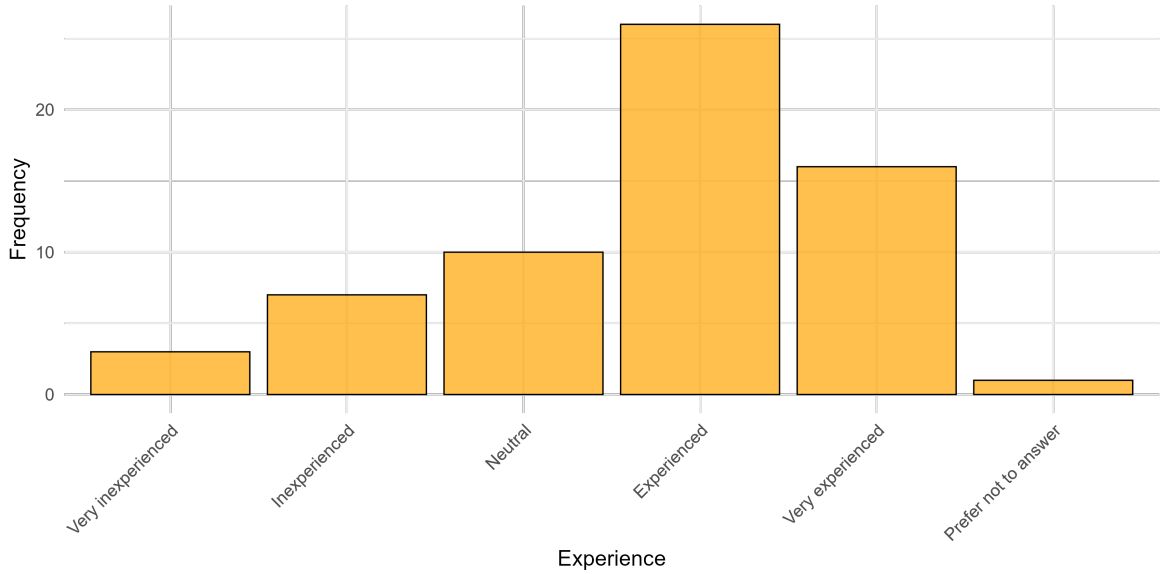


Figure 4.9: Distribution of self-assessment of programming experience with object oriented programming of the participants.

Based on the work of Siegmund et al. [45], who extracted comparison with classmates and experience in logical programming as the main factors for programming experience, it can be concluded that the dataset contains a rather mixed group.

4.1.3 Effects in Empirical Data

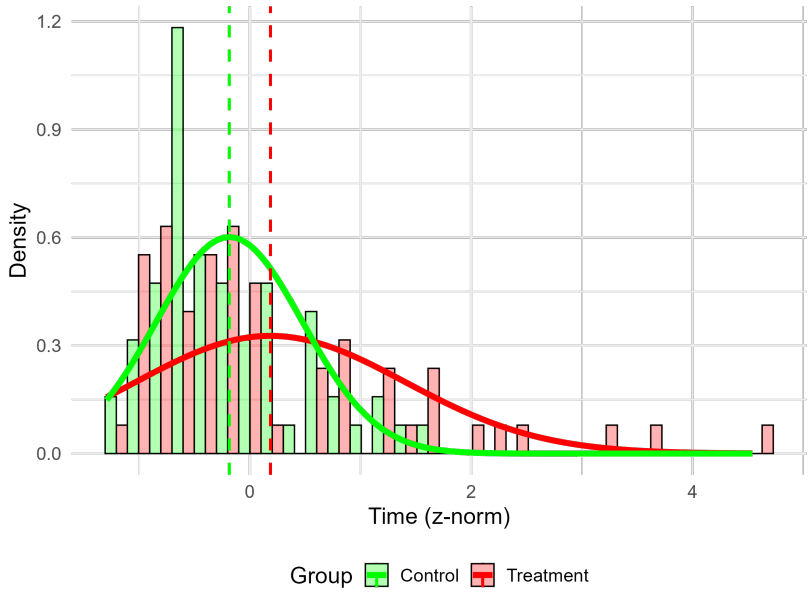
The following chapter presents and analyzes the empirical data to answer research questions one to six. For the sake of clarity, only the summarized data from both snippets per effect are presented here. A graphical representation of the results for the individual snippets can be found in the Appendix A.

The assumption of normality was not met in any of the distributions. For the sake of completeness, the results of the Shapiro-Wilk normality test are presented. However, given that the t-test is considered to be stable for such group sizes, it was employed in this instance [43, 55].

Each section provides a concise overview of the characteristics of the code snippets within each category, allowing for a more straightforward classification of the observed effects.

4.1.3.1 *Code Distance*

The CD class exhibited variability in the proximity of variable declaration relative to their initial utilization. In the treatment group, there were two intervening lines, whereas in the control group, the declaration and use of the variable occurred in immediate succession. The results for the CD snippets are presented in Figure 4.10. On the left, the times on the target



(a) Time on target line.

	False	True	Sum
Control	7	56	63
Treatment	8	55	63
Sum	15	111	126

(b) Error rate.

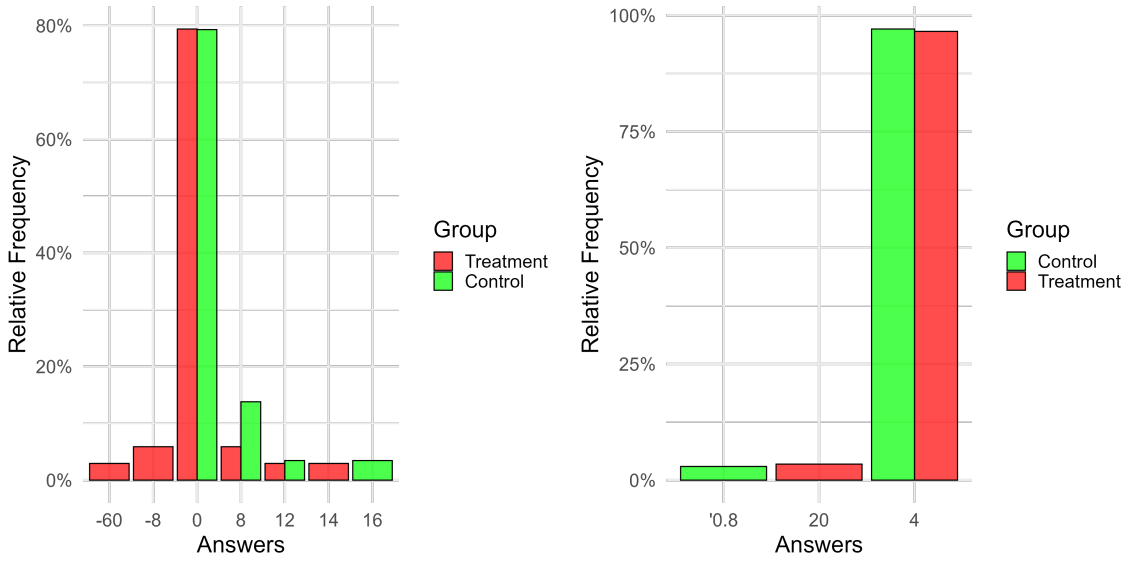
Figure 4.10: Results for the combined data of both CD snippets for treatment and control group for empirical data.

line are presented in comparison between the two groups. On the right, the comparison of the error rate between the two groups is displayed. This kind of presentation will be used for all subsequent effects presentation.

Time Data The Shapiro-Wilk test yielded a value of W = 0.845 with p = 1.31e-06 for the treatment group and a value of W = 0.932 with p = 1.813e-03 for the control group. This indicates that both groups are not normally distributed. The results of the Levene test yielded an value of F = 5.969 with p = 0.01597, indicating the presence of heteroscedasticity. Consequently, the findings from the Welch-test will be employed in the subsequent analysis. The mean value for the treatment group is 0.19, with a standard deviation of 1.22. The control group exhibited a mean value of -0.19 and a standard deviation of 0.66. With a group size of 63 per group, this yields a t-value of t = 2.125 with p = 0.03617. The effect size of Cohen's D is d = 0.379, which can be regarded as indicative of a small to medium effect.

Error Rate Data As illustrated in Table 4.10b, the number of incorrect responses was approximately equivalent in both groups. Consequently, the χ^2-test yielded a value of $\chi^2 = 0$ with p = 1, indicating that there is no statistically significant difference between the two datasets. The effect size is Φ = -0.025, which is negligible.

If one looks at the type of errors made in Figure 4.11, one can see that the latitude of possible answers in the treatment group of CD1, visualized in Listing 3.6, was slightly larger. The correct results were "0" for CD1 and "4" for CD2.



(a) CD1. (b) CD2.

Figure 4.11: Distribution of the answers for the CD snippets for empirical data.

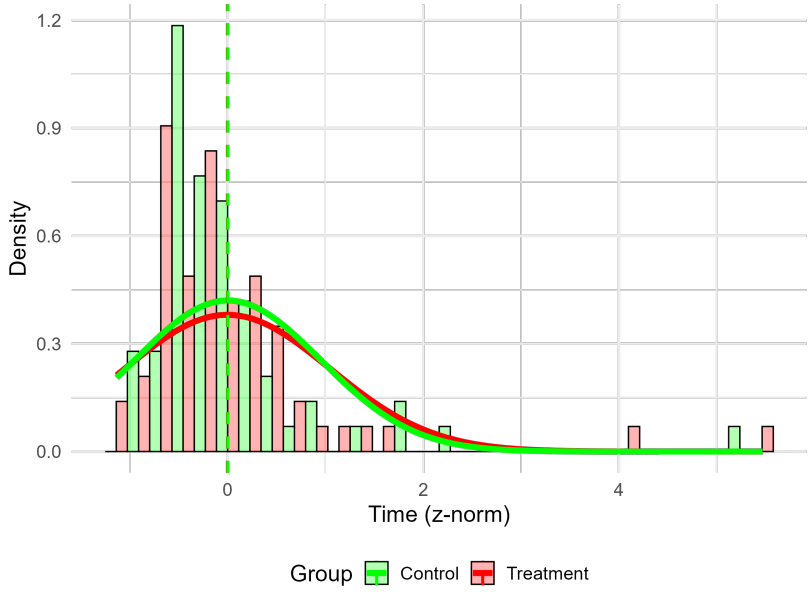
4.1.3.2 *Repeated Code*

The class of CR snippets varied in the number of variable declarations, either single for the control group or double for the treatment group. The declaration was followed by another line of code, after which the variable was used. The results for the CR snippets are presented in Figure 4.12.

Time Data The Shapiro-Wilk test yielded a value of W = 0.668 with p = 1.158e-10 for the treatment group and a value of W = 0.681 with p = 2.062e-10 for the control group. This indicates that both groups are not normally distributed. The results of the Levene test yielded an value of F = 0.124 with p = 0.725, indicating the absence of heteroscedasticity. Consequently, the findings from the t-test will be employed in the subsequent analysis. The mean value for the treatment group is 0, with a standard deviation of 1.05. The control group exhibited a mean value of 0 and a standard deviation of 0.95. With a group size of 63 per group, this yields a t-value of t = -0.021 with p = 0.983. The effect size of Cohen's D is d = 0.00374, which is negligible.

It is worth noting, however, that the non-existent mean value difference shown here is the result of a positive and a negative difference between the mean values in both code snippets. This phenomenon is illustrated in Figure A.3 and A.4. The reason for the reversal of the effect between the two snippets could not be determined in this work.

Error Rate Data As illustrated in Table 4.12b, mistakes only occurred in the treatment group. Nevertheless, the discrepancy is not statistically significant, as indicated by $\chi^2 = 1.366$ with p = 0.242. Since the condition of at least 5 observations per field is violated at this point, we use Fisher's Exact test as an alternative which gives us p = 0.244. As one can see, the two

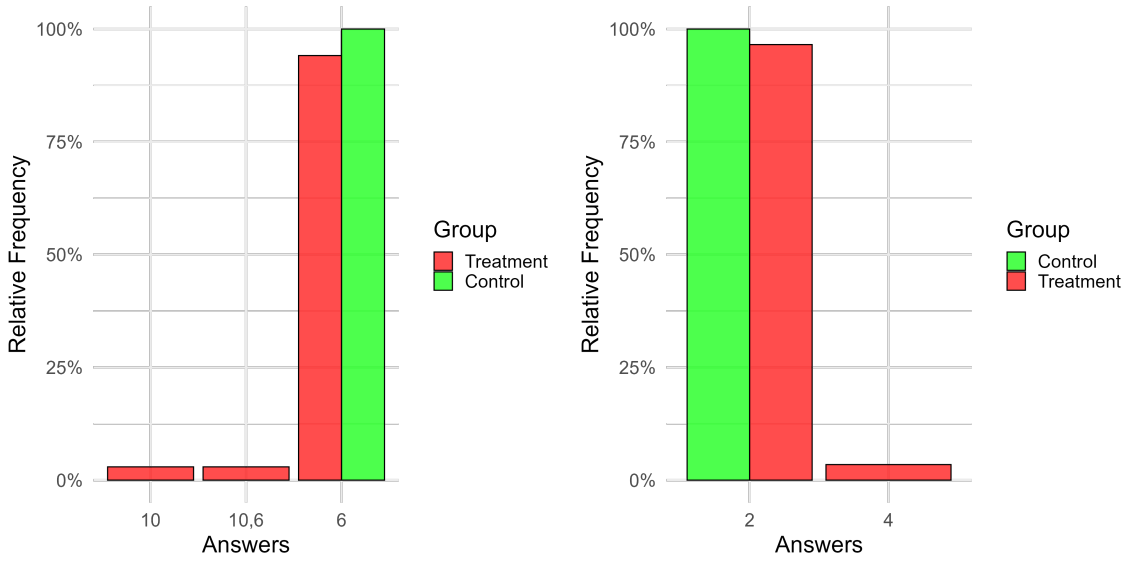


(a) Time on target line.

	False	True	Sum
Control	0	63	63
Treatment	3	60	63
Sum	3	123	126

(b) Error rate.

Figure 4.12: Results for the combined data of both CR snippets for treatment and control group for empirical data.



(a) CR1.

(b) CR2.

Figure 4.13: Distribution of the answers for the CR snippets for empirical data.

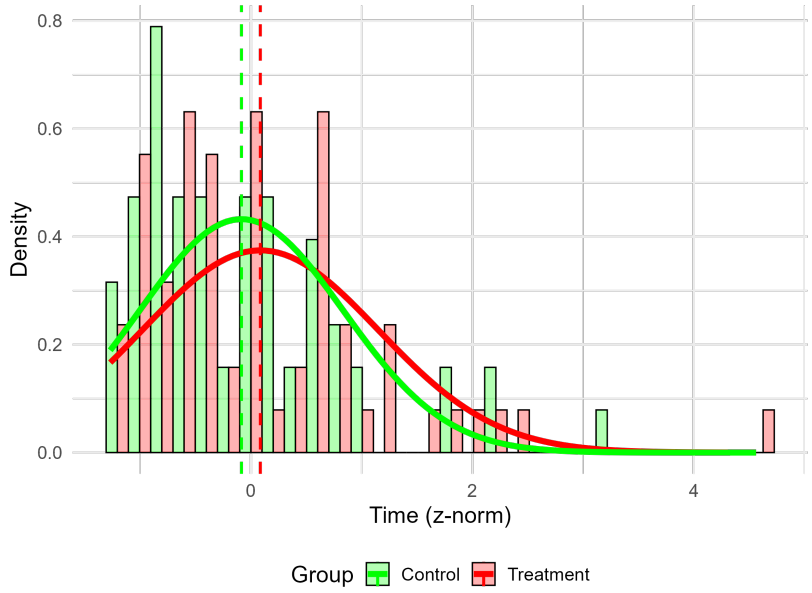
tests differ only minimally in their result. Consequently, it is plausible to conclude that the observed difference is merely a coincidence. The effect size is Φ = -0.156, which is small effect.

If we again look at the type of errors made in Figure 4.13, it is noticeable for CR1, visualized in Listing 3.8, that at least the expected error of not remembering the overwriting occurred

once in the treatment group. The "10.6" could be interpreted as a uncertainty about the functionality of variable overwriting. The correct results were "6 "for CR1 and "2" for CR2.

4.1.3.3 *Declaration Redeclaration Distance*

The DR Snippets class represents an interaction between the two previously described snippets. It serves to vary the distance between the declaration and the redeclaration of a variable. In the case of the treatment group, a double declaration was created at the beginning, followed by two lines of filler code, after which the usage is presented. For the control group, the filler lines were inserted between the declaration and redeclaration of the variable. The results for the DR snippets are presented in Figure 4.14.



(a) Time on target line.

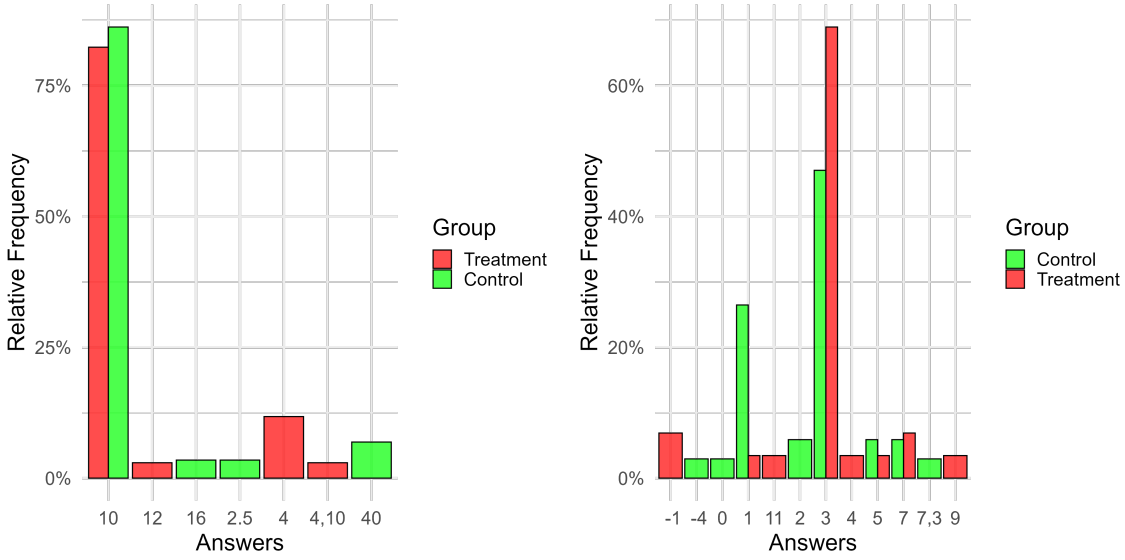
	False	True	Sum
Control	22	41	63
Treatment	15	48	63
Sum	37	89	126

(b) Error rate.

Figure 4.14: Results for the combined data of both DR snippets for treatment and control group for empirical data.

Time Data The Shapiro-Wilk test yielded a value of W = 0.883 with p = 2.226e-05 for the treatment group and a value of W = 0.89 with p = 3.821e-05 for the control group. This indicates that both groups are not normally distributed. The results of the Levene test yielded an value of F = 0.5398 with p = 0.464, indicating the absence of heteroscedasticity. Consequently, the findings from the t-test will be employed in the subsequent analysis. The mean value for the treatment group is 0.09, with a standard deviation of 1.07. The control group exhibited a mean value of -0.09 and a standard deviation of 0.92. With a group size of 63 per group, this yields a t-value of t = -0.96 with p = 0.339. The effect size of Cohen's D is d = 0.171, which is small effect.

Error Rate Data As illustrated in Table 4.14b, there occurred more mistakes in the control group than in the treatment group, and there were more mistakes overall. Nevertheless, the



(a) DR1. (b) DR2.

Figure 4.15: Distribution of the answers for the DR snippets for empirical data.

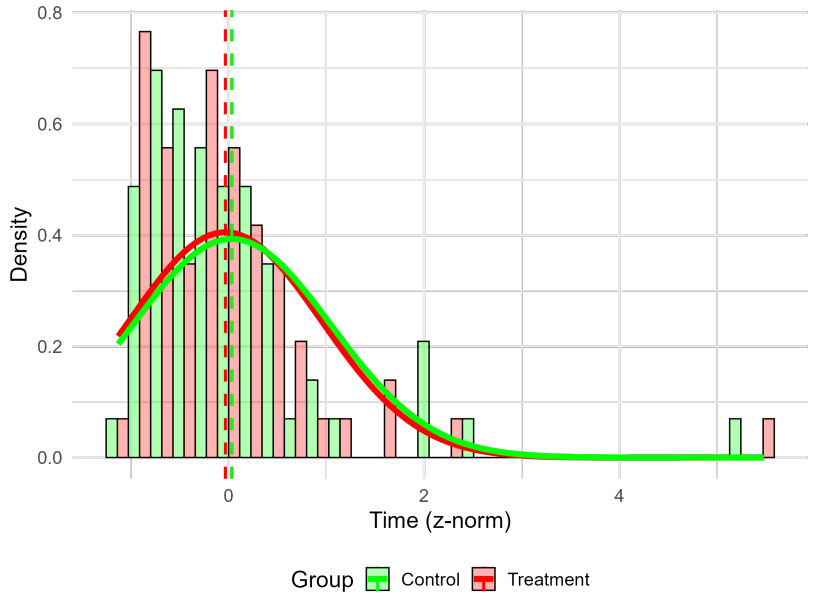
discrepancy is not statistically significant, as indicated by $\chi^2 = 1.378$ with $p = 0.24$. Consequently, it is plausible to conclude that the observed difference is merely a coincidence. The effect size is $\Phi = 0.122$, which is small effect. It is not possible to ascertain the reason for the accumulation of errors in this form of code snippets.

If we look at the errors made in Figure 4.15, we see in DR1 an accumulation of "4" for the treatment group and "40" for the control group. "4.10" could again be interpreted as inconclusiveness. If we look into the snippet in Listing 3.10, "4" would be the expected error if the overwriting was forgotten. The answer "40" could be explained by forgetting to divide "q" by "4". For DR2, visualized in Listing 3.11, there is an accumulation at "1" for the control group. This error could occur if the order of the subtraction is reversed, as these were not queried in the initialized order. The correct results were "10" for DR1 and "3" for DR2.

4.1.3.4 *Repeated Distance*

The class of RP snippets varied the distance between a double declaration and its usage. In the treatment group, there were two fill code lines between the double declaration and the usage, which were drawn in front of the double declaration for the control group. The results of RP snippets are presented in Figure 4.16.

Time Data The Shapiro-Wilk test yielded a value of $W = 0.734$ with $p = 2.404e\text{-}09$ for the treatment group and a value of $W = 0.759$ with $p = 8.328e\text{-}09$ for the control group. This indicates that both groups are not normally distributed. The results of the Levene test yielded an value of $F = 0.045$ with $p = 0.8323$, indicating the absence of heteroscedasticity. Consequently, the findings from the t-test will be employed in the subsequent analysis. The mean value for the treatment group is -0.03, with a standard deviation of 0.98. The control



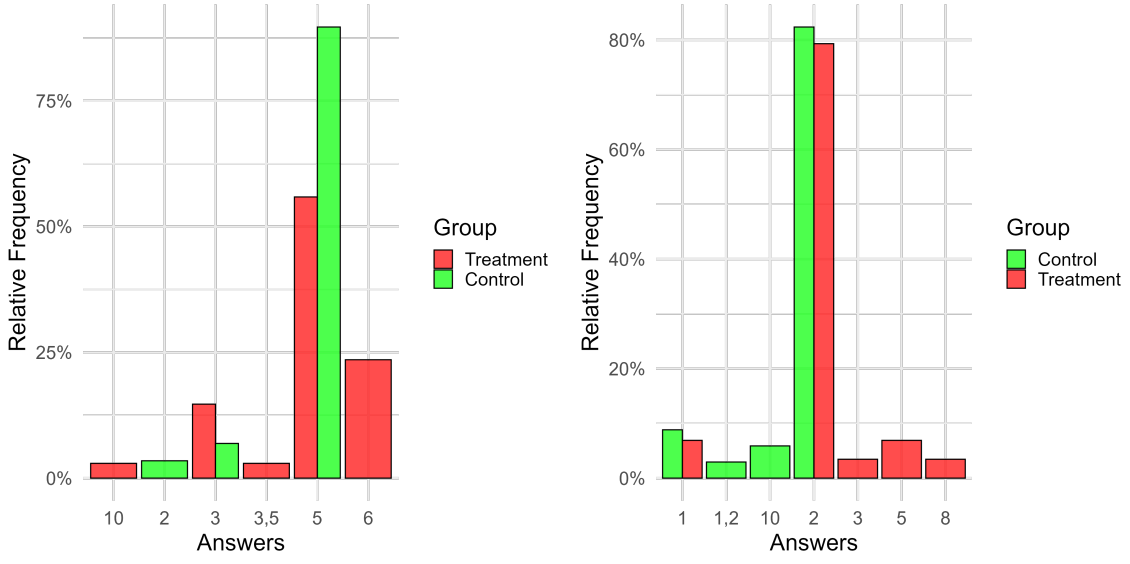
(a) Time on target line.

	False	True	Sum
Control	9	54	63
Treatment	21	42	63
Sum	30	96	126

(b) Error rate.

Figure 4.16: Results for the combined data of both RP snippets for treatment and control group for empirical data.

group exhibited a mean value of 0.03 and a standard deviation of 1.01. With a group size of 63 per group, this yields a t-value of t = -0.3697 with p = 0.712. The effect size of Cohen's D is d = 0.0659, which is negligible.



(a) RP1.

(b) RP2.

Figure 4.17: Distribution of the answers for the RP snippets for empirical data.

Error Rate Data As illustrated in Table 4.16b, there occurred more mistakes in the treatment group. This discrepancy has a lower probability of occurring by chance than the other test values in this analysis, with a test value of $\chi^2 = 5.294$ with $p = 0.0214$. The effect size is Φ = -0.224, which is small effect.

Let's take a final look at the distribution of the errors made in Figure 4.17. For the treatment group, there is a concentration of "6" and "3" for RP1. If we inspect RP1, visualized in Listing 3.12, "3" would be the expected error if the test subjects forgot to overwrite the variable with the new declaration. The answer "3.5" can again be interpreted as an ambivalent answer. The answer "6" could arise if one remembers "t" with "2" instead of "3". For RP2, visualized in Listing 3.13, an accumulation at "1", which is the expected error if the subjects forgot to overwrite, and at "5" and "10", where "10" could be explained by the fact that in the control group one was supposed to subtract "4" from "w" at the beginning and forgot to do so. The correct results were "5" for RP1 and "2" for RP2.

4.1.3.5 *Bonferoni-Holm Correction*

Now that we have considered all eight tests on the empirical data, they must be examined for statistical significance. Of all the eight results presented, only two fall below the threshold of $p = 0.05$. Since all test statistics greater than this cannot be made significant even with the correction, we exclude them from further consideration here. The two remaining values are the difference in times for CD with $p = 0.0362$ and the difference in the probability of errors in the answers for RP with $p = 0.0214$. As a remainder, the BH correction gives us bounds of $\frac{0.05}{8}, \frac{0.05}{7}, \ldots, \frac{0.05}{2}, \frac{0.05}{1}$ for the significance levels in ascending order of size. So the limits for the first two are $p < 0.00625$ and $p < 0.00714$. Therefore both values exceed the bounds and the test statistics are not large enough to consider a chance finding unlikely.

In conclusion, although there are trends in the data that support our initial hypotheses, the differences found were too small to rule out chance at this level of variability and sample size.

4.1.4 Model Fitting

The outcomes of the individual optimizations are presented in the following section. We started tuning the ans parameter for time and error rate as well as the combination of both. After that, we used the findings to extend the experiments and tune rt, ans and lf. The BO ran for long time, but only the best parameter combination results per experiment are described in detail.

4.1.4.1 *Tuning only ans in Recommended Default Range*

The first experiments started by optimizing the default range of ans. We used the recommended range from 0.2 up to 0.8 for the ans parameter. Therefore the search space was small and we only needed 100 steps to converge stably for time and error rate optimizations

and only 300 for the combined one. The results on the single snippet level for the best ans parameter are visualised in Appendix B.

