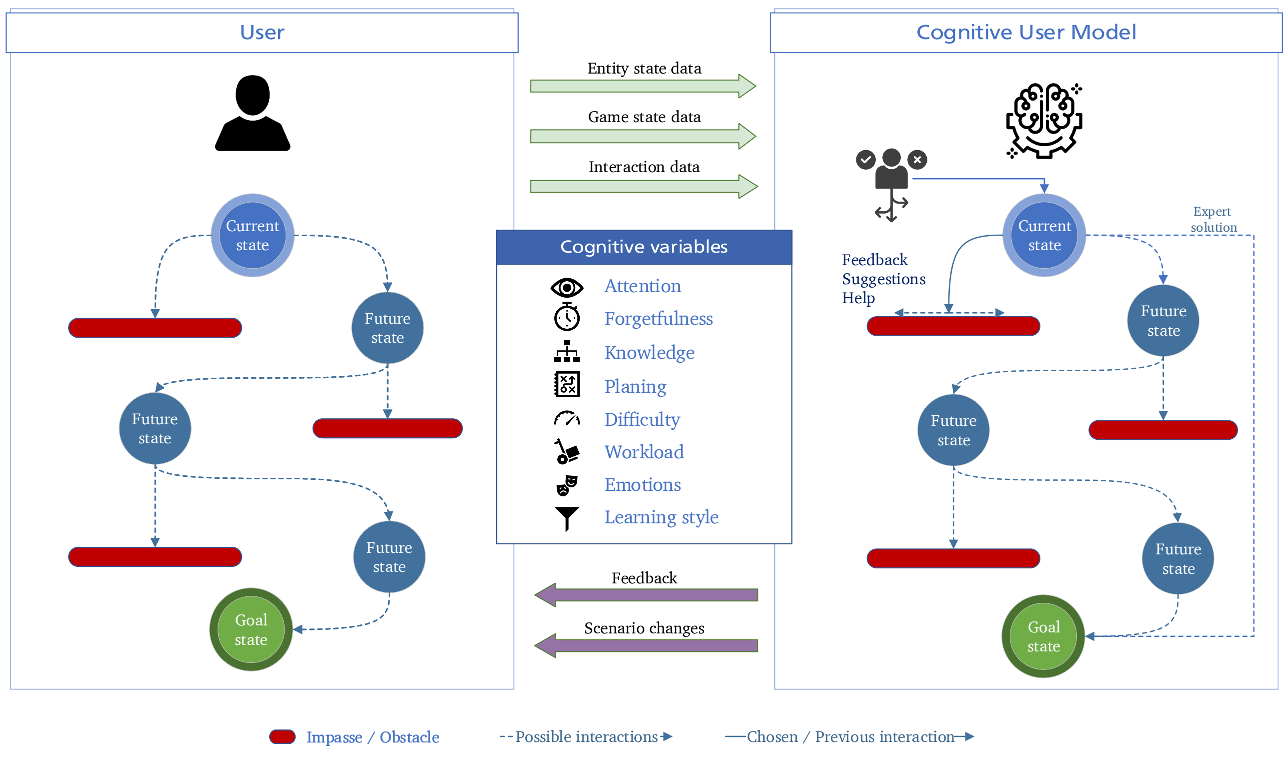
Realizing Cognitive User Models for Adaptive Serious Games

Realisierung kognitiver Benutzermodelle für adaptive Lernspiele
Master-Thesis von Paul Michael Aydinbas
18. März 2019

TECHNISCHE UNIVERSITÄT DARMSTADT

FG Psychologie der Informationsverarbeitung
Fachbereich Humanwissenschaften



In Zusammenarbeit mit der Abteilung Interoperabilität und Assistenzsysteme (IAS) des Fraunhofer-Instituts für Optronik, Systemtechnik und Bildauswertung IOSB

Realizing Cognitive User Models for Adaptive Serious Games
Realisierung kognitiver Benutzermodelle für adaptive Lernspiele

Vorgelegte Master-Thesis von Paul Michael Aydinbas

1. Gutachten: Prof. Constantin A. Rothkopf, TU Darmstadt
2. Gutachten: Dipl.-Inf. Alexander Streicher, Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung IOSB

Tag der Einreichung:

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Paul Michael Aydinbas, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum / Date: ____________________

Unterschrift / Signature: ______________________________

Abstract

Playing games is fun, promotes a wide range of cognitive skills, and has motivational, emotional, and social benefits. What could be more obvious than to use games for learning, an otherwise considered strenuous activity. Educational serious games follow the premise that games are effective learning environments, primary because of their motivational function.
However, in order to keep learners engaged over a longer period of time, games must strike the right balance between being challenging and fun. To allow for experiencing a state of "flow", the game has to be adaptive so it can react to the learner's needs. Adaptivity, though, requires a dynamic assessment of the learner's current cognitive state. The challenge is to intervene adaptively at the right time.
Two methods from the field of computational cognitive science were tested for their applicability to the realization of cognitive user models: Soar 9, a hybrid cognitive architecture and hierarchical Bayesian models (HBMs). Soar was not considered useful for the implementation of cognitive user models, but approaches for the integration of Soar into adaptive systems were demonstrated. The probabilistic programming framework PyMC3 was used to implement four HBMs. The models made use of the Cognitive Load Theory to describe the relationship between the characteristics of the learner and the characteristics of the learning material. The models were validated extensively.
A detailed model comparison was conducted to obtain the best model. The final model is able to explain three observational variables for data sets with different subjects and concepts. The model can accurately predict individual differences as well as group differences. The model's output are probability distributions that allow for arbitrary inferences about the model's parameters and provide the uncertainty associated with these inferences.
The comparison of Soar with PyMC3 showed a clear advantage of PyMC3 for realizing cognitive user models. PyMC3 and HBMs met all the requirements for a cognitive user model. The developed Python package CogIUM can be used to build, train, and test cognitive user models. To the best knowledge of the author, this is the first application of a HBM for realizing cognitive user models for adaptive serious games. Several issues with the final model were identified and discussed. In addition, possible future work to further evaluate the model was presented.

Zusammenfassung

Spielen bereitet Freude, fördert ein breites Spektrum an kognitiven Fähigkeiten und hat motivationale, emotionale sowie soziale Vorteile. Was läge demzufolge näher, als Spiele zum Lernen zu nutzen, eine ansonsten als eher anstrengend empfundene Aktivität. Lernspiele folgen der Prämisse, dass Spiele effektive Lernumgebungen sind, primär aufgrund ihrer Motivationsfunktion.
Um Lerner jedoch über einen längeren Zeitraum zu binden, müssen Spiele das richtige Gleichgewicht zwischen Herausforderung und Spaß finden. Um einen Zustand des „Flows“ zu erleben, muss das Spiel adaptiv sein, damit es auf die Bedürfnisse des Lerners reagieren kann. Anpassungsfähigkeit erfordert jedoch eine dynamische Bewertung des aktuellen kognitiven Zustands des Lerner. Die Herausforderung besteht darin, zum richtigen Zeitpunkt adaptiv zu intervenieren.
Zwei Methoden aus dem Bereich der computergestützten Kognitionswissenschaft wurden auf ihre Eignung für die Realisierung kognitiver Nutzermodelle getestet: Soar 9, eine hybride kognitive Architektur, und hierarchische Bayesianische Modelle (HBMs). Soar wurde nicht als geeignet für die Implementierung kognitiver Benutzermodelle angesehen, aber es wurden Ansätze für die Integration von Soar in adaptive Systeme aufgezeigt. Mit dem probabilistischen Programmierframework PyMC3 wurden vier HBMs implementiert. Die Modelle nutzten die Theorie der kognitiven Last (CLT), um den Zusammenhang zwischen den Eigenschaften des Lerners und den Eigenschaften des Lernmaterials zu beschreiben. Die Modelle wurden umfassend validiert.
Ein detaillierter Modellvergleich wurde durchgeführt, um das beste Modell zu ermitteln. Das endgültige Modell ist in der Lage, drei Beobachtungsvariablen für Datensätze mit unterschiedlichen Personen und Konzepten zu erklären. Das Modell kann sowohl individuelle Unterschiede als auch Gruppenunterschiede genau vorhersagen. Die Ergebnisse des Modells sind Wahrscheinlichkeitsverteilungen, die beliebige Inferenzen in Bezug auf die Modellparameter zulassen und zudem ein Maß für die Unsicherheit der Inferenz liefern.
Der Vergleich von Soar mit PyMC3 zeigte einen klaren Vorteil von PyMC3 bei der Realisierung kognitiver Nutzermodelle. PyMC3 und hierarchische Bayesianische Modelle erfüllten alle Anforderungen an ein kognitives Benutzermodell. Das entwickelte Python-Paket CogIUM kann zum Erstellen, Trainieren und Testen kognitiver Benutzermodelle verwendet werden. Nach bestem Wissen des Autors ist dies die erste Anwendung eines hierarchischen Bayesianischen Modells zur Realisierung kognitiver Benutzermodelle für adaptive Lernspiele. Mehrere Probleme mit dem endgültigen Modell wurden identifiziert und diskutiert. Außerdem wurden mögliche zukünftige Arbeiten zur weiteren Bewertung des Models vorgestellt.

List of Acronyms

CLT	Cognitive Load Theory
CogIUM	cognitive intelligent user model
ECL	extraneous cognitive load
EDM	Educational Data Mining
ELAI	E-Learning Artificial Intelligence
ESS	effective sample size
GCL	germane cognitive load
HBM	hierarchical Bayesian model
HDI	highest density interval
ICL	intrinsic cognitive load
ITS	Intelligent Tutoring Systems
KL	Kullback-Leibler divergence
LA	Learning Analytics
LMS	Learning Management System
LOO-CV	leave-one-out cross-validation
LRS	Learning Record System
MAE	mean absolute error
MCMC	Markov chain Monte Carlo
RMSE	root mean squared error
WME	working memory element
xAPI	Experience API
ZPD	zone of proximal development

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

„Wenn wir“ sagtest Du, „die Menschen nur nehmen, wie sie sind, so machen wir sie schlechter; wenn wir sie behandeln als wären sie, was sie sein sollten, so bringen wir sie dahin, wohin sie zu bringen sind.“
If you treat an individual as they are, they will remain how they are. But if you treat them as if they were what they ought to be and could be, they will become what they ought to be and could be.

In *Wilhelm Meister's Lehrjahre (Book VIII, Chapter four)*
Johann Wolfgang von Goethe

In an ever more complex world that changes more and more rapidly, we should pose the question: What are the skills of the 21st century that we need to prevail and that we should taught our children so that they are prepared for the upcoming future? The relevant skill set of the 21st century is dramatically different from the skills the current educational system values. In the past, traditional educational practices emphasized only one correct answer and promoted conformity and standardization instead of individual solutions. Today, we need critical thinking, creativity, collaboration, and communication. Critical thinking includes the skills scientific reasoning, systems thinking, computational thinking, decision making, and problem solving. Creativity is made of divergent thinking, innovative thinking, originality, inventiveness, and the ability to view failure as an opportunity. Being able to collaborate means being able to work effectively and respectfully with diverse teams, make compromises to accomplish goals, and share responsibility. Finally, communication means to articulate thoughts and ideas in a variety of forms. When classical teaching methods are no longer appropriate for the modern world's requirements, what are the alternatives? (Qian and Clark, 2016)

The rest of this introduction gives a possible answer to this question and states the main topics that guide the research questions of this thesis.

The quote from Goethe already mentioned the importance of individual education—to promote and foster the individual strengths and abilities. New developments in the fields of computer science, artificial intelligence and cognitive science provide complete new opportunities for education. Artificial intelligence research led to systems with a deeper understanding of knowledge, especially procedural knowledge, that is, knowledge about how people perform and solve tasks. Research in cognitive science led to a deeper understanding of how people think, solve problems, and learn. Most importantly, education is no longer understood as a universal, one-sided approach. Cognitive research taught us that individual differences and preferred learning styles influence the learning process. Furthermore, education should be based on individual, one-to-one tutoring instead of conventional teaching. Ideally, students are guided in asking their own questions and gathering evidence that allows them to find answers to those questions. Likewise, teaching meta-cognitive skills can help students to learn more effectively when focusing on their own learning approaches. However, these methods are nearly impossible to implement in classrooms without technology. (Woolf, 2009)

Computers have been used for educational purposes since 1959. These first computer-based or computer-aided instruction systems have been shown to increase student scores by 10 % to 20 %, to decrease the time to achieve goals by one third, and to improve class performance (Woolf, 2009). However, these early systems had several drawbacks, for example inflexible frame-based methods with a pre-defined sequence of topics, and no individual treatment of students, no matter the student's performance. Thus, computer-based instruction systems were developed further into Intelligent Tutoring Systems (ITS). ITS differ from traditional computer-based instruction by generating individual responses to students' input (Woolf, 2009). Besides the ongoing discussion about the overall effect of ITS on learning, it seems justified to say that ITS can be "very effective instructional tools" (Kulik and Fletcher, 2016, p. 67). Students, who learned with the help of an ITS, outperformed students from conventional classes in 92 % of 50 controlled evaluations, and the improvement in performance due to ITS use can be considered "substantively important" in 78 % of the 50 studies.

What are the visions of the field? What should an ideal intelligent tutor software be capable of? Woolf (2009) lists two important key aspects:

ITS treat students individually Intelligent tutors consider each student's background, learning style, and current needs and choose appropriate learning material according to this knowledge to meet the exact need of the student. In addition, ITS infer the student's emotion and affective state and can respond appropriately to these emotions. Examples of affective states are frustration and boredom.

ITS know how to teach The software not only contains the academic material but also qualitative models of each domain to be taught. An ITS follows a student's reasoning about the domain knowledge, engages in discussions, and answers questions on various topics. Secondly, new tutor software is easily build and added onto existing tutors.

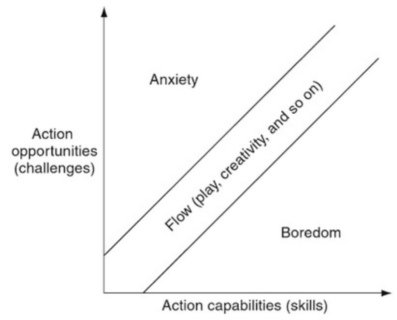
Besides many success stories of ITS that provide learning support, make use of natural language, model complex pedagogical strategies, recognize and respond to differences in student emotion, most of those points are hardly fulfilled even by modern ITS and remain visions (Baker, 2016). In addition, the development of ITS remains "costly and expensive" and inspired a new research field looking into ITS authoring tools (Dermeval et al., 2017).

Although ITS might use simple quizzes or window-based applications as medium, a more natural choice are games. Schools are still designed with the goal of producing “standardized learners and, most importantly, sort students into those groups”, whereas games foster creative thinking and problem solving (Squire, 2005). Play as one form of gaming has been an important medium of learning throughout human history (Pellegrini, 2009a), serving both immediate as well as deferred benefits by increasing behavioral and cognitive flexibility (Pellegrini, 2009b). Nowadays, digital games have become mainstream with an ever growing acceptance: 99 % of boys and 94 % of girls are playing digital games and time spent on playing digital games for this group of age ranges from approximately 7 to 10 hours per week (Plass et al., 2015)—and there is no reason to believe that the appeal of digital games has declined over time (Winn and Heeter, 2009) or decreases with age (De Schutter, 2011).

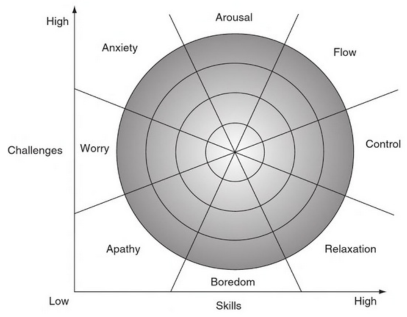
The research field of game-based learning is especially dedicated to research questions about learning outcomes in games. One definition given by Qian and Clark (2016, p. 51) states that “Game-based learning describes an environment where game content and game play enhance knowledge and skills acquisition, and where game activities involve problem solving spaces and challenges that provide players/learners with a sense of achievement”. A game in this sense is “a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome” (Plass et al., 2015, p. 259). Good games are neither too easy, which would result in boredom, nor too difficult, which would result in frustration. Good games aim for the “sweet spot”, where players have a chance to succeed if they are engaged (Plass et al., 2015). This idea is closely related to the state of *flow* and a player’s *zone of proximal development*.

The flow model describes the characteristics of optimal experience and its proximal conditions and tries to understand the phenomenon of intrinsically motivated activities (Nakamura and Csikszentmihalyi, 2009). The conditions for entering the state of flow are twofold: a perceived challenge, that can be overcome with the current skills, and clear proximal goals as well as immediate feedback about the progress towards this goal (Figure 1.1a on the following page) (Nakamura and Csikszentmihalyi, 2009). There exists several updated versions of the flow model that differentiate the challenge/skill terrain further into four or eight experiential “channels” (Figure 1.1b on the next page). Besides the experience of flow, there is another kind of experience in these updated models that might be intrinsically rewarding: one that is characterized by conservation of energy.

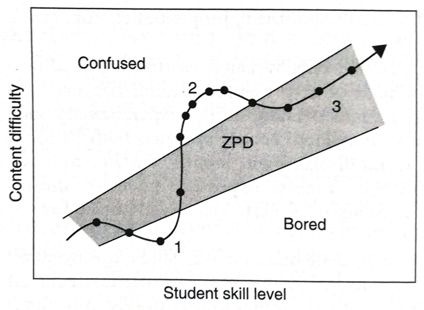
The second, similar concept of zone of proximal development (ZPD) was introduced by Vygotsky (1978) as “the distance between the actual development level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or collaboration of more capable peers”. ZPD is usually used to describe apprenticeship-learning approaches and can be characterized from both cognitive and affective perspectives. Instructional materials should not be too difficult or too easy, and the learner should not experience boredom, confusion, or frustration (Figure 1.1c on the following page) (Woolf, 2009). Plass et al. (2015) list more arguments for the use of games for learning, like their motivational function, a wide range of ways to engage learners, multiple ways of making a game adaptive and of personalizing experiences, and their tolerance for failure.


Anxiety
Action opportunities (challenges)
Flow (play, creativity, and so on)
Boredom
Action capabilities (skills)

(a) The original model of flow state (from Nakamura and Csikszentmihalyi, 2009, p. 196). Flow is experienced when there is a balance between the experienced challenge and the actor's perceived skill set.



(b) The extended model of flow state (from Nakamura and Csikszentmihalyi, 2009, p. 201). In comparison to the original model, balance of experienced challenge an perceived skills is not enough but has to be above the actor's average levels of challenge and skill.



(c) Operational definition of Vygotsky's (1978) zone of proximal development (from Woolf, 2009, p. 126). The diagram depicts a student's trajectory through time in the space of tutorial content difficulty versus a student's evolving skill level.

Figure 1.1.: The flow model and concept of zone of proximal development, two prominent concepts to describe engaging games.

What are the main components of successful games and which game elements support the learner's learning success? The basic structure of all games consists of three key elements: a challenge, a response, and feedback (Figure 1.2 on the next page) (Plass et al., 2015). When we talk about educational games, that is, games that are a subset of serious games with a clear focus on learning, most researchers agree on the following building blocks of games: game mechanics, visual aesthetics, narrative, incentives, musical score, and the learning objective. These design features facilitate different forms of engagement and contribute to a playful experience (Figure 1.3 on the following page). There are two very important principles among the cognitive foundations of game-based learning: scaffolding along with relevant feedback and dynamic assessment (Plass et al., 2015). The first describes a teaching strategy that "controls aspects of a task that are beyond the learner's capabilities, thereby allowing the learner to complete a task that he or she would not be able to do on their own" (Plass et al., 2015, p. 266). This strategy is directly linked to the concept of ZPD. For effective scaffolding to take place the task being solved must fall within the learner's ZPD. Scaffolding can be realized by tutorial levels and appropriate feedback and support, ideally implemented as dynamic instead of static feedback. Plass et al. note, however, that scaffolding has been much more limited in games for learning due to the increased difficulty in doing the dynamic assessment. Effective scaffolding requires an accurate and ongoing assessment of learner's knowledge and skills. This is true for all forms of adaptivity. Process and product data, that is, data from both the activities of the learner and from anything created by the learner within the game, are key information that can be obtained from games to facilitate dynamic assessment.

The goal of dynamic adaptive systems is to keep the learner motivated, to uphold the learner's intrinsic motivation for interaction and learning, and, ultimately, to increase the learning outcome. Dynamic

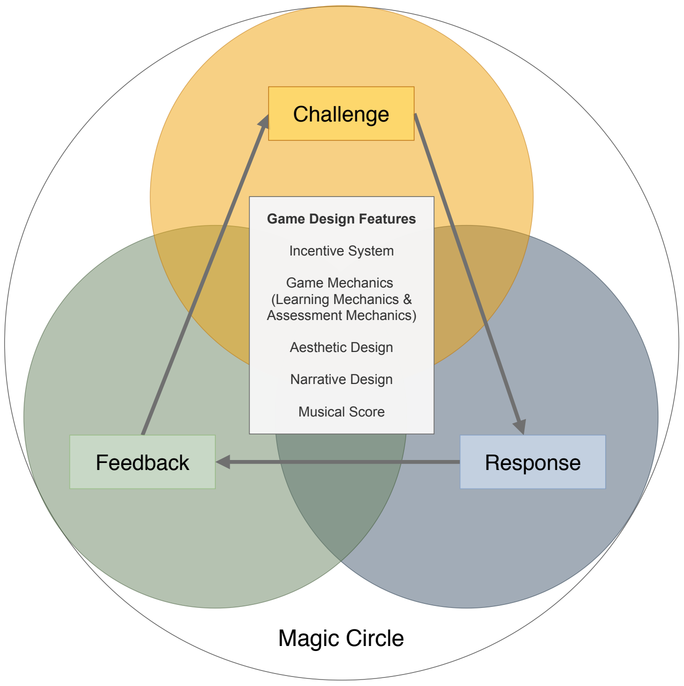


Figure 1.2.: A simple model for game-based learning (based on Plass et al., 2015, p. 262). The key components of any game are a challenge, a response, and feedback. A loop is generated when the feedback of the player leads to a new challenge posed by the game. The game design features are at the center of the learning experience.