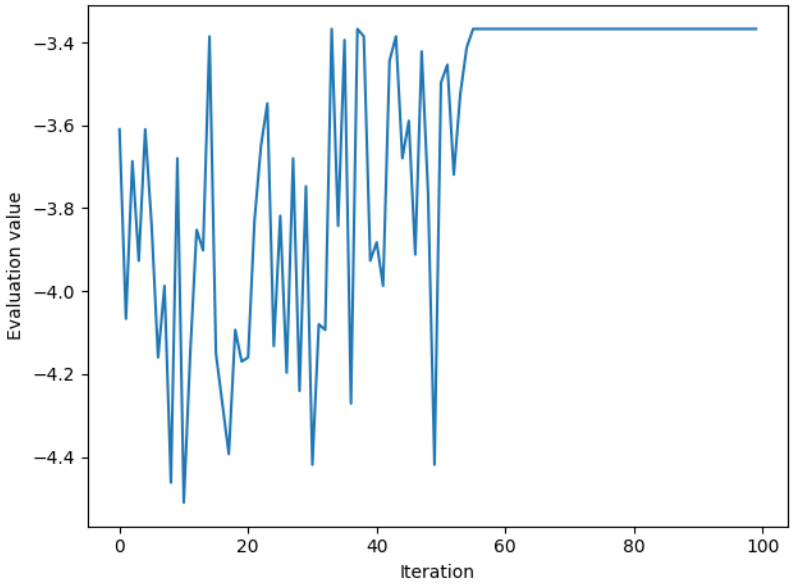
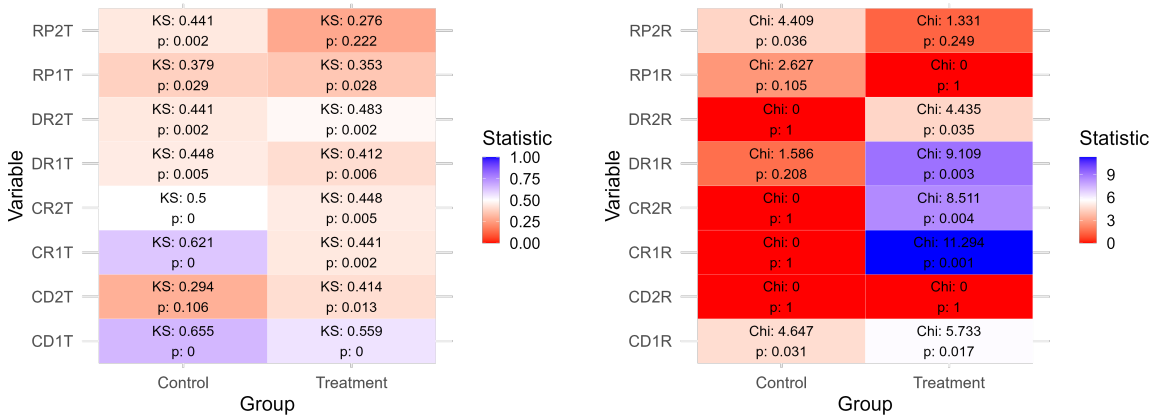


Figure 4.18: Evolution of the BO algorithm with 50 start points for the tuning ans for time data.

Time For the time data, the best reachable target value for the negated sum of squared KS statistics was -3.367. This value was reached for the first time in the 50 initial points, specifically point 34. As one can see in Figure 4.18, this value was quite stable at the end. The value for the ans parameter is 0.5366.



(a) Heatmap for matching the time distribution using KS statistics.

(b) Heatmap for matching the error rate distribution using χ^2 statistics.

Figure 4.19: Heatmaps for the tuning ans for time data.

Figure 4.19 shows how accurate this approximation was in terms of time data and error rate. In the following we will use this form of representation for the quality of the approxi-

mation.

The KS statistics and their significance for the individual code snippets are shown on the left side in Figure 4.19a. Notice that only two snippets are not significantly different from each other, even though we have optimized for time.

On the right side of the Figure 4.19b are the χ^2 values and the significances for each snippet. Note that in some marginal cases, namely when the column sums were zero, we skipped the χ^2 test and assumed a value of 0 and a significance of 1 directly. This is necessary because the χ^2 test is not defined for 0 sums. The reason this shortcut is possible is that the row sums are always the same due to the same number of simulations as the empirical data. Consequently, if the row sums are equal and one column sum is 0, the distribution in the other column must be identical. Notice that the fit is more spread out here. Thus, we have a good fit for samples where there were hardly any errors in the empirical data, such as CD2 and the control groups CR1 and CR2, but for groups where the number of errors in the empirical data was higher, such as DR2 and RP1.

The question following from this is in which direction the time data do not fit and whether there is a systematic error.

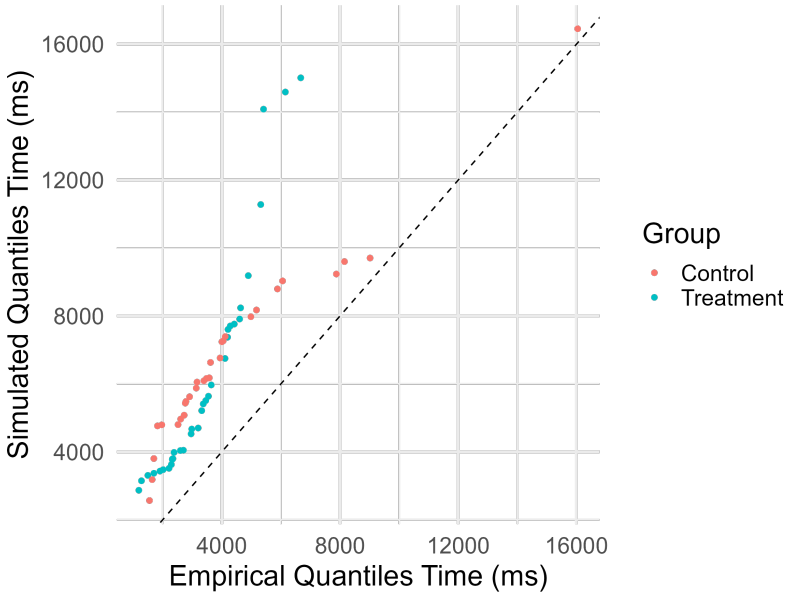


Figure 4.20: QQ-Plot for CR1 with optimized ans for time data.

To gain insight into this phenomenon, a detailed examination of the distribution is essential. The code snippet CR1 is used here as an illustrative example, although the observation is equally applicable to the other code snippets. These can be found in the Appendix in Figure B.1 The empirical and simulated data distributions have been plotted in a QQ-plot (see Figure 4.20). Given that the group sizes are identical, each data point represents a single measurement. The dashed diagonal represents the ideal line, indicating that if all points lie

on this line, the distributions are identical.

A noteworthy observation at this stage is that all points lie above the diagonal. This indicates that the simulated times are systematically slower than the empirical times. Consequently, the times for the simulations must be accelerated. Given the linear nature of the shift, it is reasonable to suggest that the lf parameter should be incorporated into the optimization process.

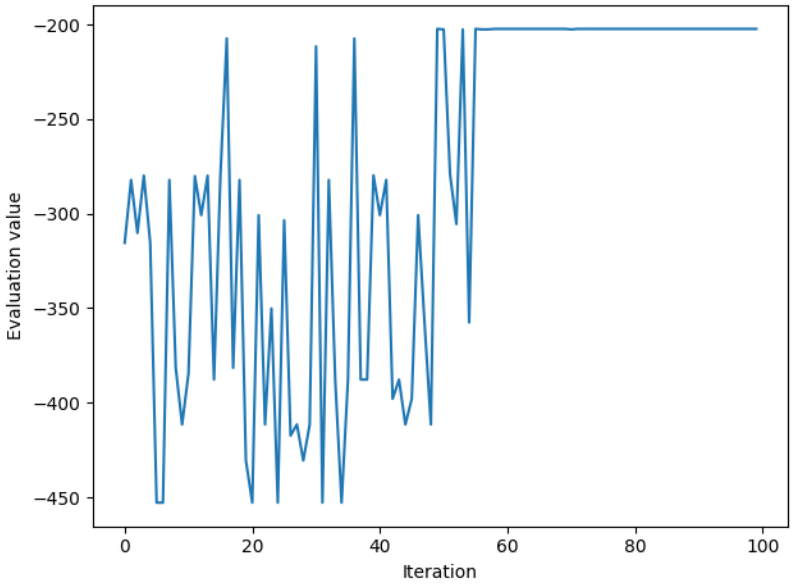
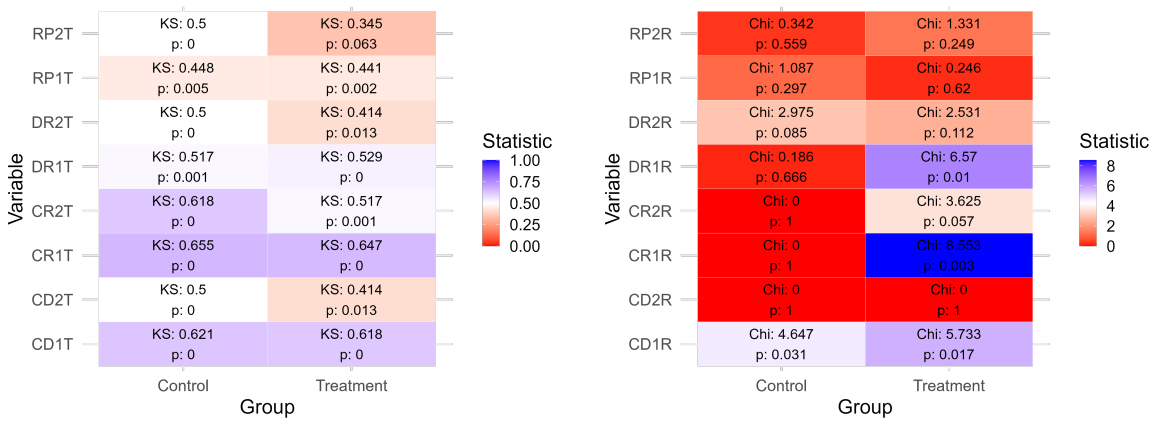


Figure 4.21: Evolution of the BO algorithm with 50 start points for the tuning ans for error rate data.

Error Rate For the error rate data, the best reachable target value for the negated sum of squared χ^2 values was -202.3575. This value was reached for the first time in the last execution in the 50 initial points. As one can see in Figure 4.21 this value was stable. The value for the ans parameter was 0.2084, which is different from the results for time optimization. At this moment, it becomes evident why it is challenging to offset the values against one another. The KS statistic and χ^2 statistics have different ranges and χ^2 would dominate the KS statistic.

Figure 4.22 shows again the accuracy of matching. Since we optimized for the error rate data now, the heatmap in Figure 4.22b contains smaller χ^2 values compared to the results optimizing for time presented in Figure 4.19b.
As expected, the fit of the time data is worse than if one optimize for time. This can be seen from the colouration of Figure 4.22a, which shows more blue, compared to Figure 4.19a

Now we needed to investigate why some data don't match. To do this, we looked at the snippets with the worst fit, specifically CR1 and DR2. The empirical and simulated results comparisons are shown in Table 4.2 and Table 4.3.



(a) Heatmap for matching the time distribution using KS statistics.

(b) Heatmap for matching the error rate distribution using χ^2 statistics.

Figure 4.22: Heatmaps for the tuning ans for error rate data.

Table 4.2: Crosstables to compare simulated and empirical data for CR1 with optimized ans for error rate data.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	13 (2)	21 (32)	34 (34)
Total	13 (2)	50 (61)	63 (63)

Annotation: Empirical data in brackets.

Table 4.3: Crosstables to compare simulated and empirical data for DR2 with optimized ans for error rate data.

	False	True	Total
Control	10 (18)	24 (16)	34 (34)
Treatment	16 (9)	13 (20)	29 (29)
Total	26 (27)	37 (36)	63 (63)

Annotation: Empirical data in brackets.

It is particularly noticeable that for CR1 the number of errors for the simulations was significantly higher for the treatment snippet. This indicates that the retrieval threshold was too low and the ans was causing the wrong chunk to be retrieved. The same is true for the DR2 treatment group. Interestingly, this effect was reversed in the control group of DR2, where there were more errors in the empirical data. However, as mentioned in Chapter 4.1.3.3, these errors are not the expected ones resulting from incorrect retrieval. Accordingly, the model cannot simulate these guessed values. In our model, a failed retrieval would most likely simulate this. The rt had to be changed for this as well.

Given all these observations, it is reasonable to suggest that the rt parameter should be incorporated into the optimization process.

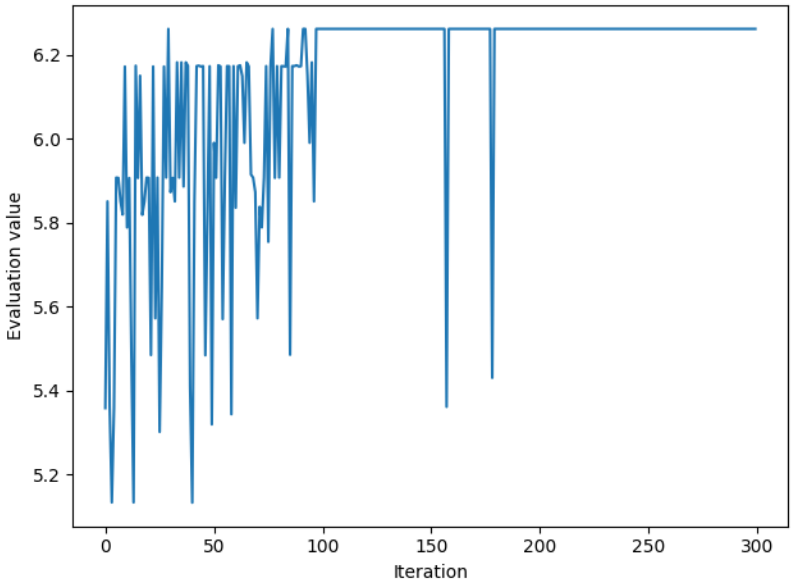
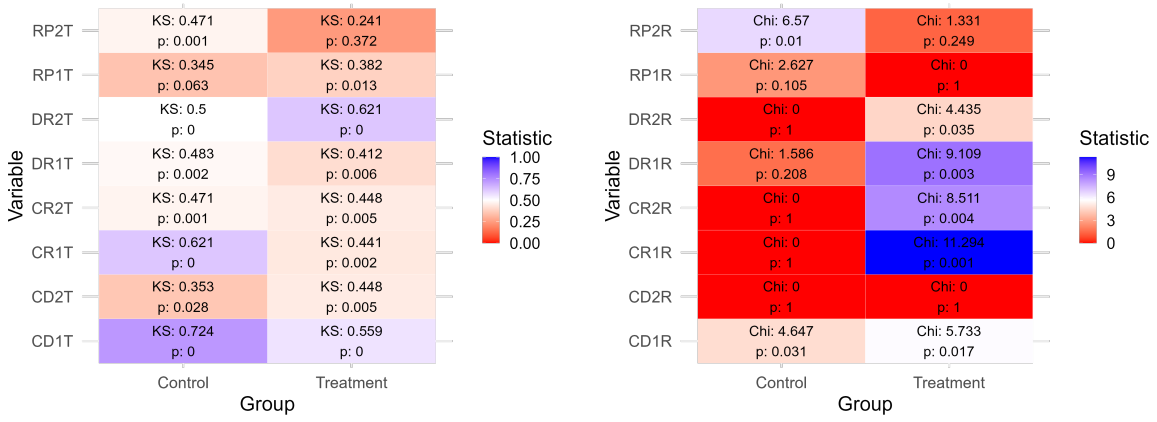


Figure 4.23: Evolution of the BO algorithm with 50 start points for the tuning ans for combined data.

Combined For the combined time and error rate optimization, the best reachable target value for the sum of squared significance values was 6.262. This value was reached for the first time in the 50 initial points, specifically point 30. As one can see in Figure 4.23 this value was quite stable at the end. The value for the ans parameter was 0.6523. This shows once again how important the focus of the optimization is. The parameters were different depending on what is being optimized for. However, this shows how important a good underlying model is, because a model with parameters can be drastically adapted to the data.



(a) Heatmap for matching the time distribution using KS statistics.

(b) Heatmap for matching the error rate distribution using χ^2 statistics.

Figure 4.24: Heatmaps for the tuning ans for combined data.

Figure 4.24 illustrates that the combined optimization results in a split fit. The time data in Figure 4.24a fits better than tuning only for results (see Figure 4.22a) but worser than

tuning only for time (see Figure 4.19a). Interestingly, the results for the fit of the result data are even similar to optimizing on time data only, but worse than optimizing on the result data.

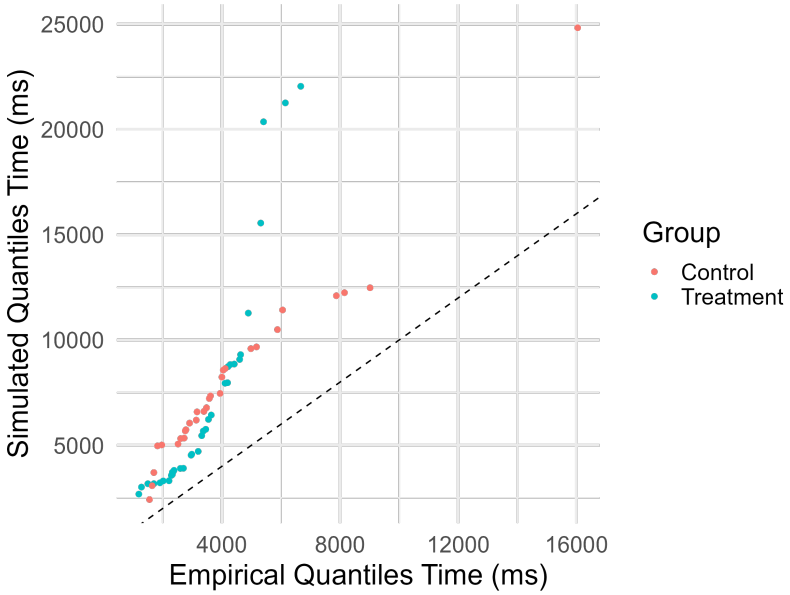


Figure 4.25: QQ-Plot for CR1 time data with optimized ans for combined data.

Table 4.4: Crosstables to compare simulated and empirical data for CR1 with optimized ans for combined data.

	False	True	Total
Control	0 (6)	29 (23)	29 (29)
Treatment	0 (7)	34 (27)	34 (34)
Total	0 (13)	63 (50)	63 (63)

Annotation: Empirical data in brackets.

Table 4.5: Crosstables to compare simulated and empirical data for DR2 with optimized ans for combined data.

	False	True	Total
Control	17 (18)	17 (16)	34 (34)
Treatment	18 (9)	11 (20)	29 (29)
Total	35 (27)	28 (36)	63 (63)

Annotation: Empirical data in brackets.

In Figure 4.25 one can see the linear shift to the top, which enforced the decision to incorporate the lf in the next optimization step.

The matching of the result data visualized in Table 4.4 and Table 4.5 shows that there is potential. The simulation yields more errors than the empirical data shows, specially in the treatment conditions. This enforced tuning the rt parameter.

4.1.4.2 *Tuning rt, ans and lf*

For tuning the three parameters together we enlarged the space for the ans parameter. Therefore for the three subsequent optimizations we used the search space from -10 to 10 for rt, 0.01 to 2 for ans and 0.1 to 5 for lf. We let the BO algorithm run for one thousand simulations. Like before, we present the results for the best case for all three optimization processes. The results on the single snippet level for the best rt, ans and lf parameter combination are visualised in Appendix C.

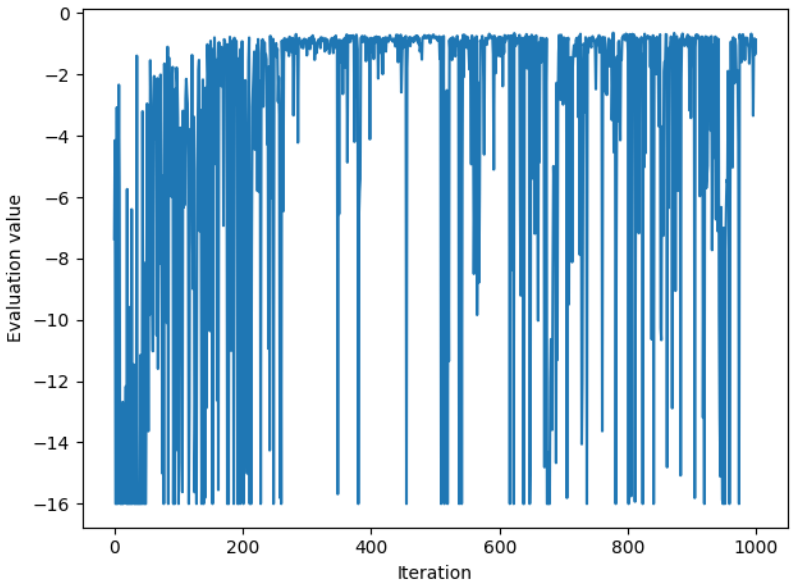
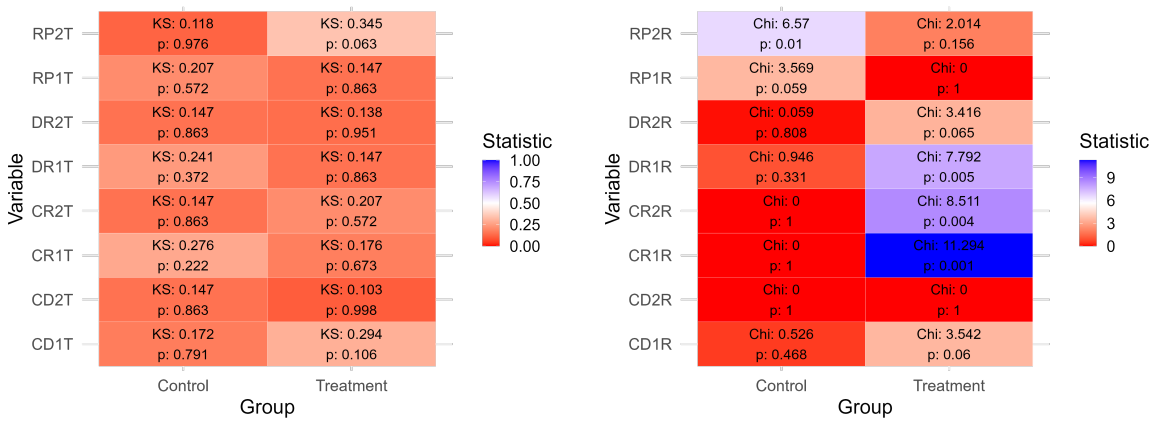


Figure 4.26: Evolution of the BO algorithm with 50 start points for the tuning rt, ans and lf for time data.

Time For the time data, the best reachable target value for the negated sum of squared KS statistics was -0.638 and therefore only one fifth of the result tuning only ans. This value was reached at point 778 and the parameters are: rt = -5.2685, ans = =0.8307 and lf = 0.4473. As one can see in the evolution of the BO algorithm in Figure 4.26 this result may be not optimal, since there is no clear convergence so far. However, due to time constraints, the simulation was ended after 1000 episodes, which took about 4 days.

The fit of the time has improved significantly with the addition of the two parameters, as can be seen in Figure 4.27a. All KS values are below 0.4 and all significance values are above 0.05, which means that the probability that the simulated and empirical data have the same distribution is correspondingly high. Expanding the parameter space did not change the results for error rate much. Most of the results stayed the same, some areas got worse or



(a) Heatmap for matching the time distribution using KS statistics.

(b) Heatmap for matching the error rate distribution using χ^2 statistics.

Figure 4.27: Heatmaps for the tuning rt, ans and lf for time data.

even better.

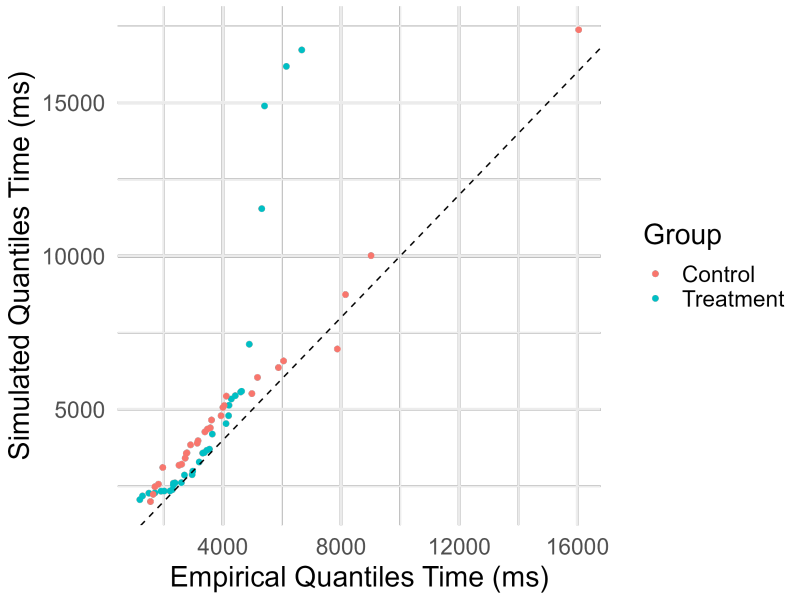


Figure 4.28: QQ-Plot for CR1 with optimized rt, ans and lf for time data.

The fact that the addition of the parameters has improved the fit of the time can be seen in the QQ plots. While the points were still clearly above the diagonal in the pure optimization of ans (see Figure 4.20) One can see in Figure 4.28 and that they have moved closer to the line, in some cases even lying on the line. The lf of 0.4473 is less than half the default value.

Error Rate For the error rate data, the best reachable target value for the negated sum of squared χ^2 values was -115.7197, nearly half of only tuning ans. This value was reached at point 832 and the parameters are: rt = -1.5124, ans = 0.2073 and lf = 1.442. This parameter set

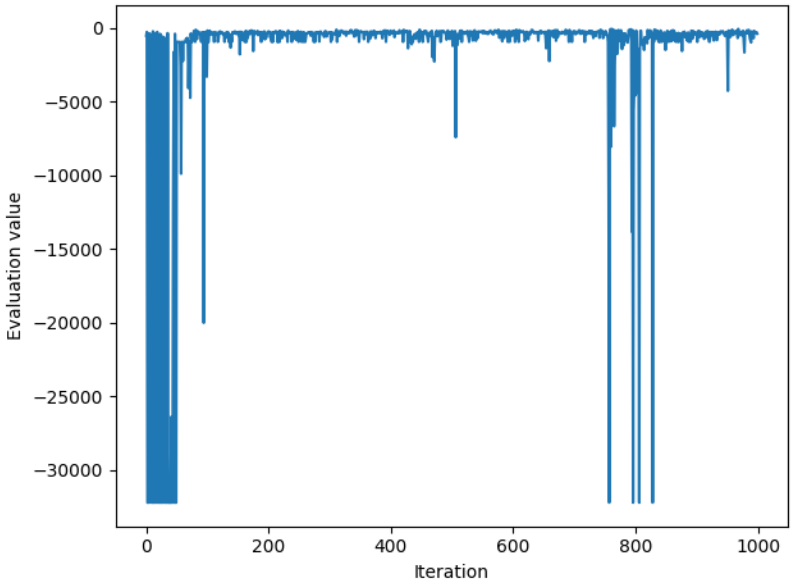
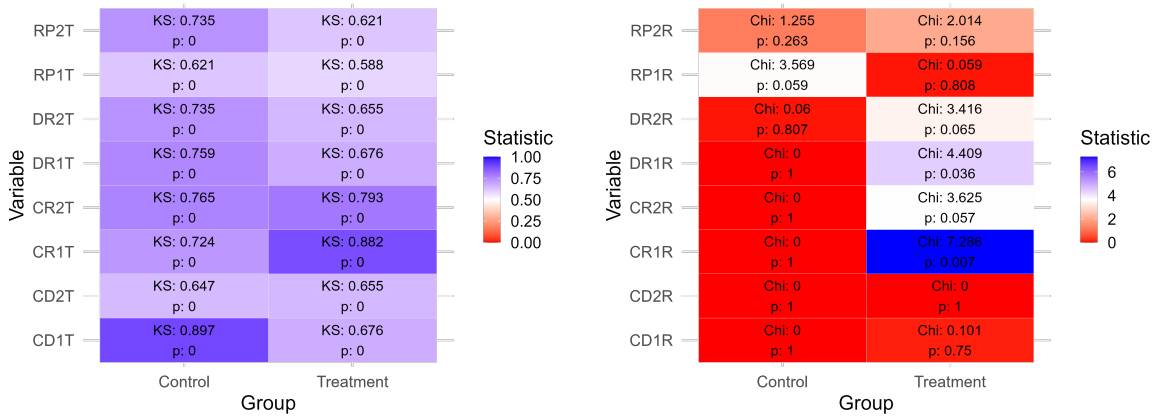


Figure 4.29: Evolution of the BO algorithm with 50 start points for the tuning rt, ans and lf for error rate data.

is very different compared to optimising time. The increased latency factor in particular stands out compared to the default value. As one can see in Figure 4.29 the high range for the hyperparameters leads to a high range for the χ^2 values.



(a) Heatmap for matching the time distribution using KS statistics.

(b) Heatmap for matching the error rate distribution using χ^2 statistics.

Figure 4.30: Heatmaps for the tuning rt, ans and lf for error rate data.

As can be seen in Figure 4.30b, the matching of the result data is the best so far. Only two fields have a significance value of less than 0.05. The drawback is that the time data do not match at all, and all 16 time distributions differ significantly between simulations and empirical data.

Table 4.6: Crosstables to compare simulated and empirical data for CR1 with optimized rt, ans and lf for error rate data.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	12 (2)	22 (32)	34 (34)
Total	12 (2)	51 (61)	63 (63)

Annotation: Empirical data in brackets.

Table 4.7: Crosstables to compare simulated and empirical data for DR2 with optimized rt, ans and lf for error rate data.

	False	True	Total
Control	20 (18)	14 (16)	34 (34)
Treatment	17 (9)	12 (20)	29 (29)
Total	37 (27)	26 (36)	63 (63)

Annotation: Empirical data in brackets.

It is worth noting that although the fit has improved significantly, not much has changed in our example snippets. Both in CR1 (see Table 4.6) and for DR2 (see Table 4.7), the simulation predicts more errors in the treatment group than were actually observed.

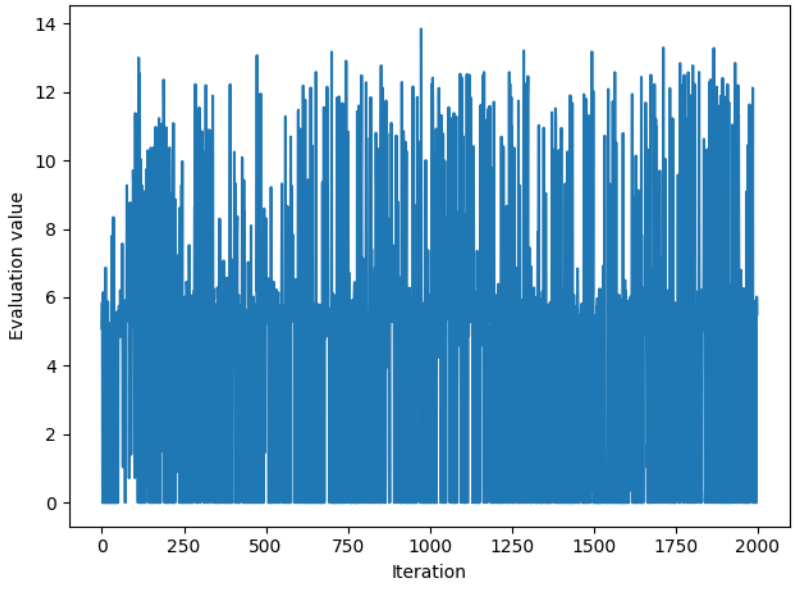
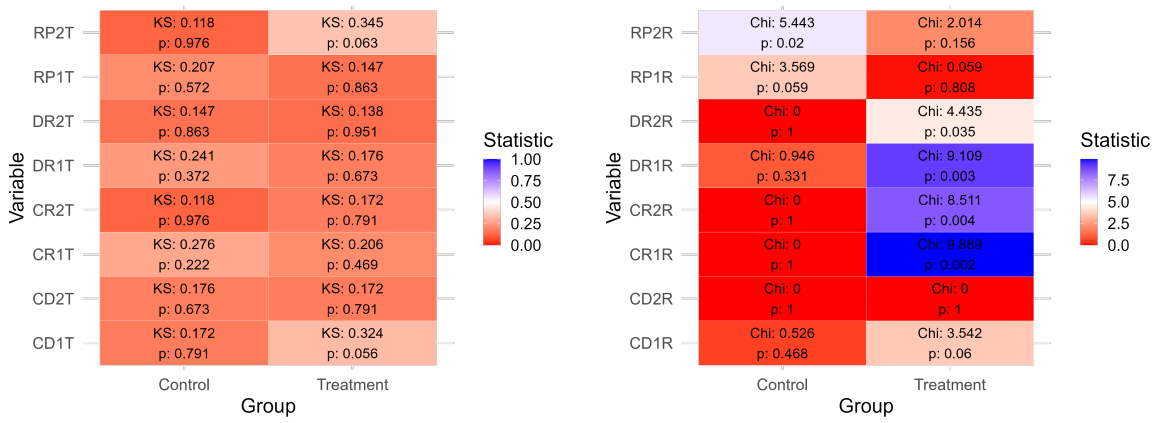


Figure 4.31: Evolution of the BO algorithm with 50 start points for the tuning rt, ans and lf for combined data.

Combined For the combined time and error rate optimization, the best reachable target value for the sum of squared significance values was 13.84. This value was reached at

point 975 and the parameters are: rt = -4.968, ans = =0.7984 and lf = 0.4622. This parameter set is similar to the one obtained by the optimization only on time. And as one can see in Figure 4.31, there is possible potential for improvement, since there is no convergence so far.



(a) Heatmap for matching the Time Distribution using KS statistics.

(b) Heatmap for matching the Error Rate Distribution using χ^2 statistics.

Figure 4.32: Heatmaps for the tuning rt, ans and lf for combined data.

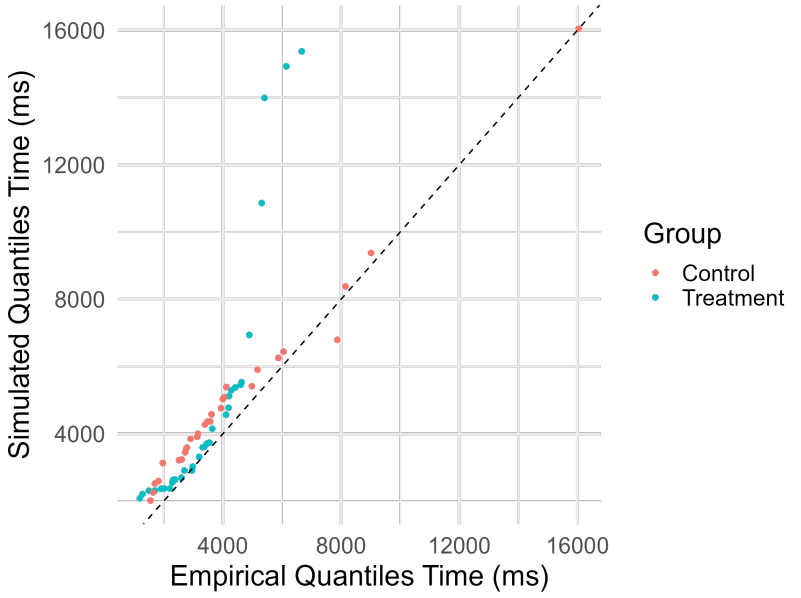


Figure 4.33: QQ-Plot for CR1 with optimized rt, ans and lf for combined data.

As one can see in Figure 4.32 the results are nearly the same compared to the pure time optimization. There is no significant value below 0.05 for the time data and only five for the outcome data, which is even slightly worse than the results for pure time optimization. In the QQ Plot in Figure 4.33 we can see the shift to the diagonal again, which illustrates the similarity of the data.

Table 4.8: Crosstables to compare simulated and empirical data for CR1 with optimized rt, ans and lf for combined data.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	14 (2)	20 (32)	34 (34)
Total	14 (2)	49 (61)	63 (63)

Annotation: Empirical data in brackets.

Table 4.9: Crosstables to compare simulated and empirical data for DR2 with optimized rt, ans and lf for combined data.

	False	True	Total
Control	17 (18)	17 (16)	34 (34)
Treatment	18 (9)	11 (20)	29 (29)
Total	35 (27)	28 (36)	63 (63)

Annotation: Empirical data in brackets.

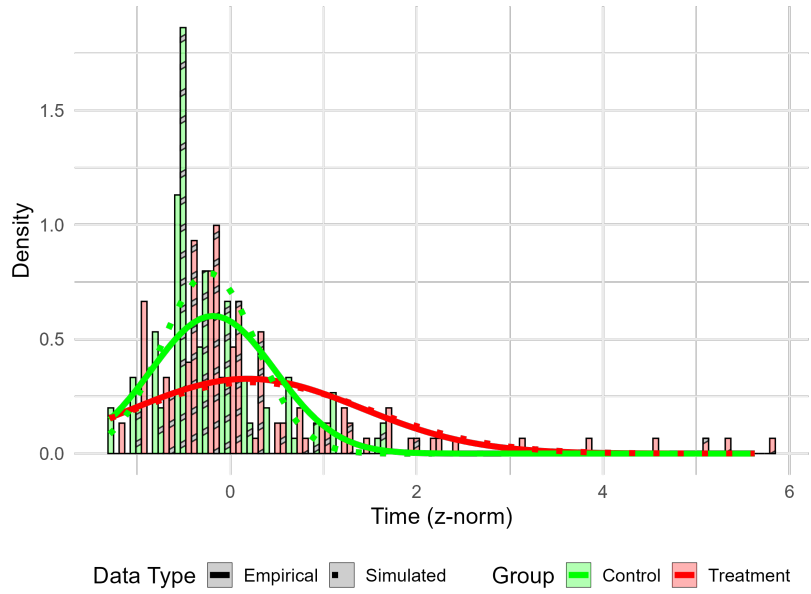
For the result data, the same observation as before can be made: the simulation predicts a higher error rate than was observed for the treatment group in CR1 (see Table 4.8) and DR2 (see Figure 4.9).

4.1.5 Model Evaluation

Finally, we have to evaluate the model. Therefore, we performed the same evaluation for the artefact effects as for the empirical data. For the evaluation we used the results from the combined optimization in Section 4.1.4.2. The results are compared to the empirical findings. For this we used the z standardization and combined both snippets of a class. For single snippets the comparision is visualized in Appendix D. For all values given in this section, the corresponding values from the empirical analysis are written again in brackets to allow a direct and simple comparison.

4.1.5.1 *Code Distance*

For the CD class the control group matches near perfectly. The comparison of the results for the CD snippets is shown in Figure 4.34. The distribution of the time values is shown above. The dashed lines and striped bars are the simulated data, while the solid lines and full-colored bars again reflect the empirical values we examined in Section 4.1.3. The bottom part shows the evaluation of the results. The empirical data is shown in brackets. As one can see, the row sums and the total sum are identical, which was the goal of our simulation with the same amount of data. This type of presentation will be used for all subsequent presentations of effects.



(a) Time on target line.

(b) Error rate.

	False	True	Total
Control	5 (7)	58 (56)	63 (63)
Treatment	1 (8)	62 (55)	63 (63)
Total	6 (15)	120 (111)	126 (126)

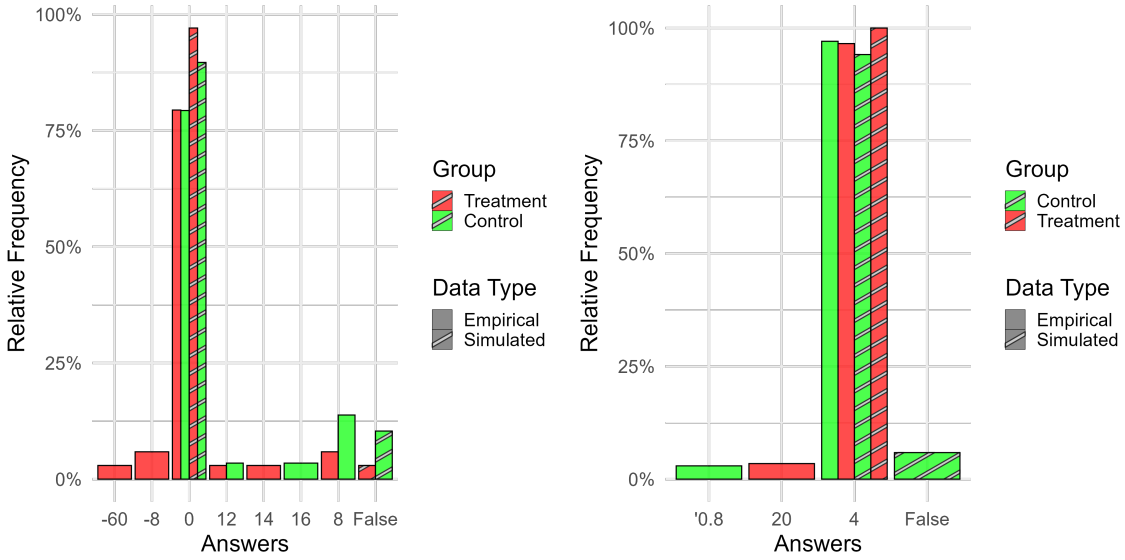
Annotation: Empirical data in brackets.

Figure 4.34: Results for the combined data of both CD snippets for treatment and control group compared between empirical data and simulated data.

Time Data As illustrated in Figure 4.34a, the distribution curves of the treatment group exhibit minimal variation, whereas the distributions of the control group demonstrate a more pronounced peak compared to the empirical data.

The Shapiro-Wilk test yielded a value of W = 0.585 (0.845) with p = 4.615e-12 (1.31e-06) for the treatment group and a value of W = 0.784 (0.932) with p = 3.294e-08 (1.813e-03) for the control group. This indicates that both groups are not normally distributed. The results of the Levene test yielded an value of F = 4.2081 (5.969) with p = 0.04234 (0.01597), indicating the presence of heteroscedasticity. Consequently, the findings from the Welch-test will be employed in the subsequent analysis. The mean value for the treatment group is 0.23 (0.19), with a standard deviation of 1.28 (1.22). The control group exhibited a mean value of -0.23 (-0.19) and a standard deviation of 0.51 (0.66). With a group size of 63 (63) per group, this yields a t-value of t = 2.6383 (2.125) with p = 0.009991 (0.03617). The effect size of Cohen's D is d = 0.470 (0.379), which can be regarded as indicative of a small to medium effect.

A comparison of the empirical data with the simulated data indicates that the primary conclusions derived from the artefact analysis remain broadly consistent. Both datasets exhibit non-normal distributions and heteroscedasticity. The mean values of the groups are somewhat further apart, with both standard deviations corresponding well with the empirical values. As a result of the slightly larger discrepancy, the t-value increases slightly, and the p-value decreases. Nevertheless, the statistical significance of the difference will only become apparent after the BH correction. The effect increases slightly, but is still classified as a medium effect.



(a) CD1. (b) CD2.

Figure 4.35: Distribution of the answers for the CD snippets for simulated data.

Error Rate Data In contrast to the time data, there has been a change in the error rate data. As illustrated in Table 4.34b, while the empirical data indicates that the number of errors between the control and treatment groups is nearly identical, the simulated data indicates that the control group makes more errors.

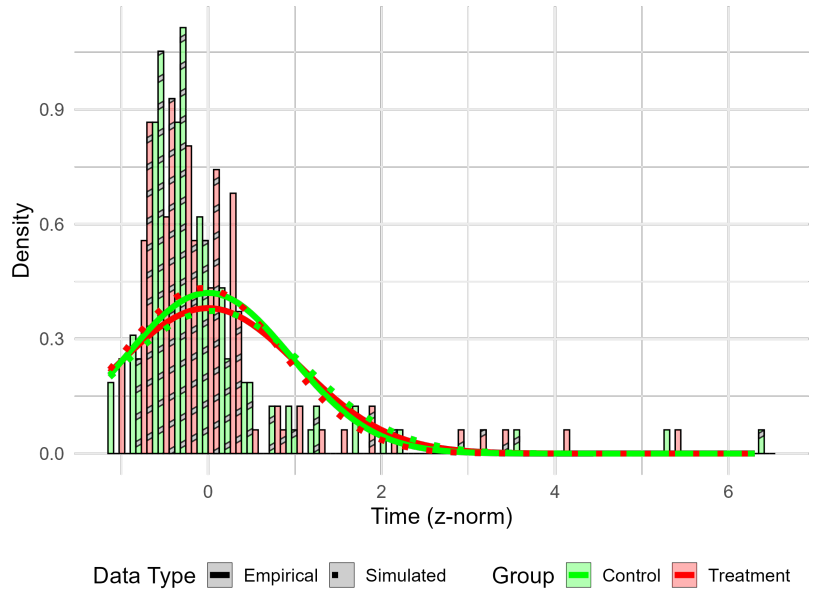
Consequently, the χ^2 test yielded a value of $\chi^2 = 1.575\,(0)$ with $p = 0.2095\,(1)$. Since the assumption of at least five observations per field is again violated here, Fisher's Exact test provides $p = 0.2074$ as an alternative. Although the difference between treatment and control is not statistically significant, it differs from the empirical data found. The effect size is $\phi = 0.149$ (-0.025), which indicates a small effect, and the direction of the effect is even reversed.
The results remain unchanged by the changes, but the change in the treatment group should be taken into account in the further interpretation, even if the difference has not changed the statistical significance.

If we now look at the distribution of the answers in Figure 4.35 for the simulated data, it is noticeable that the ACT-R model made fewer different errors. The "False" class has

been added and includes all those cases in which a retrival failure resulted in no chunk being retrieved from memory, which is why no further calculation was possible. For the CD snippets, either the correct answers or "False" are output. The correct answers remain unchanged, of course.

4.1.5.2 *Repeated Code*

While the distributions of times in the class of CR snippets exhibited only slight differences between empirical and simulated data, the result data exhibited a substantial change.



(a) Time on target line.

(b) Error rate.

	False	True	Total
Control	1 (0)	62 (63)	63 (63)
Treatment	25 (3)	38 (60)	63 (63)
Total	26 (3)	100 (123)	126 (126)

Annotation: Empirical data in brackets.

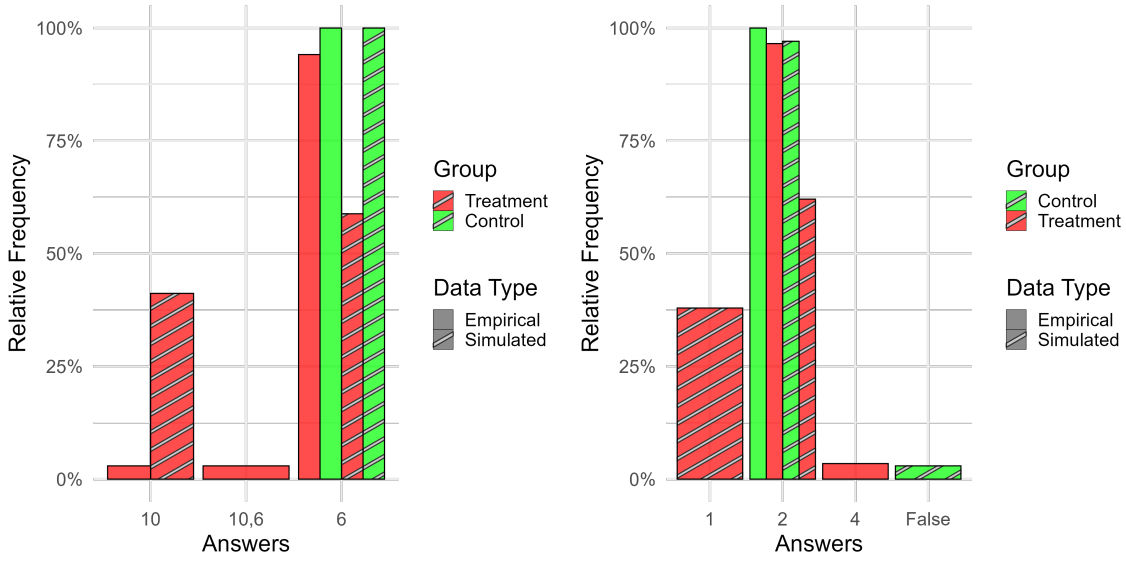
Figure 4.36: Results for the combined data of both CR snippets for treatment and control group compared between empirical data and simulated data.

The comparison of the results for the CR snippets is shown in Figure 4.36.

Time Data As illustrated in Figure 4.36a, both distributions exhibited a similar mean value, yet the distribution of the treatment group exhibited less variation than the distribution of the control group in the simulated data.

The Shapiro-Wilk test yielded a value of W = 0.683 (0.668) with p = 2.175e-10 (1.158e-10) for the treatment group and a value of W = 0.585 (0.681) with p = 4.564e-12 (2.062e-10) for the control group. This indicates that both groups are not normally distributed. The results of the Levene test yielded an value of F = 0.0028 (0.124) with p = 0.958 (0.725), indicating the absence of heteroscedasticity. Consequently, the findings from the t-test will be employed in the subsequent analysis. The mean value for the treatment group is -0.06 (0), with a standard deviation of 0.92 (1.05). The control group exhibited a mean value of 0.06 (0) and a standard deviation of 1.07 (0.95). With a group size of 63 (63) per group, this yields a t-value of t = -0.682 (-0.021) with p = 0.497 (0.983). The effect size of Cohen's D is d = 0.121 (0.00374), which is a small effect.

The observations from the empirical data can be transferred as far as possible. The effect size increases due to the higher mean difference, but the difference is not significant. All conclusions from the empirical data apply to the simulated data.



(a) CR1. (b) CR2.

Figure 4.37: Distribution of the answers for the CR snippets for simulated data.

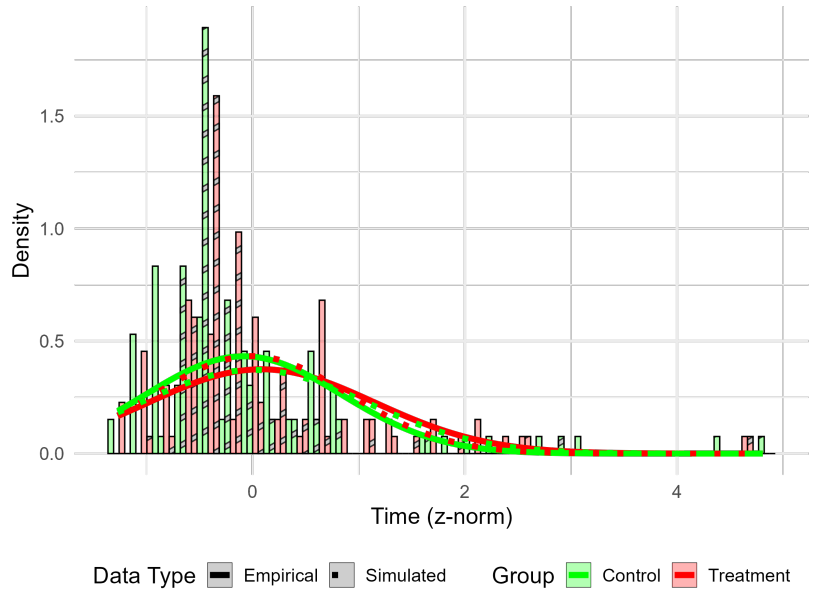
Error Rate Data As illustrated in Table 4.36b, there was a clear change in the frequency of errors in the treatment group. This difference is even highly significant, as indicated by the value of the $\chi^2 = 25.636$ (1.366), with a p = 4.122e-07 (0.242). For comparison, the Fisher's Exact test returns p = 4.999e-08 (0.244). This value is smaller than the smallest value of the BH correction, which is 0.00625. Consequently, the discrepancy observed in the simulated data is statistically significant, representing a clear divergence from the empirical data. The effect size is -0.471 (-0.156), which is classified as a medium effect.

This discrepancy represents a clear divergence between the model's predictions and the empirical findings. If one looks at the distribution of responses in Figure 4.37, one can see the difference between the empirical and simulated data. The treatment group in the

simulated data has an increased probability of emitting the response that occurs when the redeclaration was forgotten than was actually observed in the empirical data. This can be seen in the increased values of "10" for CR1 and "1" for CR2.

4.1.5.3 *Declaration Redeclaration Distance*

The times for the DR snippets agree well between the empirical data and the simulation, although the simulated error rate is higher than the empirical error rate.



(a) Time on target line.

(b) Error rate.

	False	True	Total
Control	25 (22)	38 (41)	63 (63)
Treatment	37 (15)	26 (48)	63 (63)
Total	62 (37)	64 (89)	126 (126)

Annotation: Empirical data in brackets.

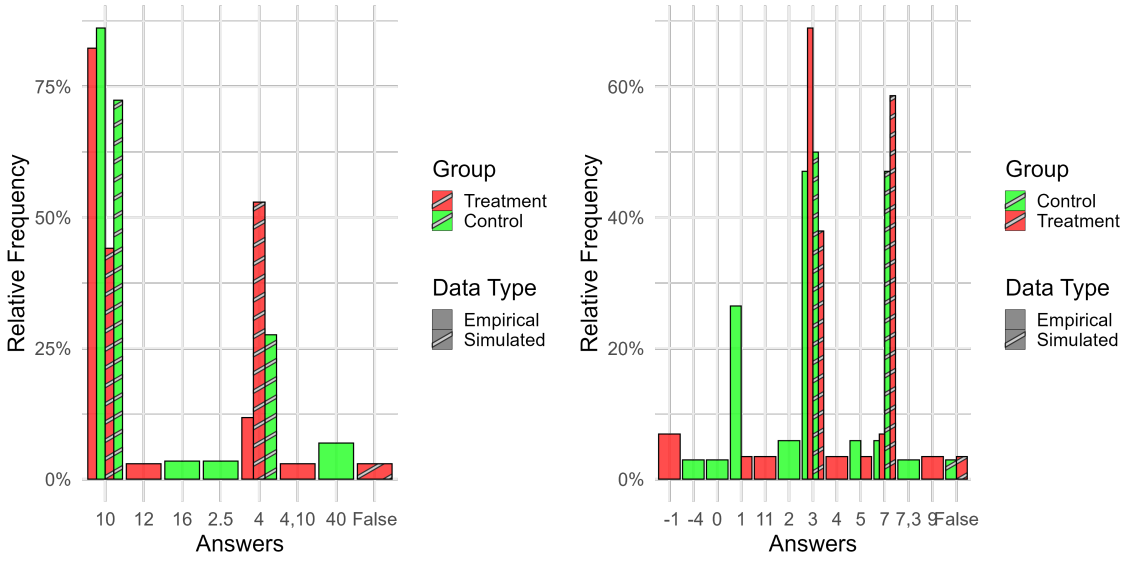
Figure 4.38: Results for the combined data of both DR snippets for treatment and control group compared between empirical data and simulated data.

The results for the DR snippets are presented in Figure 4.38.

Time Data As illustrated in Figure 4.38a, both distributions have swapped their position from empirical to simulated data. Thus, the time on the treatment snippets for the simulations lies more on the distribution of the control snippets for the empirical data and vice versa.