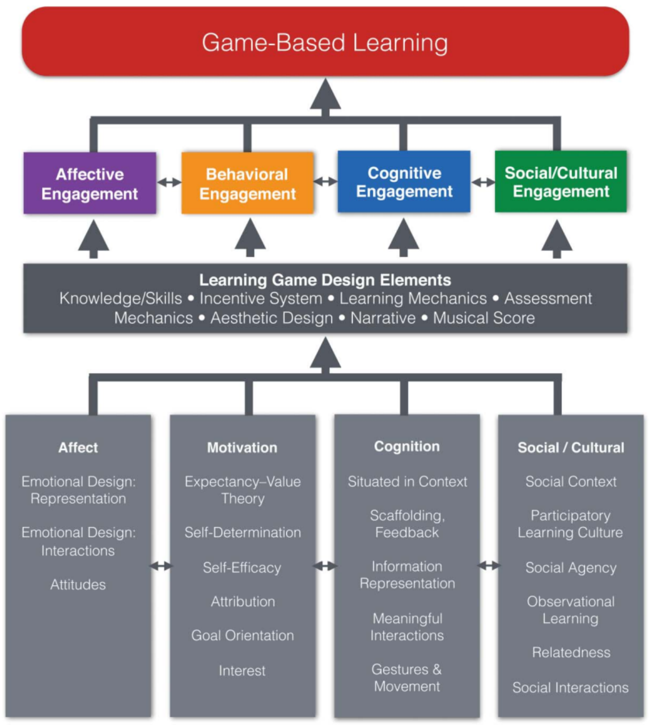


Figure 1.3.: An integrated framework for game-based learning (from Plass et al., 2015, p. 263). Games are uniquely qualified to implement existing models of learning and offer multiple types of engagement. Design features ensure the playful experience.

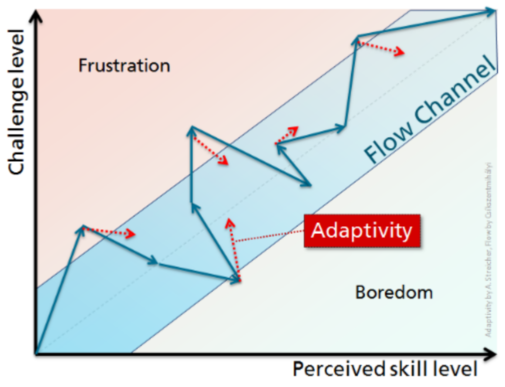


Figure 1.4.: How to stay in the flow channel (from Streicher and Smeddinck, 2016, p. 341). Adaptive measures (dotted arrow-lines) aim at keeping the user's interaction route through the game (solid arrow-lines) within the area where a flow state can be experienced.

adaptive systems can help achieving these goals by adapting the educational game to the needs of the learner according to their knowledge level, skill, and experience. They must consider the "heterogeneity of the users and their varying knowledge levels, cultural backgrounds, usage surroundings, skills, etc." (Streicher and Smeddinck, 2016, p. 334). The aforementioned balance of risk of failure or challenge and the chance to overcome the challenge is not constant over time because a learner's skill is "developing, deteriorating, or temporarily boosted or hindered" during or between different gaming sessions (Streicher and Smeddinck, 2016, p. 342). Here, a dynamic system can adapt the level of challenge to meet the learner's current level of skill and keep them within the flow channel (Figure 1.4). Parasuraman et al. (2000) present a model for types and levels of automation to help decide which system functions should be automated. Shute and Zapata-Rivera (2012) present a four-process adaptive cycle that is based on accurate diagnosis of learner characteristics, for example knowledge, skill, and motivation (Figure 1.5 on the following page). The four stages are capture, analyze, select, and present. For the purpose of this work the most important part is the analyze stage in which a model of the learner is created and maintained in relation to the domain. Information about the learner is typically gained by inferences on current states, that is, the system can infer the learner's knowledge or their intended next actions based on the learner's performance in the learning domain or from the learner's interactions. The student model is the central aspect of any dynamic adaptive system for educational games. Once a student model has been built and trained, in a next step adaptive measures can be selected based on the learner's current status as represented in the student model and the purpose of the system (Shute and Zapata-Rivera, 2012).

1.1 Motivation

Educational serious games, that is, serious games with a clear focus on learning and knowledge acquisition, can lead to better performance, better concept retention, and greater commitment along with the general incentives gaming can offer (Boyle et al., 2016). However, serious games require a permanent balance – between gaming and learning and between the challenge posed by the game versus the player's ability – to assure immersive gaming and effective learning (Kickmeier-Rust and Albert, 2012; Kickmeier-Rust, Mattheiss, et al., 2011). To achieve both enjoyment and successful learning at the same time, appropriate adaptation is a key requirement. Adaptation in this context means that a system is able to automatically

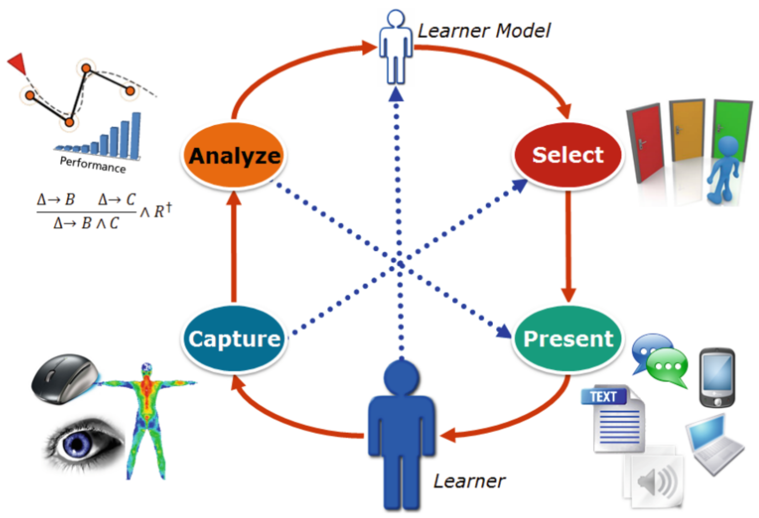


Figure 1.5.: The four-process adaptive cycle to promote learning (from Streicher and Smeddinck (2016, p. 353), based on Shute and Zapata-Rivera (2012)). The learner is connected to appropriate educational material through the use of a learner model.

change its own characteristics to meet the user's current needs (Oppermann, 1994). Adaptation and personalization not only enable effective learning but can lead to "superior gaming experience and educational gains" (Kickmeier-Rust and Albert, 2012, p. 25).

Besides the merits of adaptive systems, current computer simulations and digital game-based learning systems show a profound lack of concepts for didactic adaptivity to the learner's needs (Streicher and Roller, 2017). Furthermore, such systems are most often handcrafted, "one-of-a-kind solutions" (Sottilare and Gilbert, 2011, p. 2).

Intelligent Tutoring Systems (ITS) are facing similar problems. ITS are systems with rich, dynamic models of student knowledge with the ability to adapt their model over time, as the student's understanding becomes more profound (Woolf, 2009). However, creating sophisticated ITS remains costly, complex, and relies on collaborative expertise (Bell, 2015; Woolf, 2009).

A system's ability to show adaptive behavior in an one-to-one tutoring setting relies on its capability to understand the learner's current state. This includes modeling the learner's "unobserved cognitive state" which comprises their affect, readiness to learn, and their comprehension of the material, none of which is directly accessible to observation (Sottilare and Gilbert, 2011, p. 4). To start the adaptation process, the system must constantly measure the current state of the user to be able to react to undesirable or unfavorable states just at the right time (Streicher and Smeddinck, 2016). Timing is a very crucial aspect for the learner's acceptance of the system because the system's response to the learner's actions has to be supportive and not obstructive. A response given too early – for example due to an erroneous prediction of the learner's perceived difficulty level – might hinder the learner from experiencing a challenging problem that allows them to go beyond their current level of mastery of a certain domain or knowledge and thus prevents an optimal learning experience (see the concept of flow, Figure 1.4 on the previous page). Likewise, a response given too late – for example due to the model's inability to accurately capture the learner's current cognitive state – might fail at counteracting a learner's growing feeling of frustration or boredom or a loss in motivation and thus, again, prevents an optimal learning experience. Consequently, a key and open research question for any adaptive system is when to adapt (Streicher and Smeddinck, 2016).

In summary, educational serious games need a build-in adaptivity to react at the right time to the learner's current state. To reach this goal, an ongoing dynamic assessment is necessary. Solutions to this problem exist, but they are either proprietary solutions or overly complex and domain-dependent. Any solution to the problem of realizing dynamic adaptivity for serious games has to implement a model of the learner's internal state along with important cognitive variables. The appropriateness and accuracy of this cognitive user model will be essential to the overall goal of creating adaptive systems that rely on these models.

1.2 Research Objectives and Intended Approach

This thesis tries to find answers to the aforementioned open research question: When to adapt? However, it cannot be the goal to fully answer this question or even to analyze its complete scope. Instead, and as mentioned above, the thesis starts with the foundation of any adaptive system in the context of e-learning – the cognitive user model. Having realized a cognitive user model that accurately captures at least parts of the learner's real internal cognitive state, the model can be used subsequently to infer the best point in time for an adaptive response.

To realize cognitive user models for adaptive systems in an e-learning context, the practicality of two techniques from the field of cognitive modeling will be examined: cognitive architectures and (hierarchical) Bayesian models. The goal is to build and train a cognitive intelligent user model (CogIUM) that both accurately represents a part of the learner's real internal cognitive state and enables the adaptive system to infer appropriate actions to maintain an optimal learning experience for the learner and, in this way, ensures the learner's learning success.

My procedure in this thesis can be summarized by the following steps:

Define scenarios In a first step, eligible scenarios for the system's adaptive behavior have to be defined that frame the search for possible techniques. These scenarios are based on a concrete digital educational serious game and cover only a small part of all the possible actions the game offers. Again, the goal of this thesis is not to develop a model for the whole game but rather to demonstrate that, in general, the chosen modeling technique is appropriate for achieving the research goal. The output of this step will be twofold: First, a set of concrete (inter-)actions plausible for a learner to have executed within the gaming environment. Secondly, a set of observable variables as a direct output of the interactions between the learner and the game along with their domain.

Method selection and model building In a second step and in accordance with these scenarios, appropriate methods from the field of cognitive modeling have to be systematically chosen. As stated earlier, I will limit the set of choices to methods or approaches from the field of cognitive architectures and the field of (hierarchical) Bayesian models. Having selected a method, a cognitive user model of the learner has to be built and trained. This model should allow the adaptive system to assess the learner's current cognitive state with respect to the scenario. Such an assessment could involve affective variables (personality, emotions, or mood), readiness to learn (attention, engagement, and motivation), comprehension, mental load, exhibited knowledge, competence states or skills, incongruent behaviors, individual preferences, traits or aptitudes, learning styles, and many more (Kickmeier-Rust and Albert, 2012; Sottilare and Gilbert, 2011). It is not the goal of this thesis to model all of these cognitive variables, but rather to select the ones that can be modeled up to a degree that will allow for helping to answer the overall research question of when to adapt.

Model validation In a third step, the chosen cognitive user model will be applied to realistic data of a concrete digital educational serious game for image evaluation to validate the concept. Multiple models will be compared to each other to show the process of model selection. The best model will be evaluated on different data sets to demonstrate the model's applicability to a versatile but plausible range of data expected to be gained from learner interactions with the educational serious game.

The main goal of this thesis is to a) evaluate how methods from computational cognitive modeling can be used to realize cognitive user models for digital serious games with a focus on image evaluation, b) list the advantages and disadvantages of the different approaches, and c) in general, estimate their practicality for real world applications. Besides the requirements for a general cognitive model, it is also necessary to account for psychological variables associated with the domain of education like the accomplishment of educational objectives and knowledge acquisition.

1.3 Project Environment

This master thesis was written in cooperation with the department Interoperability and Assistance Systems (IAS)[1] of the Fraunhofer Institute of Optronics, System Technologies, and Image Exploitation IOSB. The IAS offers solutions for human interaction with complex information- and technology-based systems. Their research focuses on computer-supported assistance systems, information management, cooperative work, and decision-making processes. Their work focuses on stations for the analysis of aerial and satellite images along with assistance systems, among others; topics that are directly related to this thesis.

Dipl.-Inf. Alexander Streicher, my advisor and second reviewer of this thesis, is the contact person for two of the IAS research topics: intelligent tutoring interfaces for technology enhanced learning and adaptive learning systems. The topic of this thesis is directly related to the second line of research.

The IAS has already developed solutions for the problem of sustainable learning and adaptive systems. One of their products is the E-Learning Artificial Intelligence (ELAI) Framework (Streicher and Roller, 2017), based on the four-process adaptive cycle for adaptive learning systems (Shute and Zapata-Rivera, 2012). The core of ELAI is an intelligent tutoring controller that, by interpreting the collected learner user data, adjusts the simulation or game accordingly to the learner's needs. The ELAI Framework consists of an ELAI game engine adapter, which has to be programmed for a particular game, and an externalized ELAI Controller, which hosts the actual e-learning "intelligence" with an interpretation and an influence

[1] https://www.iosb.fraunhofer.de/servlet/is/12696

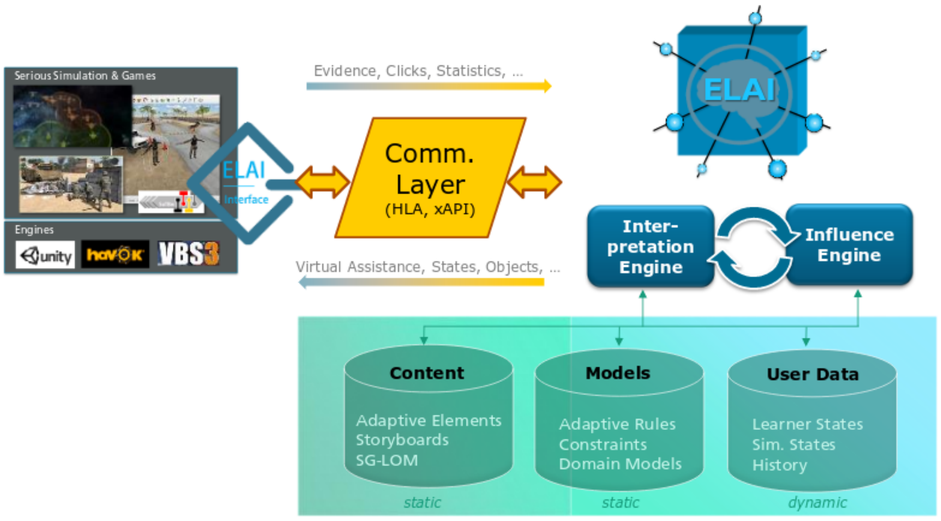


Figure 1.6.: The E-Learning Artificial Intelligence framework (from Streicher and Roller, 2017, Fig. 1). The framework consists of a game engine adapter, a communication layer based on HLA with xAPI payloads, and an external intelligent tutoring agent. The agent interprets the user data with the help of different models to suggest adaptations based on the available content.

engine. So far, the ELAI adapter has been realized for the Unity, Havok, and VBS3 game engines[2]. The purpose of this thesis, the realization of intelligent, cognitive user models, is directly related to the externalized ELAI Controller.

Multiple serious games were developed at the IAS: Seek and Find for Image Reconnaissance (SaFIR) plus adaptivity (SaFIRa) (Streicher, Roller, and Biegemeier, 2017), Exercise Trainer (EXTRA) (Streicher, Szentes, et al., 2016), and Lost Earth 2307[3]. Lost Earth 2307 was developed to help in the training of image interpreters and supports related learning objects like gaining basic knowledge in interpreting and analyzing aerial and satellite imagery and understanding processes of the Reconnaissance-Cycle. Lost Earth is currently ported from the Havoc game engine to the Unity game engine and an early version of this port is used as the concrete digital serious game for this thesis.

1.4 Structure

The rest of this thesis is structured as follows: Chapter 2 Conceptual Background (page 12) provides the reader with the theoretical background about computational cognitive models in cognitive science, about the Experience API as a standard for describing and exchanging user interactions, and about the Cognitive Load Theory, one example of a learner attribute that plays a central role in the realized cognitive intelligent user model.

Chapter 3 Literature Review (page 53) begins with the current state of the art of cognitive architectures, and a short overview of current approaches to realize student models in Intelligent Tutoring Systems. Next, different ways of model comparison and model selection for Bayesian models are presented. The presented metrics are used in the implementation chapter to decide which of the built models to keep.

[2] see https://www.iosb.fraunhofer.de/servlet/is/77629/
[3] https://www.iosb.fraunhofer.de/servlet/is/58015/

The end of this chapter deals with how to measure cognitive load according to the Cognitive Load Theory, which is important to verify the model's external validation.

In the previous section Research Objectives and Intended Approach I listed three concrete steps to find answers to the research questions. Chapter 4 Concept (page 69) deals with the first step, chapter 5 Implementation (page 103) with the third step, while the second step is divided among the two chapters because model building requires both a concept as well as an implementation. Chapter 4 begins with a description of possible interactions between the learner and the cognitive user model. After that follows a description of possible interaction patterns for the serious game Lost Earth along with a description of observable variables that can be gained from the interactions between a user and the game and that serve as the input for the cognitive user model. The last section covers the concepts to realize cognitive user models with both of the approaches cognitive architectures and Bayesian models.

The actual implementation is covered in the chapter 5. I describe the library PyMC3 that allows for probabilistic computations and the developed Python package CogIUM to build, train, and evaluate cognitive user models. The main contribution of this chapter is section 5.3 Validation (page 115), in which I validate different cognitive user models, give an example of how to extend the model to additional observations and validate the final model on different data sets. I also discuss the findings regarding the model comparison process. The remainder of this chapter is used to discuss how the model can be further extended.

The second last chapter 6 Application Example (page 146) gives a short example of how to use the CogIUM Python package, what the output of the model looks like and how to interpret the results.

The thesis concludes with chapter 7 Conclusion and Recommendations (page 152), which gives a summary of the work along with the main contributions, states open research questions, and lists possible links to future work.

2 Conceptual Background

This chapter lays the theoretical foundations for all concepts and methods used in this thesis. First, the main concepts of the field of computational cognitive modeling are presented with a focus on cognitive architectures and hierarchical Bayesian models. The sections provide an answer to the question how user behavior can be modeled and understood. Next, a standard for capturing user data is presented: the Experience API. The final section presents a cognitive theory that helps to understand how different learning material can influence a learner's performance: the Cognitive Load Theory.

2.1 How to Model User Behavior – Paradigms in Cognitive Science

Cognitive science is the interdisciplinary study of the human mind, combining the perspective of many different fields such as philosophy, psychology, artificial intelligence, robotics, neuroscience, linguistics, and anthropology (Friedenberg and Silverman, 2006; Thagard, 2005). Cognitive science wants to explain the various kinds of human thinking with a focus on methods and strategies the mind utilizes to solve the huge variety of problems that humans have to face from day to day. Most cognitive scientists agree to the idea that knowledge in the mind consists of mental representations in the form of rules, concepts, images, and analogies. In addition, cognitive science states that people have mental procedures that operate on these mental representations to produce thought and action. Thagard (2005) calls this central hypothesis the Computational-Representational Understanding of Mind (CRUM). Although CRUM might be wrong and the "mind as computer" metaphor (Table 2.1) has fallen into disfavor (Chown, 2004, p. 2), this analogy of the mind as an information processor that manipulates mental representations with the help of computational procedures, like a computer manipulates data structures with the help of algorithms, has been the "most theoretically and experimentally successful approach to mind ever developed" (Thagard, 2005, p. 11).

Computational cognitive modeling "embodies descriptions of cognition in algorithms and programs" and thus "provides detailed descriptions of mechanisms (i.e., static aspects) and processes (i.e., dynamic aspects) of cognition." (Sun, 2009, p. 125). Sun notices that computational models can match actual human data in a variety of ways and thus can be validated differently. First, such models can be broad or narrow with respect to the covered behavioral data, precise or imprecise, and descriptive or normative. Furthermore, he states three types of correspondences between computational models and human behavior in an increasing order of precision. In the *behavioral outcome modeling*, a computational model yields nearly the same types of behaviors as humans do, given the same circumstances. In

Table 2.1.: The "mind as a computer" metaphor (based on Thagard, 2005, p. 11).

Entity	Requirements	Output
Program	data structures + algorithms	→ running programs
Mind	mental representations + computational procedures	→ thinking

qualitative modeling, the computational model can produce the same qualitative behaviors resembling human performance. In *quantitative modeling*, the computational model's performance is no longer only close to human performance but manages to produce exactly the same quantitative behaviors that were observed in humans as indicated by some qualitative performance measure. (Sun, 2009)

Besides its focus on computational modeling, the primary method of cognitive psychology is experimentation with human participants. The experiment is the ultimate test of any theory (Chown, 2004). To be able to answer questions about the nature of the mind, the findings of a psychological experiment need to be interpretable within a theoretical framework that posits mental representations and procedures. To develop these theoretical frameworks, building and testing computational models intended to be analogous to mental operations is one of the most promising ways (Thagard, 2005). Therefore, a typical paper in a cognitive science conference proceeding will entail the presentation of the findings of a single experiment or a set of psychological experiments on some area of cognition, a model to account for the data, and computer simulations of the model (Chown, 2004).

McClelland (2009, p. 16) argues that the "essential purpose of cognitive modeling is to allow investigation of the implications of ideas, beyond the limits of human thinking." Any discovery made by or gained through such a model is based on what McClelland calls a particular set of specified properties to avoid the word assumptions. Assumptions give rise to consequences and based on observations of this type researchers attempt to draw implications for the nature of human cognition. Because cognitive science's main focus is to capture the essence of human cognitive abilities, it is important to consider both the sufficiency of a model as well as its ability to explain experimental data. When assessing the adequacy and appropriateness of a model, McClelland (2009, p. 21) suggests posing the following question: "Does a model carry out some task as well as humans do, and does it achieve optimality to the same degree and deviate from optimality in the same way as humans?".

Before we come to a description of two of the main paradigms in the field of cognitive science, I want to take a closer look at the term model. A computational model makes the structures and processes postulated by a cognitive theory more precise by interpreting them by analogy with computer programs. Vague representations must be specified by computational ideas about data structures as well as mental processes must be defined algorithmically (Thagard, 2005). A good model of cognition is both predictive and prescriptive. The first describes the model's ability to describe people's likely behavior in different scenarios. The second describes the model's limitations in cognition and possible ways to overcome these limitations (Chown, 2004). Again, the ultimate measure of the model's value is the model's ability to make accurate predictions. However, the fact that a model is consistent with a particular body of facts does not mean that the model is correct, or that the properties it embodies resemble the system it is intended to model. The only thing that can be said is that this set of ideas cannot be ruled out (McClelland, 2009). On the other side, the claim that a modeler has a "virtually infinite toolkit of things to play with in fitting a model to data" (McClelland, 2009, p. 23) is simply not the case, and arbitrarily changing the model's parameter will not ensure that a given model can be fit to any body of data. Instead, real progress actually can and does occur because achieving a good fit to a complex body of data is not at trivial task and its success is not assured (Sun, 2009).

When comparing different information processes, it may be helpful to describe them at different levels. The tri-level hypothesis postulates three different levels to evaluate mental or artificial information

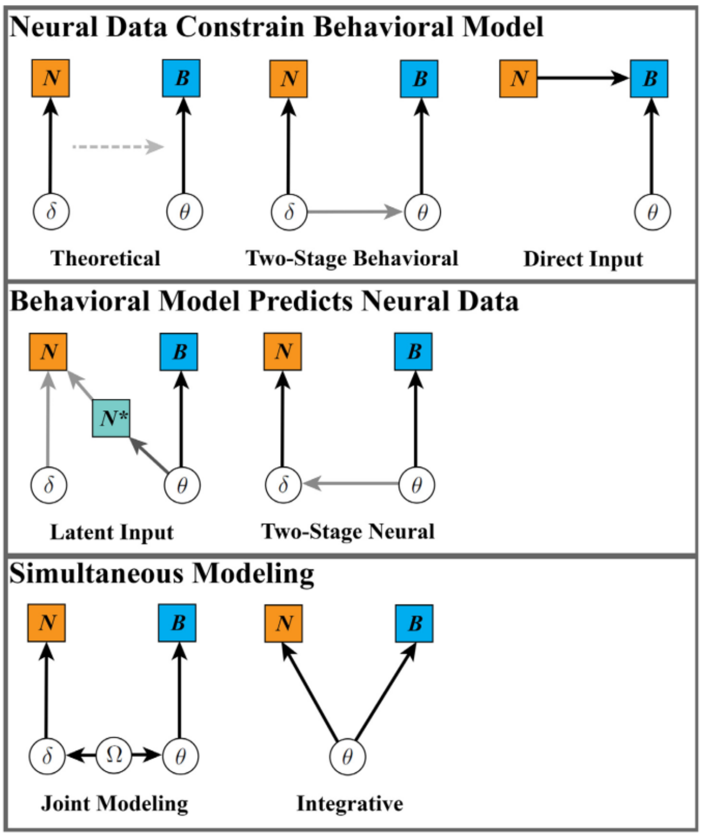


Figure 2.1.: Three main categories of modeling approaches in cognitive neuroscience (from Turner et al., 2017, p. 67). N represents the neural data, B represents the behavioral data, N^* represents simulated internal model states, and θ, δ, and Ω represent model parameters. The authors count the cognitive architecture ACT-R to the simultaneous modeling approaches.

processing events (Marr, 1982). The first and most abstract level of analysis is the computational level. This level is concerned with the two tasks of specifying the goal of computation and the problem it attempts to solve as well as the purpose or reason for the process. The second, more narrowed down level of analysis is the algorithmic or programming level. This level is concerned with formal procedures that act on informational representations and attempt to answer the question what information processing steps are being used to solve the problem. The third and most specific level of analysis is the implementation or hardware level. This level is concerned with the physical realization of the information processor and its constituting parts. (Friedenberg and Silverman, 2006)

Another common distinction between computational models is the distinction between understanding computation from a symbolic or a connectionist perspective. First the symbolic perspective. A formal symbol manipulator is syntactic or rule-governed and operates on representations that are independent of the content of those representations. The manipulations are physically and take some time to occur. Knowledge is localized in the form of symbols. In the connectionist view, instead, knowledge is represented as a pattern of activation or weight distributed throughout a network, and processing happens not in discrete stages but rather in parallel (Friedenberg and Silverman, 2006). As we will see in the next section General artificial intelligence – cognitive architectures, the boundaries between these different categories are not as absolute as they may appear and modern systems incorporate ideas from both domains (Chown, 2004; Kotseruba and Tsotsos, 2018).

Because cognitive science aims for understanding and explaining the human mind, patterns of neural activity from studies in cognitive neuroscience has to be considered along with behavioral data from cognitive experiments. Neuroscience can help to better understand and improve models due to a additional source of constraints and information regarding information flow, modularity, mechanisms, and timing (Chown, 2004). Often enough, both groups of mathematical psychologists and cognitive neuroscientists keep within their restricted tight single-level focus, thinking in terms of Marr's (1982) level of analysis, and opportunities from combining both approaches are missed (Turner et al., 2017). Turner et al. have analyzed several approaches in cognitive neuroscience according to how they model the link between neural and behavioral data (Figure 2.1 on the previous page). They argue for at least three general categories of approaches. In the first category, neural data constrains the behavioral model. In the second category, the reverse can be found, and a behavioral model predicts neural data. The third category comprises approaches which build a single model that jointly accounts for both neural and behavioral data, that is simultaneous modeling. Cognitive architectures like ACT-R (Anderson, 2007) that aim for an integrative approach, can be put in the third category as they integrate neural and behavioral measures which allows the ACT-R model to be used in both exploratory and confirmatory research (Turner et al., 2017). The Bayesian modeling approach is much richer in the ways it can be applied, but Turner et al. (2017) list a concrete example of a hierarchical Bayesian model used in a joint modeling framework to build the connection between neural and behavioral measures. Although integrative models might be the most desirable way to go, it is naturally the most difficult one with the most constraints to be met.

An in-depth description about the different cognitive modeling paradigms can be found in Sun (2008) and commentaries to the different frameworks in McClelland (2009). The covered paradigms are connectionist models, Bayesian models, dynamical systems approaches, declarative/logic-based models, and cognitive architectures. In this thesis I try to answer the research questions using two concrete models from the aforementioned paradigms: Soar, a cognitive architecture, and hierarchical Bayesian models. The following two sections will introduce the main ideas behind these two paradigms.

2.1.1 General artificial intelligence – cognitive architectures

Simon (1957) once declared that "AI can have two purposes. One is to use the power of computers to augment human thinking. The other is to use a computer's artificial intelligence to understand how humans think."

The term *cognitive architecture* was first coined by Allen Newell and his colleagues in their work on unified theories of cognition (Newell, 1994). Historically, the aim of cognitive architectures was threefold: to capture the basic mechanisms of human cognition (e.g., reasoning, control, learning, memory, attention, adaptivity, perception, and action), to form the basis for the development of "cognitive capabilities through the ontogeny over extended periods of time", and to achieve human level intelligence, which is also called General Artificial Intelligence (Lieto, Bhatt, et al., 2018, p. 1). All cognitive architectures have the common goal of creating unified theories of cognition.

Sun (2007, p. 2) defines a cognitive architecture as follows:

> "A cognitive architecture is a broadly-scoped, domain-generic computational cognitive model, capturing the essential structure and process of the mind, to be used for a broad, multiple-level, multiple-domain analysis of behavior."

Cognitive architectures focus on the constant and task-independent aspects of cognition (Vernon, Metta, et al., 2007). Cognitive architectures aim for a broad and cross-domain analysis of cognition by providing a framework that is used to model cognitive phenomena. As a framework a cognitive architecture specifies essential structures, the division of modules, the relations among models, and other important aspects. By choosing a cognitive architecture, the modeler agrees to the embodied fundamental theoretical assumptions of the framework and limits themself to only those models that are possible to realize within the framework. It is important to understand that cognitive architectures "cannot accomplish anything in their own right" (Vernon, Metta, et al., 2007, p. 162). Only when the cognitive architecture is combined with particular knowledge to form a cognitive model, actual intelligent behavior can emerge. In most cognitive systems human designers are responsible for determining the knowledge that should be incorporated into the cognitivist system.

The early scientific vision of the 'cognitivist' approach to Artificial Intelligence (AI) aimed at "understanding and reproducing, in computational systems, the full range of intelligent behavior that we observe in humans" (Langley, 2012, p. 3). Besides the ongoing effort, artificial systems with human-like and human-level intelligence are still out of reach (Lieto and Radicioni, 2016). In general, when it comes to the explanatory role we attribute to artificial models and systems, we can differentiate between the methodological approach of functionalism and the 'structural' approach. The former postulates a weak equivalence between cognitive processes and AI procedures. Thus, comparisons between the 'natural mind' and 'artificial software' are restricted to macroscopic equivalence of the functional organization of the two systems. The latter sees an epistemological need of artificial models whose 'functions' are implemented in such a way that they resemble the biological and cognitive 'structures' and 'constraints' of human cognition. A detailed comparison of the cognitivist and emergent paradigms of cognition can be found in Vernon, Metta, et al. (2007). Only when the artificial system can be considered a good 'proxy' of a target cognitive system it can play an explanatory role about it and help to gain results useful in "refining or rethinking theoretical aspects concerning the target biological system used as source of inspiration" (Lieto and Radicioni, 2016, p. 2).

Despite the differences between these perspectives, both the cognitivist and the emergent perspective are compliant with the structural approach and the design of a cognitive architecture is usually driven by a set of general desirable desiderata (Lieto and Radicioni, 2016; Sun, 2004; Vernon, Hofsten, et al., 2016). The role of the desiderata is to ensure a cognitive architecture's capacity for development that is "driven by both exploratory and social motives" (Vernon, Hofsten, et al., 2016, p. 120). Desiderata are also important to characterize and classify cognitive architectures as psychologically oriented (Hélie and Sun, 2014). Instead of focusing on narrow domains, as 'expert systems' do, psychologically oriented cognitive architectures aim to capture human level performance in a wide variety of domains. This is possible because the cognitive architecture includes only minimal initial structures and independently learns from its own experiences. Hélie and Sun (2014) list five essential desiderata to model autonomous learning in a psychologically-realistic way. Sun (2007) states three reasons for the importance of psychologically oriented cognitive architectures: they help advancing the understanding of human cognition, they serve as a foundation for understanding human behavior, and they are ‘intelligent’ systems that are cognitively realistic and more human-like.

Sun (2007) names four different levels of analysis that range from most macroscopic to the most microscopic: the sociological level, the psychological level, the componential level, and the physiological level. A cognitive architecture can play a role as a ‘centerpiece’, combining and aligning different strands of research. On one hand, detailed mechanisms are developed within a cognitive architecture, probably tied to low-level cognitive processes. On the other hand, the cognitive architecture as a whole may provide very high level cognitive abilities.

The literature differentiates between three major classes of cognitive architectures: cognitivist, connectionist, and hybrid (Gudivada, 2016; Kotseruba and Tsotsos, 2018). Cognitivist architectures use explicit symbolic representations to represent information. Cognitivist architectures are also called symbolic architectures and artificial intelligence approaches. Despite being quite successful, they lack generality to be useful across domains (Gudivada, 2016). In connectionist architectures information is processed by simple, connected computational units, which communicate in parallel. The units in the network receive stimuli through their incoming connections, perform some non-linear computation on it, and affect other neurons through their outgoing connections. Connectionist architectures are also called emergent architectures. Finally, hybrid cognitive architectures use a combination of symbolic and emergent architectures for their components.

Newell and Simon (1976) formulated two hypotheses that help with better understanding cognitivist architectures

1. The Physical Symbol System Hypothesis: A physical symbol system has the necessary and sufficient means for general intelligent actions.
2. Heuristic Search Hypothesis: The solutions to problems are represented as symbol structures. A physical-symbol system exercises its intelligence in problem-solving by search, that is, by generating and progressively modifying symbol structures until it produces a solution structure.

Because a symbol system in the sense of Newell and Simon comprises two recursive loops in which processes can produce processes and patterns can designate patterns (Figure 2.2 on the following page), the system is able to build “ever more abstract representations and reason about those representations” as well as modify itself as a function both of its processing and of its representations (Vernon, Metta, et al., 2007, p. 155).

he information presented up to this point should suffice to grasp a general understanding of what cognitive architecture try to achieve, how they are related to cognitive models, and how they can be characterized. More information about the current state of the art of cognitive architectures can be found in the related section 3.1.1 Cognitive architectures (page 53) of the chapter Literature Review. As a concrete cognitive architecture to realize CogIUMs Soar was chosen. Soar and ACT-R are two of the most popular, longest actively developed and best supported cognitive architecture. I decided to use Soar because it seemed easier to start with and to get productive. The next section will have a closer look on Soar, its structure, evolution, and latest development.

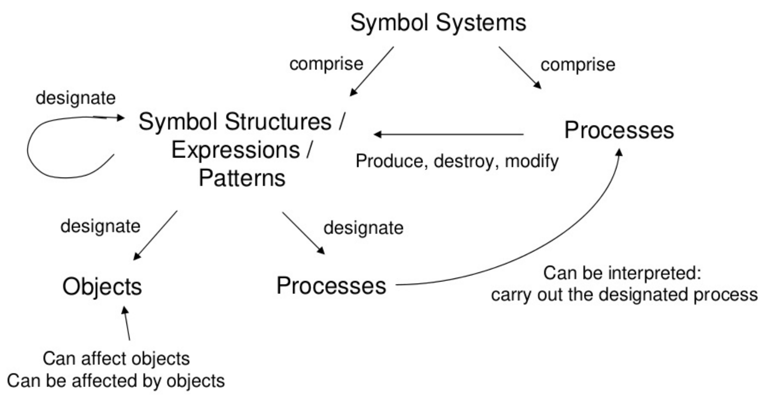


Figure 2.2.: The main components and processes of a physical symbolic system (from Vernon, Metta, et al., 2007, p. 154).

The Soar cognitive architecture

Soar is one of the oldest and most successful cognitive architectures and was developed by Laird, Newell, et al. (1987). The architecture existed since the mid 1982 and started as a classical symbolic manipulation system: Soar uses an explicit symbolic representation of its tasks, which in return are manipulated by symbolic processes. Knowledge is encoded in symbolic structures and guides the system's behavior. The system's behavior is further controlled by a general scheme of goals and sub goals that represent what the system wants to achieve at any given state in time. The development of Soar is guided by the search for a minimal set of mechanisms that are sufficient to realize the complete range of intelligent human behavior. Although Soar is often categorized as a typical member of the cognitivist architectures, other authors put Soar in the hybrid category as it combines symbolic concepts and rules with sub-symbolic elements such as activation values, spreading activation, and reinforcement learning (Gudivada, 2016; Kotseruba and Tsotsos, 2018).

Soar distances itself from other architectures by embodying mechanisms and organizational principles that express hypotheses about the nature of the architecture for intelligence. I will shortly summarize these characteristics because they help in gaining a clear picture of the distinctive features of Soar, and in understanding how Soar operates and what possible outcomes of the architecture might look alike. The order of the characteristics along with their descriptions is in accordance with (Laird, Newell, et al., 1987). Lehman et al. (1998) list another but very similar six characteristics of what Soar posits about any human cognitive behavior.

Problem Space Hypothesis Every task in Soar is formulated as finding a desired state in a problem space. Operators are procedures that, when applied to a current state, yield a new state. Without episodic memory, which was only recently implemented in Soar, there is no knowledge about any states of the problem space but the current one, and Soar can generate new states only by applying appropriate operators to the current state. Similarly, Soar does not know about other possible or valid states in the problem state because they do not pre-exist as data structures but have to be generated by applying operators to existing states.

Universal subgoaling

Any decision in Soar can be related to searching a problem space. If the available knowledge is sufficient, Soar can advance in choosing an operator and applying it to the current state. However, if the current knowledge is insufficient (e.g., which operator to choose, how to perform the chosen operator, how to evaluate the result), Soar creates a subgoal with the aim to deduce the missing knowledge. This can be done for any problematic decision and is called "universal subgoaling". This behavior is recursive so that Soar involves a tree of subgoals and problem spaces.

Long-term knowledge is represented as a production system

The uniform production system delivers control knowledge, as when to propose appropriate operators, how to decide between proposed operators or when to reject proposed operators, and procedural knowledge. The actual data structures are held in the system's short-term working memory and the long-term storage in form of productions lead to actions that manipulate or generate the data structures. All satisfied productions (i.e., the conditions of the production meet the current state) are fired in parallel.

Fixed decision procedure

A fixed decision procedure determines the next action based on preferences. Preferences are one type of data element that represents knowledge about which operators are suitable for Soar in the current situation. Preferences are limited to a build-in set of concepts: acceptability, rejection, better (best, worse, and worst), and indifferent.

Automatic subgoaling

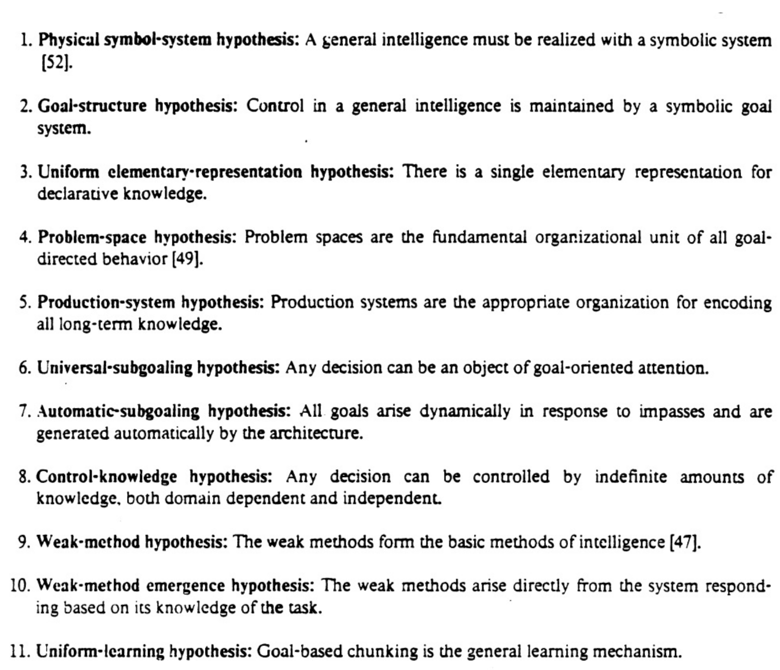
Insufficient knowledge leads to impasses which in return create goals to be overcome. Whenever Soar cannot continue with problem solving due to insufficient knowledge, an impasse is reached. The architecture detects impasses due to an inability of the fixed decision procedure to conclude the next action. This is when the architecture creates a new sub goal under the current goal to overcome the impasse of a failed decision procedure. Goals are only created in response to an impasse. The build-in four different impasse types are enough to create all types of sub goals.

Continuous monitoring of (sub)goal termination

Soar continuously monitors for the termination of all active goals in the goal hierarchy (see previous point). Whenever a goal is terminated, Soar proceeds immediately from the point of termination. As all goals besides the goals on the first level were created as subgoals to overcome impasses, this means that the decision at the higher level will be made immediately. All working memory elements that were created in the process of overcoming an impasse and thus belong to the state of the terminated goal are automatically removed.

Basic problem-solving methods are a side-product

So-called weak methods, such as hill climbing, means-end analysis, etc., are not directly implemented in Soar but are realized by adding search-control productions to the production system that express knowledge about the task. Soar will conduct in a manner that is similar to these weak methods if it has the knowledge to do so.



1. **Physical symbol-system hypothesis:** A general intelligence must be realized with a symbolic system [52].
2. **Goal-structure hypothesis:** Control in a general intelligence is maintained by a symbolic goal system.
3. **Uniform elementary-representation hypothesis:** There is a single elementary representation for declarative knowledge.
4. **Problem-space hypothesis:** Problem spaces are the fundamental organizational unit of all goal-directed behavior [49].
5. **Production-system hypothesis:** Production systems are the appropriate organization for encoding all long-term knowledge.
6. **Universal-subgoaling hypothesis:** Any decision can be an object of goal-oriented attention.
7. **Automatic-subgoaling hypothesis:** All goals arise dynamically in response to impasses and are generated automatically by the architecture.
8. **Control-knowledge hypothesis:** Any decision can be controlled by indefinite amounts of knowledge, both domain dependent and independent.
9. **Weak-method hypothesis:** The weak methods form the basic methods of intelligence [47].
10. **Weak-method emergence hypothesis:** The weak methods arise directly from the system responding based on its knowledge of the task.
11. **Uniform-learning hypothesis:** Goal-based chunking is the general learning mechanism.

Figure 2.3.: Eleven basic hypotheses about the structure of an architecture for general intelligence (from Laird, Newell, et al., 1987, p. 57). The first two are almost universally accepted among researchers of AI systems of any scope. On the contrary, the weak-method emergence hypothesis is unique to Soar.

Learning through chunking Chunking is a learning mechanism that learns continuously by automatically and permanently caching the results of problem solving to overcome impasses (see automatic subgoaling). The created production is added to the long-term memory and fires in similar situations, making the result of the previous problem solving directly available to overcome the impasse without the need for solving the same problem twice. This learning mechanism is related to the phenomenon called chunking in human cognition. Soar learns both operator implementations and search control by fine-grained task decomposition and the ability to abstract away all but the relevant features of problem solving.

The authors of Soar admit that there are most certainly more and other properties underlying human cognitive capabilities, and other ways to interpret the same behavior. However, as an architecture that reflects this particular view, Soar allows to easily implement this view. Therefore, after constructing a model within the architecture, it will be easy to describe the model's behavior as goal-oriented because the architecture supports that view directly (Lehman et al., 1998). Lehman et al. remind us that the architecture alone is not enough to elicit behavior because behavior is a function of both the architecture and content.