The Shapiro-Wilk test yielded a value of W = 0.672 (0.883) with p = 1.398e-10 (2.226e-05) for the treatment group and a value of W = 0.573 (0.89) with p = 2.991e-12 (3.821e-05) for the control group. This indicates that both groups are not normally distributed. The results of the Levene test yielded an value of F = 0.0047 (0.5398) with p = 0.946 (0.464), indicating the absence of heteroscedasticity. Consequently, the findings from the t-test will be employed in the subsequent analysis. The mean value for the treatment group is -0.01 (0.09), with a standard deviation of 0.92 (1.07). The control group exhibited a mean value of 0.01 (-0.09) and a standard deviation of 1.07 (0.92). With a group size of 63 per group, this yields a t-value of t = -0.084029 (-0.96) with p = 0.9332 (0.339). The effect size of Cohen's D is d = 0.015 (0.171), which is negligible.

Interestingly, both distributions have actually swapped their distributions and moved closer together. The effect on time, which was already barely present in the empirical data, became even smaller and statistically less significant.



(a) DR1. (b) DR2.

Figure 4.39: Distribution of the answers for the DR snippets for simulated data.

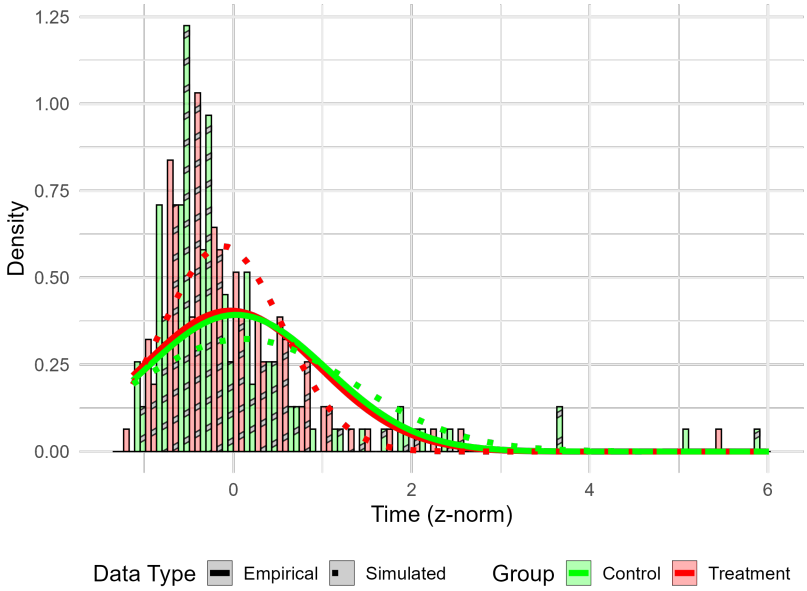
Error Rate Data As illustrated in Table 4.38b, there were even more errors in the simulations than in the empirical data and the direction of the effect was reversed. If the control group was more prone to errors in the empirical data, the treatment group was more prone to errors in the simulations. This effect is a candidate for statistical significance, as indicated by $\chi^2 = 3.8422$ (1.378) with p = 0.04998 (0.24). Whether this effect is statistically significant can only be concluded with the BH correction. The effect size is Φ = -0.191 (0.122), which is small effect. The changed sign illustrates once again the reversal of the effect direction.

However, the model predicted the high error rate in this snippet class, which should be taken as an insight from these results. But, it is noticeable in the distribution of the answers, shown in Figure 4.39, that the possible answers in the empirical data are much more widely

scattered than in the simulated data. The errors in the simulated data are either due to a retrieval failure or an incorrect retrieval of the already overwritten variable.

4.1.5.4 *Repeated Distance*

For RP snippets, we observe an amplification of the temporal effect and a mitigation of the effect on the error rate. Simultaneously, the error rate itself increases.



(a) Time on target line.

(b) Error rate.

	False	True	Total
Control	26 (9)	37 (54)	63 (63)
Treatment	29 (21)	34 (42)	63 (63)
Total	55 (30)	71 (96)	126 (126)

Annotation: Empirical data in brackets.

Figure 4.40: Results for the combined data of both RP snippets for treatment and control group compared between empirical data and simulated data.

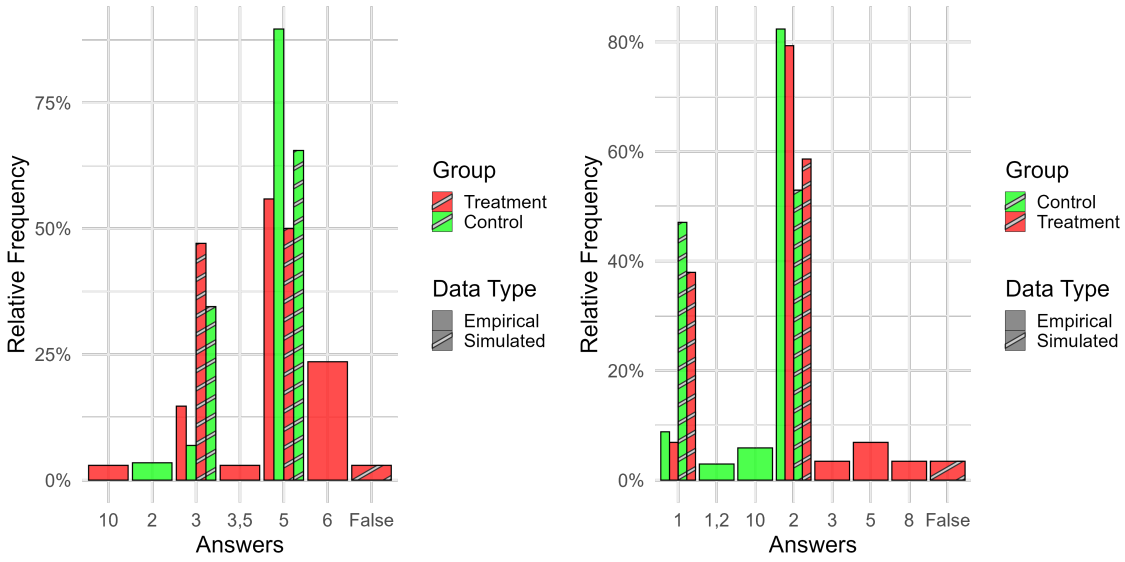
The results of RP snippets are presented in Figure 4.40.

Time Data As illustrated in Figure 4.40a, the distributions in the empirical data were quite close to and overlapping with each other. In the simulated data, the means have moved slightly further apart, but the most standout difference lies in the variance between the two groups. The treatment group has less variance than the control group.

The Shapiro-Wilk test yielded a value of W = 0.86456 (0.734) with p = 5.417e-06 (2.404e-09) for the treatment group and a value of W = 0.65564 (0.759) with p = 6.954e-11 (8.328e-09) for

the control group. This indicates that both groups are not normally distributed. The results of the Levene test yielded an value of F = 1.4051 (0.045) with p = 0.2381 (0.8323), indicating the absence of heteroscedasticity. Consequently, the findings from the t-test will be employed in the subsequent analysis. The mean value for the treatment group is -0.12 (-0.03), with a standard deviation of 0.68 (0.98). The control group exhibited a mean value of 0.12 (0.03) and a standard deviation of 1.23 (1.01). With a group size of 63 (63) per group, this yields a t-value of t = -1.3266 (-0.3697) with p = 0.1871 (0.712). The effect size of Cohen's D is d = 0.236 (0.0659), which is a small effect.

The interpretation of the effect changes very little. The effect remains statistically insignificant and has only gained slightly in strength.



(a) RP1.

(b) RP2.

Figure 4.41: Distribution of the answers for the RP snippets for simulated data.

Error Rate Data As illustrated in Table 4.40b, the error rate increased in both groups in the simulated data, with a stronger increase in the control group, resulting in significantly smaller differences between the two groups.This is reflected in the changed test statistics. With a test value of $\chi^2 = 0.12907(5.294)$ and p = 0.7194 (0.0214), this effect has lost statistical significance and is no longer a candidate for being considered significant after the BH correction. The effect size is Φ = -0.048 (-0.224), indicating a now negligible effect.

This change demonstrates that an effect observed in the empirical data can disappear in the simulated data. As one can see in the distribution of answers in Figure 4.41, one of the biggest changes is that the control groups in both snippets make an error due to an incorrect retrieval more often. This accumulation seems to cancel out the difference between treatment and control in the simulations.

4.1.5.5 *Bonferoni-Holm Correction*

Now that we have considered all eight tests on the simulated data, they must be examined for statistical significance. For the simulated data, three of the eight tests fall below the threshold of $p = 0.05$. As with the empirical data, we do not consider the other tests further. The three remaining values are the difference in times for CD with $p = 0.009991$ and the differences in the error rate for CR with $p = 4.122e\text{-}07$ and DR with $p = 0.04998$.

As a reminder, the BH correction gives us bounds of $\frac{0.05}{8}$, $\frac{0.05}{7}$, ..., $\frac{0.05}{2}$, $\frac{0.05}{1}$ for the significance levels in ascending order of size. So the limits for the first three are $p < 0.00625$, $p < 0.00714$, and $p < 0.00833$. 4.122e-07 is smaller than 0.00625. Therefore, the error rate effect for the CR snippets is significant. 0.009991 is greater than 0.00714, and 0.04998 is greater than 0.00833. Thus, these two effects are no longer considered significant after the BH correction. In conclusion, one highly significant effect remains in the simulated data, which was not found in the empirical data.

4.2 Discussion

We will now critically reflect on, categorise and discuss the obtained results.

4.2.1 Artefacts

First, we will discuss the artefacts. Unfortunately, none of the effects were significant, so for research questions one to six, we must initially conclude that there is no statistically significant effect, neither for time nor for error rate. Nevertheless, it would be short-sighted to stop our analysis here.

We observed two effects with a very low probability of occurring purely due to chance. One is the longer time for the treatment group with the CD snippets, and the other is the higher error rate for the treatment group with the RP snippets. These two effects did not reach significance because we explored a variety of different effects. Had these been the only two effects examined, they would have been significant even under the BH correction. Furthermore, the sample size was smaller than our targeted 100 participants, which would have increased the test power.

In the following sections, we will examine each artefact individually and discuss the insights we can draw from the results.

4.2.1.1 *Code Distance*

What stood out with the CD snippets was the longer time participants needed for the treatment snippets. This difference was more pronounced for CD2, in which the filler lines were not used in the target line. Although there were fewer errors in this task, the difference

in time was quite prominent.

This could indicate that the use of variables far from their declaration involves greater cognitive load, as it takes more time to retrieve the variable's value from memory. When multiple variables are used, this effect seems to be mitigated. This could be because in working with multiple variables at least one variable declaration will always be farther away, making their order less impactful. Future research could investigate whether there are more or less advantageous declaration orders when using multiple variables.

However, it is evident that the presence of multiple variables increases the range of possible errors, and the overall error rate is higher in the CD1 snippet. Since this increase was observed in both the treatment and control groups between CD1 and CD2, it suggests that the distance to the single variable is not the cause, as this pattern was used in both snippets. Instead, it is likely due to the fact that CD1 uses multiple variables on the target line.

Therefore, we can conclude that using multiple variables in a single line may increase error susceptibility during comprehension, and the distance from a declaration to its use may increase the time required for understanding. The interaction of both factors could increase the workload on working memory, thereby overall hindering code comprehension.

4.2.1.2 *Repeated Code*

For the CR snippets, the time data for the two snippets canceled each other out. While the treatment group was faster than the control group for CR1, the control group was faster for CR2. Unfortunately, no explanation for this could be found. Both snippets differ only in the variable names and values, and in the calculation on the target line: Multiplication in one case and division in the other. For both snippets, the errors made occurred exclusively in the treatment group, reflecting the effect observed by Gopstein et al. [25].

In summary, there are slight indications that repeated code might increase the likelihood of making errors during code comprehension. However, this effect was so weak that it lacked statistical significance and could just as easily be attributed to chance.

4.2.1.3 *Declaration Redeclaration Distance*

For the DR snippets, the treatment group took slightly longer than the control group. The temporal proximity seems to influence how easily the declaration can be distinguished from the redeclaration.

In general, it can be said that both snippets have an increased error probability and that the expected error (incorrectly recalling the outdated value) occurred more frequently in the treatment group than in the control group. In DR2, however, a different source of error dominates in the control group: the order of variable declarations seems to make a difference. While in the treatment group the variables are accessed in the reverse order of their usage in the term, this changes in the control group because we had moved the redeclaration directly before the target line. This seems to encourage the error of swapping the other two

variables during the subtraction. This extends the discussion begun in Section 4.2.1.1 that there could be order effects in variable declaration, which positively or negatively affect comprehensibility and error probability.

In summary, the findings from this type of snippet indicate that the proximity of a redeclaration to the original declaration could negatively impact code comprehension by increasing the time needed to comprehend. Additionally, it could increase the error probability. However, caution should be exercised during refactoring, as there could be a dominant effect due to the order of the variables.

4.2.1.4 *Repeated Distance*

The observations for the DR snippets are further supported by the observations for the RP snippets. The lack of a difference in time might be due to the double declaration being shifted as a block, while the other variables are used on the target line in both snippets.

Moreover, it is evident that a double declaration farther from the target line increases the error rate more than when it occurrs closer to the target line. This is particularly noticeable in RP1, where the expected error due to the forgotten redeclaration appears, but the number of different errors is higher in the treatment group. Since this effect only narrowly missed statistical significance, we believe it would be worthwhile to investigate this effect again with greater test power.

4.2.1.5 *Insights from All Artefacts*

The results of our empirical study across all artefacts suggest several insights regarding code comprehension time and error rates.

Effects That Increase Time Several factors were observed which increase the time required for code comprehension. First, variables that are declared far from their usage might involve greater cognitive load, increasing the time needed to retrieve the variable's value from memory. Additionally, the proximity of a redeclaration to the original declaration could increase the time needed to understand the code.

Effects That Increase Error Rates Several factors were observed to increase the error rates during code comprehension. The presence of multiple variables might increase the range of possible errors, as reflected in the higher error rate. Double declarations could increase the likelihood of errors during code comprehension. Although this effect was weak and lacked statistical significance, it was consistently observed across different snippets. A double declaration farther from the target line increased the error rate more than when it was closer. Additionally, the order in which variables are declared might influence error rates.

Conclusions on What to Consider Based on these insights, several recommendations can be made to reduce code comprehension time and error rates. It is important to consider the distance between variable declarations and their usage to reduce cognitive load and the

required time for understanding them. Caution should be exercised with double declarations, as they might increase error rates, particularly when occurring far from the target line. Furthermore, attention should be paid to the order of variable declarations, as it might impact error rates and comprehensibility. During refactoring, the proximity of redeclaration and the order of variable declarations should be carefully considered to minimize cognitive load and error susceptibility.

4.2.2 Socio-Demographic Data

In addition to the data on the code snippets, we collected a substantial amount of metadata to better analyze the composition of our study participants. Due to the extensive analysis of artefacts and model predictions, analyses exploring the interaction between this metadata and our investigations were somewhat neglected. These could be part of a deeper analysis of the dataset.

For example, there are tendencies indicating that participants with more programming experience are less prone to errors caused by double declarations compared to others. This can be observed when programming experience is measured as the average self-assessment compared to peers and logical programming skills, and then split the group at value three. Those with higher self-assessments make these errors less frequently.

Such and other effects might still be hidden within the dataset, awaiting discovery. Consequently, it might be necessary to extend the model to include these parameters to account for these effects, potentially through different sets of hyperparameters for experienced and inexperienced programmers. However, a larger dataset should be used for this exploration, as splitting the data further reduces the sample size, making model fitting more challenging due to the dominance of noise.

Therefore, while the collected socio-demographic and other metadata can serve as a basis for additional meta-analyses, the limited time frame and scope of this work prevents us from delving further insights into this aspect. We hope others may feel called to undertake this task.

4.2.3 Model Fitting

Tuning the model fitting exclusively with the parameter ans demonstrated that the tuning results vary depending on the target value, making it challenging to simultaneously optimize for both time and error rate. Consequently, it is evident that the optimal parameter set might differ based on the optimization objective, and achieving a balanced optimization for both metrics remains a challenge.

For all optimizations, the treatment groups CR1, CR2, and DR1 proved difficult to optimize effectively. This difficulty arises because the empirical data for these groups exhibited fewer errors than the model simulations predicted with its current rules. This suggests that the

model may require further refinement to more accurately represent these cases.

Using the significance level as a combined measure warrants critical evaluation, as the resulting interpretations can differ substantially. For instance, a p-value of 1 for the KS statistic is almost never observed, whereas it frequently appears 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.

Optimizing with a reduced parameter set proved worthwhile, as we were able to simulate time data with high precision using only three parameters. This underscores the suitability of the underlying rule set for the given task. Since we did not modify many parameters, it must be the rules themselves that enable this fit.

Exclusive optimization for the error rate compromised the time metric, suggesting potential loopholes between parameters and rules that allow the model to produce outputs better fitting the data at the expense of time. Conversely, this does not seem to apply, as the final results of pure time optimization and combined optimization on the significance level differed minimally in terms of resulting parameters and data fit.

4.2.4 Model Evaluation

In this section, we will discuss the extent to which the final tuned model simulates the indicated effects in the empirical data and what insights we can derive from this.

4.2.4.1 *Code Distance*

In the CD snippets, the effect of the treatment group taking slightly longer on the target line than the control group was even more pronounced. This could be partially due to the smaller variance of the control group in the simulated data and the larger difference between the groups. This might indicate that the model effectively captures the impact of greater distance between declaration and usage.

However, the model seems to poorly represent the error probability in these snippets. Not only is the error rate lower than in the empirical data, but the types of errors are obviously more limited. While many different errors appeared empirically in CD1, the model can only produce the correct answer or a "false" answer due to retrieval failure in this type of snippet. This reveals an area for potential improvement. By using similarity matrices, the retrieved values could be made more probabilistic, allowing for unexpected outliers and answers without a retrieval failure, which, due to the architecture, affects the time.

The fact that more errors occurred in the control group could be because the failures can only be caused by a retrieval failure. Since the noise component is significantly involved in the calculation of chunk activation, it might be simply by chance that the control group had more retrieval failures than the treatment group.

In summary, for this type of snippet, the model seems to overestimate the effect of distance on time and underestimate its impact on the error rate.

4.2.4.2 *Repeated Code*

For the CR snippets, the model's predictions deviate from the empirical findings. While the temporal effects of the two snippets offset each other in the empirical data, the simulated data shows that the treatment group is faster than the control group. This is initially counter intuitive, as the treatment group should be confused by the double declaration. This effect can be explained by the fact that chunks that are triggered more frequently have higher activation and are thus retrieved faster. Therefore, the treatment group may have shorter retrieval times than the control group.

However, this effect of shorter retrieval time comes with a downside: a significantly increased error rate, which is statistically highly significant. This prediction of the model deviates considerably from the empirical data, where only a slight trend was observed, which could just as well be due to chance. This discrepancy clearly indicates that the model still requires optimization to accurately predict error probability, as it significantly overestimates it in this case.

In summary, for this type of snippet, the model significantly overestimates the error probability and slightly overestimates the temporal effect.

4.2.4.3 *Declaration Redeclaration Distance*

For the DR snippets, the model seems to slightly underestimate the effects on time. The difference between the groups in the simulation is smaller than in the empirical data. However, both the effect in the empirical data and the effect in the simulated data are negligible. Overall, the fit of the temporal data could be described as good. It appears that in the model, the timing of the redeclaration makes little difference.

The error rate, however, presents a different picture. There is a significantly higher probability that the treatment group will make an error compared to the control group. This is because the model cannot capture the potential sequence effect between declaration and usage in snippet DR2, as evidenced by the fact that the simulation never provides response "1". The model's errors are limited to retrieval failures and incorrect retrievals of the old variable value.
This reveals a weakness in the model: It cannot represent such effects. One possible enhancement could involve using a similarity matrix to assign similarity scores to variable names. For instance, "q" and "u" sound relatively similar, and "q" and "p" look quite similar, which might explain the effect. Therefore, this could be a meaningful extension.

In summary, for this type of snippet, the model accurately represents the timing but overestimates the error probability associated with redeclaration while failing to capture errors caused by sequence effects.

4.2.4.4 *Repeated Distance*

For the RP snippets, the model predicts a larger temporal effect than what was actually observed in the empirical data. Interestingly, the treatment group is faster than the control group, even though in the treatment group the double declaration was further from the target line than in the control group. This could be because the other variables used in the target line were moved further away due to refactoring, and the double declaration was even further away in the treatment group. This might explain the temporal effect in the simulation. However, this effect was not observed in the empirical data, indicating that the model overestimates the impact here.

Conversely, the model underestimates the influence on the error rate when a double declaration is further from or closer to the target line. While there is a slight effect in the empirical data, it is almost entirely absent in the simulation. Moreover, the types of errors differ. Notably, the model is unable to generate the response "6" for snippet RP1, which may be attributed to the misassignment of values to variables, and predominantly predicts errors due to incorrect retrieval processes.
Again, applying a similarity matrix would likely be beneficial here, making the possible responses more probabilistic.

In summary, for this type of snippet, the model overestimates the temporal effect and underestimates the effect on the error rate. This discrepancy is partly due to the types of errors observed in the empirical data, which the model cannot represent due to its rules.

4.2.4.5 *Insights from Model Evaluation*

The evaluation of our ACT-R model provides several important insights into its performance and areas for improvement. The model shows varying degrees of accuracy in predicting the temporal effects and error rates associated with different types of code snippets. Below, we summarize these insights, highlighting where the model performs well and where further refinement is needed.

Temporal Effects The model demonstrates a reasonable capability to simulate the temporal effects observed in code comprehension tasks. It effectively captures the general trend that increased cognitive load, such as when variable declarations are far from their usage, leads to longer comprehension times.

Furthermore, it predicts where code rearrangements do not cause a difference because other variables take longer to be retrieved, suggesting a balanced effect. However, the model often tends to overestimate these temporal effects. Specifically, it seems to overestimate the effect that variables which are declared more frequently are quicker to remember.

Error Rates The model's ability to predict error rates is less accurate. It frequently overestimates or underestimates the likelihood of errors. The model generally captures the idea that certain coding patterns, such as double declarations or complex variable interactions, increase the probability of errors. However, it tends to simplify the types of

errors and does not fully capture the diversity and nuance of errors observed in empirical data.

Model Improvements To address the identified shortcomings, several improvements can be made to enhance the model's accuracy. Enhanced error prediction could be achieved by incorporating similarity matrices, allowing the model to generate a broader range of errors that better reflect empirical observations. This would make error predictions more probabilistic and realistic.

Temporal adjustments, refining the model's parameters and rules, can help to better balance the time predictions. Adding rules to account for more complex interactions between variable declarations and usage could improve temporal accuracy. Developing mechanisms to capture sequence effects, particularly in the order of variable declarations, can help the model more accurately simulate error patterns seen in the empirical data. Continuously aligning model predictions with empirical data through iterative refinement and validation can ensure the model remains robust and accurate across different code snippets and scenarios.

In conclusion, while the ACT-R model shows potential in predicting certain temporal effects and error rates, significant room for improvement remains. By addressing these areas, the model can become a more powerful tool for understanding and optimizing code comprehension.

4.2.5 Why Should We Care? Analysing the Relevance of Our Findings

We have extensively discussed the potential causes of certain effects and the capabilities and limitations of our ACT-R model. But what can be done with all these insights?

Returning to the analogy of the software developer as a production unit, overburdening resources such as cognitive workload can impair productivity or even lead to errors. Therefore, unnecessary workload should be avoided. One challenge is to identify unnecessary workload, and an even greater challenge is to potentially predict it.
This is where the ACT-R model comes into play. In this work, we found indications that different code formats producing the same output might take varying amounts of time to comprehend and might differ in their susceptibility to comprehension errors.

Our ACT-R model was able to accurately predict parts of these effects, and despite there being significant room for improvement, the potential is evident.
In the domain of cognitive modeling, we tread well-worn paths, as the effects identified here are more or less known and based on simple foundational structures of ACT-R. This work does not provide new insights in the field of cognitive modeling. Why, then, are these results still interesting? It is the practical implication that accompanies these fundamental insights. We have shifted the focus. For instance, while models by Danker and Anderson [16] and

Lebiere [35] deal with the mathematical understanding and development of humans, where the stimulus material is merely a means to an end to gain insights into brain function, we take it a step further and also bring the stimulus material into focus.

Of course, it is in our interest to understand how the human mind processes code. However, we are equally concerned with making statements about the interaction between the stimulus material and the brain to make specific assertions about the stimulus material itself, such as whether it is difficult or easy to understand.

The model could now be used to predict which formatting of new, unseen code would require the least time to understand or have the lowest probability of being misunderstood. In the context of modern Large Language Models (LLMs) that can quickly generate large amounts of code, such an evaluation metric could facilitate the writing of more human-readable code.

One might argue that any of the many LLM versions available today could just as well be trained to produce more readable code. This is where the strength of cognitive models lies. While LLMs operates purely probabilistically, cognitive models are rule-based, meaning we can directly derive experiences from the models and gain insights into why something is the way it is. This offers the advantage of a precise understanding of the problem, simplifying the search for solutions and being able to explain it.

Of course, the model presented in this work is not sufficient to evaluate complex code, as it is only applicable to our highly reduced framework. However, we believe that the principle can be easily extended to incorporate more functionality, offering the possibility of developing an objective evaluation method from such a cognitive model.

Furthermore, a precise model that encompasses complex semantics could deepen our understanding of how code comprehension works. ACT-R offers considerable potential to predict eye movements and even brain activity such as the BOLD effect. This could provide a more nuanced understanding of code comprehension and open the door to associating the complexity of code directly with the necessary cognitive effort.

4.3 Threats to Validity

In the following, the most critical points of the methodology are reflected once again. This critical reflection of the experimental design is guided by Feitelson [21], who summarized the usual pitfalls in code comprehension studies.

Internal Validity

The stimulus material consists of short and very simple code snippets to ensure that both the test subjects and the ACT-R model can process them. The reasons for this include the reduced framework, line-by-line presentation, and limited working memory capacity.

Additionally, the selected snippets were artificially created by the authors. Although inspired by real examples, these effects may have been inadvertently induced by the design. These include effects based on the similarity of variable names and numbers, as well as sequence effects related to declaration and usage. We addressed these issues by using predefined schemas for generating the snippets and by randomizing variable names. The code snippets were tested in a pilot study after randomization, and no potential difficulties were identified during that phase.

The sample was not a purely random collection of participants, as recruitment was conducted through personal networks, on campus, and via survey pools. Consequently, there may be confounding factors in participant characteristics that could influence the results. We tried to minimize the influence of potential confounding variables by employing a mixed-subject design and randomly assigning participants to groups.

One problem that stood out in the results is the model's inability to produce errors other than incorrect retrievals. Furthermore, the model may contain bugs that were not identified in the pilot study, in addition to its lack of functionality. To test the model, we observed participants during the pilot study but did not detect a wide variation in errors, which limited our ability to further analyze these observations. We chose not to modify the model post hoc, as we could not rule out the possibility of being influenced by the already known results.

Regarding the optimization of hyperparameters, it should be noted that we worked with a limited set of possible hyperparameters, which was due to the constrained scope of this thesis. It is possible that other algorithms, aside from BO, might find better parameters more efficiently. Additionally, integrating different parameters could reveal other interactions and lead to different results. Our decision to use the BO algorithm was informed by findings from the literature, as was our choice to work with as few hyperparameters as possible.

In pursuit of reproducibility, a random seed key is used. This feature diverges from the standard ACT-R model, where each run is either identical or entirely random. In this variant, while the experiments themselves exhibit variability, the resultant data remains consistent. Furthermore, the ACT-R model operates through the same interfaces as humans, utilizing visual input and manual output, thereby enhancing comparability with human test subjects. Only a minor adjustment was necessary for the input of the results.

A critical consideration at this point is whether the simplified code in the experiments aligns too closely with the comprehensibility parameters of ACT-R. Given that both the model and the code style were developed in parallel, it is plausible that the model interprets different lines of code differently than a human would.
We tried to minimize the influence by adapting models originally designed for mental arithmetic and modifying them for code processing.
This may explain why the latency factor was smaller than its default value: the way the process was modelled was so inefficient that, despite using math facts as a simplification, the retrieval of chunks had to be accelerated to match the empirical data.