Taking all the previous points into consideration, it is of little surprise that the inventors or Soar call it a "problem-solving architecture", because Soar accomplishes all of its tasks in problem space. To implement any task as search in a problem space requires a set of "task-implementation functions" that realize the retrieval or generation of the problem space, operates for this problem-space, an initial state representing the start situation, and new states as a direct result of operators applied to existing states.

Besides the task-implementation functions, the architecture needs ‘search-control functions’, which allow the selection of a problem space, a state from available states, and an operator to apply to the state. Together, task-implementation and search-control functions are all that is needed to realize problem-space search. (Laird, Newell, et al., 1987) Soar takes most, if not all, of the eleven hypothesis it embodies to the limit: All tasks in Soar are represented in the problem-space, all learning is based on goal-based chunking[1], and all long-term memory is a production system (Figure 2.4 on the next page).

As mentioned earlier, Soar has not stopped there but instead has evolved into an architecture that now uses several memory modules, additional learning strategies and new non-symbolic representations of knowledge (Figure 2.5 on the following page). However, the core processing cycle is still driven by procedural knowledge implemented as production rules. Nevertheless, with the newest extension of Soar, Laird (2012) had to admit that it was necessary to “departure from some of the original hypotheses”. It holds no longer true, that rules are sufficient to represent all long-term knowledge, that a single learning mechanism is sufficient for all learning and that symbolic representations are sufficient for all short-term and long-term knowledge. Instead, Soar 9 supports multiple long-term memory systems (procedural, episodic, and semantic), multiple learning mechanisms (chunking, reinforcement learning, semantic, and episodic learning), and multiple representations of knowledge (symbolic, numeric, and imagery-based representations) (Laird, 2012). A more in depth description of the latest version of Soar along with examples of ow a cognitive architecture should be described and evaluated can be found in Laird’s book (Laird, 2012).

Because Soar 9 can be seen as a truly hybrid architecture with non-symbolic components, I will briefly summarize these new components to help the reader understand their implications and possible contributions (Figure 2.5 on the next page)).

Up until Soar 9, all operator preferences were symbolic, and no possibility existed to repent environmental reward. That changed with Soar 9 and the introduction of reinforcement learning for numeric preferences. Now it is possible to specify numeric preferences. For operator selection all numeric preferences are combined, and a Boltzmann distribution-based algorithm is used to select the next operator. After an operator applies, reinforcement learning lead to an update of all rules that created numeric preferences for that operator based on any new reward and the expected future reward. Such, reinforcement learning allows Soar agents to “improve their decision making over time as it receives feedback from the environment” (Laird, 2012).

Very similar to ACT-R, Soar 9 now supports encoding knowledge directly as semantic knowledge, which represents facts about the word. Knowledge in semantic memory cannot be accessed directly, though, but has to be retrieved via creating cues in working memory in a special buffer. Based on this cue and a so-called base-level activation, which favors results that are more recently and/or frequently accessed over less activated memory chunks, the best match in semantic memory is found and retrieved into working memory. With semantic memory, a Soar agent is now able to build up declarative knowledge over time.

Analogously to semantic memory, Soar 9 also introduces episodic memory, which is a storage for memories of what was experiences over time. Episodic memory is task-independent and thus available for every problem. It encodes instances of the structures that occur in working memory at the same time, so

[1] This is not true for the latest version of Soar, but we will come to this in a minute.

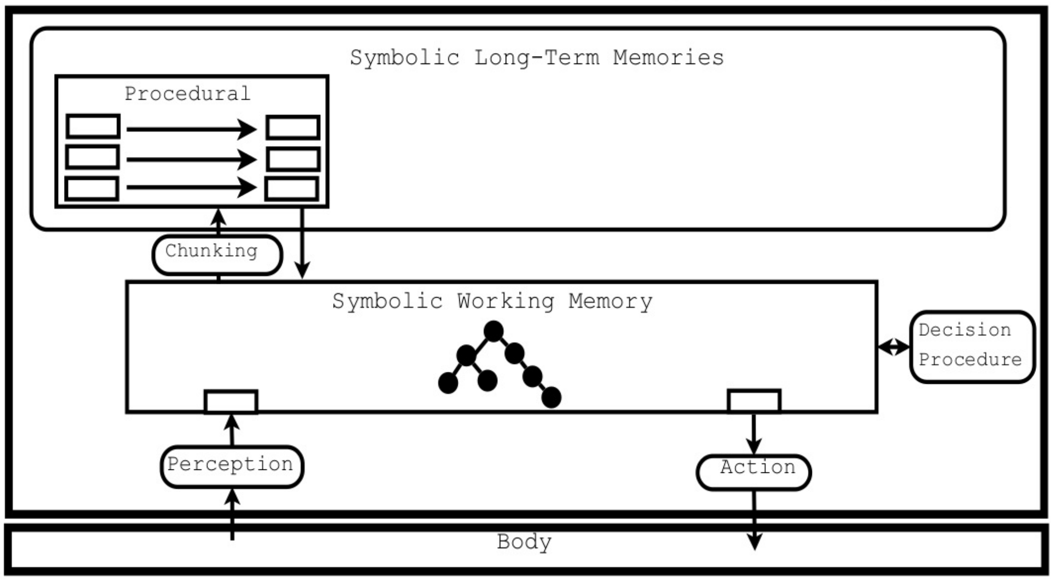


Figure 2.4.: Structure of the original classic Soar (from Laird, 2012, p. 1). Soar originally consisted of a single long-term memory, which is encoded as production rules, and a single short-term memory, which is encoded as a symbolic graph structure. The symbolic working memory holds the agent's knowledge of the current situation, derived from perception, and the knowledge retrieved from its long-term memory. The decision procedure selects appropriate operators based on the knowledge of the task, or detects impasses when knowledge is insufficient. Working memory can only be changed by the application of operators. Chunking is Soar's learning mechanism that converts the results of problem solving in subgoals into rules.

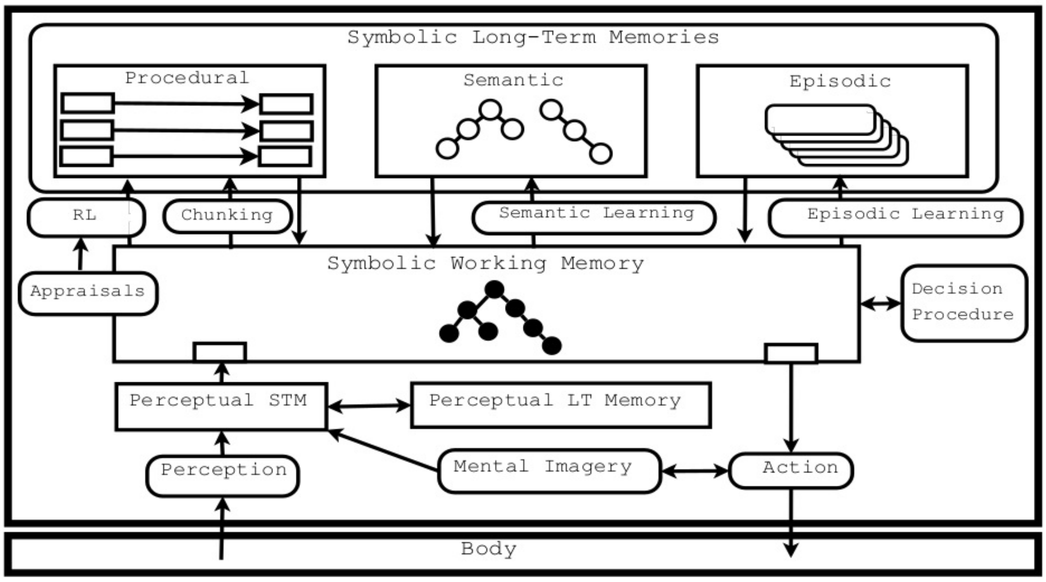


Figure 2.5.: Structure of Soar 9 (from Laird, 2012, p. 2). In its latest version, Soar was extended significantly. Next to the classical procedural long-term memory, Soar 9 supports semantic, episodic and perceptual long-term memory. Chunking as the traditional single learning mechanism is complemented by semantic learning, episodic learning and reinforcement learning. Finally, Soar 9 now incorporates emotions, appraisals and feelings to guide reinforcement learning.

that it is possible to remember the context of past experiences as well as temporal relationships. Like with semantic memory, episodic memory is not accessed directly, but again via creating cues, which are in this case partial specification of working memory. Based on this cue, the best match in episodic memory is found and the contents of the working memory of this episode is restored in a separate working memory buffer. Episodic memory gives a Soar agent "advanced cognitive capabilities such as virtual sensing, internal simulation, and prediction, learning action models, and retrospective reasoning and learning." (Laird, 2012)

All the previous mentioned extensions depend on the existing symbolic short-term working memory that encodes the agent's understanding of the current world situation. However, symbolic representations are not suited for all kind of data and processing tasks, like visual-spatial reasoning. The natural way of handle visual-spatial reasoning is by processing visual imagery, the very reason why Soar 9 was extended by a short-term memory especially for constructing and manipulating images, as well as a long-term memory that contains images that can be retrieved into the short-term memory. There is of course still a connection to the symbolic system, which controls visual imagery by issuing commands to construct, manipulate, and examine visual images. With visual imagery a Soar agent is now able to not only solve spatial reasoning problems "orders of magnitude faster", but also to process information not possible with only symbolic reasoning (Laird, 2012).

Finally, Soar 9 now includes a computational implementation of a specific appraisal theory to support emotions that arise due to an continuous evaluation of a situation. This evaluation is theorized to happen along multiple dimensions, such as goal relevance, goal conduciveness, causality, control, etc.. The appraisal detector in Soar 9 registers the outcome of these evaluations in form of emotions, which in return influence mood. Both mood and emotions determine feelings. Individual appraisals produce either categorical or numerical values for the current feeling and this intensity becomes the intrinsic reward for reinforcement learning. Thus, emotions are implemented as a form of intrinsic motivation via rewards or penalties for reinforcement learning. (Laird, 2012)

Lehman et al. (1998, pp. 31–32) have nicely summarized the principle work flow with Soar:

> "First, we specify the domain knowledge Joe needs; that's our content. Next, we tie the different types of domain knowledge to the different parts of the goal context: goals, problem spaces, state structures (including percepts and any other working memory elements that trigger actions), and operators. Finally, we specify the relationships between problem spaces by the impasses that will arise and the kinds of knowledge that will be missing, and consequently learned".

A synopsis of the mechanisms and structures of the classical Soar architecture, previous to Soar 9, entails the goal context as central organizing element, the working memory and long-term memory as the two main knowledge representations, the perception/motor interface to interact with the external world, the decision cycle with a single selected operator that leads from the current state to a new state, the impasses as signals of lack of knowledge, which provide an opportunity for learning, and four learning mechanisms: chunking, reinforcement learning, episodic learning, and semantic learning. (Lehman et al., 2006).

Before we leave this section and the introduction to Soar, I want to give the reader some more information about the internals of Soar to foster an intuition about how Soar works.

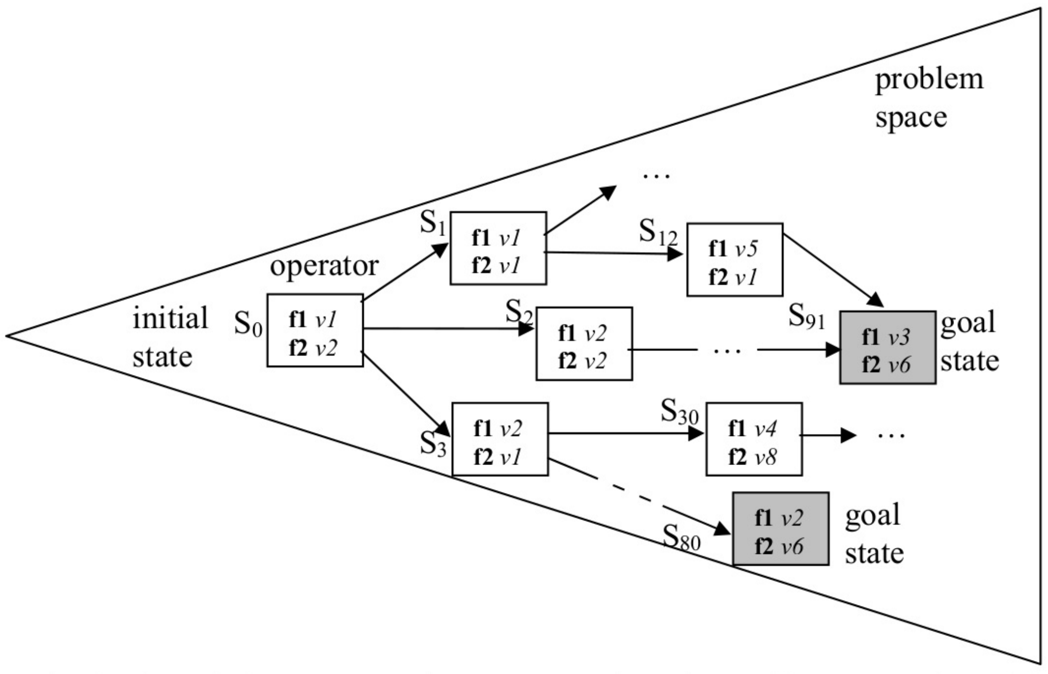


Figure 2.6.: Problem solving represented as movement through a problem space (from Lehman et al., 2006, p. 10). The problem space is widening over time to represent the "ever-expanding set of possibilities" that unfold over time. Squares represent states. Goal states are shaded and indicate the state the system tries to achieve. Each state is characterized by features (**bold** face) and their corresponding values (*italic* face). Arrows represent operators that are the only way to transform a current state into a new state. All variables are indicated with arbitrary symbols.

First, we have a closer look at the problem space (Figure 2.6). The problem space is made of states (squares) which represent possible isolated instances of the problem in the real world. A state is described by attributes or features (in bold face) and their possible values (in italics). The values are not limited to atomic values but instead can be themselves a set of features and values, allowing the representation of richly interconnected sets of objects. We will come to this later. A state is a representation of all the aspects of a situation, internal and external, that the agent may need to consider choosing its next action. Each model of a particular behavior must include a description of the initial situation, which translates to an initial state (S_0) in the problem space. Likewise, there has to be a description of desired goal states (shaded squares) with features and values that indicate that the goal has been achieved. Given the initial and goal states, an agent tries to find a path through the problem space that allows him to travel from the initial state to a goal state via operators (arrows). As was said earlier, at every point in time, there is only a single state designated as the current state which represents the current situation of the agent and the world. Movement from the current state to a new state is only possible through the application of an operator to the current state. The application of an operator to the current state transforms the state by changing some of its features and values. To avoid a random, unguided movement through problem space, the agent has to be guided by the principle of rationality: if the agent has knowledge about the preferences of operators in a given situation it will select the operator from which it knows that it will lead to one of its desired goals. (Lehman et al., 2006)

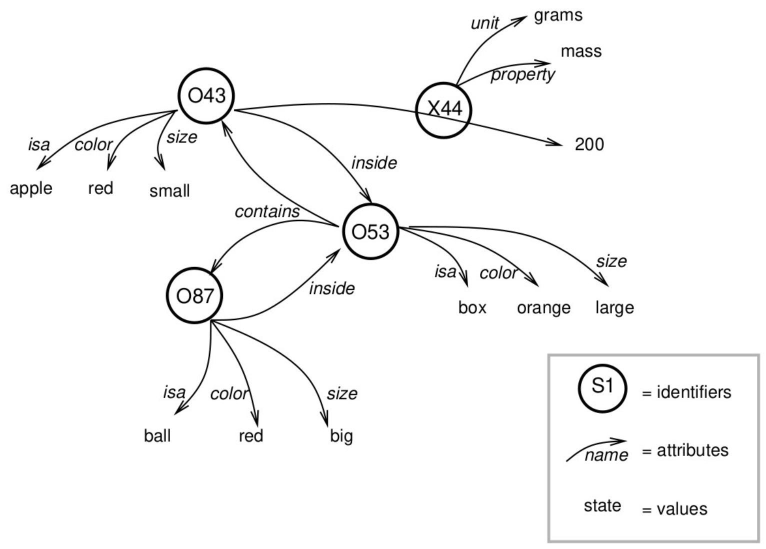


Figure 2.7.: Working memory as a semantic net (from Laird, Congdon, et al., 2017, p. 47). Each working memory element is defined by at least one identifier-attribute-value triple.

All knowledge about the current situation is held in Soars working memory (Figure 2.7). This comprises the current state with all known objects and elements as well as operators. The working memory is organized in working memory elements. Each WME represents exactly one piece of information about an object, like `<B1> <is a> <block>`, or `<B1> <is named> <A>`. All WMEs are represented by an identifier-attribute-value triple. The identifier tells Soar to which internal object the WME belongs to. Several WMEs can specify information about the same object by sharing the same identifier. Each WME describes a different attribute of the object, for example, its name or type or location. In return, each attribute has a value associated with it, which can be atomic or another WME triple. In the latter case, objects are linked to other objects and relations between objects are formulated, such as `<T1> <is ontop of> <F1>`. The working memory in Soar is implemented as a set, which means that there can never be two WMEs at the same time that have the same identifier-attribute-value triple. However, multi-valued attributes are supported. How are WMEs created? Most WMEs are created through the actions of productions (as specified by the production rule's right hand side) and through the environment by the input-link for sensory data. Working memory is not only a set, but it is also a graph structure where identifiers are nodes, attributes are links, and constants are terminal nodes (Figure 2.7). The objects in working memory illustrated by Figure 2.7 are given by the following Soar syntax

```
(O43 ^isa apple ^color red ^inside O53 ^size small ^X44 200)
(O87 ^isa ball ^color red ^inside O53 ^size big)
(O53 ^isa box ^size large ^color orange ^contains O43 O87)
(X44 ^unit grams ^property mass)
```

The second important memory type in Soar is the long-term memory that holds procedural knowledge in the form of productions. Procedural knowledge specifies how to respond to different situations in working memory. Because Soar is a rule-based system, productions are formulated as rules. A rule has a set of conditions and a set of actions. If the conditions of a production rule are satisfied by WMEs of the current working memory, the production fires, and the actions are performed. The first part of a production rule is often called the left hand side, whereas the second part of a production rule is referred to as right hand side. In its simplest form, a production can test for the presence or absence of objects in working memory. Of course, there are much more sophisticated possibilities for formulating production rules, which will work with variables instead of hardcoded attribute values to match a wider range of situations. The procedural knowledge found in Soar can be categorized into four distinct types of knowledge: inference rules, operator proposal knowledge, operator selection knowledge, and operator application rules. (Laird, Congdon, et al., 2017)

As we have seen, Soar is based on the hypothesis that all deliberate behavior can be described as goal-oriented. This is implemented in Soar as the selection and application of operators to a state. A state represents the current situation; an operator specifies how a state can be transformed; and a goal is a description of a desired outcome of the problem-solving activity. At a very abstract level, the Soar program can be seen as a sequence of operator selections and operator applications to the current state, until a goal state has been reached. On a finer-grained scale, the Soar program proceed through a number of decision cycles, each consisting of five phases (Figure 2.8 on the following page):

Input	Sensory data is processed, and the results come into the working memory.
Proposal	All matching productions fire in parallel to elaborate the current state, propose operators for the current situation, and compare proposed operators.
Decision	Either a new operator is selected, or an impasse is detected which results in the creation of a new state.
Application	Only the productions that describe operator application fire in parallel to apply the selected operator.
Output	If applicable, output commands are sent to the external environment.

The proposal and application phase are not monolithic but entail as many elaboration cycles as necessary to reach quiescence, which is the case when the set of matching rules remains unchanged. In each elaboration cycle, working memory is changed by the actions of all matching productions. If the actions of the fired productions led to changes in working memory that now satisfy the conditions of further productions, a new elaboration cycle will begin. Interestingly, the limitation to a single operator application per decision cycle imposes a cognitive bottleneck in the architecture (Lehman et al., 1998). With this information it is possible to state a simplified version of the Soar algorithm (Listing 2.1 on page 28, from Laird, Congdon, et al. (2017, p. 27)).