This potential discrepancy, however, falls beyond the scope of this thesis and remains an open question for subsequent studies.

Construct Validity

Within the framework of the ACT-R model, it is crucial to acknowledge that the comprehension of code processing in the human brain may not precisely align with the model's assumptions. The model functions as a simplification of reality designed for analytical purposes, necessitating further refinement and validation through subsequent experimental investigations. Notably, many issues associated with human subjects, such as learning effects and diverse knowledge backgrounds, are absent in the model. Since there was no existing model for code comprehension, we used a model for simulating mental arithmetic as a starting point and source of inspiration, as it seemed most appropriate given the similarity of the tasks. These types of models have a long history and are well-validated, so our modifications for handling variables are likely the only aspect warranting further investigation in future studies.

External Validity

To create snippets that are suitable for both the model and human participants, we had to rely on many assumptions and simplifications. These include a greatly simplified syntax, a line-by-line reading order, and the use of random letters as variables. Therefore, it is questionable whether the observations found here can be generalized to more complex programs or if they were merely favoured by the specific context.
To best meet these stringent requirements, we based our code on Python and drew inspiration from real-world code examples. We sought to minimize the influence of variable naming by employing randomization techniques to the greatest extent possible.

As mentioned in Section 3.6.1, referral-chain sampling does not provide a representative sample. Consequently, the findings may only apply to this selected sample, and different effects might be observed in a replication study. Various confounding variables, such as the participants' expertise, could influence the outcomes and act as confounders. Our sample is not representative, as the socio-demographic data indicates that we primarily studied men from Germany, aged 25 to 30, with a university degree and relatively high programming expertise.
To ensure that the sample was as heterogeneous as possible, we utilized a variety of starting points and channels to broaden the pool of potential participants. This approach was partially successful, as evidenced by the diversity in age distribution and nationalities.

The parameters of the model were tuned retrospectively based on the data collected in this study. Consequently, we cannot determine how well these parameters would generalize to other datasets or different contexts. The influence of the hyperparameters on the model's accuracy can be seen in the significantly lower accuracy of time predictions when optimizing for error rate. Thus, it can be concluded that these production rules, combined with the post hoc determined hyperparameters for these empirical data, allow for high accuracy in timing and moderate accuracy in error rate.

However, the production rules of the ACT-R model were implemented and established prior to any knowledge of the data and have proven to be quite accurate, at least in predicting time. This ensured that the data had no influence on the development of the model's production rules.
Whether these findings can be generalized to other datasets remains an open question.

5

Related Work

The purpose of this work was to implement a first simulatable cognitive model for source code comprehension based on ACT-R. Therefore, the literature research focussed on three different types of publications.
Firstly, we searched for publications that have developed theoretical models of how code comprehension is performed cognitively. These served as a basis for how code comprehension works and which specific domains it covers.
Publications that have created theoretical algorithms or concrete implementations based on the theoretical models were the second focus. These served as a starting point for an implementation. In addition, problems or limitations encountered by the authors could provide indications of what to look out for in a concrete implementation.
The last part included a search for topics in the field of software engineering that dealt with ACT-R. This showed the relevance of the work in the context of this work, which is why concrete and validated models can be helpful for further research.
In the following, the nine most relevant papers on which this thesis is based are briefly described, discussed and categorized.

The theoretical basics form the starting point of the literature research. The first step for mental models was laid by Brooks [6] in 1978. In his essay, he proposed that understanding a program is a successive refinement of the hypothesis about what the program does. Various parts of the program serve as cues to substantiate the hypotheses.
Half a decade later, he formulated his model, now known as the 'top-down model' [7] which is schematically visualized in Figure 5.1. Top-down means that in the process of program comprehension, from the first moment the process starts, hypotheses are generated as to what the program or individual parts of it do. These hypotheses are tested and refined. The testing is done on indicators or patterns that Brooks called 'beacons'. This is a highly complex and individual process. Therefore, it would go beyond the scope of this paper and was not be included in our model.

At the same time, Shneiderman [44] came up with his 'bottom-up model'. He followed an open approach and from his explorative studies, he put together his model. A graphical representation can be seen in Figure 5.2. The term 'chunk' was used for the first time in this context. In his model, chunks are information units that summarize the functional units of the program in a hierarchically structured way. His findings also show that programs are not memorized line by line but functionally in the form of an internal semantic representation. He also emphasized the difference between semantic knowledge and syntactic knowledge. Semantic means programming concepts that are independent of the language,

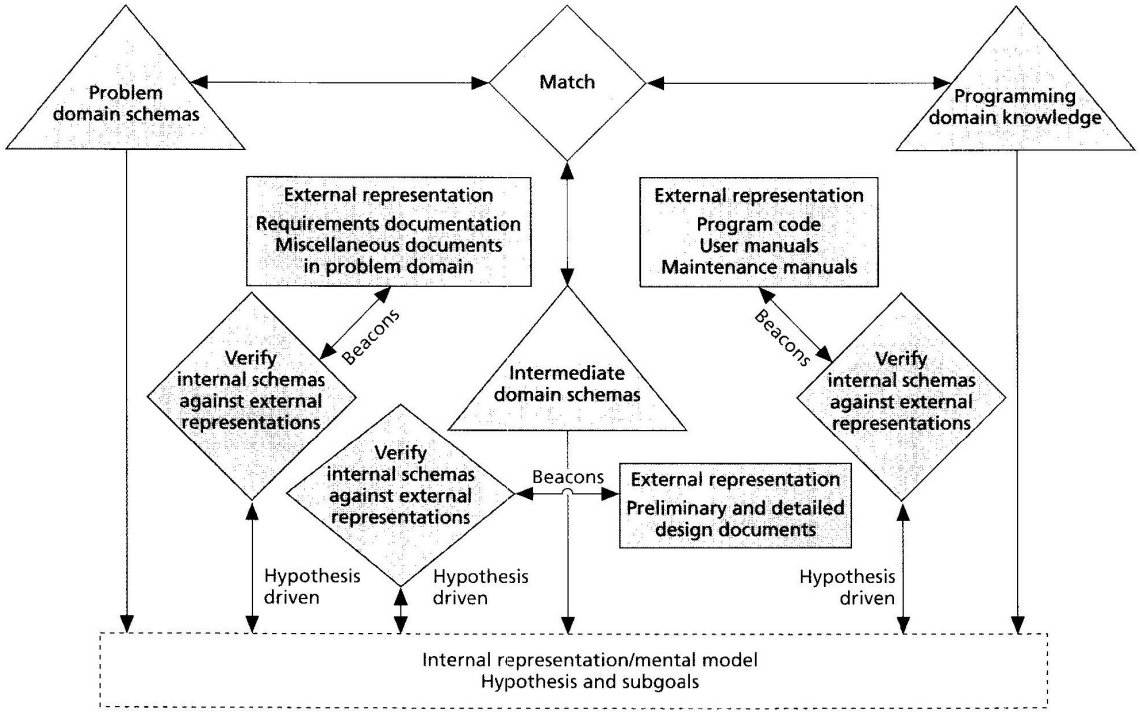


Figure 5.1: Brooks top down model [52].

while syntactic represents specific knowledge in the language. Our model was partly build on these bottom-up processes. It is limited to low-level details, in particular to concrete values of variables. These must be extracted from the available source code and remembered in short-term memory.

Some years later, Letovsky [37] came up with the first mixed model. According to his research, three components are involved in the process of code comprehension.
The first is the knowledge base, which differs between individuals and represents the entire knowledge of the developer.
The second is the internal model that is created based on previous recognitions. This contains information about the specification of the program, a precise understanding of the implementation, and also specific annotations about how the implementation achieves the specification.
The third component is the assimilation process by which the internal model is further developed based on the stimulus material. This is a kind of question-and-answer game between the developer and the stimulus material. If there is a discrepancy between the internal model and the material, an attempt is made to clarify how this can be resolved. The assimilation process makes use of both bottom-up and top-down processes.
Of particular interest in this work is the fact that Letovsky [37] mentions at the very end of his notes how valuable a computer-based model of human understanding of source code would be for the development of new tools that can assist developers in programming. Unfortunately, he did not continue this work himself.

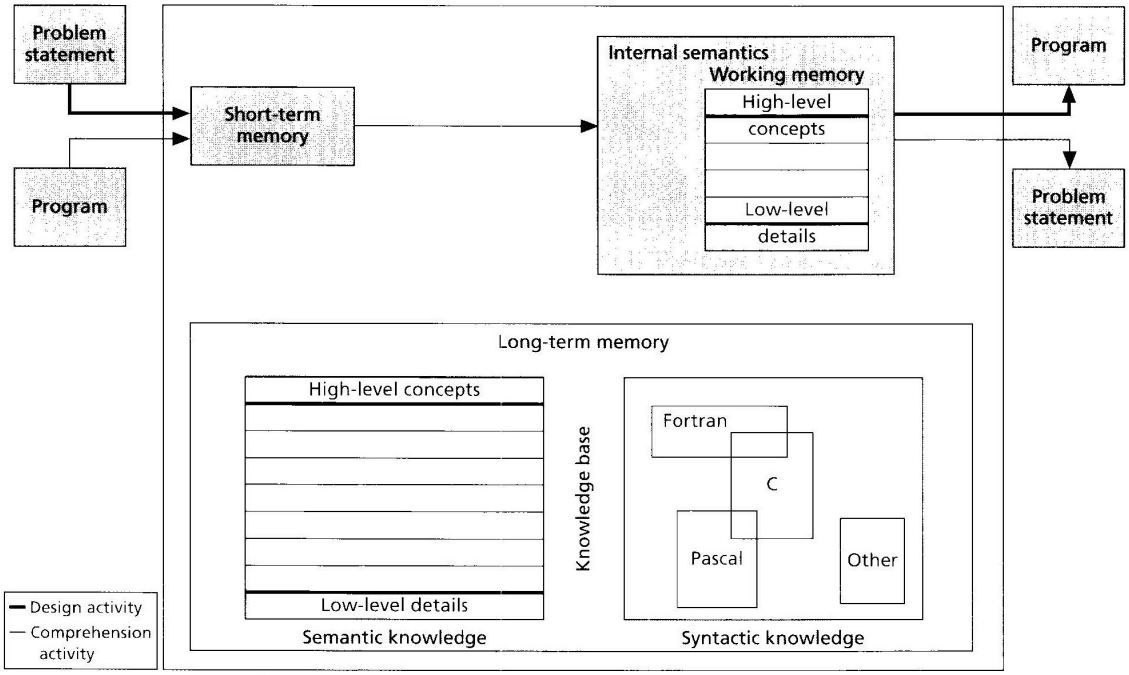


Figure 5.2: Schneiderman and Mayer bottom up model [52].

Tiemens [50] has also formulated a mixed model which he calls the 'Comprehension Support Tool', which aims to support developers in their work by supporting the process of comprehension and thus reducing cognitive workload. In contrast to Letovsky [37], Tiemens [50] also goes into more detail at this point about the design of a possible implementation and precisely describes how possible use cases could look like and how they might provide support. Similar to Letovsky [37], he also assumes that an implementation requires a knowledge base, a mental model and an assimilation process. On closer inspection, the proposed implementation turns out to be a large collection of many complicated sub-problems, such as the representation of knowledge or mental processes, how they interact with each other or how they change over time. This is perhaps the reason for the lack of concrete implementations to date.

The next major milestone was set by Von Mayrhauser and Vans [52] by combining numerous models [7, 37, 42, 44, 46] into one metamodel. A representation of the metamodel can be seen in Figure 5.3. The knowledge base, which bundles knowledge about the different classes, is also at the centre here. This model also contains both top-down and bottom-up processes, in this case, called the program model process. In contrast to the previous mixed models, it also contains a situation model, which takes into account the context in which the processes take place. Familiar elements from the previous models such as chunks and beacons can be found again. Von Mayrhauser and Vans [52] also emphasized that it is rarely one process alone that dominates the comprehension of the program, but that it is a constant interaction and progression of processes that leads to the overall comprehension of a program. This model shows the three major areas, top-down model, bottom-up model, situation model and their interaction for the first time in one single model. As we did not

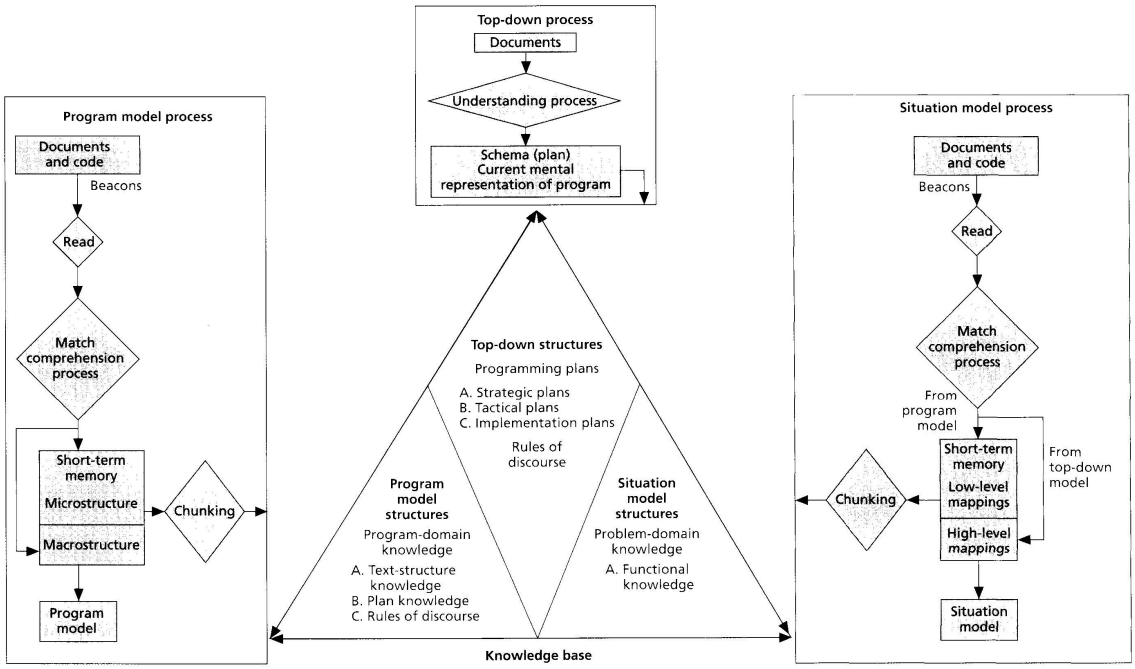


Figure 5.3: Von Mayerhauser and Vans integrated metamodel [52].

believe it makes sense to tackle such a complex model all at once, we limited ourselves to the bottom-up process, which in this model is called the program model process, and only to a small part of the code execution. This means that the chunks that have to be formed only included the value of variables and the knowledge base only covered basic arithmetic knowledge. The aim of further work, however, could be to extend the model to include the other processes.

In the same year, Cant et al. [8] also published their CCM. This serves to quantify the cognitive processes involved in the code, in particular chunking and tracing. In this case, tracing means being able to resolve the local dependencies of variables.

$$C_i = R_i + \sum_{j \in N} C_j + \sum_{j \in N} T_j \tag{5.1}$$

In terms of the CCM, the complexity of a chunk i, as can be seen in equation 5.1, is made up of the direct complexity of the chunk R_i, the complexity of the sub-chunks C_j and the complexity of the tracing dependencies T_j. Each of these complexities is once again made up of individual components that relate to different components of visual perception and complexity during processing. This form of the model is interesting in that it seems easy to implement due to its formulaic structure. However, on closer inspection, it is not clear how exactly a chunk breaks down into its sub-chunks. Nevertheless, our model integrated this approach in that a formula evaluation has a certain runtime which could be understood as complexity R_i and depended on other variables used in the equation which could be understood as sub-chunks C_j. The exact temporal calculators, however, were taken from ACT-R. Tracing was not used by our model because we only considered a purely linear

execution.

Hansen et al. [28] took up this approach almost two decades later and developed an approach for implementing an ACT-R model for program comprehension using the CCM. They also emphasize that the advantages of ACT-R lie in the availability of perception and motor modules and the fact that BOLD can later also be simulated using ACT-R, which further expands the field of potential experiments. They see a use case for their approach above all in directly quantifying how complicated code is and making more efficient use of test subjects through this pre-quantification. Although the work describes concrete plans on how an implementation can be carried out successively, the process described is extremely complex, which is why a concrete ACT-R model is still a work in progress. We have taken this as a warning and have limited ourselves to the small part of the mental execution of code in our model.

An implemented ACT-R model was used by Chiarelli [10] to investigate how self-explanation is used when learning Python concepts. In particular, simulations were used to examine how novices behave in comparison to experts. However, this work has some methodological weaknesses. For example, the simulated data was not compared with experimental data from real test subjects. Furthermore, the description is also slightly misleading, as the core of the model is based on simple label matching to make queries to declarative memory using labelled stimulus material. However, the work shows that it is possible to gain insights into code comprehension with ACT-R.

A recent contribution to ACT-R in the field of software engineering is provided by Leung and Murphy [38]. They presented the potential of LLMs that can be adapted to human needs and designed based on ACT-R to create developer tools. The advantages of both sides could be combined. LLMs can summarize effective knowledge and also create code, while cognitive models describe what generates a high cognitive load when dealing with code. The possible assistance systems should provide support at the points that require a high cognitive load, namely the declarative memory, the production memory and the working memory. The potential applications range from explaining code to writing and automatic test creation. This work is again of a more theoretical and hypothetical nature but shows that even in times of breakthroughs in the field of artificial intelligence, cognitive models allow a different perspective on modern software development.

Table 5.1 summarizes the publications discussed before and shows how these works have influenced our work.

Table 5.1: Overview of Related Work

Reference	Contribution	Our Adaptation
Brooks [7]	First developed the theoretical functional principle of a top-down model, based on a successive refinement of hypotheses about the program.	Used as the theoretical foundation for understanding code comprehension.
Shneiderman [44]	Built on exploratory studies to develop the bottom-up model, which involves progressively summarizing the model.	The process we refer to in this work is essentially a bottom-up process.
Letovsky [37]	Combined top-down and bottom-up models into a mixed model, noting influences from both processes in his studies. Also considered a model that could be simulated.	Used as the theoretical basis for understanding code comprehension.
Tiemens [50]	Also integrated top-down and bottom-up approaches into a mixed model, with a focus on practical implications.	Used as the theoretical basis for understanding code comprehension.
Von Mayrhauser and Vans [52]	Synthesized multiple models into a meta-model.	Used as the theoretical basis for understanding code comprehension.
Cant et al. [8]	Developed the Cognitive Complexity Metric (CCM), a formula for describing code complexity.	Used implicitly as a measure of time complexity for productions and retrievals.
Hansen et al. [28]	Theorized how to implement CCM within ACT-R.	Served as a starting point for our work due to their extensive considerations and identified issues.
Chiarelli [10]	Developed an ACT-R model for self-explanation of Python.	Demonstrated that ACT-R is already part of software engineering research.
Leung and Murphy [38]	Theoretically assessed the potential of cognitive models in the era of LLMs, showing how to implement a functioning ACT-R model.	Provided insight into how a functioning model can still contribute new knowledge in the age of AI.

6 Concluding Remarks

6.1 Conclusion

Understanding code is a daily task for developers, and it is crucial to make this process as simple and efficient as possible. Research on code comprehension is ongoing, as the distinctions between easily comprehensible and difficult code are not yet fully understood. Furthermore, the exact definition and measurement of these differences remain ambiguous.

In this study, we have revisited a old concept of mental models for code comprehension and explored the extent to which a practical model can be realized in ACT-R. To this end, we developed a reduced framework that is compatible with both the ACT-R model and human participants. Using this framework, we designed a series of code snippets expected to elicit specific effects in the participants, and the model was tasked with making predictions for these snippets.

The data collected from the participants were used to adjust the model's hyperparameters. Although the effects were not statistically significant, the model demonstrated a high degree of precision in predicting the timing data. However, the model revealed some weaknesses in predicting error rates.

In summary, this study has shown that a cognitive model has the potential to serve as a metric for code complexity. With further refinement, such a model could accurately predict the likely time and error susceptibility of code, providing an objective criterion for differentiating various inputs. Therefore, it is essential to continue pursuing this approach in future research.

6.2 Future Work

As emphasized at the beginning of this work, the concept of a cognitive model for studying code comprehension is not new. However, this work presents the analysis of the first simple executable model within the ACT-R framework, aiming to open a new field of methodological approaches. Consequently, there are numerous opportunities for building upon this work. In this concluding section, we will outline several potential directions for future research.

6.2.1 Model Extension

6.2.1.1 *Multi Line Reading*

A possible extension of the model is to incorporate the capability to process multiple lines of code. This would require integrating a logic that directs eye movements to switch from line to line. However, it is possible that text is not read strictly from left to right and top to bottom [41]. Therefore, analyzing eye-tracking data could be useful to identify specific reading patterns. The advantage is that ACT-R can also simulate eye-tracking data, which allows the model to be validated against eye-tracking experiments.

Extending the model to handle multiple lines could present both opportunities and challenges. One challenge is how to manage retrieval failures when multiple lines are displayed. If a person forgets the value of a variable, they will likely perform a kind of recovery process, meaning they scan the code again and, upon finding a trigger point that jogs their memory, return to the previous point. Modelling peripheral vision and quick saccades to accurately represent movements between lines can be problematic.

On the other hand, this extension opens up many possibilities, such as integrating more complex constructs like loops or conditional expressions that span multiple lines. Multi-line reading is a prerequisite for these integrations. Additionally, classes, functions, and comments could be incorporated into the model. Therefore, we see that an essential future step is to extend the model to handle multi-line reading.

6.2.1.2 *Similarity Matrices*

As discussed in this work, the error space in the empirical data is larger than that in the model. Therefore, it is essential to extend the model to simulate the errors observed in the empirical data.

One potential approach is to utilize similarity matrices. This function in ACT-R allows for the description of the similarity between chunks, enabling the activation of a chunk not only for an exact match but also proportionally for similar chunks. This addition makes the activation calculation more complex, as it now includes a similarity component.

Both variable names and numbers can be similar to each other, leading to potential confusion. Based on the errors observed here, this extension could be necessary to better encompass the range of human errors and more accurately predict potential issues in code comprehension.

6.2.2 Snippet Modification

As described in the results, trends for effects are noticeable in the data, but no statistically significant differences were found. This could be due to the sample size as well as the

nature of the snippets used. Different snippets of the same type yielded varying results, and unintended effects were detected in the data.

To address these issues in future studies, one approach is to expand the snippet portfolio based on the insights gained from this work. For instance, to enhance the effects of distance, additional filler lines could be introduced to amplify the effect. Additionally, one could more precisely differentiate whether other variables appear in the target line or not. It may also be worthwhile to consider directly incorporating and investigating unexpected effects, such as order effects, into the snippet samples.

Overall, there are numerous opportunities to improve snippet selection to better highlight the effects under investigation. A limiting factor in this context is the line-by-line presentation. Since humans have limited memory capacity, we can only introduce and utilize a limited number of variables and present a certain length of code before attention wanes. However, exploring these boundaries more thoroughly could enable more comprehensive experiments even with the line-by-line presentation.

6.2.3 Replication Studies

Since this work aims to revive a novel method for studying code comprehension, one of the most crucial steps for the future is replication. With a larger sample size, some of the effects suggested here might achieve statistical significance. Additionally, other models could be developed to identify potential design flaws.

Investigating different snippets and data collection methods could help control for confounding variables more effectively. Moreover, a reanalysis of the hyperparameter optimization process with the current dataset could lead to a more robust model that optimizes both time and error rate. Other effects that might be hidden in the collected dataset could also be analyzed and replicated.

It is essential to continue this line of research to prevent it from meeting the same fate as its predecessors, where the topic fades into obscurity for the next several years.

6.2.4 Modularity

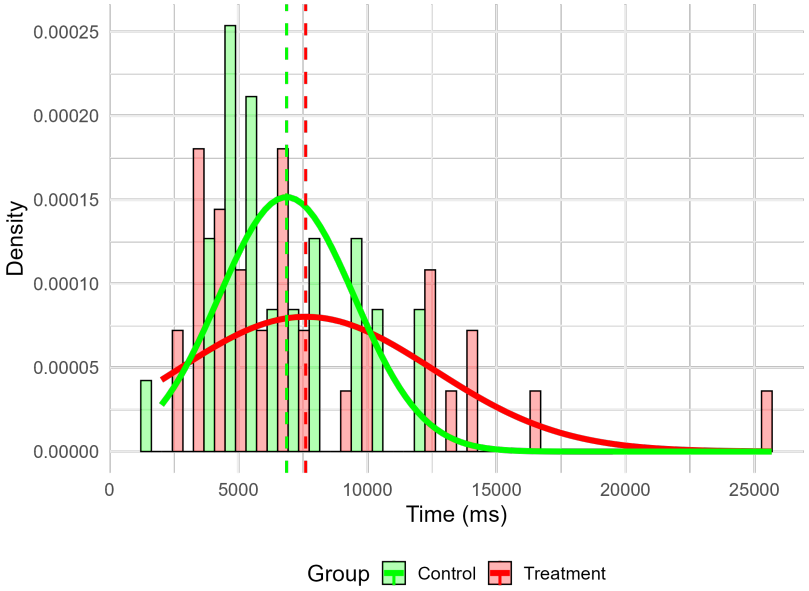
A crucial step to prevent this topic from fading into obscurity once again could be embracing modularity. In the context of a large research community, a framework could be established, and multiple submodels could be developed to model various aspects of code comprehension. For instance, the capabilities for conditional expressions, loops, lambda expressions, and so on could be divided into different submodules, allowing parallel and independent development of the ACT-R model.

The challenge lies in developing a main module that integrates and coordinates all the other submodules. It is essential to ensure that the model does not drift too far from modelling human cognitive processes and acts more like a compiler due to excessive optimization. The balance between cognitive psychology and computer science must be carefully maintained.

One approach we propose is the modularization of the maths module by separating different arithmetic operations and adding new ones. This would be a good starting point to explore the feasibility of modularization within a manageable scope that does not require as many interactions with other submodules as, for example, a loop. Additionally, unlike other proposed modules, it can be applied within the line-by-line reading behaviour.

A

Empirical Results Single Snippets

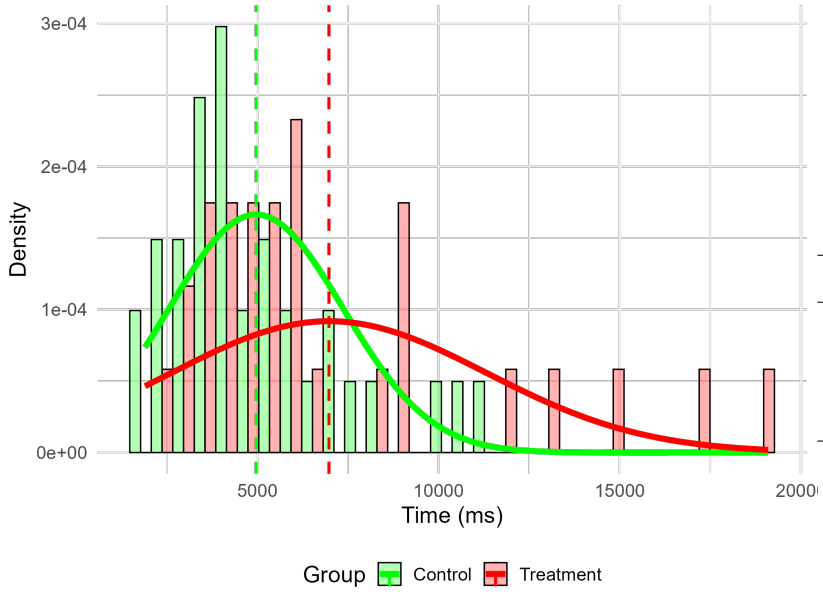


(a) Time effect.