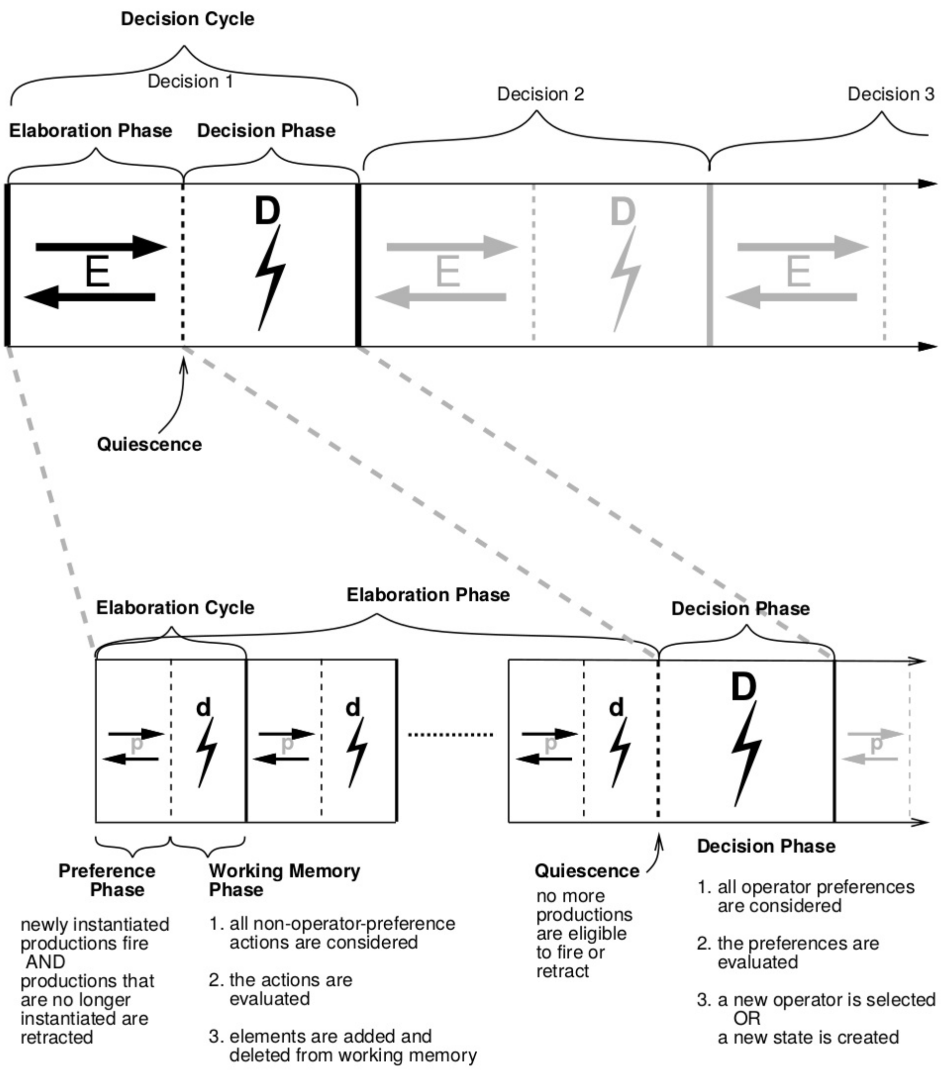

Decision Cycle
Decision 1
Decision 2
Decision 3
Elaboration Phase
Decision Phase
E
D
Quiescence
Elaboration Cycle
Elaboration Phase
Decision Phase
p
d
D
Preference Phase
newly instantiated productions fire AND productions that are no longer instantiated are retracted
Working Memory Phase
1. all non-operator-preference actions are considered
2. the actions are evaluated
3. elements are added and deleted from working memory
Quiescence
no more productions are eligible to fire or retract
Decision Phase
1. all operator preferences are considered
2. the preferences are evaluated
3. a new operator is selected OR a new state is created

Figure 2.8.: Soar's decision cycle in detail (from Laird, Congdon, et al., 2017, p. 26). Each decision cycle consists of the two phases elaboration and decision. In the elaboration phase matching production rules fire in parallel until *quiesence* is reached. In the decision phase all operator preferences are evaluated and the best operator is selected. If a decision cannot be made due to a lack of knowledge, an impasse is created.

Listing 2.1: A simplified version of the Soar algorithm.

```
Soar
   while (HALT not true) Cycle;

Cycle
   InputPhase;
   ProposalPhase;
   DecisionPhase;
   ApplicationPhase;
   OutputPhase;

ProposalPhase
   while (some i-supported productions are waiting to fire or retract)
      FireNewlyMatchedProductions;
      RetractNewlyUnmatchedProductions;

DecisionPhase
   for (each state in the stack,
       starting with the top-level state)
   until (a new decision is reached)
      EvaluateOperatorPreferences; /* for the state being considered */
      if (one operator preferred after preference evaluation)
         SelectNewOperator;
      else      /* could be no operator available or */
         CreateNewSubstate;  /* unable to decide between more than one */

ApplicationPhase
   while (some productions are waiting to fire or retract)
      FireNewlyMatchedProductions;
      RetractNewlyUnmatchedProductions;
```

2.1.2 Handle uncertainty – Bayesian modeling

As we have established in section 2.1 How to Model User Behavior – Paradigms in Cognitive Science (page 12), cognitive scientists view the brain as an information processor. Central to this view is the idea that new information is inferred from information that has been derived from the senses, like linguistic or visual input, or from memory. But the process of inference from old to new is typically uncertain with multiple sources of uncertainty: we have to estimate the quality of our sensory input, the importance of different sensory sources, the trustworthiness of different knowledge sources, the relevance of prior experiences and so on. Most of cognition and learning requires uncertain conjecture from partial or noisy information (Griffiths et al., 2010). Probability theory is, in essence, a calculus for inference conducted under uncertainty (Chater et al., 2006). To understand the Bayesian point of view it is important to understand the subjective interpretation of probability.

The 'frequentist' interpretation of probability handles probabilities as limiting relative frequencies of repeated identical 'experiments', such as coin flips or dice rolls. In comparison, in cognitive science, probabilities refer to 'degrees of belief' (Chater et al., 2006). Beliefs are peoples' prior assumptions about the range of possible values an observation can have along with a notion about how plausible the different values are within this range. A person's weak belief can be represented with a uniform distribution of values, meaning that, for this person, any value within a certain range is equally credible. Another person might have a much stronger belief about the same quantity, leading to only a few credible values from the same range. Thus, two people seeing the same event, but having different belief states, will have different subjective probabilities. In addition, the particular pattern of prior information and evidence will never be repeated, so we cannot define the probability of this event as a limiting frequency (Chater et al., 2006).

Probabilistic analyses of perceptual, linguistic, learning or motor tasks can be understood as the process to understand what is believed, and what can be inferred. Subjective interpretation of probability generally involves evaluating conditional probabilities. We are interested in the state of the world, what we call a hypothesis, given certain observations. A typical every day task could be to determine from our sensory input whether we know the person in front of us or not. Two alternative hypotheses would describe the fact that we either know the person or not. Or we hear a human voice and want to infer from which person this might have come from. In this case our set of hypotheses would describe a set of possible persons we might expect to have uttered this sound. Hypotheses are not restricted to categories, though. In another task, the alternative hypotheses might be the speed of an object that we see in the dark and of which we are unsure. In general, hypotheses can take any form, from categorical values over numeric values to structured symbolic representation, as long as they specify a probabilistic distribution over observations (Griffiths et al., 2010).

Denoting the set of possible hypotheses with $\mathscr{H}$ that might explain observed data $\mathscr{D}$ and assigning each hypothesis h of $\mathscr{H}$ a prior probability $p(h)$ before observing $\mathscr{D}$, then Bayes' rule indicates that the conditioned probability $p(h|\mathscr{D})$ assigned to h after having observed $\mathscr{D}$ should be:

$$\underbrace{p(h|\mathscr{D})}_{=:\text{ Posterior}} = \frac{\overbrace{p(\mathscr{D}|h)}^{=:\text{ Likelihood}} \cdot \overbrace{p(h)}^{=:\text{ Prior}}}{p(\mathscr{D}) = \sum_{i=1}^{H} p(\mathscr{D}|h_i) \cdot p(h_i)} \tag{2.1}$$

This allows us to derive the probability in question, also known as *posterior* probability, as a product of a) the probability of the data if the hypothesis were true, called the *likelihood*, and b) the degree of belief in the hypothesis prior to the observations, called the *prior* probability. To obtain a proper probability distribution the quotient is normalized by the probability of the data. Probability theory tells us how a learner should revise their degrees of belief in a set of hypotheses in light of the information provided by observed data.

When referring to a Bayesian approach, what is meant by this is the interpretation of probabilities as subjective probabilities within the Bayesian theorem. When compared to cognitive architectures, it is important to understand that Bayesian methods are a way to think about the nature of inferences and not a framework that specifies computation like symbolic rule-based processing vs. connectionist networks (Chater et al., 2006). Bayesian methods can be applied on all three of Marr's levels of analyses. Thus, Bayesian methods can very well be implemented within a cognitive architecture. Furthermore, probabilistic models can be defined over a broad range of candidate representations like causal graphs, phrase structure grammars, logical rules or theories (Griffiths et al., 2010).

The probabilistic framework allows us to address key questions about cognition (Griffiths et al., 2010): How much information is needed? What representations subserve the inferences people make? What constraints on learning are necessary? The probabilistic approach to modeling cognitive behavior begins by identifying ideal solutions to inductive problems and the representations that might be involved. Representations and inductive biases are selected according to what is thought necessary to perform the same operations as the brain, assuming only that those operations can be described as forms of probabilistic inference. (Griffiths et al., 2010)

To build a probabilistic model we have to start with a formal characterization of an inductive problem. This includes the following steps:

1. To specify the hypotheses under consideration,
2. to specify the relation between these hypotheses and observable data, and
3. to specify the prior probability of each hypothesis.

One of the core strengths of the probabilistic framework is the transparent account of the assumptions that allow a problem to be solved. In addition, the framework makes it easy to alter these assumptions and explore the consequences of different assumptions.

The Bayesian approach utilizes not only probabilistic models but probabilistic generative models, which are described by Tenenbaum et al. (2011, p. 1280) as "a kind of mental model that describes the causal processes in the world giving rise to the learner's observations as well as unobserved or latent variables that support effective prediction and action if the learner can infer their hidden state." The use of probabilistic generative models instead of deterministic models allows for handling the learner's uncertainty about the true states of latent variables and the true causal processes at work. Tenenbaum et al. note that generative models are abstract in two senses: they are not only a description of a specific situation but also of a broader class of situations, and they capture the "essential world structure" that is responsible for the learner's observations and allows generalization.

Hierarchical Bayesian modeling

Whenever the Bayesian model should account for data from multiple participants, we have a design choice to made. On the one side we could fit the model to individual participants, avoiding any aggregation of the data. At the other end we could fit the model to aggregated data, which introduces stability by averaging data but also the risk for artifacts. But simple nonhierarchical models are usually inappropriate for hierarchical data, they either cannot fit large datasets or tend to 'overfit' (Gelman et al., 2014, p. 101). A middle way is provided by hierarchical Bayesian model (HBM). HBMs take into account the data from all participants simultaneously, but, unlike fits of individual participants, postulate some degree of dependence between participants. (Farrell and Lewandowsky, 2018)

In HBMs individual variation is considered to be governed by an orderly distribution (Figure 2.9 on the following page). This distribution across individuals is called the *parent distribution* and can be interpreted as a *population distribution* from which the individual parameter values are sampled (Gelman et al., 2014). The parent distribution characterizes the distribution of parameters that determine the priors for each individual. For that reason, the parent distribution is also sometimes known as a "hyperprior distribution" (Gelman et al., 2014, p. 107) because it determines the shape of the priors for each individual. Therefore, a HBM is always also a theory of individual differences. (Farrell and Lewandowsky, 2018)

In fact, any cognitive model can be instantiated as a hierarchical model. During the process of fitting the model to the data, individual parameters for each subject and parameters of the parent distribution are estimated simultaneously. Hierarchical models are so powerful because they try to explain individual data and the group the participants belong to as well as possible (Farrell and Lewandowsky, 2018).

One way to represent Bayesian models in general, and especially HBMs, is with the help of graphical models. There is no agreed standard notation for representing graphical models, but the notation introduced by Lee and Wagenmakers (2014) should be regarded as such. In graphical models, all variables are represented as *nodes* and their dependencies are indicated by arrows (see Table 2.2 on the next page). Plates are used to indicate replications, for example for multiple trials. A graphical model depicts the observed variables as well as the model components, that is, parameters or predictions of the model. The model components are referred to as latent or non-observed variables and are further divided into stochastic and deterministic variables. Continuous variables are denoted with circular nodes, whereas discrete variables are square nodes. Observed variables, that is, the data, are shaded in gray, irrespective of the shape of the node, whereas unobserved variables are left unfilled. Latent variables that are deterministic, such as the predictions of the model that is computed from its parameters, are represented by nodes with a double border.

As a short introduction into graphical notation and Bayesian models, I will describe the "Exam score" example presented in Lee and Wagenmakers (2014, p. 79) (graphical model given in Figure 2.10 on page 33). In this example we have observations of 15 people answering each 40 true-or-false questions. The observed data suggest that there are at lest two groups of people, the ones who just guessing and the ones who had some level of knowledge. We first assume that the number of correct answers k_i for each subject can be described by a binomial distribution that has two parameters, the success probability θ_i and the number of total questions n. The last is known and fixed for all subjects. The first is a latent variable

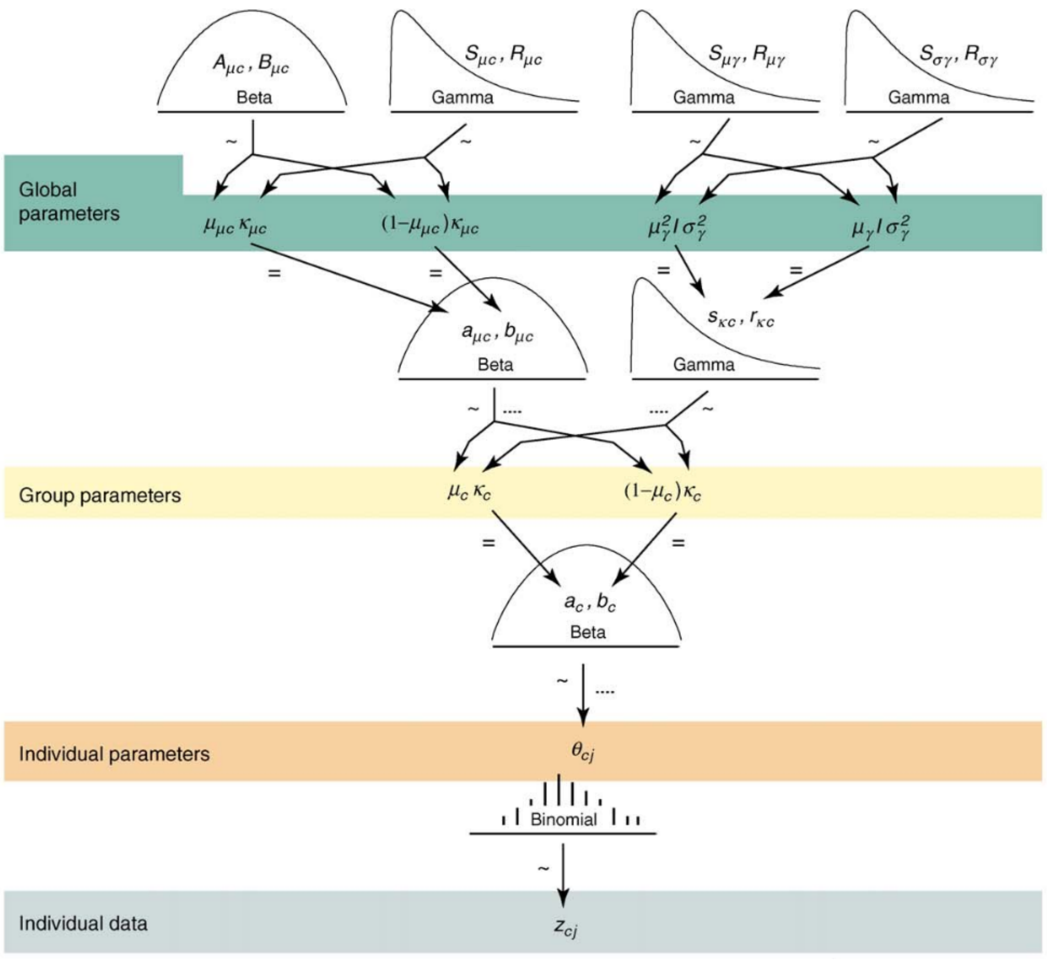
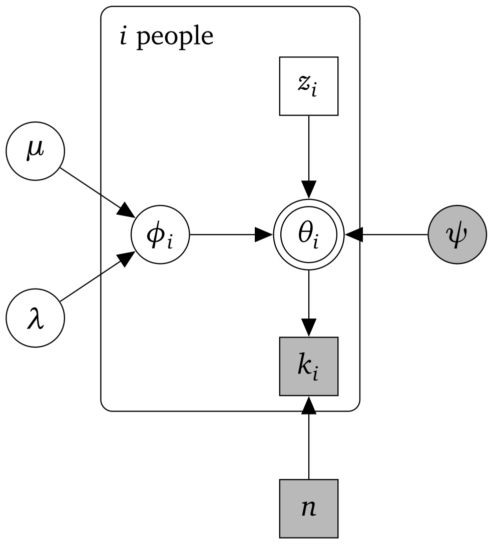


Figure 2.9.: Exemplary hierarchical Bayesian model for an experiment about category learning (from Kruschke, 2010b, p. 296). Without going into detail here, the model distinguished between individual, group level and global parameters. Global parameters connect different groups so that data from one group can influence estimates of other groups. The model contains 12 group and global parameters, and 240 individual parameters.

Table 2.2.: Notation for nodes used in graphical models (based on Farrell and Lewandowsky, 2018, p. 205).

	Type of Variable	
Status of Variable	Discrete	Continuous
Observed	■	●
Unobserved		
Stochastic	□	○
Deterministic	⧈	◎



Priors

$z_i \sim \text{Bernoulli}(0.5)$
$\mu \sim \text{Uniform}(0.5, 1)$
$\lambda \sim \text{Gamma}(0.001, 0.001)$

Intermediate variables

$\phi_i \sim \text{Normal}(\mu, \lambda)_{I(0,1)}$

Deterministic variables

$\psi \leftarrow 0.5$

$$\theta_i \leftarrow \begin{cases} \phi_i \text{if } z_i = 1 \\ \psi \text{if } z_i = 0 \end{cases}$$

Observable variables

$k_i \sim \text{Binomial}(\theta_i, n)$

Figure 2.10.: Graphical model that implements the exam scores example (Lee and Wagenmakers, 2014, p. 79). The notation for (hierarchical) Bayesian models was introduced in Table 2.2 on the preceding page.

we wish to infer from the data. However, because we assume at least two groups, θ_i is a deterministic variable because once we knew the group, we also know θ_i. If the subject belongs to the group of guessers, we assign a success probability of $\psi = 0.5$, so this value is fixed. If, on the other side, the subject belongs to the group that has learned, we assume that the success probability is at least as good as guessing, that is, something between 0.5 and 1. We model the success probability ϕ_i of the second group as a normally distributed variable with mean value μ and precision λ. We do not assume anything about the mean value and choose a uniform prior over the range of $[0.5, 1]$. The precision is also set to a common non-informative prior. By choosing a parent distribution for the parameters of the success rate of the knowledge group, we have created a hierarchical model. Each subject is modeled individually, that is the model infers for each subject an individual success rate θ_i, but all subjects in the knowledge group are connected via the parent distribution and share the same parameters μ and λ. The Gaussian distribution is a "convenient (but not perfect) choice for this 'individual differences' distribution (...) at least in the absence of any richer theory" (Lee and Wagenmakers, 2014, p. 79).

A noticeable phenomenon of HBMs is a shift in the hierarchical estimates towards the overall average. This type of behavior is common for hierarchical models, and is known as 'shrinkage towards the mean' Lambert (2018, p. 437). This is an intended by-product of using HBMs, because it takes probability mass away from the outlier estimates. The most extreme parameter estimates are shifted the most. HBMs take probability mass away from outlier estimates, and reallocate it towards those points with higher certainty. This behavior makes HBMs more robust than heterogeneous models because they are less susceptible to overfitting. Another benefit of HBM is a greater sample size than for heterogeneous models, because they partially pool data across groups. Due to the increased sample size HBMs achieve higher precision for group-level estimates. In general, the benefits of HBM grow as the number of groups increases, and when the data are sparser for each group. This approach works best when there are more than 10 groups, otherwise there is not enough data to reach an overall consensus Lambert (2018). For an in-depth introduction to HBMs I refer the reader to Gelman et al. (2014) and Kruschke (2015).

Bayesian data analysis

In the last section we have discussed the basics and merits of probabilistic generative models. These explanatory models make semantic ascriptions to the functional from or the parameters of the model, that is, they postulate a mechanistic meaning. Merely descriptive models, on the other side, summarize the relations between variables without ascribing mechanistic meaning to the model. The main idea is that Bayesian inference should guide data analysis regardless of whether Bayesian explanatory models account for cognition.

Bayesian data analysis is favorable over traditional data analysis as the latter has many "well-documented problems" like the concept of 'statistically significant', point estimates of parameter values and the meaning of confidence intervals, all of which are ill defined because they are based on p values (Kruschke, 2010b, p. 293). Bayesian analysis involves no *p*-values and inferences provide rich and complete information summarized in the posterior distribution, which is the main output of any Bayesian data analysis.

Kruschke emphasizes that Bayesian methods for data analysis are distinct from Bayesian models of mind. In the former, any useful descriptive model of the data has parameters estimated by normative, rational methods. But the descriptive models have no "necessary relation or commitment to particular theories of the natural mechanisms that actually generated the data" (Kruschke, 2010b, p. 293). Kruschke and Liddell differentiate between three applications of Bayesian methods in psychology and other sciences. All three applications use a parameterized model that uses Bayesian inference to reallocate prior beliefs. What differs between the applications is the "semantic referent of the model and its parameters" (Kruschke and Liddell, 2018a, p. 171). The first application is generic data analysis in which the model describes trends in the data, without any necessary reference to a process that generated the data. The second application is psychometric models in which the data is known to have been produced by a mind. The third application is the Bayesian model of a mind in which the mind itself is conceived as a "Bayesian statistician, taking data from the world and updating its internal state by using Bayesian inference."

In Bayesian data analysis, the descriptive model is easily customizable to the specific situation – by specifying which of the variables are observed and which are latent and thus to be inferred by the model or which information in form of prior knowledge restrains the inference process – without the computational restrictions of the traditional null hypothesis significance testing. The descriptive model is useful for summarizing the data and could be either a generic domain-independent model such as linear regression or a domain-specific model. Like with probabilistic generative models, the analysis starts with the specification of the current uncertainty of parameter values, summarized by prior distributions, that is acceptable to a "skeptical scientific audience" (Kruschke, 2010b, p. 295). Bayesian inference then yields a complete posterior distribution by reallocating beliefs according to the data. Belief is shifted from parameter values that are less consistent with the data to parameter values that are more consistent with the data (Kruschke, 2010a). The posterior summarizes the relative credibility of every possible combination of parameter values and holds the complete information about correlations of credible parameter values (Kruschke, 2010b). It is straightforward to computer power and replication probabilities from the gained posterior probability distributions, as well as credible intervals for the parameter estimates. According to Kruschke, p. 297, the posterior distribution over the parameter values is our “best representation of the world” based on the information we have observed and that is available.

In summary, the essence of Bayesian inference is reallocation of credibility across possibilities. The distribution of credibility initially reflects our prior knowledge about the world, which can be quite vague or very specific. Then new data are observed, and the credibility is re-allocated. Bayesian analysis is the mathematics of re-allocating credibility in a logically coherent and precise way.