(b) Error rate effect.

	False	True	Sum
Control	6	23	29
Treatment	7	27	34
Sum	13	50	63

Figure A.1: Empirical results for the CD1.

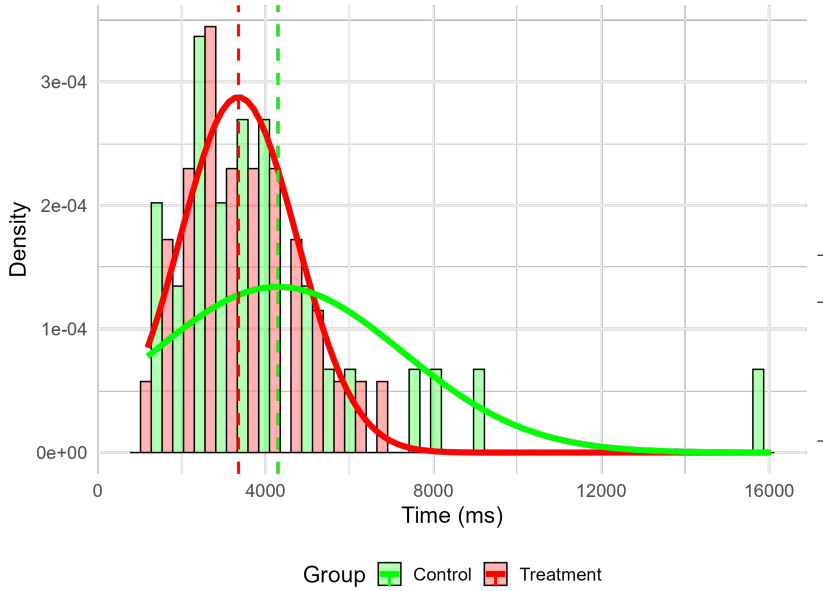


(a) Time effect.

(b) Error rate effect.

	False	True	Sum
Control	1	33	34
Treatment	1	28	29
Sum	2	61	63

Figure A.2: Empirical results for the CD2.

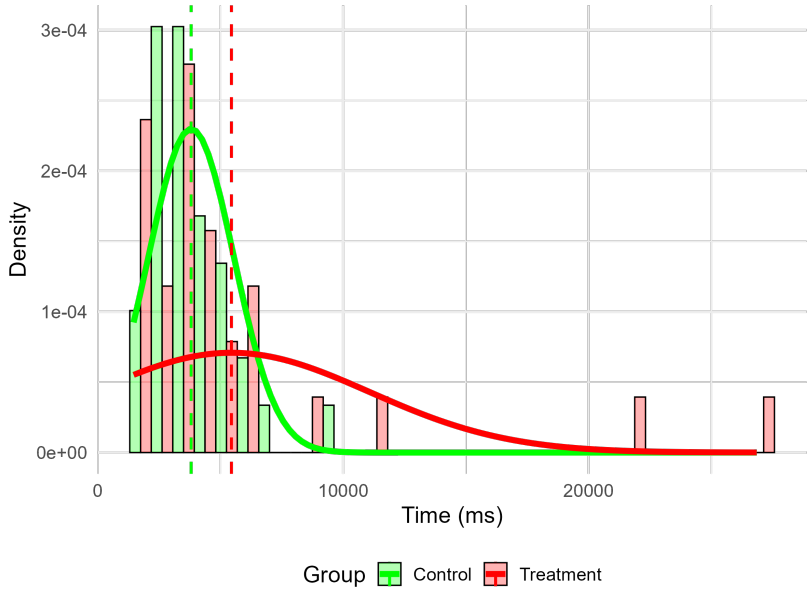


(a) Time effect.

(b) Error rate effect.

	False	True	Sum
Control	0	29	29
Treatment	2	32	34
Sum	2	61	63

Figure A.3: Empirical results for the CR1.

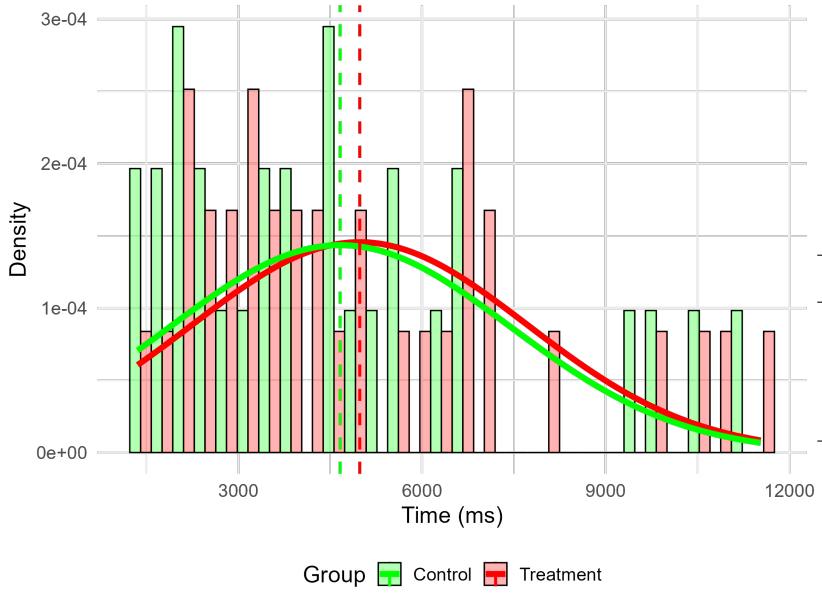


(a) Time effect.

(b) Error rate effect.

	False	True	Sum
Control	0	34	34
Treatment	1	28	29
Sum	1	62	63

Figure A.4: Empirical results for the CR2.



(a) Time effect.

(b) Error rate effect.

	False	True	Sum
Control	4	25	29
Treatment	6	28	34
Sum	10	53	63

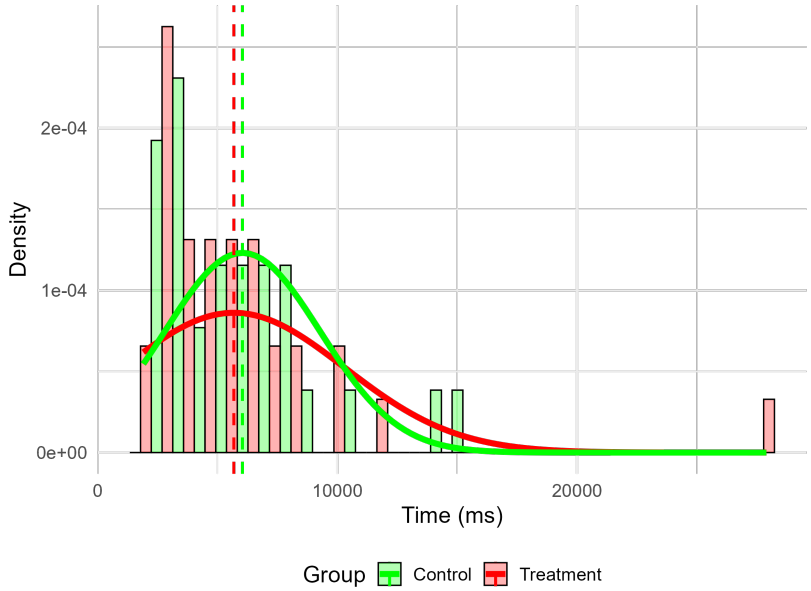
Figure A.5: Empirical results for the DR1.

(a) Time effect.

(b) Error rate effect.

	False	True	Sum
Control	18	16	34
Treatment	9	20	29
Sum	27	36	63

Figure A.6: Empirical results for the DR2.

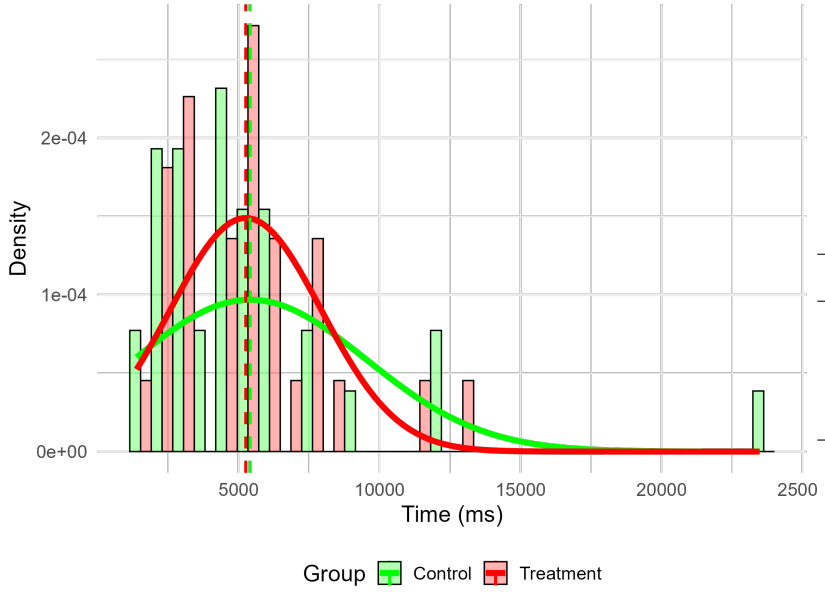


(a) Time effect.

(b) Error rate effect.

	False	True	Sum
Control	3	26	29
Treatment	15	19	34
Sum	18	45	63

Figure A.7: Empirical results for the RP1.



(a) Time effect.

(b) Error rate effect.

	False	True	Sum
Control	6	28	34
Treatment	6	23	29
Sum	12	51	63

Figure A.8: Empirical results for the RP2.

B

Accuracy Parameter Tuning activation noise s for Single Snippets

Table B.1: Error rates for optimizing ans on time data.

(a) CD1.

	False	True	Total
Control	0 (6)	29 (23)	29 (29)
Treatment	0 (7)	34 (27)	34 (34)
Total	0 (13)	63 (50)	63 (63)

Annotation: Empirical data in brackets.

(b) CD2.

	False	True	Total
Control	0 (1)	34 (33)	34 (34)
Treatment	0 (1)	29 (28)	29 (29)
Total	0 (2)	63 (61)	63 (63)

Annotation: Empirical data in brackets.

(c) CR1.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	15 (2)	19 (32)	34 (34)
Total	15 (2)	48 (61)	63 (63)

Annotation: Empirical data in brackets.

(d) CR2.

	False	True	Total
Control	0 (0)	34 (34)	34 (34)
Treatment	11 (1)	18 (28)	29 (29)
Total	11 (1)	52 (62)	63 (63)

Annotation: Empirical data in brackets.

(e) DR1.

	False	True	Total
Control	9 (4)	20 (25)	29 (29)
Treatment	19 (6)	15 (28)	34 (34)
Total	28 (10)	35 (53)	63 (63)

Annotation: Empirical data in brackets.

(f) DR2.

	False	True	Total
Control	17 (18)	17 (16)	34 (34)
Treatment	18 (9)	11 (20)	29 (29)
Total	35 (27)	28 (36)	63 (63)

Annotation: Empirical data in brackets.

(g) RP1.

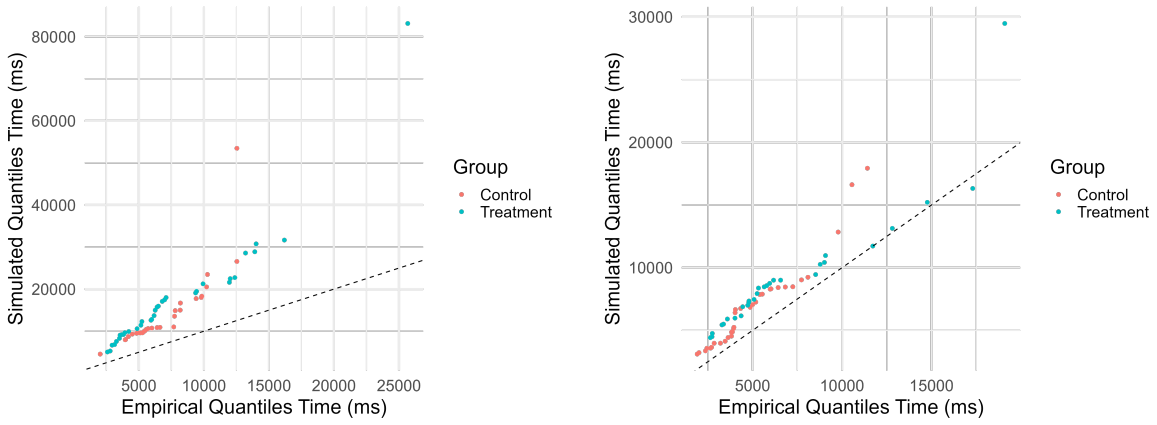
	False	True	Total
Control	9 (3)	20 (26)	29 (29)
Treatment	16 (15)	18 (19)	34 (34)
Total	25 (18)	38 (45)	63 (63)

Annotation: Empirical data in brackets.

(h) RP2.

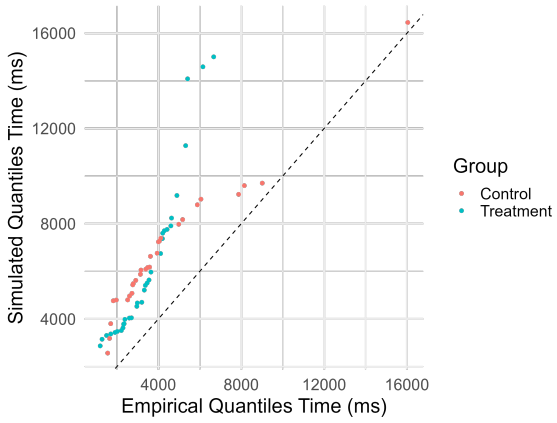
	False	True	Total
Control	15 (6)	19 (28)	34 (34)
Treatment	11 (6)	18 (23)	29 (29)
Total	26 (12)	37 (51)	63 (63)

Annotation: Empirical data in brackets.

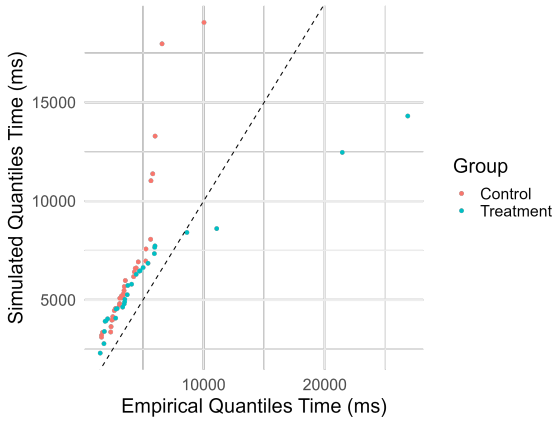


(a) CD1.

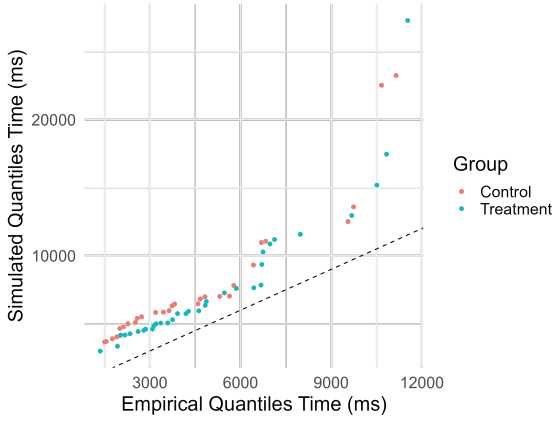
(b) CD2.



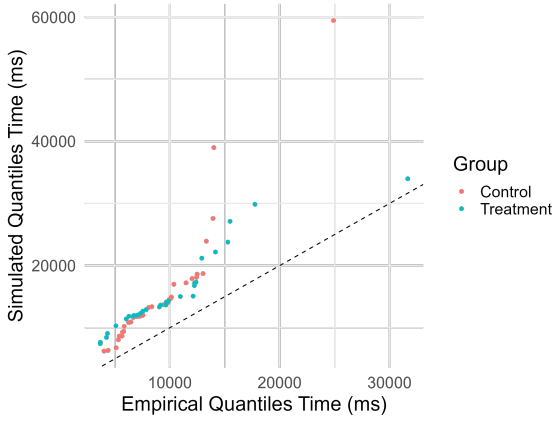
(c) CR1.



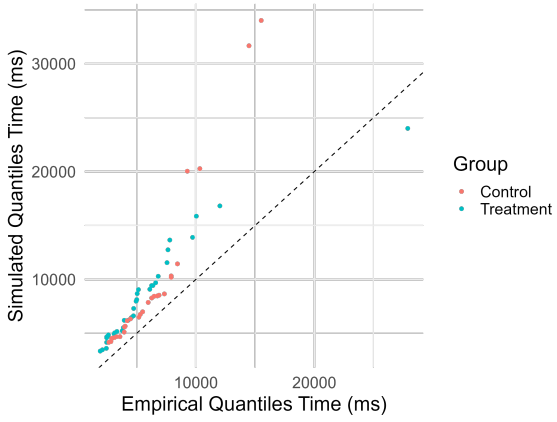
(d) CR2.



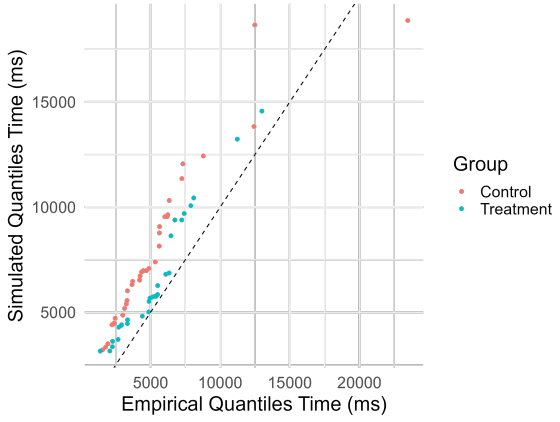
(e) DR1.



(f) DR2.

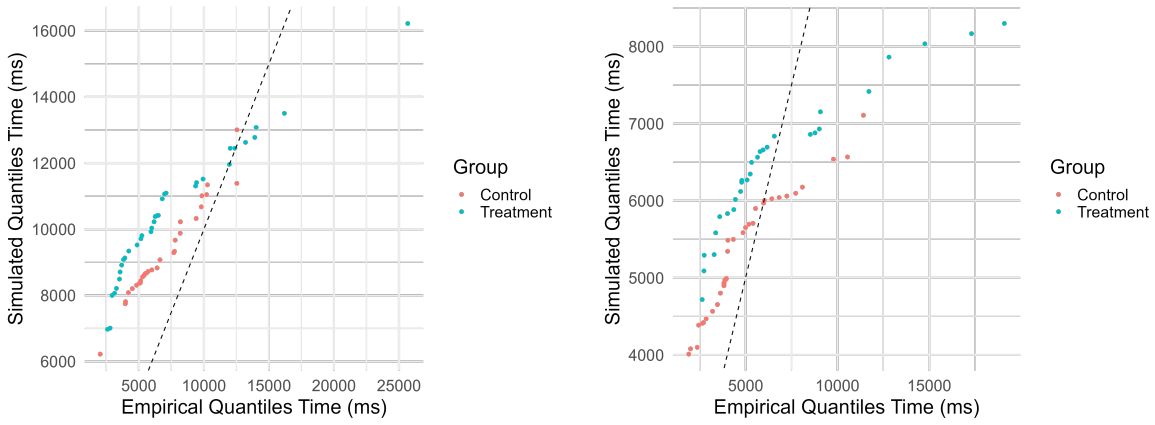


(g) RP1.



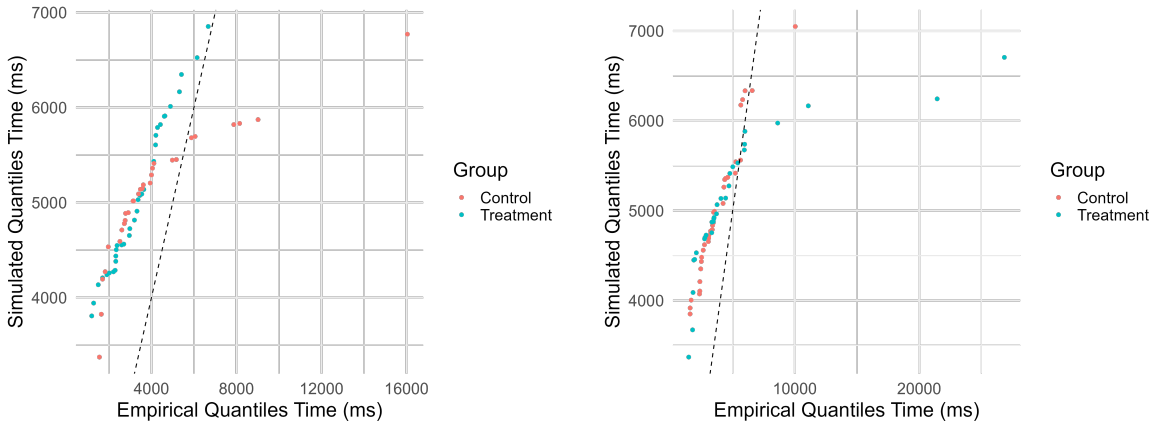
(h) RP2.

Figure B.1: QQ-Plot for optimizing ans on time data.



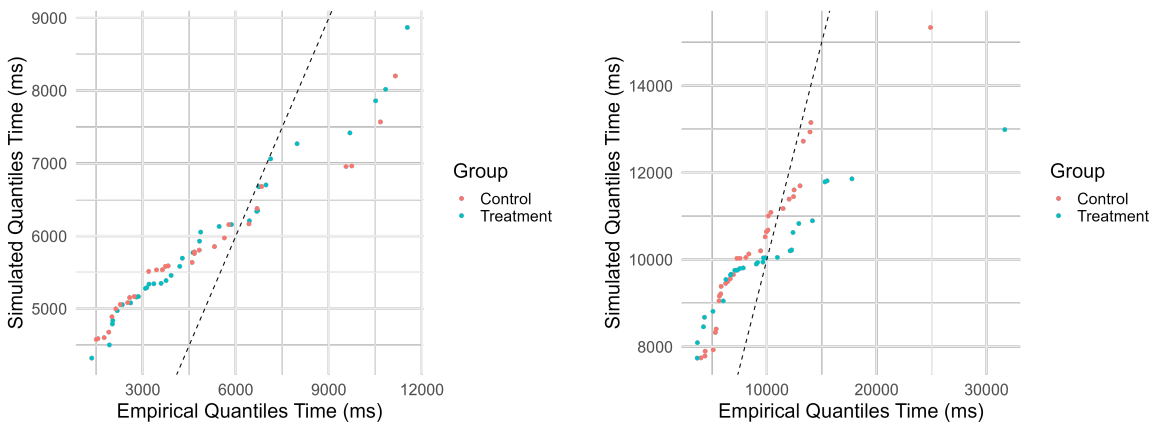
(a) CD1.

(b) CD2.



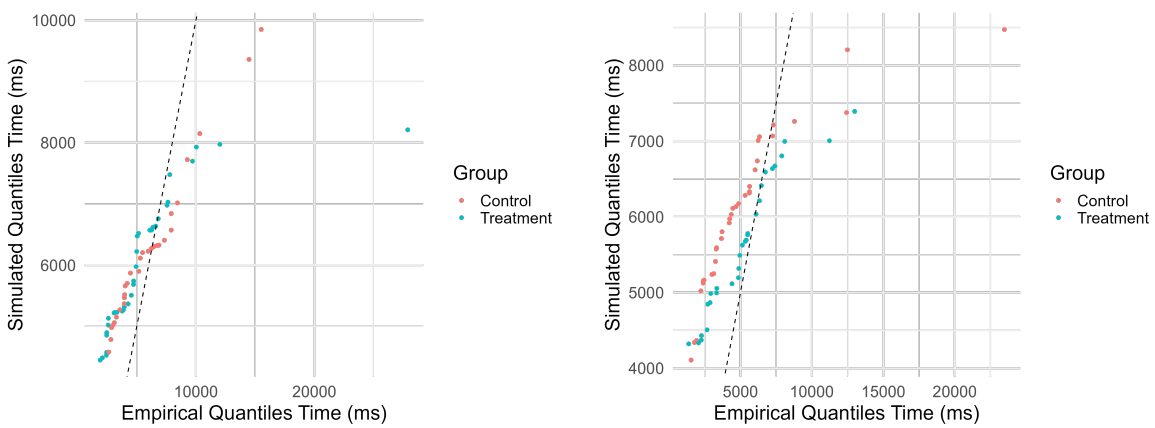
(c) CR1.

(d) CR2.



(e) DR1.

(f) DR2.



(g) RP1.

(h) RP2.

Figure B.2: QQ-Plot for optimizing ans on error rate data.

Table B.2: Error rates for optimizing ans on error rate data.

(a) CD1.

	False	True	Total
Control	0 (6)	29 (23)	29 (29)
Treatment	0 (7)	34 (27)	34 (34)
Total	0 (13)	63 (50)	63 (63)

Annotation: Empirical data in brackets.

(b) CD2.

	False	True	Total
Control	0 (1)	34 (33)	34 (34)
Treatment	0 (1)	29 (28)	29 (29)
Total	0 (2)	63 (61)	63 (63)

Annotation: Empirical data in brackets.

(c) CR1.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	13 (2)	21 (32)	34 (34)
Total	13 (2)	50 (61)	63 (63)

Annotation: Empirical data in brackets.

(d) CR2.

	False	True	Total
Control	0 (0)	34 (34)	34 (34)
Treatment	7 (1)	22 (28)	29 (29)
Total	7 (1)	56 (62)	63 (63)

Annotation: Empirical data in brackets.

(e) DR1.

	False	True	Total
Control	2 (4)	27 (25)	29 (29)
Treatment	17 (6)	17 (28)	34 (34)
Total	19 (10)	44 (53)	63 (63)

Annotation: Empirical data in brackets.

(f) DR2.

	False	True	Total
Control	10 (18)	24 (16)	34 (34)
Treatment	16 (9)	13 (20)	29 (29)
Total	26 (27)	37 (36)	63 (63)

Annotation: Empirical data in brackets.

(g) RP1.

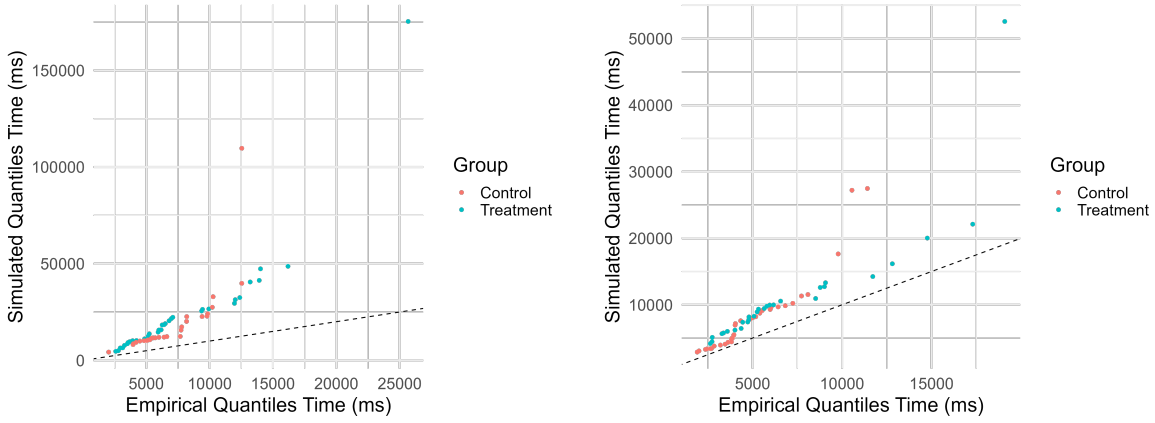
	False	True	Total
Control	7 (3)	22 (26)	29 (29)
Treatment	12 (15)	22 (19)	34 (34)
Total	19 (18)	44 (45)	63 (63)

Annotation: Empirical data in brackets.