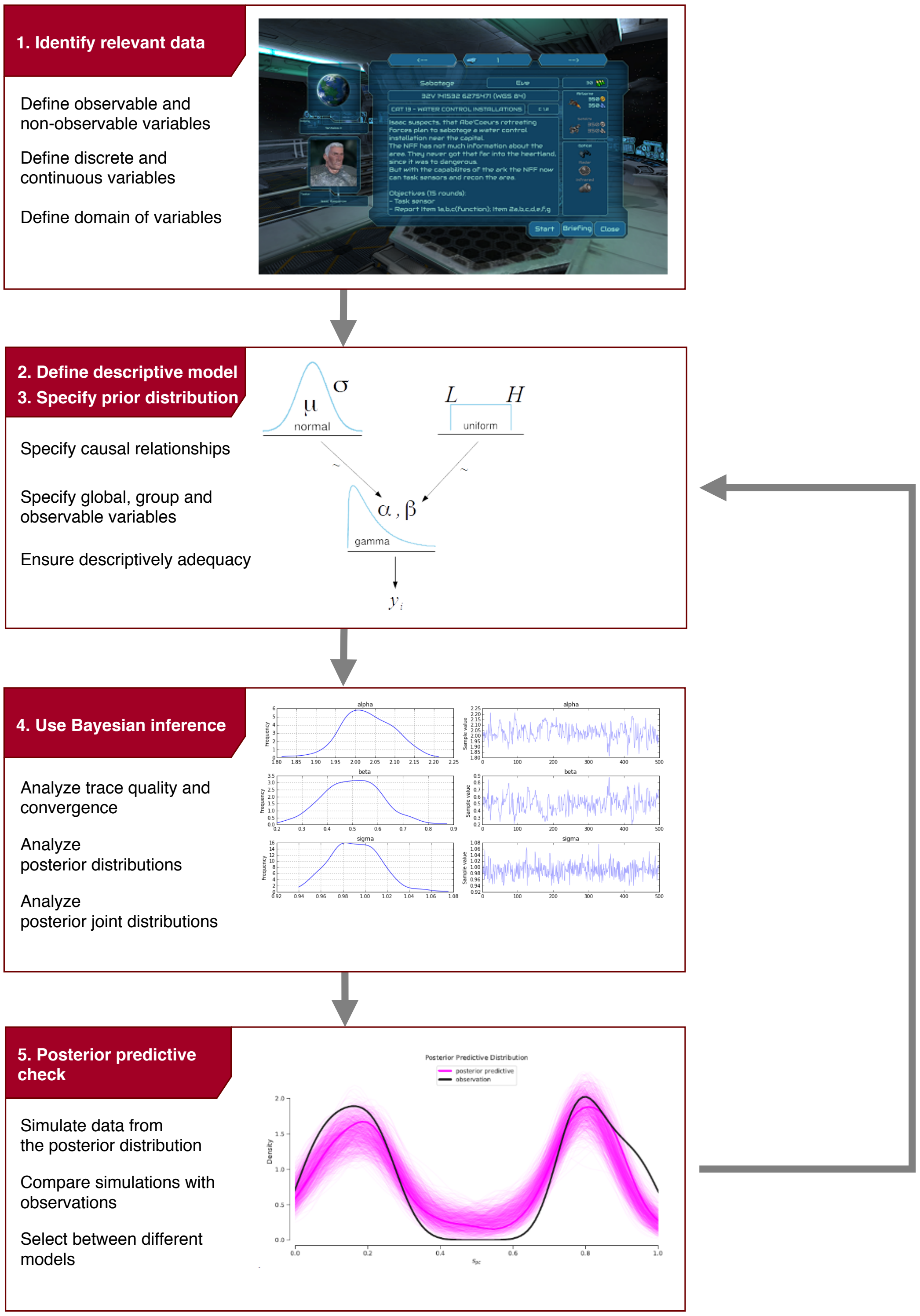
Although it was said that the descriptive models are no explanatory models, there are of course desiderata for a mathematical description of the data (Kruschke, 2015). First, the model should be specified by comprehensible distributions with meaningful parameters. Second, the mathematical description should be “descriptively adequate”, which means that there are no important systematic deviations between the trends in the data and the form of the model. However, what is an apparent discrepancy and what is not might change during the scientific progress of the analysis.

To better understand the process of Bayesian data analysis, I will repeat the 5 steps of Bayesian data analysis presented in Kruschke (2015) (Figure 2.11 on the next page):

1. Identify the data relevant to the research question. This involves the measurement scales of the data, and the definition of variables that are to be predicted and variables that are predictors.
2. Define a descriptive model for the relevant data. The mathematical form and its parameters should be meaningful. The model should be appropriate to the theoretical purposes of the analysis.
3. Specify a prior distribution of the parameters. The prior should capture the assumptions of the analyst and be agreeable to other skeptical scientists.
4. Use Bayesian inference to re-allocate credibility across parameter values. Interpret the posterior distribution with respect to the research questions.
5. Check that the posterior predictions mimic the data with reasonable accuracy, which is called “posterior predictive check”. If not, consider a different descriptive model and repeat the analysis.

Gelman et al. (2014) summarizes the process of Bayesian data analysis somewhat shorter in three steps: First, setting up a fully probability model, that is, a joint distribution for all observable and non-observable quantities in a problem. Secondly, calculating and interpreting the appropriate posterior distribution, that is, the conditional probability distribution of the unobserved quantities, given the observed data. Thirdly, evaluating the fit of the model and the implications of the resulting posterior distribution: how well does the model fit the data, are the conclusions reasonable, and how sensitive are the results to the modeling assumptions of step one? The approach presented in the sections 4.3 and 5.1 on page 80 and on page 103 is closely aligned with the steps presented here and I will describe in detail the assumptions made as well as the output of each step.

What is the posterior predictive check? With a posterior predictive check, one wants to know if the model mimics the data reasonably well. There is no single, unique way to ascertain whether the model predictions systematically and meaningfully deviate from the data (Kruschke, 2015). One approach, that I will follow in this work, is to plot a summary of predicted data from the model against the actual data. If a systematical deviation between the actual data and the predicted form is found, then alternative descriptive models might be elaborated. A descriptive model can be modified in many different ways, including reparametrization and hierarchical extensions. We will come to this later.



1. Identify relevant data
Define observable and non-observable variables
Define discrete and continuous variables
Define domain of variables
2. Define descriptive model
3. Specify prior distribution
Specify causal relationships
Specify global, group and observable variables
Ensure descriptively adequacy
μ
σ
normal
L
H
uniform
~
α , β
gamma
y_i
4. Use Bayesian inference
Analyze trace quality and convergence
Analyze posterior distributions
Analyze posterior joint distributions
alpha
beta
sigma
Frequency
Sample value
5. Posterior predictive check
Simulate data from the posterior distribution
Compare simulations with observations
Select between different models
Posterior Predictive Distribution
posterior predictive
observation
Density
Start
Briefing
Close

Sabotage

Eve

32V 141532 6275471 (WGS 84)

CAT 13 - WATER CONTROL INSTALLATIONS

Isaac suspects, that Abe'Coeurs retreating forces plan to sabotage a water control installation near the capital.
The NFF has not much information about the area. They never got that far into the heartland, since it was to dangerous.
But with the capabilites of the ark the NFF now can task sensors and recon the area.

Objectives (15 rounds):
- Task sensor
- Report Item 1a,b,c(function); Item 2a,b,c,d,e,f,g

Figure 2.11.: The five steps of Bayesian data analysis according to Kruschke (2015) with exemplary output. The output of the first step are latent and observable variables. Latent variables are not directly observable and should be inferred from the observable variables. In the second and third step a descriptive probabilistic Bayesian model is built that represents the causal relationships between the latent and observable variables. Bayesian inference in the fourth step allows for computing the posterior distribution over the model parameters. Finally, the model is validated by comparing the model's predictions with the original observations.

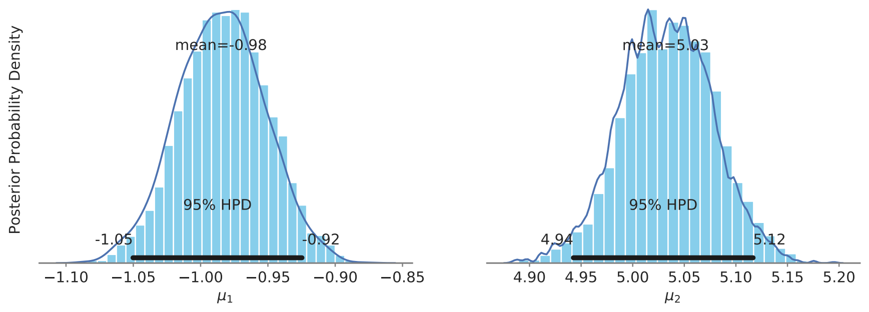


Figure 2.12.: A posterior distribution gained by Bayesian inference from a probabilistic model. In this example, the mean values μ_1, μ_2 of a two-dimensional multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with unknown mean vector and known co-variance matrix were inferred. Each posterior is represented by the frequency of the values in form of a histogram and a kernel density estimation of this histogram. HPD refers to the HDI, which is described in the text.

The main output of the Bayesian inference is a posterior distribution for all latent model parameters (Figure 2.11 on the preceding page). The posterior distribution assigns each parameter value a credibility value that is a combination of its prior probability and its likelihood. In Bayesian data analysis a posterior distribution is often summarized by the so-called highest density interval. The HDI is not particular to the posterior distribution but can be given for any density. The HDI indicates which points of a distribution are most credible, and which cover most of the distribution. It specifies an interval such that every point inside this interval has higher credibility than any point outside the interval. The exact value of this credibility can be specified arbitrarily. Formally, the values of x in the 95 % HDI are those such that $p(x) > W$ where W satisfies

$$\int_{x:p(x)>W} p(x)dx = 0.95.$$

W is some value for which this assumption holds true (Kruschke, 2015). The width of the HDI is a direct measure of uncertainty of belief. If the HDI is wide, then beliefs are uncertain, as the possible values of the latent parameters span over a huge range. If the HDI is narrow, then beliefs are much more certain, as the range of possible values of the latent parameters are limited to a small range. For further details along with a discussion about the influence of different choices for the prior and the influence of different sample sizes I refer the reader to (Kruschke, 2015).

Markov Chain Monte Carlo

So far, we have seen that the posterior distribution is computed from the prior distribution and the likelihood function of the data. But nothing was said about how this computation is executed exactly. Bayes' theorem describes the way in which the prior distribution and the likelihood function are combined to obtain the posterior distribution, but the computation itself involves the computation of the product of two probability densities and the computation of the marginal probability of the data in the nominator. Often, these calculations are analytically intractable for realistic and more complex applications. The main idea of a class of methods called Markov chain Monte Carlo is to approximate the posterior distribution in the form of a large number of values sampled from that distribution.

In general, all simulation (stochastic) methods are based on drawing random samples θ^s from an otherwise intractable target distribution $p(\theta)$ and estimating the expectation of any function $f(\theta)$ (Gelman et al., 2014),

$$E[f(\theta)|y] = \int f(\theta) \cdot p(\theta|y) d\theta \approx \frac{1}{S} \sum_{s=1}^{S} f(\theta^s). \tag{2.2}$$

MCMC assumes that the posterior distribution is specified by a function that is easily evaluated. In addition, the value of the likelihood function must be computable for any specified values of $\mathcal{D}$ and θ (Kruschke, 2015). The samples gained from the posterior distribution by MCMC can be used to estimate the central tendency of the posterior, its HDI, etc. When the preconditions are met, sample values from the target distribution are generated by following a random walk through parameter space. The walk starts at some arbitrary point (and each chain has its own starting point) where the target distribution is non-zero. At each time step the random walk proposes a move to a new position in parameter space and has to decide whether or not to accept the proposed move. The proposal distributions can take on many different forms with the goal to efficiently explore the regions of the parameter space where the target distribution has most of its mass. The decision is based on computing the ratio $p_{move} = {}^{P(\theta_{proposed})}/_{P(\theta_{current})}$ (Kruschke, 2015).

The first part of the name MCMC stems from the fact, that any simulation that samples random values from a distribution is called a Monte Carlo simulation, named after the famous casino locale. In addition, each step in the generated random walks is completely independent of the steps before the current position. Any such process in which the probability of the next step is dependent only on the current one and not on all previous steps is called a (first-order) Markov process, and a succession of such steps is a Makov chain (Kruschke, 2015).

Given sufficient time, the history of the Markov chain, $\{q_0, \ldots, q_N\}$, denoted samples by the Makov chain, becomes a quantification of the target distribution. We can estimate expectations across the entire parameter space by averaging the target function over this history (Betancourt, 2017),

$$\hat{f}_N = \frac{1}{N} \sum_{n=0}^{N} f(q_n). \tag{2.3}$$

With more and more samples the Marko chain will explore the "typical set" of the target distribution better and better, and will converge to the true expectations,

$$\lim_{N \to \infty} \hat{f}_N = E_\pi[f]. \tag{2.4}$$

Because time and memory are limited resources in real world applications, the asymptotic behavior is of limited use in practice and robust methods are necessary that ensure that the Markov chains reach a sufficiently close approximation of the target distribution after only a finite number of transitions. (Betancourt, 2017).

The exploration of parameter space by a Markov chain typically happens in three phases (Betancourt, 2017) [Figure]. In the first phase the Markov chain converges (or tries to converge) towards the typical set from its initial position in parameter space. The samples from this first phase are normally excluded from the final set of samples (which is called the burn-in of the MCMC sampler), because they are not representative for the form of the target distribution but were necessary for finding the right region in parameter space. The second phase begins once the Markov chain finds the typical set and now a first exploration of the typical set takes place. This initial exploration is extremely effective and the accuracy of the MCMC estimators rapidly improves. The third and last phase consists of an ongoing exploration where the Markov chain refines its exploration of the typical set, but the precision of the estimators improves much slower.

When the MCMC sampler has entered the third phase, the MCMC estimators satisfy a Central Limit Theorem

$$\hat{f}_N^{MCMC} \sim \mathcal{N}\left(E_\pi[f],\ \text{MCMC-SE}\right), \tag{2.5}$$

where the MCMC standard error is defined as

$$\text{MCMC-SE} \equiv \sqrt{\frac{\text{Var}_\pi[f]}{\text{ESS}}}. \tag{2.6}$$

The effective sample size is defined as

$$\text{ESS} = \frac{N}{1 + 2\sum_{k=1}^{\infty} ACF(k)}, \tag{2.7}$$

where ACF(k) is the lag-k auto-correlation of f over the history of the Markov chain. When consecutive samples of a Markov chain are highly auto-correlated, then ESS will be low and the MCMC standard error will be high due to a lack of sufficient non-correlated samples. Thus, a high ESS value is favorable. The effective sample size quantifies the number of exact samples from the target distribution necessary to give an equivalent estimator precision and hence the effective number of exact samples "contained" in the Markov chain (Betancourt, 2017, p. 16). Kruschke (2015) recommends an ESS of 10,000 for reasonably accurate and stable estimates of the limits of the 95 % HDI. As we will see, such large numbers for ESS are hard to achieve for more complex models. Gelman et al. (2014) note that, in general, fewer simulations are needed to estimate posterior medians of parameters, probabilities near 0.5, and low-dimensional summaries than extreme quantiles, posterior means, probabilities of rare events, and higher-dimensional summaries. The "guess-and-check" strategies of classical but still widely used MCMC algorithms like Metropolis-Hastings are "doomed to fail in high-dimensional space" due to the exponential number of directions in which to guess (Betancourt, 2017). More modern approaches like

Hamiltonian Monte Carlo exploit information about the geometry of the typical set and generate coherent exploration of smooth target distributions which yields better computational efficiency than other MCMC algorithms, along with stronger guarantees on the validity of the resulting estimators (Betancourt, 2017). Betancourt (2017) gives an introduction to the ideas behind Hamiltonian Monte Carlo.

How can we diagnose convergence? Visually, convergence means that different chains of a MCMC run meet in the same region of parameter space. Divergence could mean that different chains keep stuck in different regions of the parameter space or that they evolve over time, but yet do not converge to the same region. A common measure for chain convergence is the Gelman-Rubin statistic, which the authors Gelman et al. call the potential scale reduction. The idea is to evaluate mixing and stationarity simultaneously. Mixing is assessed via a between-sequence variance B, whereas stationarity is assessed via within-sequence variances W. The potential scale reduction $\hat{R}$ is estimated by a weighted average of this two variances,

$$\widehat{var}^{+}(\psi|y) = \frac{n-1}{n}W + \frac{1}{n}B \tag{2.8}$$

$$\hat{R} = \sqrt{\frac{\widehat{var}^{+}(\psi|y)}{W}}. \tag{2.9}$$

In the limit of infinite samples $\hat{R}$ declines to 1, which means perfect convergence. If the potential scale reduction is high, then we have reason to believe that the chains have not fully converged yet and further simulations may improve the inference about the target distribution. There is no hard threshold for when $\hat{R}$ is considered a high value, but according to Lee and Wagenmakers (2014, p. 80) values for $\hat{R}$ higher than 1.1 are "(deeply) suspect". This threshold is also the reported value in the probabilistic programming library PyMC3 that is used in this thesis. A further discussion of methods for monitoring convergence of iterative simulations can be found in Brooks and Gelman (1998).

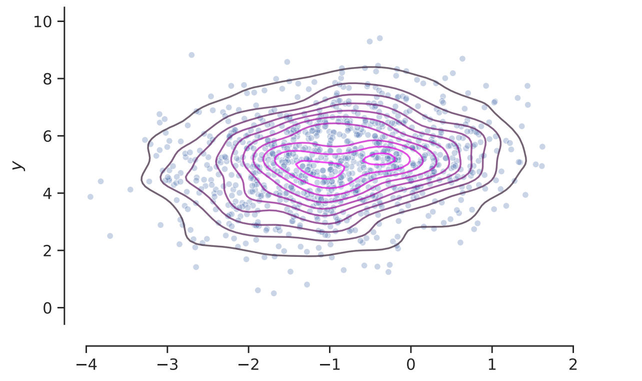
Table 2.3.: Summary statistics of the MCMC chains for the toy example of this section.

	Mean	SD	MCMC-SE	$\text{HDI}_{2.5}$	$\text{HDI}_{97.5}$	n_{eff}	$\hat{R}$
μ_1	−0.98	0.03	0.0	−1.05	−0.92	4007	1.0
μ_2	5.03	0.04	0.0	4.94	5.12	4045	1.0

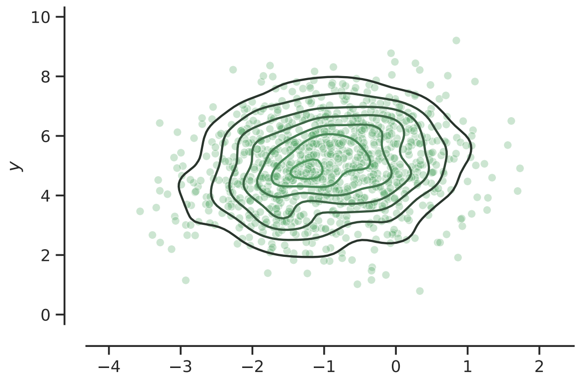
Example: An MCMC example

I conclude this section with a small toy example to introduce the most important plots and results when working with MCMC methods for Bayesian data analysis. Consider a two-dimensional data set of 1000 data points $\mathscr{D} = \{(x_i, y_i)\},\ i = 1, \ldots, N = 1000$ (Figure 2.13a on the next page). We want to model these data with a multivariate normal distribution, hence $y \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$. For the sake of simplicity we assume that we know the co-variance matrix $\boldsymbol{\Sigma}$ well enough so that we are only concerned with finding values for the mean values of the multivariate normal distribution. For our model this means that we have the observations (x_i, y_i) along with the co-variance matrix $\boldsymbol{\Sigma}$ and our latent unknown variables are μ_1 and μ_2.

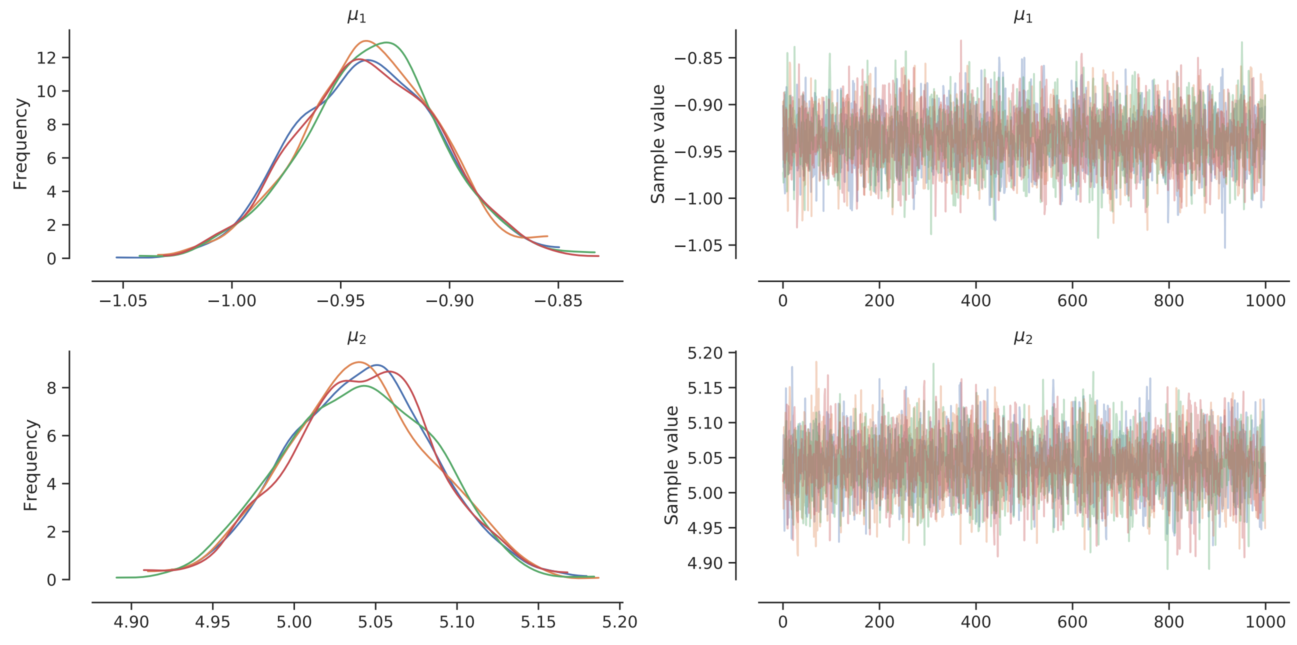
I chose a Hamiltonian Monte Carlo sampler as MCMC algorithm and drew 1000 samples after a tuning phase of 500 samples. I ran four chains simultaneously, which resulted in 6000 samples in total. The traceplot for each parameter shows us that the different chains have converged after the tuning phase to the same region in parameter space (Figure 2.13c on the following page), which improves our confidence in the results. The main result of the analysis is the posterior distribution for the latent parameters μ_1, μ_2 (Figure 2.12 on page 37). Having the posterior distribution, we can easily compute the 95 % HDIs along with point estimates like the mean or the median for the parameters. The width of both the 95 % HDI is very small, further increasing the confidence into the model's belief. The summary statistics for the trace show small MCMC errors, small 95 % HDI and good values for $\hat{R}$, indicating convergence of the four chains. The effective number of samples n_{eff} is around 66 % of the overall sample size, which is good enough for a good first estimate of central tendencies as long as we are not interested in rare events or the exact tails of the distribution (Table 2.3). Finally, looking at the predictions from the posterior predictive distribution, we can safely assume that our model was able to capture the essence of the observable data (Figure 2.13b on the next page).