(h) RP2.

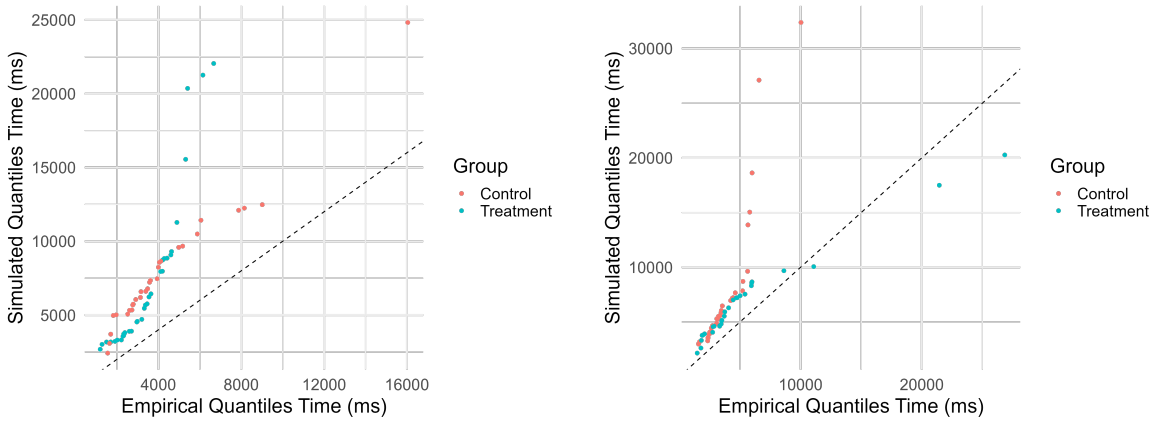
	False	True	Total
Control	9 (6)	25 (28)	34 (34)
Treatment	11 (6)	18 (23)	29 (29)
Total	20 (12)	43 (51)	63 (63)

Annotation: Empirical data in brackets.



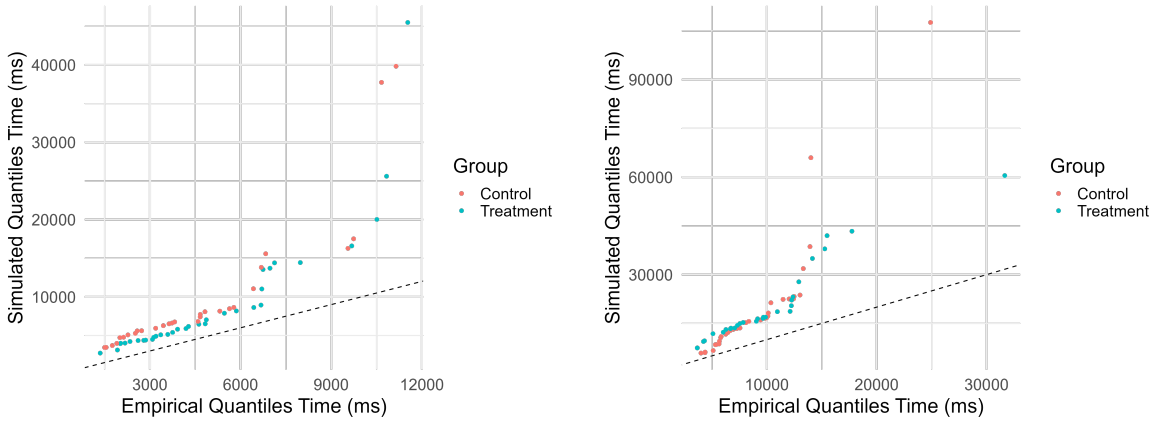
(a) CD1.

(b) CD2.



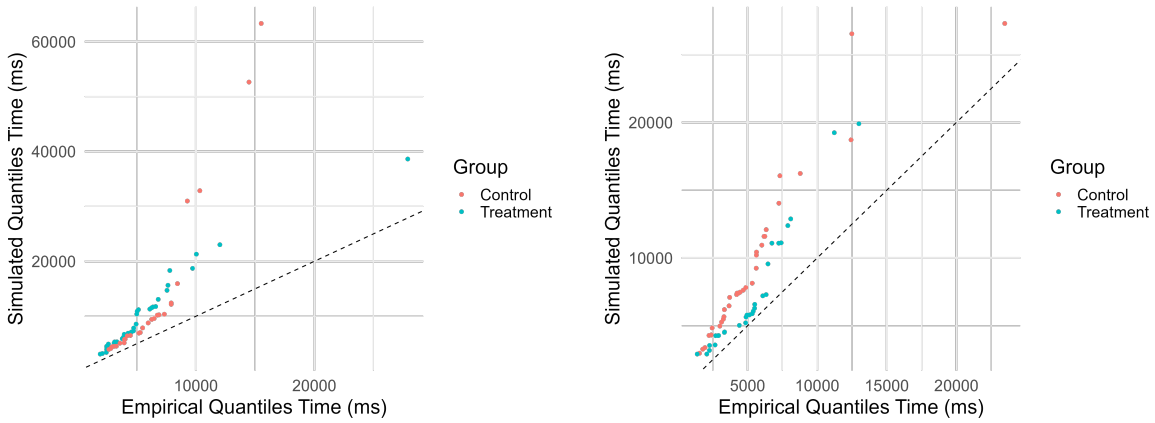
(c) CR1.

(d) CR2.



(e) DR1.

(f) DR2.



(g) RP1.

(h) RP2.

Figure B.3: QQ-Plot for optimizing ans on combined data.

Table B.3: Error rates for optimizing ans on combined data.

(a) CD1.

	False	True	Total
Control	0 (6)	29 (23)	29 (29)
Treatment	0 (7)	34 (27)	34 (34)
Total	0 (13)	63 (50)	63 (63)

Annotation: Empirical data in brackets.

(b) CD2.

	False	True	Total
Control	0 (1)	34 (33)	34 (34)
Treatment	0 (1)	29 (28)	29 (29)
Total	0 (2)	63 (61)	63 (63)

Annotation: Empirical data in brackets.

(c) CR1.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	15 (2)	19 (32)	34 (34)
Total	15 (2)	48 (61)	63 (63)

Annotation: Empirical data in brackets.

(d) CR2.

	False	True	Total
Control	0 (0)	34 (34)	34 (34)
Treatment	11 (1)	18 (28)	29 (29)
Total	11 (1)	52 (62)	63 (63)

Annotation: Empirical data in brackets.

(e) DR1.

	False	True	Total
Control	9 (4)	20 (25)	29 (29)
Treatment	19 (6)	15 (28)	34 (34)
Total	28 (10)	35 (53)	63 (63)

Annotation: Empirical data in brackets.

(f) DR2.

	False	True	Total
Control	17 (18)	17 (16)	34 (34)
Treatment	18 (9)	11 (20)	29 (29)
Total	35 (27)	28 (36)	63 (63)

Annotation: Empirical data in brackets.

(g) RP1.

	False	True	Total
Control	9 (3)	20 (26)	29 (29)
Treatment	16 (15)	18 (19)	34 (34)
Total	25 (18)	38 (45)	63 (63)

Annotation: Empirical data in brackets.

(h) RP2.

	False	True	Total
Control	17 (6)	17 (28)	34 (34)
Treatment	11 (6)	18 (23)	29 (29)
Total	28 (12)	35 (51)	63 (63)

Annotation: Empirical data in brackets.

C

Accuracy Parameter Tuning retrieval threshold, activation noise s and latency factor for Single Snippets

Table C.1: Error rates for optimizing rt, ans and lf on time data.

(a) CD1.

	False	True	Total
Control	3 (6)	26 (23)	29 (29)
Treatment	1 (7)	33 (27)	34 (34)
Total	4 (13)	59 (50)	63 (63)

Annotation: Empirical data in brackets.

(b) CD2.

	False	True	Total
Control	2 (1)	32 (33)	34 (34)
Treatment	0 (1)	29 (28)	29 (29)
Total	2 (2)	61 (61)	63 (63)

Annotation: Empirical data in brackets.

(c) CR1.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	15 (2)	19 (32)	34 (34)
Total	15 (2)	48 (61)	63 (63)

Annotation: Empirical data in brackets.

(d) CR2.

	False	True	Total
Control	1 (0)	33 (34)	34 (34)
Treatment	11 (1)	18 (28)	29 (29)
Total	12 (1)	51 (62)	63 (63)

Annotation: Empirical data in brackets.

(e) DR1.

	False	True	Total
Control	8 (4)	21 (25)	29 (29)
Treatment	18 (6)	16 (28)	34 (34)
Total	26 (10)	37 (53)	63 (63)

Annotation: Empirical data in brackets.

(f) DR2.

	False	True	Total
Control	16 (18)	18 (16)	34 (34)
Treatment	17 (9)	12 (20)	29 (29)
Total	33 (27)	30 (36)	63 (63)

Annotation: Empirical data in brackets.

(g) RP1.

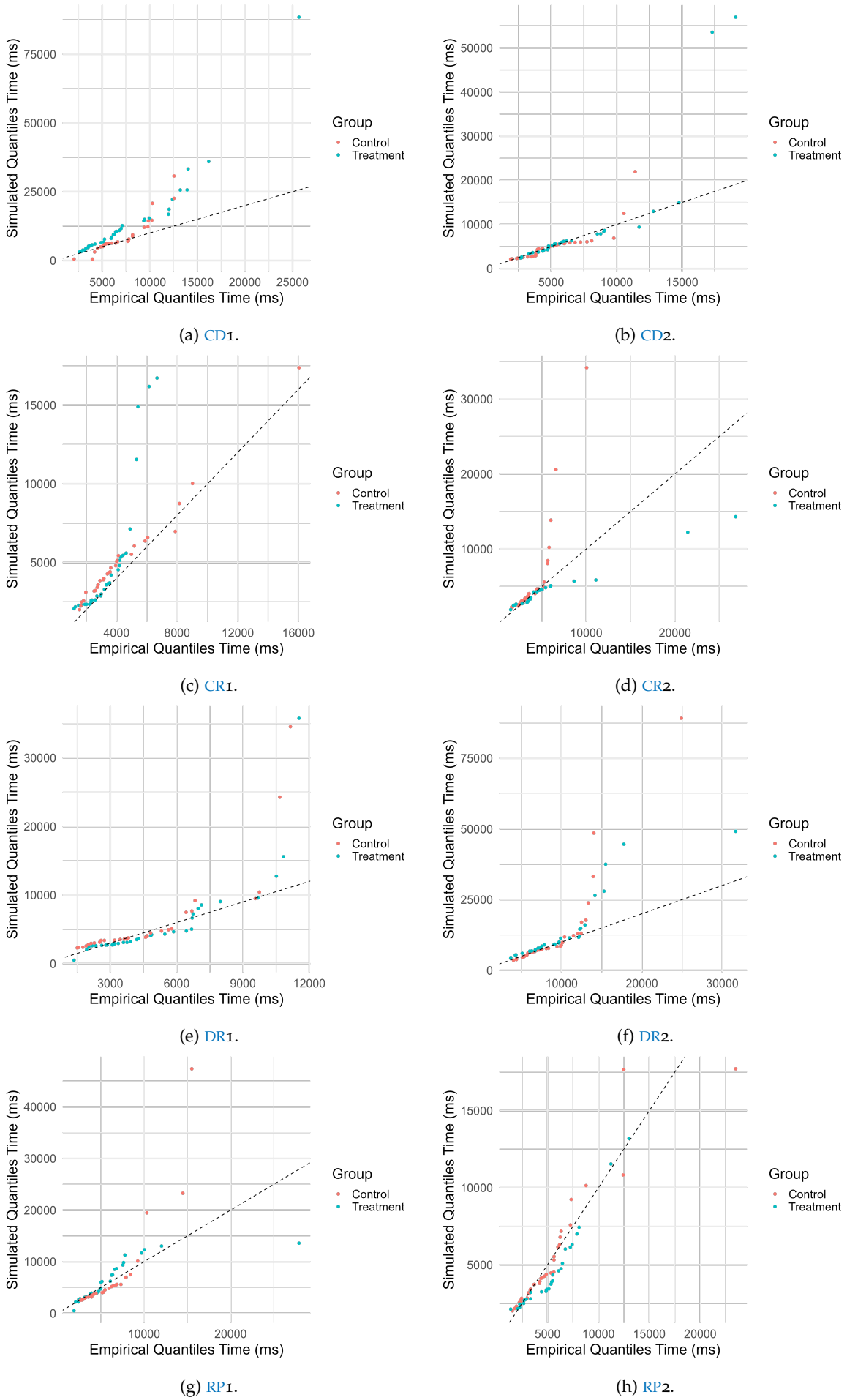
	False	True	Total
Control	10 (3)	19 (26)	29 (29)
Treatment	16 (15)	18 (19)	34 (34)
Total	26 (18)	37 (45)	63 (63)

Annotation: Empirical data in brackets.

(h) RP2.

	False	True	Total
Control	17 (6)	17 (28)	34 (34)
Treatment	12 (6)	17 (23)	29 (29)
Total	29 (12)	34 (51)	63 (63)

Annotation: Empirical data in brackets.



(a) CD1.

(b) CD2.

(c) CR1.

(d) CR2.

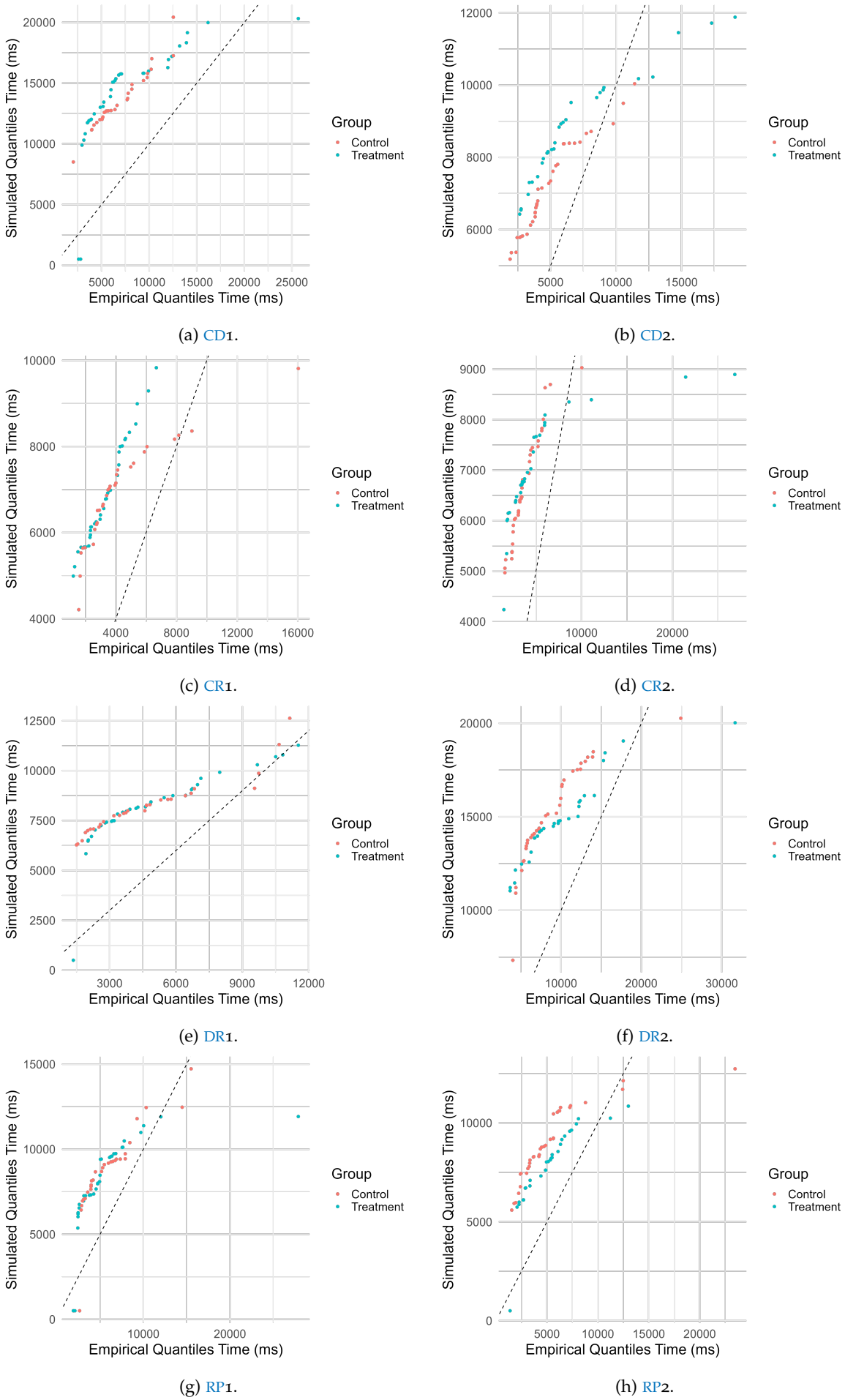
(e) DR1.

(f) DR2.

(g) RP1.

(h) RP2.

Figure C.1: QQ-Plot for optimizing rt, ans and lf on time data.



(a) CD1.

(b) CD2.

(c) CR1.

(d) CR2.

(e) DR1.

(f) DR2.

(g) RP1.

(h) RP2.

Figure C.2: QQ-Plot for optimizing rt, ans and lf on error rate data

Table C.2: Error rates for optimizing rt, ans and lf on error rate data

(a) CD1.

	False	True	Total
Control	5 (6)	24 (23)	29 (29)
Treatment	5 (7)	29 (27)	34 (34)
Total	10 (13)	53 (50)	63 (63)

Annotation: Empirical data in brackets.

(b) CD2.

	False	True	Total
Control	1 (1)	33 (33)	34 (34)
Treatment	1 (1)	28 (28)	29 (29)
Total	2 (2)	61 (61)	63 (63)

Annotation: Empirical data in brackets.

(c) CR1.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	12 (2)	22 (32)	34 (34)
Total	12 (2)	51 (61)	63 (63)

Annotation: Empirical data in brackets.

(d) CR2.

	False	True	Total
Control	1 (0)	33 (34)	34 (34)
Treatment	7 (1)	22 (28)	29 (29)
Total	8 (1)	55 (62)	63 (63)

Annotation: Empirical data in brackets.

(e) DR1.

	False	True	Total
Control	3 (4)	26 (25)	29 (29)
Treatment	15 (6)	19 (28)	34 (34)
Total	18 (10)	45 (53)	63 (63)

Annotation: Empirical data in brackets.

(f) DR2.

	False	True	Total
Control	20 (18)	14 (16)	34 (34)
Treatment	17 (9)	12 (20)	29 (29)
Total	37 (27)	26 (36)	63 (63)

Annotation: Empirical data in brackets.

(g) RP1.

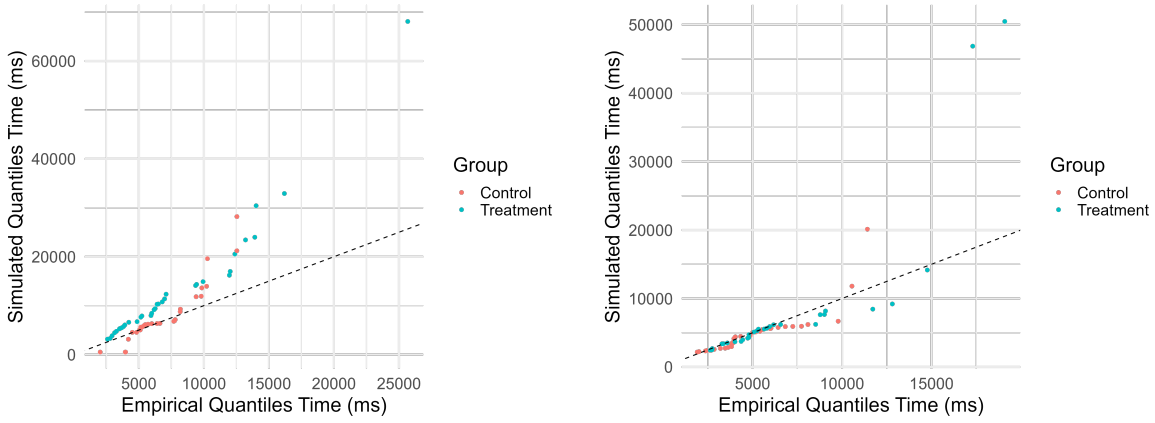
	False	True	Total
Control	10 (3)	19 (26)	29 (29)
Treatment	17 (15)	17 (19)	34 (34)
Total	27 (18)	36 (45)	63 (63)

Annotation: Empirical data in brackets.

(h) RP2.

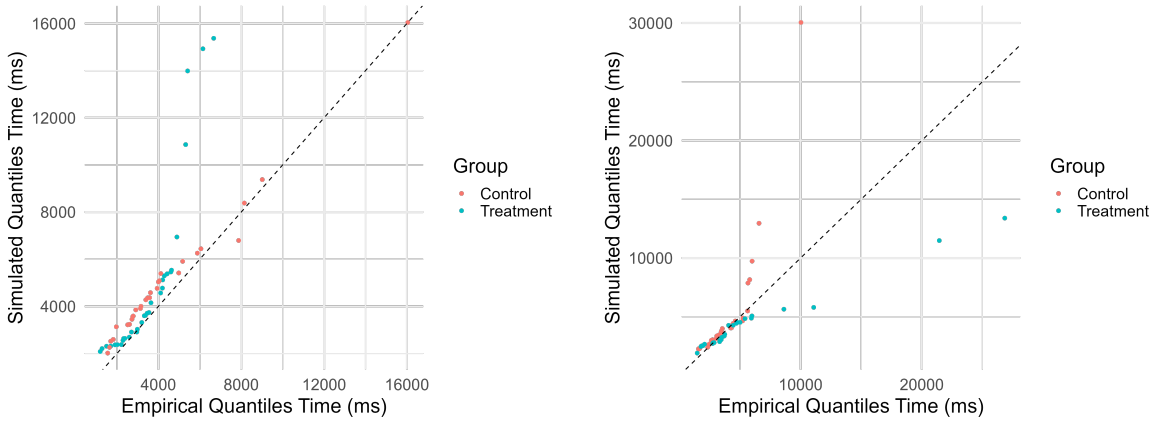
	False	True	Total
Control	11 (6)	23 (28)	34 (34)
Treatment	12 (6)	17 (23)	29 (29)
Total	23 (12)	40 (51)	63 (63)

Annotation: Empirical data in brackets.



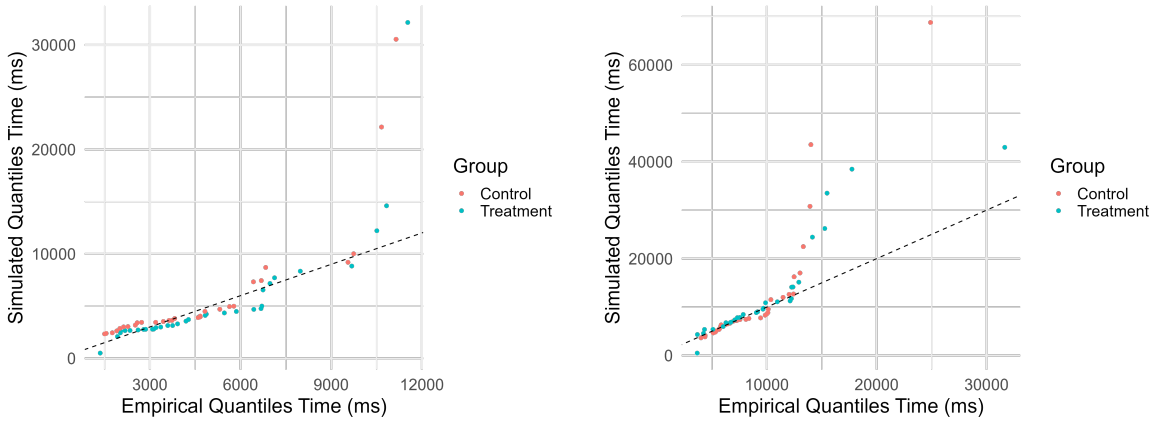
(a) CD1.

(b) CD2.



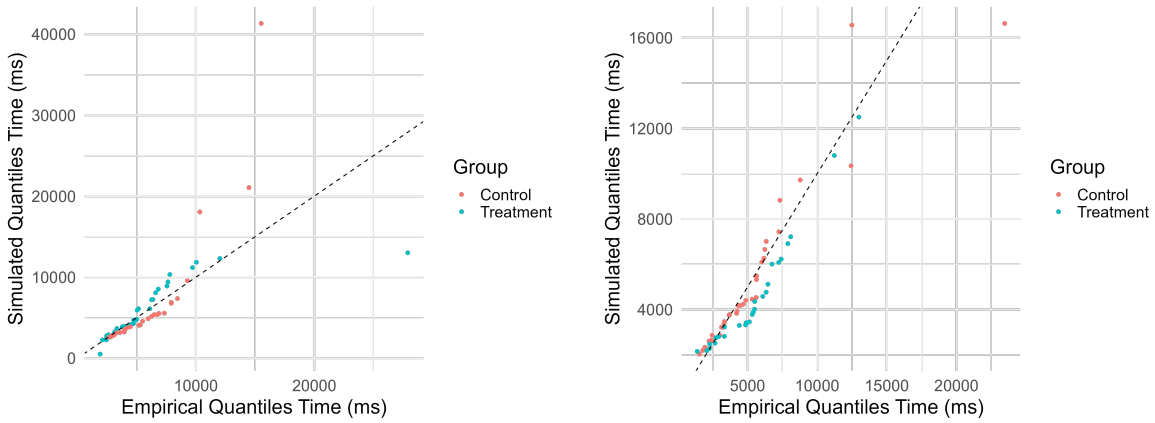
(c) CR1.

(d) CR2.



(e) DR1.

(f) DR2.



(g) RP1.

(h) RP2.

Figure C.3: QQ-Plot for optimizing rt, ans and lf on combined data

Table C.3: Error rates for optimizing rt, ans and lf on combined data

(a) CD1.

	False	True	Total
Control	3 (6)	26 (23)	29 (29)
Treatment	1 (7)	33 (27)	34 (34)
Total	4 (13)	59 (50)	63 (63)

Annotation: Empirical data in brackets.

(b) CD2.

	False	True	Total
Control	2 (1)	32 (33)	34 (34)
Treatment	0 (1)	29 (28)	29 (29)
Total	2 (2)	61 (61)	63 (63)

Annotation: Empirical data in brackets.

(c) CR1.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	14 (2)	20 (32)	34 (34)
Total	14 (2)	49 (61)	63 (63)

Annotation: Empirical data in brackets.

(d) CR2.

	False	True	Total
Control	1 (0)	33 (34)	34 (34)
Treatment	11 (1)	18 (28)	29 (29)
Total	12 (1)	51 (62)	63 (63)

Annotation: Empirical data in brackets.

(e) DR1.

	False	True	Total
Control	8 (4)	21 (25)	29 (29)
Treatment	19 (6)	15 (28)	34 (34)
Total	27 (10)	36 (53)	63 (63)

Annotation: Empirical data in brackets.

(f) DR2.

	False	True	Total
Control	17 (18)	17 (16)	34 (34)
Treatment	18 (9)	11 (20)	29 (29)
Total	35 (27)	28 (36)	63 (63)

Annotation: Empirical data in brackets.

(g) RP1.

	False	True	Total
Control	10 (3)	19 (26)	29 (29)
Treatment	17 (15)	17 (19)	34 (34)
Total	27 (18)	36 (45)	63 (63)

Annotation: Empirical data in brackets.