(a) Two-dimensional observations (x_i, y_i) obtained from a two-dimensional multivariate normal distribution with $\boldsymbol{\mu} = [-1, 5]$ and $\boldsymbol{\Sigma} = [[1, 0.25], [0.25, 2]]$. The density is approximated by a kernel density estimation.



(b) 1000 samples drawn from the posterior predictive distribution of the model. The density is approximated by a kernel density estimation.



(c) Traceplots for the two model parameters μ_1, μ_2 for each of the four chains. (Left) Posterior distribution as kernel density estimation for each of the four chains. (Right) Traceplot of the 1000 samples for each of the four chains drawn after the tuning phase.

Figure 2.13.: Example of Bayesian data analysis fitting a multivariate normal distribution with unknown mean and known co-variance matrix to two-dimensional data. Plots belong to the MCMC example

.

2.2 How to Capture User Behavior – Experience API

Learning Analytics (LA) is a new trend in the e-learning community that applies data-driven approaches to e-learning applications (Blanco et al., 2013). The aim is to gather and analyze educational data. The learning experience itself is increasingly taking place within Learning Management System (LMS) deployed by educational institutions. One problem of these LMSs is that thy lack standardized data structures. Thus, LA tools tend to be tied to specific implementations of LMS and databases (Blanco et al., 2013).

One of the key components of LA and Educational Data Mining (EDM) are the learner's sensors which capture the data generated during the interaction with a LMS. The captured data is stored in a Learning Record System (LRS) to be consumed by the LA services. A key requirement to facilitate interoperability among the architecture components and to enable the integration of new ones is a specification that describes a universal format of how data are captured, stored and, retrieved.

Interoperability is defined by Bakhouyi et al. (2017) as "the ability of different e-learning systems and software applications to ensure the harmonization of content between them, allow the sharing of educational content between different environments, to exchange data and to use the information exchanged." The objectives of e-learning interoperability standards is to provide a compliant data model and communication protocol. Once these standards are integrated into e-learning systems, users can start to create and use content from multiple systems and from different suppliers.

Several initiatives have tried to develop standards for e-learning content interoperability. Among these initiatives are the IMS Global Learning Consortium, the IEEE Learning Technology Standards Committee (LTSC), the Aviation Industry CBT Committee (AICC), and the Advanced Distributed Learning (ADL) Initiative (Bakhouyi et al., 2017). ADL has developed a set of technical guidelines called SCORM (Sharable Content Object Reference Model) to better support flexible and lifelong learning. The next evolution of SCORM was the Tin Can API, an open source standard that is flexible and serves to track learning experiences and activities as well as store learning data. In 2013, the third version of the Tin Can API was published under the name Experience API (xAPI). It is a Representational state transfer (REST) wen service based on JavaScript Object Notation (JSON) for its data format. The Experience API (xAPI) specification, has become the de facto standard due to its simple data model and the number of vendors that have adopted it (Vidal et al., 2015).

Berdun and Armentano (2018, p. 1) define a user profile as a "representation or description of different aspects of a user in a computer application". Today, intelligent systems automatically learn the collaborative profiles of users by means of the observations of the users behavior (Berdun and Armentano, 2018). The traditional method to obtain information about the user is by means of self-perception questionnaires, which have many disadvantages. Instead, it is desired to build profiles from the observations of the user behavior in an digital game that tests the users ability. Berdun and Armentano give an example of how to build profiles from game actions based on the coding scheme Multiple Level Group Observation System (SYMLOG). They collected all actions taken by users in a game session like actions that the user executes in the regular course of the game, spontaneous decisions, participation in collective decision making situations, and the state of the game variables that condition the performed action. The rules used for mapping user actions to SYMLOG attributes followed the structure `<user>` ∧ `<action>` ∧ `<state>` ⇒ `<SYMLOG>`.

xAPI focuses on defining and interoperable data model for storing data about students' learning experience and an API for sharing these data among systems. The central element in xAPI is the LRS, which can reside inside a LMS or in an independent server. At any time, activities can send in their collected data over the xAPI web service. xAPI uses RESTful HTTP requests and can be used with any programming language.

The xAPI data model is based on the concept of Activity Streams (*Activity Streams 2.0* 2017) and highly influenced by the socio-cultural framework Activity Theory, which stores a user's activity as a statement of the form "I did this." The xAPI data model is an extension of this simple idea to track all aspects of the learning experience. The basic structure of an xAPI statement is as follows (Figure 2.14 on page 46):

```
<actor> <verb> <object>, with <result>, in <context>.
```

The three properties `actor`, `verb`, and `object` are required but can be complemented with `result` and `context` elements. A complete example with all properties is given in Listing 2.2 on the next page. CMI-5 is a standard that was officially launched in June 2016 by ADL. It represents the combined effort of AICC and ADL to resolve the problems and shortcomings of the SCORM standards and its xAPI extension by adding extra rules. This new specification was adopted by more then 170 organizations in June 2016 (Bakhouyi et al., 2017).

There exist a range of validation tools for xAPI statements. Rabelo, Lama, Vidal, et al. (2017) have analyzed four of the most promising tools and have found that SmartLAK is the most suitable tool to perform the validation and verify the xAPI data model: statements, contexts, and activities. SmartLAK also provides a GUI from which users can directly check their JSON-formatted xAPI statements. In addition, SmartLAK offers Intelligent Analytics Services (IAS) that process the LRS data to provide valuable information that help teachers to gain a better understanding of what is happening in the course or to facilitate the teaching and learning process (Rabelo, Lama, Amorim, et al., 2015).

Serrano-Laguna et al. (2017) developed an interaction model that establishes a basis for applying LA into serious games and presented an implementation of their model with the xAPI. They identified common targets, that is, objectives of the players' action, across all surveyed serious games: a) *completable*, which a player can start, progress on and complete within a serious game, b) *alternative*, which is a set of options among which the player has to choose at a given point in the game, c) *meaningful variable*, which is a value inside the game world with a special significance and which can be set by the player, and d) *custom interaction*, which is an extension to track events not covered by the other event types above. They provided a mapping of the interaction events, of the event actions and the target types to the associated xAPI equivalence and showed the practicality of their implementation with the Countrix serious game as an example of use of the Serious Game xAPI profile.

Listing 2.2: xAPI statement according to CMI-5 specifications, from the official xAPI specification.

```
{
   "id":"2a41c918-b88b-4220-20a5-a4c32391a240",
   "actor":{
      "objectType":"Agent",
      "name":"Gert Frobe",
      "account":{
         "homePage":"http://example.adlnet.gov",
         "name":"1625378"
      }
   },
   "verb":{
      "id":"http://adlnet.gov/expapi/verbs/failed",
      "display":{
         "en-US":"failed"
       }
   },
   "object":{
      "id":"https://example.adlnet.gov/AUidentifier",
      "objectType":"Activity"
   },
   "result":{
      "score":{
         "scaled":0.65,
         "raw":65,
         "min":0,
         "max":100
      },
      "success":false,
      "duration":"PT30M",
      "extensions":{
         "https://w3id.org/xapi/cmi5/result/extensions/progress":100
      }
   },
   "context":{
      "registration":"ec231277-b27b-4c15-8291-d29225b2b8f7",
      "contextActivities":{
         "category":[
            {
               "id":"https://w3id.org/xapi/cmi5/context/categories/moveon"
            },
            {
               "id":"https://w3id.org/xapi/cmi5/context/categories/cmi5"
            }
         ]
      },
      "extensions":{
         "https://w3id.org/xapi/cmi5/context/extensions/sessionid":"458240298378231"
      }
   },
   "timestamp":"2012-06-01T19:09:13.245+00:00"
}
```

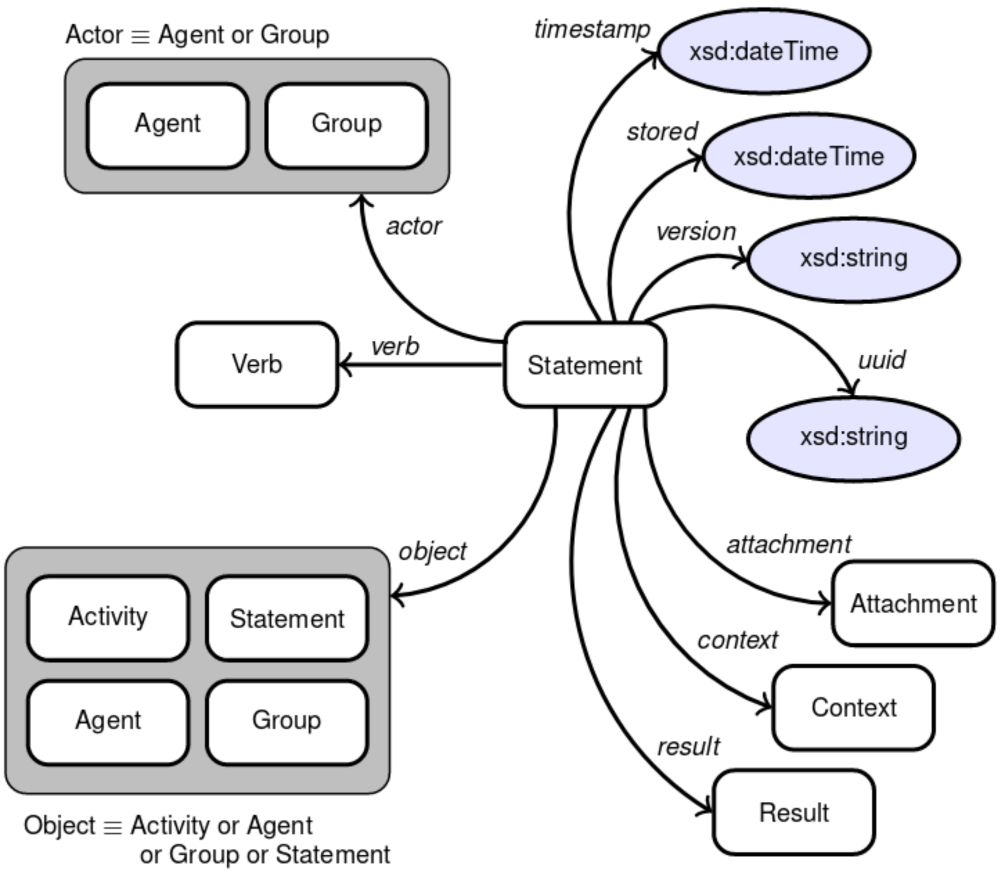


Figure 2.14.: The semantic network of the *statement* model of the xAPI specification (from Vidal et al., 2015, p. 268). A statement is used to represent a learning event and as such consists of three required properties: the actor, the verb, and the object (left hand side). Besides the required properties, xAPI statements are allowed to store additional information about results, the context, the authority, and can have an attachment as part of the learning record. The remaining four properties are data properties.

2.3 How to Think About User Behavior – Learner Attributes

Researchers and developers from the educational community started not long ago exploring the potential adoption of sophisticated analytic techniques to evaluate rich data sources. Two areas under development oriented towards the inclusion and exploration of data-driven capabilities in education are Educational Data Mining (EDM) and Learning Analytics (LA). EDM is concerned with "developing, researching, and applying computerized methods to detect patterns in large collections of educational data that would otherwise be hard or impossible to analyze due to the enormous volume of data within which they exist" (Romero and Ventura, 2012, p. 12). And LA is "the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and environments in which it occurs"[2]. For the purpose of this thesis, the field of LA can offer valuable insights into what constitutes a good learning experience and which learner attributes are already in the focus of interest.

Papamitsiou and Economides (2014) provide a literature review of 40 key studies published between 2008 and 2013, which revealed four distinct major directions of the LA/EDM empirical research. The majority of studies investigated issues related to student/student behavior modeling and prediction of performance, followed by increase of students' and teachers' reflection and (self-)awareness and improvement of provided feedback and assessment services. Modeling students' learning behavior also comprises modeling affective and metacognitive states. Most of these studied used learners' interactions with the learning environment, automatically-generated data during the activity, response times and sequence of actions to infer learning strategies. Regarding the student's motivation, studies found that demographics and factors like achievement rates and final performance were associated to students' motivation to remain engaged and actively enrolled in courses. Because motivation is one the main cognitive variables that a CogIUM might infer from users' interactions, these findings are directly relevant for this work. For the task of performance prediction the number of quizzes passed was the main determinant of performance, while other measures like frequencies of the events and time-spent could identify activities that are related to higher or lower marks. In another study, engaged concentration and frustration were correlated with positive learning outcomes, while boredom and confusion are negatively correlated with performance. Regarding the task of predicting dropout and retention studies could show that monitoring students' activity and applying data-driven machine learning methods on students' profiles and log files from LMS databases allowed for detecting students at risk at an early state. Studies also suggest to consider additional learner attributes like experience level indicators, learning interests, learning styles, learning goals and competences and background information, their recent navigation history or learner's affective traits in recommendation processes.

Serrano-Laguna et al. (2017) performed a review of serious games found in the literature to detect the common interactions tracked in serious games. The most common form of interaction strategy encountered in games is event-based, where the game logs pre-specified events when they occur. Most of these events include at least a timestamp, when the event was generated, and a user id, identifying the player that originates the event. The analyzed serious games tracked the completion (binary value) or the level of completion achieved (percentage) by the players, whether the game was fully completed or a

2 https://tekri.athabascau.ca/analytics/

more fine-grained level of detail. Furthermore, they tracked the in-game choices performed by players in a given context, most commonly questions with multiple answers. Serious games also gathered meaningful measurable variables to calculate the players' learning outcomes. The most common variables were score, number of in-game deaths and kills or coins collected. All these variables can reveal the level of success in the learning goals involved. Of course, most of these games also collected game-specific events, for example chat logs, number of times a player asked for in-game help, and biometric information using several external devices.

Kickmeier-Rust, Mattheiss, et al. (2011) present a psycho-pedagogical framework for multi-adaptive educational games that utilizes the formal framework of the Competence-based Knowledge Space Theory (CbKST), which is a cognitive framework, to realize micro level adaptation. The basic idea of CbKST is to separate the observable performance and underlying latent skills or competencies. The relationship between the skills and learning objectives are established by skill and problem functions. CbKST provides a probabilistic approach to assessment with probability distributions over all possible skill states, and with each action the probabilities of those states that include the relevant skill are updated. CbKST allows for monitoring and interpreting the learner's behavior in the game and enabled the authors to assess the player's motivation.

Kickmeier-Rust and Albert (2012) give an overview about the assessment in educational serious games. They list the following performance related aspects: scores, task completion rates and times, task success rates, task success depth, distance covered and progress in the game world, exhibited knowledge, competence states, or skills, and incongruent behaviors as indicators for succeeding by chance. These aspects should be analyzed and interpreted to assess the following important dimensions: individual preferences, progress, results and scores, traits and aptitudes, prior knowledge and ability and prior achievements. Another model that supports cognitive aspects is the Cognitive Trait Model (CTM), which enables student modeling on the basis of cognitive abilities and resources (Lin et al., 2003).

In this thesis I wanted to realize a CogIUM that is based on a theory which gives a plausible explanation for how cognitive variables might lead to the observed performance of a learner in an educational serious digital game. Ideally, the theory should incorporate affective states as well as motivation and prior knowledge, all aspects we have seen to be important for predicting the learner's performance. One such theory is Cognitive Load Theory (CLT) which was chosen as the theory behind the realized CogIUM. I will present the theory along with its main components and how they affect a learner's performance in the next section.

2.3.1 Cognitive Load Theory

What distinguished an expert from a novice? How are experts able to process much more information than novices without obvious difficulties? Sweller (1988) answered these questions with the concept of cognitive load. Experts differ from novices in three major ways: they can store a higher number of problem state configurations, also called *chunk size*, they use more effective problem solving strategies by recognizing each problem and each problem state from previous experience and know appropriate moves, also called *schema*, and they use more relevant features in categorizing problems according to acquired schemas. Thus, schema acquisition constitutes a primary factor when determining a person's problem solving skill.

To explain the phenomenon that some problem-solving strategies like means-end analysis infer with learning, Sweller assumed that the cognitive load imposed on a person using a complex problem solving strategy might interfere with learning during problem solving. The steps required to conduct the strategy use much of the available cognitive-processing capacities, so that little is left for schema acquisition. Sweller could show in experiments that changing the strategy to a nonspecific goal strategy indeed enhanced the development of problem-solving expertise.

Why should schema acquiring help in processing more information or freeing resources? Merriënboer and Sweller (2005) explain that the central assumption is that working memory as a short-term memory is limited and stores about seven elements, but operators on just two to four elements at a given time. Importantly, the capacity of working memory is limited only when dealing with new information obtained through sensory memory. Working memory has no known limitations when retrieving information from long-term memory, on the contrary, long-term memory alter the characteristics of working memory. This is due to cognitive schemata stored in long-term memory. Schemata organize information or knowledge that is processed in working memory, freeing capacities of working memory for other processes. In contrast, when dealing with novel information for which no schema-based central executive is available, working memory capacity becomes a limiting factor. If no knowledge can be used to organize information, it must be organized randomly for the first time and the organization then is tested for effectiveness. Schemas can be acquired through a great deal of practice, but only for those aspects of performance that are consistent across problem solving situations, such as routines. More on the aspects of the human cognitive architecture can be found in Sweller et al. (1998).

Sweller et al. introduced the Cognitive Load Theory (CLT) to provide guidelines for an optimal instruction design in a manner that encourages learner activities that optimize intellectual performance. CLT postulates that working memory load may be affected either by intrinsic cognitive load (ICL) or extraneous cognitive load (ECL) (Merriënboer and Sweller, 2005). Whereas ICL represents the intrinsic nature of the material being learned and depends on the number of interactive elements that must be processed simultaneously in working memory, ECL, in contrast, is load that is not necessary for learning, that is, for schema construction and automation. A high ECL can lead to a range of undesirable effects (Merriënboer and Sweller, 2005; Sweller, 1994, 2010; Sweller et al., 1998). ICL cannot be altered by instructional design because the element interactivity of the material or task is fixed. Materials with high element interactivity are difficult to understand and the only way to deal with it is to develop cognitive schemata that incorporate the interacting elements. ECL, instead, can be altered by instructional interventions. ICL and ECL are additive and together sum up to the total cognitive load experienced by the learner. A third type of load had to be included to explain findings where high variability in materials presented to learners increased the total cognitive load during practice but yielded better schema construction and transfer of learning. germane cognitive load (GCL) describes the resources working memory dedicates to schema construction and automation and thus contribute to learning. Although the triarchic theory of cognitive load is not supported by all researchers, there is strong evidence that favors the triarchic theory of cognitive load over a unitary theory of cognitive load (DeLeeuw and Mayer, 2008; Klepsch et al., 2017).

The clearest and most accurate description of the CLT is given in an article by Sweller (2010), in which he not only states that both ICL and acecl can be explained by element interactivity, but clarifies misconceptions about GCL that were emerging in the literature (Figure 2.15 on the following page). As was said, ICL is imposed by the natural complexity of information that must be understood and material that must be learned and is independent of the instructional design. However, ICL is not independent of the learner, because the learner's knowledge level determines the amount of cognitive load that is imposed by ICL. An expert with rich schemas experiences the complexity of a given material differently from a novice with no task knowledge. For a given task and given learner knowledge level, however, ICL is fixed and cannot be altered other than by changing the task itself. ECL is cognitive load imposed by instructional procedures that are less than optimal. ECL is entirely independent of the characteristics of the learner. But it can be minimized by optimizing the instructional design and CLT is primarily concerned with techniques designed to reduce this type of cognitive load. GCL differs from ICL and ECL as it is determined entirely by the characteristics of the learner. GCL refers to the working memory resources that are devoted to deal with the ICL of the learning material. Thus, GCL is independent of the information presented and it does not contribute to the imposed working memory load, it is purely a function of the working memory resources devoted to deal with the interacting elements. GCL can, however, be altered by the motivational level of the learner. If the level of motivation is constant, the learner ha no control over GCL. In summary, cognitive load can be due to element interactivity associated with either intrinsic or extraneous cognitive load. Working memory resources that deal with ICL are germane to the task and so are referred to as GCL. Workin memory resources that deal with ECL do not contribute to learning but must be allocated if the instructional procedures demand those resources.