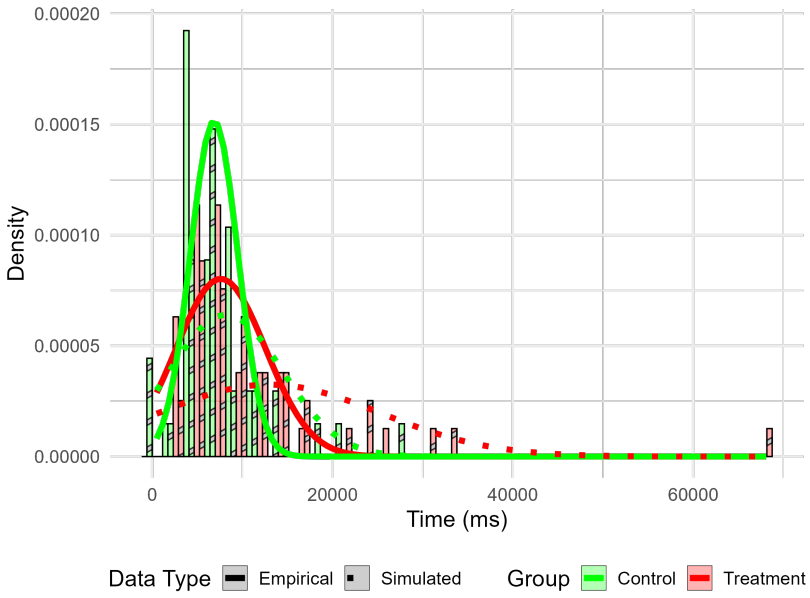
(h) RP2.

	False	True	Total
Control	16 (6)	18 (28)	34 (34)
Treatment	12 (6)	17 (23)	29 (29)
Total	28 (12)	35 (51)	63 (63)

Annotation: Empirical data in brackets.

D

Comparison of Simulated Data and Empirical Data for Single Snippets



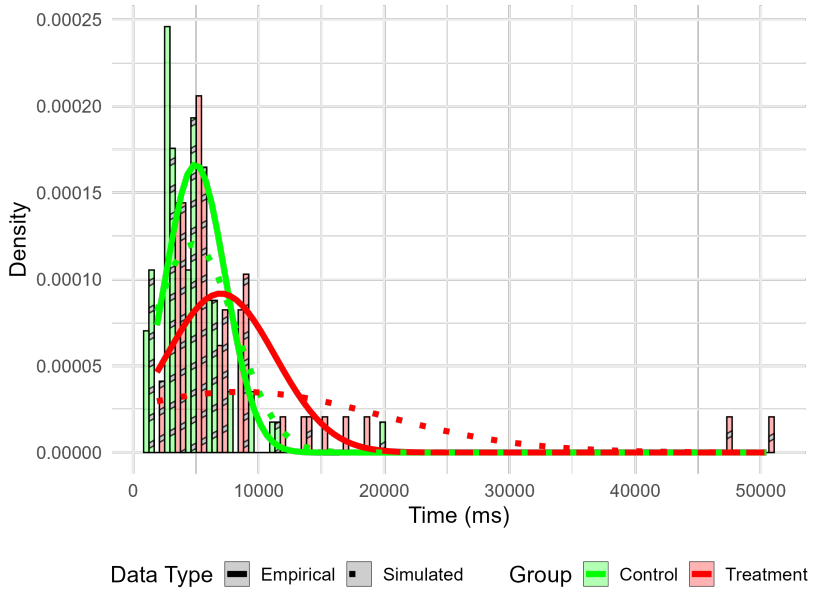
(a) Time effect.

(b) Error rate effect.

	False	True	Total
Control	3 (6)	26 (23)	29 (29)
Treatment	1 (7)	33 (27)	34 (34)
Total	4 (13)	59 (50)	63 (63)

Annotation: Empirical data in brackets.

Figure D.1: Comparison of simulation and empirical data for CD1.



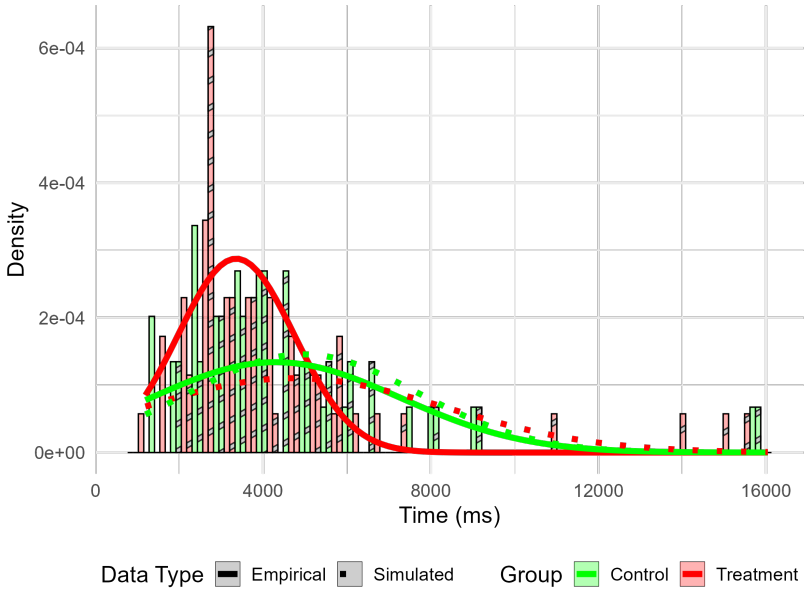
(a) Time effect.

(b) Error rate effect.

	False	True	Total
Control	2 (1)	32 (33)	34 (34)
Treatment	0 (1)	29 (28)	29 (29)
Total	2 (2)	61 (61)	63 (63)

Annotation: Empirical data in brackets.

Figure D.2: Comparison of simulation and empirical data for CD2.



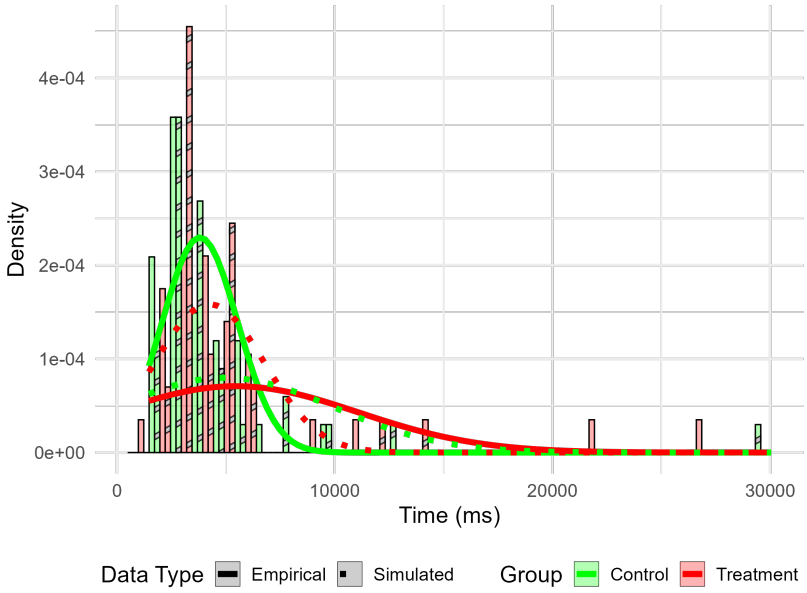
(a) Time effect.

(b) Error rate effect.

	False	True	Total
Control	0 (0)	29 (29)	29 (29)
Treatment	14 (2)	20 (32)	34 (34)
Total	14 (2)	49 (61)	63 (63)

Annotation: Empirical data in brackets.

Figure D.3: Comparison of simulation and empirical data for CR1.



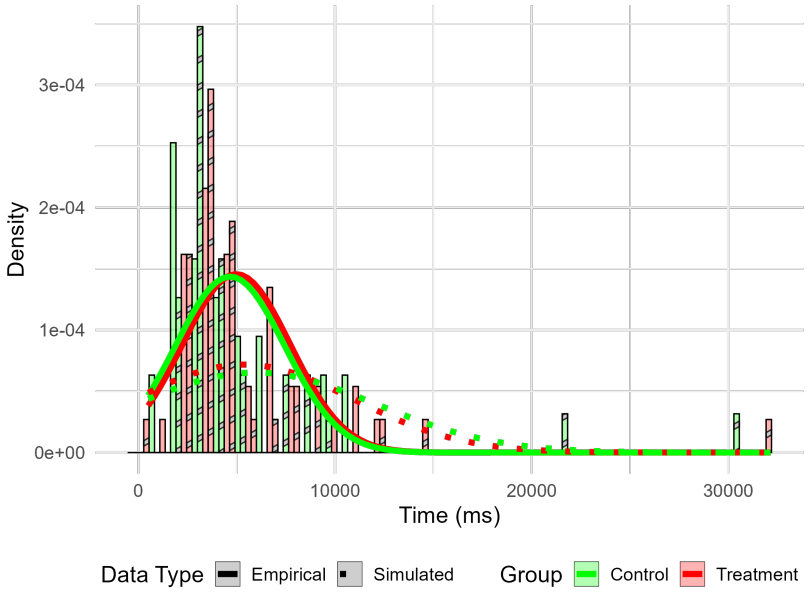
(a) Time effect.

(b) Error rate effect.

	False	True	Total
Control	1 (0)	33 (34)	34 (34)
Treatment	11 (1)	18 (28)	29 (29)
Total	12 (1)	51 (62)	63 (63)

Annotation: Empirical data in brackets.

Figure D.4: Comparison of simulation and empirical data for CR2.



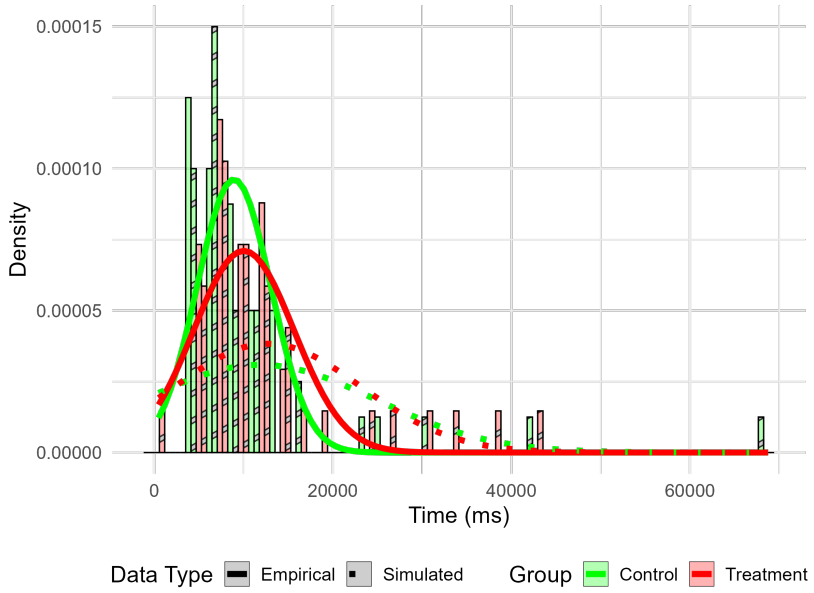
(a) Time effect.

(b) Error rate effect.

	False	True	Total
Control	8 (4)	21 (25)	29 (29)
Treatment	19 (6)	15 (28)	34 (34)
Total	27 (10)	36 (53)	63 (63)

Annotation: Empirical data in brackets.

Figure D.5: Comparison of simulation and empirical data for DR1.



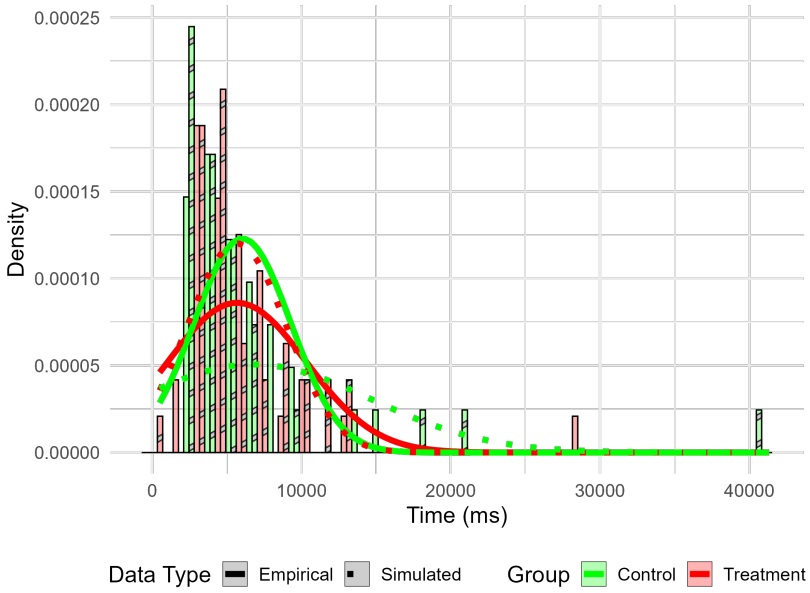
(a) Time effect.

(b) Error rate effect.

	False	True	Total
Control	17 (18)	17 (16)	34 (34)
Treatment	18 (9)	11 (20)	29 (29)
Total	35 (27)	28 (36)	63 (63)

Annotation: Empirical data in brackets.

Figure D.6: Comparison of simulation and empirical data for DR2.



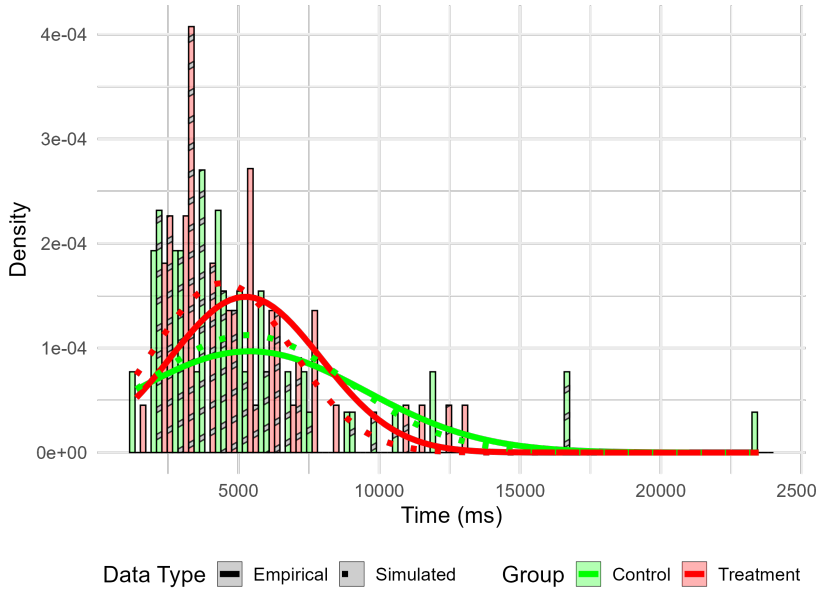
(a) Time effect.

(b) Error rate effect.

	False	True	Total
Control	10 (3)	19 (26)	29 (29)
Treatment	17 (15)	17 (19)	34 (34)
Total	27 (18)	36 (45)	63 (63)

Annotation: Empirical data in brackets.

Figure D.7: Comparison of simulation and empirical data for RP1.



(a) Time effect.

(b) Error rate effect.

	False	True	Total
Control	16 (6)	18 (28)	34 (34)
Treatment	12 (6)	17 (23)	29 (29)
Total	28 (12)	35 (51)	63 (63)

Annotation: Empirical data in brackets.

Figure D.8: Comparison of simulation and empirical data for RP2.

Statement on the Usage of Generative Digital Assistants

This work was created with the assistance of Generative Digital Assistants, specifically ChatGPT, DeepL, and Grammarly.

The Generative Digital Assistants were utilized for the following tasks:

- Translating text
- Optimizing text structure
- Generating R code for structuring diagrams and outputs
- Generating Python code for data processing structures
- Generating LaTeX code for graphic structures
- Conducting source research

The Generative Digital Assistants were explicitly not used for the following tasks:

- Generating entire text sections from brief instructions
- Automatic source integration
- Generating experimental materials
- Analyzing data
- Interpreting results

The authors are aware of the risk of erroneous information from the Generative Digital Assistants. Therefore, all generated text and code were meticulously reviewed for accuracy. No source suggestions were included in the literature of this work without verification of the original texts.

Bibliography

[1] John R. Anderson. "ACT: A simple theory of complex cognition." In: *American Psychologist* 51.4 (Apr. 1996), pp. 355–365. ISSN: 0003-066X. DOI: 10.1037/0003-066x.51.4.355.

[2] John R. Anderson. *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, Oct. 2007. ISBN: 9780195324259. DOI: 10.1093/acprof:oso/9780195324259.001.0001.

[3] 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.

[4] John Robert Anderson, Joachim Funke, Katharina Neuser-von Oettingen, and Guido Plata. *Kognitive Psychologie*. Springer Vieweg. in Springer Fachmedien Wiesbaden GmbH, 2013. ISBN: 9783642373916.

[5] Sebastian Baltes and Paul Ralph. *Sampling in Software Engineering Research: A Critical Review and Guidelines*. 2021. arXiv: 2002.07764 [cs.SE]. URL: https://arxiv.org/abs/2002.07764.

[6] Ruven Brooks. "Using a Behavioral Theory of Program Comprehension in Software Engineering." In: *Proceedings of the 3rd International Conference on Software Engineering*. ICSE '78. Atlanta, Georgia, USA: IEEE Press, 1978, 196–201.

[7] Ruven Brooks. "Towards a theory of the comprehension of computer programs." In: *International Journal of Man-Machine Studies* 18.6 (June 1983), pp. 543–554. ISSN: 0020-7373. DOI: 10.1016/s0020-7373(83)80031-5.

[8] SN Cant, DR Jeffery, and B Henderson-Sellers. "A conceptual model of cognitive complexity of elements of the programming process." In: *Information and Software Technology* 37.7 (1995), pp. 351–362. ISSN: 0950-5849. DOI: https://doi.org/10.1016/0950-5849(95)91491-H. URL: https://www.sciencedirect.com/science/article/pii/095058499591491H.

[9] "Chi-square Goodness of Fit Test." In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 72–76. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_55. URL: https://doi.org/10.1007/978-0-387-32833-1_55.

[10] V. S. Chiarelli. "Learning basic Python concepts via self-explanation: A preliminary python ACT-R model." In: *Virtual MathPsych/ICCM 2021* (July 2021). URL: mathpsych.org/presentation/623..

[11] Herbert H. Clark and C. J. Sengul. "In search of referents for nouns and pronouns." In: *Memory & Cognition* 7.1 (Jan. 1979), pp. 35–41. ISSN: 1532-5946. DOI: 10.3758/bf03196932.

[12] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, May 2013. ISBN: 9781134742707. DOI: 10.4324/9780203771587.

[13] Nelson Cowan. "The magical number 4 in short-term memory: A reconsideration of mental storage capacity." In: *Behavioral and Brain Sciences* 24.1 (Feb. 2001), pp. 87–114. ISSN: 1469-1825. DOI: 10.1017/s0140525x01003922.

[14] Nelson Cowan. "The Magical Mystery Four: How Is Working Memory Capacity Limited, and Why?" In: *Current Directions in Psychological Science* 19.1 (Feb. 2010), pp. 51–57. ISSN: 1467-8721. DOI: 10.1177/0963721409359277.

[15] Meredyth Daneman and Patricia A. Carpenter. "Individual differences in working memory and reading." In: *Journal of Verbal Learning and Verbal Behavior* 19.4 (Aug. 1980), pp. 450–466. ISSN: 0022-5371. DOI: 10.1016/s0022-5371(80)90312-6.

[16] Jared F. Danker and John R. Anderson. "The roles of prefrontal and posterior parietal cortex in algebra problem solving: A case of using cognitive modeling to inform neuroimaging data." In: *NeuroImage* 35.3 (Apr. 2007), pp. 1365–1377. ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2007.01.032.

[17] Mika Dumont, Gordon Hogenson, Saisang Cai, John Parente, Mike Jacobs, and Genevieve Warren. *Move variable declaration near reference*. 2023. URL: https://learn.microsoft.com/en-us/visualstudio/ide/reference/move-declaration-near-reference?view=vs-2022 (visited on 01/11/2024).

[18] Timon Dörzapf. *var-distance*. 2024. URL: https://github.com/Monti1811/var-distances (visited on 01/25/2024).

[19] Pamela D'Addario and Birsen Donmez. "The effect of cognitive distraction on perception-response time to unexpected abrupt and gradually onset roadway hazards." In: *Accident Analysis & Prevention* 127 (2019), pp. 177–185.

[20] Kate Ehrlich and Keith Rayner. "Pronoun assignment and semantic integration during reading: eye movements and immediacy of processing." In: *Journal of Verbal Learning and Verbal Behavior* 22.1 (Feb. 1983), pp. 75–87. ISSN: 0022-5371. DOI: 10.1016/s0022-5371(83)80007-3.

[21] Dror G. Feitelson. *Considerations and Pitfalls in Controlled Experiments on Code Comprehension*. 2021. arXiv: 2103.08769 [cs.SE].

[22] *Free Fish Tank Videos*. URL: https://www.pexels.com/search/videos/fish%20tank/ (visited on 03/10/2024).

[23] Simon Garrod and Anthony Sanford. "Interpreting anaphoric relations: The integration of semantic information while reading." In: *Journal of Verbal Learning and Verbal Behavior* 16.1 (Feb. 1977), pp. 77–90. ISSN: 0022-5371. DOI: 10.1016/s0022-5371(77)80009-1.

[24] Nancy R. Gee, Taylor Reed, April Whiting, Erika Friedmann, Donna Snellgrove, and Katherine A. Sloman. "Observing Live Fish Improves Perceptions of Mood, Relaxation and Anxiety, But Does Not Consistently Alter Heart Rate or Heart Rate Variability." In: *International Journal of Environmental Research and Public Health* 16.17 (Aug. 2019), p. 3113. ISSN: 1660-4601. DOI: 10.3390/ijerph16173113.

[25] 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'17. ACM, Aug. 2017. DOI: 10.1145/3106237.3106264.

[26] ACT-R Research Group. *About ACT-R*. 2012. URL: http://act-r.psy.cmu.edu/about/,mar2012.

[27] ACT-R Research Group. *ACT-R Tutorial*. 2024. URL: /web/20240811152346/http://act-r.psy.cmu.edu/software/ (visited on 08/11/2024).

[28] Michael Hansen, Andrew Lumsdaine, and Robert Goldstone. "Cognitive architectures: A way forward for the psychology of programming." In: Oct. 2012, pp. 27–38. DOI: 10.1145/2384592.2384596.

[29] Wanja A. Hemmerich. *StatistikGuru. Bonferroni–Holm Korrektur*. 2020. URL: https://statistikguru.de/lexikon/bonferroni-holm-korrektur.html (visited on 05/10/2024).

[30] Felix Henninger, Yury Shevchenko, Ulf Mertens, Pascal J. Kieslich, and Benjamin E. Hilbig. *lab.js: A free, open, online experiment builder*. 2024. DOI: 10.5281/ZENODO.597045.

[31] Antti Kangasrääsiö, Jussi P. P. Jokinen, Antti Oulasvirta, Andrew Howes, and Samuel Kaski. "Parameter Inference for Computational Cognitive Models with Approximate Bayesian Computation." In: *Cognitive Science* 43.6 (June 2019). ISSN: 1551-6709. DOI: 10.1111/cogs.12738.

[32] David Kean. *Move declaration near reference" can change behavior of a program for captured variables*. 2018. URL: https://github.com/dotnet/roslyn/issues/25741 (visited on 01/11/2024).

[33] "Kolmogorov–Smirnov Test." In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 283–287. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_214. URL: https://doi.org/10.1007/978-0-387-32833-1_214.

[34] Jan-Peter Krämer, Jan Oliver Borchers, Horst Lichter, and Joel Brandt. *Interacting with code: Observations, models, and tools for usable software development environments*. Tech. rep. Fachgruppe Informatik, 2017.

[35] Christian Lebiere. "The dynamics of cognition: An ACT-R model of cognitive arithmetic." In: *Kognitionswissenschaft* 8.1 (Mar. 1999), pp. 5–19. ISSN: 1432-1483. DOI: 10.1007/bf03354932.

[36] D. J. Leiner. *SoSci Survey (Version 3.5.06) [Computer software]*. 2024. URL: https://www.soscisurvey.de (visited on 05/04/2024).

[37] Stanley Letovsky. "Cognitive Processes in Program Comprehension." In: *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*. Washington, D.C., USA: Ablex Publishing Corp., 1986, 58–79. ISBN: 089391388X.

[38] Mira Leung and Gail Murphy. "On Automated Assistants for Software Development: The Role of LLMs." In: *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Sept. 2023. DOI: 10.1109/ase56229.2023.00035.

[39] Jixing Li, Shaonan Wang, Wen-Ming Luh, Liina Pylkkänen, Yiming Yang, and John Hale. "Cortical processing of reference in language revealed by computational models." In: (Nov. 2020). DOI: 10.1101/2020.11.24.396598.

[40] Julian N. Marewski and Katja Mehlhorn. "Using the ACT-R architecture to specify 39 quantitative process models of decision making." In: *Judgment and Decision Making* 6.6 (2011), 439–519. DOI: 10.1017/S1930297500002473.

[41] Norman Peitek, Janet Siegmund, and Sven Apel. "What Drives the Reading Order of Programmers?: An Eye Tracking Study." In: *Proceedings of the 28th International Conference on Program Comprehension*. ICPC '20. ACM, July 2020. DOI: 10.1145/3387904.3389279.

[42] Nancy Pennington. "Stimulus structures and mental representations in expert comprehension of computer programs." In: *Cognitive Psychology* 19.3 (July 1987), pp. 295–341. ISSN: 0010-0285. DOI: 10.1016/0010-0285(87)90007-7.

[43] Dieter Rasch and Volker Guiard. "The robustness of parametric statistical methods." In: *Psychology Science* 46 (2004), pp. 175–208.

[44] Ben Shneiderman. "Measuring computer program quality and comprehension." In: *International Journal of Man-Machine Studies* 9.4 (1977), pp. 465–478.

[45] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. "Measuring and modeling programming experience." In: *Empirical Software Engineering* 19.5 (Dec. 2013), pp. 1299–1334. ISSN: 1573-7616. DOI: 10.1007/s10664-013-9286-4.

[46] Elliot Soloway and Kate Ehrlich. "Empirical Studies of Programming Knowledge." In: *IEEE Transactions on Software Engineering* SE-10.5 (Sept. 1984), pp. 595–609. ISSN: 0098-5589. DOI: 10.1109/tse.1984.5010283.

[47] Niels A Taatgen and John R Anderson. "Constraints in cognitive architectures." In: *Cambridge handbook of computational psychology* (2008), pp. 170–185.

[48] 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.

[49] Niels Taatgen and Hedderik van Rijn. "Traces of times past: Representations of temporal intervals in memory." In: *Memory & Cognition* 39.8 (May 2011), pp. 1546–1560. ISSN: 1532-5946. DOI: 10.3758/s13421-011-0113-0.

[50] Tim Tiemens. "Cognitive Model of Program Comprehension." In: *Software Engineering Research Center Technical Report* (1989).

[51] Northeastern University. *Gender Inclusive Language in Research*. 2023. URL: https://research.northeastern.edu/app/uploads/sites/2/2023/03/Gender-Inclusive-Language-02.08.2023.pdf (visited on 04/26/2024).

[52] A. Von Mayrhauser and A.M. Vans. "Program comprehension during software maintenance and evolution." In: *Computer* 28.8 (1995), pp. 44–55. DOI: 10.1109/2.402076.

[53] Stefan Wagner and Marvin Wyrich. "Code Comprehension Confounders: A Study of Intelligence and Personality." In: *IEEE Transactions on Software Engineering* (2021), pp. 1–1. ISSN: 2326-3881. DOI: 10.1109/tse.2021.3127131.

[54] Dirk Wentura, Christian Frings, and Bernd Dirksmöller. *Kognitive Psychologie*. Springer Fachmedien Wiesbaden GmbH, 2012. ISBN: 9783531166971.

[55] Rand R Wilcox. *Introduction to robust estimation and hypothesis testing*. Academic press, 2012.

[56] Marvin Wyrich. *Source Code Comprehension: A Contemporary Definition and Conceptual Model for Empirical Investigation*. 2023. arXiv: 2310.11301 [cs.SE].