To better understand the relationship between the three components of cognitive load and when cognitive load is expected to influence learning, I have prepared a demonstration of different cases that I will discuss now (Figure 2.16 on page 52). When the complexity of the learning material or the task is low, it does not matter whether the material is poorly designed or presented or the instructional procedures are non-optimal because there are enough free working memory resources to deal with the additional ECL (first box in Figure 2.16 on page 52). When the total cognitive load is lower than the available working memory resources no effects of ECL on the learning performance might be experienced. With more complex material and higher element interactivity, things are different. When ICL is high but ECL is low, the total cognitive load might equal the working memory resources so that ECL has no effect. But with an increase in ECL the total cognitive load exceeds the working memory capacity. Because ECL has always to be dealt with, all working memory resources that are necessary to deal with ECL have to be allocated and only the rest can be used to deal with ICL. We remember that GCL is defined as the working memory resources devoted to deal with ICL. So in this example GCL has declined because ECL was increased, and learning performance will be affected negatively (second box in Figure 2.16 on page 52). The last case I want to discuss is the effect of an altered level of motivation. I assume that motivation determines the overall amount of working memory that is dedicated to learning. With other words, a high level of motivation leads to all working memory resources are dedicated to learning. If, however, the level of motivation decreases, less working memory resources are freed for the learning activity. In addition to the resources that are bound by dealing with the increased ECL now resources are missing because of a lower level of motivation. As a result, GCL has declined even further and the learner's performance has worsened (third box in Figure 2.16 on page 52).

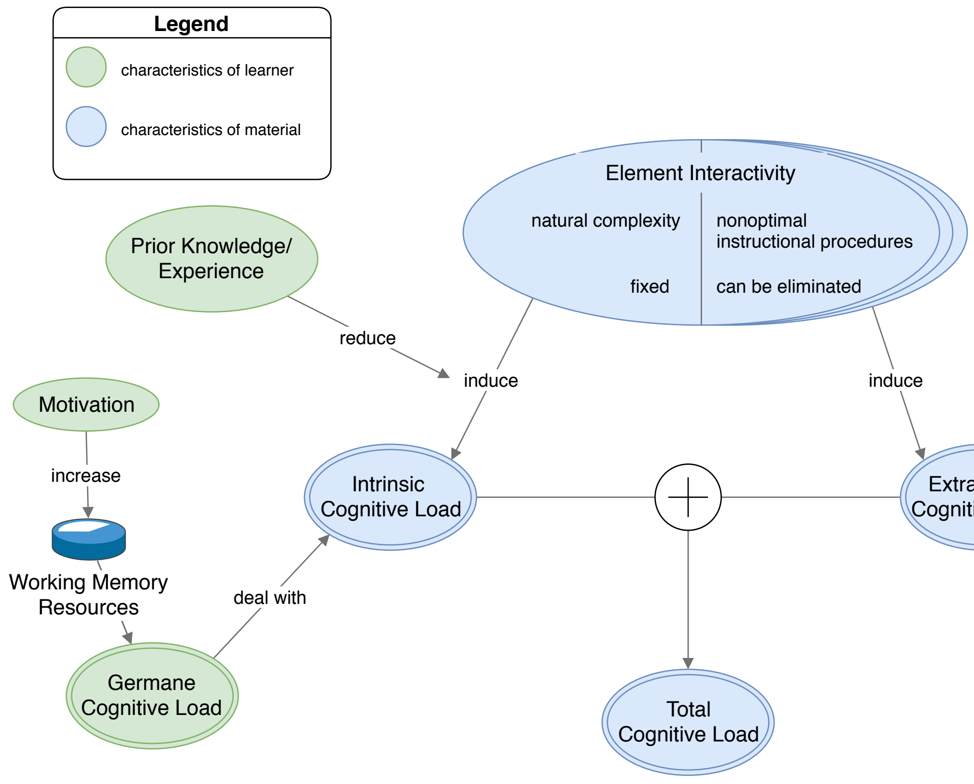
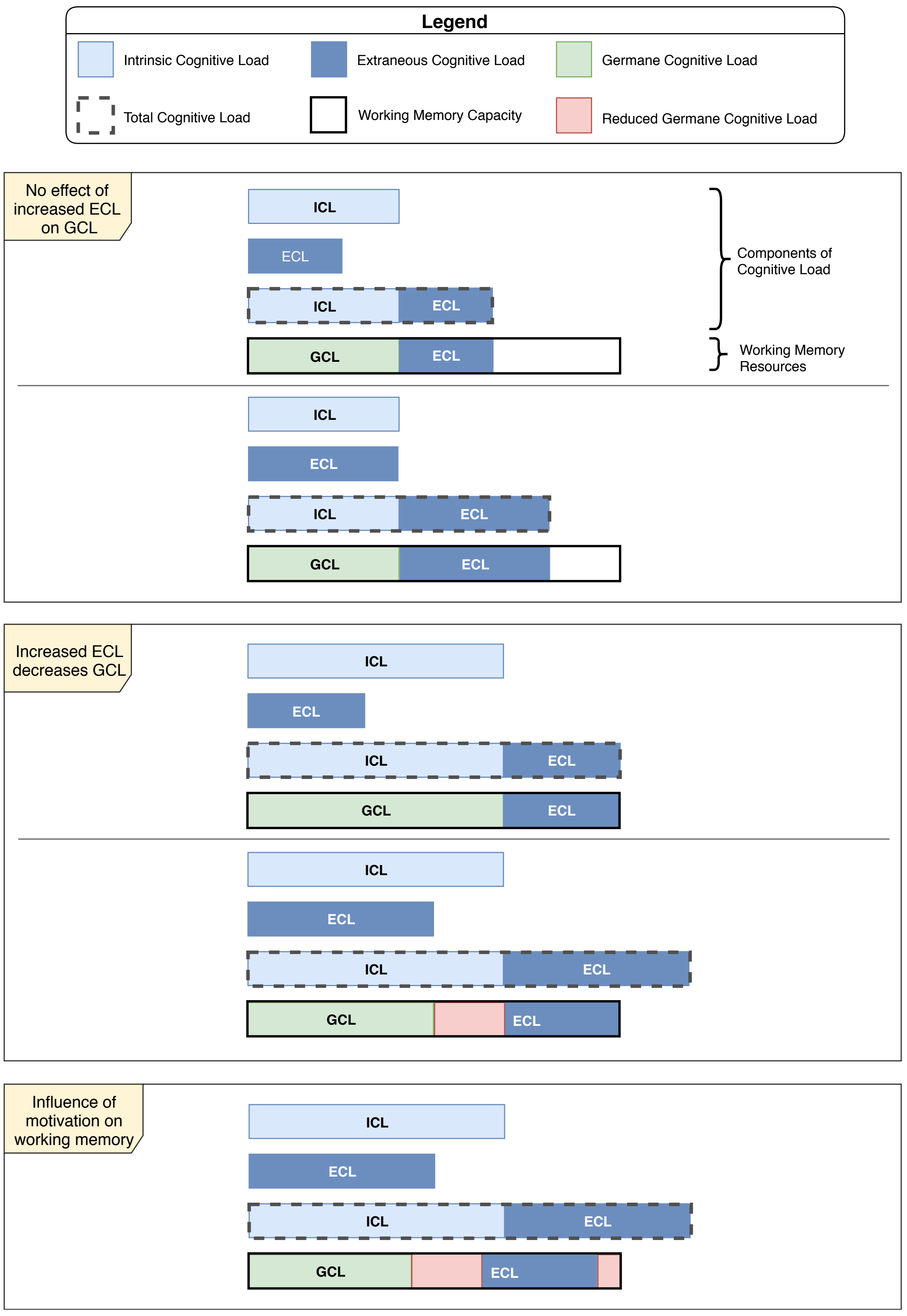


Figure 2.15.: Relationship between element interactivity and intrinsic, extraneous, and germane cognitive load according to the CLT. Both intrinsic and extraneous cognitive load are determined by element interactivity and are primarily characteristics of the learning material. Intrinsic cognitive load is fixed, but the learner's level of knowledge determines the element size. Extraneous cognitive load is imposed by the instruction design and can be minimized or eliminated. Germane cognitive load is independent of the learning material and refers to the working memory resources allocated for dealing with the intrinsic cognitive load. The overall cognitive load, as experienced by the learner, is the sum of intrinsic and extraneous cognitive load, but not of germane cognitive load. Own graphic, based on (Sweller, 2010).

CLT is a good theory to use as the foundation of a CogIUM, because cognitive load influences learning performance. Merriënboer and Sweller (2005, p. 166) state that performance is one assessment dimension of cognitive load, "because a higher cognitive load often increases the number of errors, and slows down performance." Additional dimensions are *mental load*, which originates from the interaction between task characteristics and learner characteristics like prior knowledge, and *mental effort*, which refers to the "cognitive capacity that is actually allocated to accommodate the demands imposed by the task". In addition, they report findings of adaptive e-learning based on assessment of cognitive load. In a first step the learner's expertise was assessed, which led to a dynamic selection of the next learning task in a second step. Expertise was assessed by how much mental effort the learner had to invest to achieve a certain performance. In several studies adaptive e-learning proved superior to the use of a fixed sequence of tasks.



Legend
Intrinsic Cognitive Load
Extraneous Cognitive Load
Germane Cognitive Load
Total Cognitive Load
Working Memory Capacity
Reduced Germane Cognitive Load
No effect of increased ECL on GCL
ICL
ECL
GCL
Components of Cognitive Load
Working Memory Resources
Increased ECL decreases GCL
Influence of motivation on working memory

Figure 2.16.: Influence of ECL and motivation on GCL for three different cases. For a detailed explanation see the in-text description. Own graphic, based on (Sweller, 2010).

3 Literature Review

This chapter provides an overview of the state of the art and the latest trends in computational cognitive modeling, model evaluation and model comparison, and measuring cognitive load.

3.1 Computational Cognitive Modeling

This chapter deals with current developments and methods in the field of cognitive architectures and learner models in Intelligent Tutoring Systems. The presented information may be helpful to better understand the approach, the procedure and the contribution to the research of this thesis. The literature review will indicate that the approach of this thesis in this form is unique and represents an extension of the current body of knowledge.

3.1.1 Cognitive architectures

Taatgen and Anderson (2010) note that most research o cognitive architectures involves building models of particular phenomena, and only a smaller portion of research is focused on the architectures itself. One development is use findings from neuroscience to further refine the cognitive architecture By implementing the cognitive architectureCT-R as a neural network, it was found that the model could only perform a single retrieval from declarative memory at a time. This constraint was subsequently introduced in the standard architecture and moved the ACT-R architecture closer to modeling the limitations on human cognition. Other approaches try to directly design cognitive architectures at the level of neural networks. One example is given by the Leabra architecture. Taatgen and Anderson, p. 699 criticize that it takes "substantial intellectual commitment to learn to understand models of a particular architecture and to learn to construct models." In addition, it is infeasible to compare multiple architectures. As a consequence, modelers often do not use an architecture at all but build cognitive models out of components specifically designed for the study at hand. However, if the current developments would go one, they express the hope that the choice of architecture does not imply a strong theoretical commitment, because most mechanisms that underlie human cognition will be probably included. They make a forecast that the gap between symbolic and neural networks will be bridged. Finally, they propose the amount of task-specific knowledge as a measure of simplicity when choosing between different models and architectures.

The most recent overview of the last 40 years of research on cognitive architectures was conducted by Kotseruba and Tsotsos (2018). They analyzed a set of 84 architectures, of which 49 are still actively developed. They estimate the number of existing cognitive architectures to be around 300. They focused on architectures with at least one practical application and several peer-reviews publications and mechanisms for perception, attention, action selection, memory and learning. As a first result they provide a timeline of the 84 cognitive architectures with ACT-R and Soar among the oldest. They present a taxonomy of the cognitive architectures into the three categories symbolic, emergent, and hybrid. The hybrid category is further divided into *symbolic sub-processing*, where sub-symbolic computation is limited to a self-contained module within the architecture, and *fully integrated* that contains all other types of hybrids. The majority of cognitive architectures belongs to the hybrid category, including Soar, "showing the tendency to grow even more". This finding confirms the prognosis made a decade ago that cognitive architectures will converge eventually into hybrid architectures (Sh, 2007; Taatgen and Anderson, 2010). Architectures like ACT-R and Soar, which are hybrid, combine symbolic concepts and rules with sub-symbolic elements such as activation values, spreading activation, stochastic selection process, and reinforcement learning. The authors discuss in detail the mechanisms that are supported by the different cognitive architectures with respect to perception, attention, action selection, memory, learning, reasoning, and metacognition. They conclude with a detailed overview of 900 projects that were implemented using the 84 cognitive architectures. It is noteworthy that ACT-R and Soar are the architectures that have by far the most practical applications. Soar was used in equal parts to create agents for games and puzzles, miscellaneous, and virtual agents, whereas the most work with ACT-R was done for psychological experiments. With respect to the prsented list of cognitive abilities and phenomena the authors conclude that "none of the systems we reviewed is close to supporting in theory or demonstrating in practice even this restricted subset, let alone a set of identified cognitive abilities" and that "most of the featured architectures cannot reuse the capabilities or accumulate knowledge as they are applied to new tasks. Instead, every new task or skill is demonstrated using a separate model, specific set of parameters or knowledge base." In their discussion they list a range of open research questions and future directions. The decision to use Soar 9 as the concrete cognitive architecturen this thesis is partly based on this review.

With their "standard model of the mind" Laird, Lebiere, et al. (2017) extract key aspects of structure and processing, memory and content, learning, and perception and motor from the synthesis across three existing cognitive architectures: ACT-R, Sigma, and Soar. They try to give the best consensus given the community's current understanding of the mind. Symbols in the standard model are the primitive elements over which relations an be defined and allow for the creation of complex symbolic structures, including semantic networks, ontologies, and taxonomies. Non-symbolic information has two roles in the standard model: to represent quantitative task information and to annotate the representations of task information to guide how it is processed, that is, to play the role of metadata. The standard model involves a hybrid combination of symbolic and statistical processing, embodies forms of statistical learning, including Bayesian and reinforcement learning, and uses significant amounts of parallelism. The core components of the standard model are perception an motor, working memory, declarative long-term memory, and procedural long-term memory. The authors summarize and persent the key assumptions that underlie the standard model of human-like minds and provide an analysis of the extend ACT-R, Soar, and Sigma agree in theory with the standard model and implement the corresponding capabilities. Their main finding is that while the architectures of the early-90s showed significant disagreement or a lack of theory, their current versions are "in total agreement in terms of theory and only substantially differ in the extent to which they implement perception and motor systems." They also note that the standard model remains incomplete regarding metacognition, emotion, mental imagery, direct communication and learning across modules, and social cognition.

Although the three categories, emergent, symbolic, and hybrid, are the dominant ones, there exist attempts to build models that use other approaches. Shi et al. (2007) present their Globally Connected and Locally Autonomic Bayesian Network (GCLABN) that adopts Bayesian networks to integrate the merits of rule-based systems and neural networks. The model is composed of numerous interconnected and overlapping tiny Bayesian networks to model the overall cognition. It employs a unique knowledge representation strategy, generates cognition via dynamic oscillation, and provides a white-box architecture by manipulating symbolic concepts with probabilistic reasoning.

Cognitive architectures, especially Soar, are widely used or considered for realizing agents that learn how to play games, from simple games such as Tic-Tac-Toe to complex computer strategy games such as *Civilization* or *Starcraft* (Laird, Gluck, et al., 2017). There has been success with simple games with limited numbers of objects, but current techniques have not yet scaled up to more complex games. Complexity dimensions include the number of movable pieces and places, the number and complexity of rules and their interactions, the responsiveness/speed of gameplay and the number and types of relevant spatial relations. However, to the best knowledge of the author there are no studies that investigate how cognitive architectures can be used to model the learner or to derive the learner's current cognitive state. Therefore, using Soar 9 to realize a CogIUM to represent the learner's current cognitive state would be a contribution to the field.

3.1.2 The student model in Intelligent Tutoring Systems

Classical ITS have four main components: a domain module that represents expert knowledge and includes definitions, processes, or skills, a student module that represents a student's mastery of the domain and contains both stereotypical student knowledge of the domain and information about the current student, a tutoring module that represents teaching strategies and includes methods for encoding reasoning about the feedback, and a communication module that represents methods for communicating between students and computers (Woolf, 2009). The student model in ITS is equivalent to the CogIUM in an adaptive system for serious games. A student model in ITS observes student behavior and creates a qualitative representation of the learner's cognitive and affective state. The model partially accounts for student performance like time on task and observed errors. Woolf lists several issues that have to be considered when building a student model: how to represent student knowledge, how to update information to infer the student's current knowledge and how to improve student behavior. She gives an overview of several knowledge categories along with their typical representations used to model student and domain knowledge, including semantic nets, rules, constraints, plan recognition, and machine learning. Techniques to update student models are classified based on their origin: cognitive science or artificial intelligence. Cognitive science techniques include model-tracing and constraint-based methods, whereas artificial intelligence techniques include formal logic, expert systems, plan recognition, and Bayesian belief networks.

Chrysafiadi and Virvou (2013) conducted a literature review on student modeling approaches that have been used in the past ten years. The data usually represented in a student model includes knowledge level, skills, learning preferences and styles, errors an misconceptions, motivation, affective features such as emotions and feelings, cognitive aspects such as memory, attention, solving, making decisions and analyzing abilities, critical thinking and communication skills, and meta-cognitive aspects like self-regulation, self-explanation, self-assessment and self-management. The affective states can be: happy, sad, angry, interested, frustrated, bored, distracted, focused, and confused. Some of these emotions led students to an off-task behavior, where students' attention became lost and they engaged in activities that were unrelated to learning. The authors gave a very detailed overview of student modeling approaches, among them the overlay model, the stereotype model, the perturbation model, machine learning techniques for automated observation of students' actions and behavior and for automated induction, cognitive theories, constraint-based models, fuzzy logic modeling techniques, Bayesian networks for dealing with uncertainty of student diagnosis, and ontologies for reused student models. Each of these approaches can be used either alone or in combination with one or more approaches. Chrysafiadi and Virvou presented their findings in two groups: for studies conducted between 2002 and 2008 and for studies conducted between 2009 and 2012. The most common used student modeling techniques in the years 2002 up to 2007 were overlay and stereotype modeling. Probabilistic models in form of Bayesian student models based on Bayesian networks became more popular over the next five years. In addition, researchers started to use hybrid student models, which brought together various features of different techniques of student modeling, for example Bayesian networks with machine learning algorithms. Regarding blended student models, Bayesian networks were most often combined with cognitive theories (66.67 %), constraint-based methods (50 %) and machine learning (25 %). They found that affective student modeling was performed successfully through the use of cognitive theories and/or Bayesian networks, with an increase in the adoption of fuzzy logic techniques and Bayesian networks in the development of student models in order to deal with uncertainty of learning and the student diagnosis process.

A review from Pavlik Jr. et al. (2013) gave an in-depth overview of different types of student models and their implementation. They evaluated the different approaches along various dimensions: quantitative fit, ease of understanding, generality and flexibility, cost of creation, granularity, time scale, and learning gains in practice.

Truong (2016) reviewed 51 studies that dealt with the application and integration of learning styles theories in adaptive e-learning systems. Recent surveys suggested that learning styles models were the most useful frameworks for adaptive systems development next to previous knowledge and student background. The author gives an overview of learning style theories and common predictors of student's learning styles. Potential sources of data are log files that track users' actions and interactions with the system's interface and users' history and background data that include static information such as gender, education majors and ethnicity and culture. In one example a study was able to detect learning styles by using attributes related to performance assessment such as time spent for certain type of questions, performance on the test, and time taken to check the questions. Regarding learning styles classification algorithms, the author has found that again Bayesian networks are among the most common approaches (second most popular method) next to rules-based methods (most popular). One study in the review used an educational serious game as target for the application of learning styles in developing adaptive learning systems. Feldman et al. (2014) provided empirical evidence that learning styles can be measured through students' behaviors when they were playing games.

Another review on student modeling approaches in ITS was done by Kurup et al. (2016). They presented yet another technique called Bayesian Knowledge Tracing (BKT), which is a type of student model used in adaptive tutoring that infers a student's knowledge from previous responses of a student. BKT uses four probability factors to calculate the student's mastery of a skill based on their performance history: the probability that a student is well versed in that skill, the probability that a student who has not learned the skill yet will know it after the next exercise, the probability that a student who has not learned the skill yet will answer a question correctly, and the probability that a student will answer a question incorrectly albeit they have the right knowledge. The rest of the paper deals with the challenges in estimating the parameters for the BKT model.

Because Bayesian methods in the form of Bayesian networks have attracted a lot of attention, Millán et al. (2010) dedicated a whole article to an introduction for education practitioners about the basic concepts and techniques in the context of typical student modeling problems. Bayesian networks can be used to implement all of the standard types of student models like overlay models, differential models, perturbation models and so on. The authors start with the definition of a Bayesian network as a directed acyclic graph (DAG) of random variables and the probabilistic relationships between the variables. If all joint probability distributions are known, any kind of inference can be performed. The next step in applying Bayesian networks is to compute the inference in the Bayesian framework, which results in a posterior probability distribution for each variable. Bayesian networks allow for two kinds of reasoning: diagnostic and predictive. Diagnosis is the task of identifying the most likely causes given a set of observations. Prediction, instead, tries to identify the most likely event occurrence given a set of observations. Another particular powerful aspect of Bayesian networks is that any variable can be either a source of information, if it can be observed, or object of inference based on other variables in the network. The authors explain how to derive a Bayesian model and how to define its variables, the connections between them, the structure of the model, and finally the model's parameters. Commonly used variables that represent user features include knowledge, cognitive features such as learning styles, cognitive and meta-cognitive skills, and affective attributes such as self-image, motivation, and emotional state. Commonly used evidence variables that represent all directly observable features of student's behavior include answers to questions, measurable traits of conscious behavior such as time elapsed, hints requested, and measurable variables of unconscious response such as eye movement and physiological data. The authors present a range of standard model structures that can be used to combine the model variables, among them prerequisite relationships, refinement relationships and granularity relationships. If time plays an important role in the model, that is when the state of variables change over time, then dynamic Bayesian networks (DBNs) are the tool of choice. In DBN time is discrete and a separate Bayesian network is constructed for each step. The last step that is explained by the authors is learning from data to determine which links between variables should be considered and which should be removed. This can be done using fully automatic methods or by introducing structural constraints and only learnthe reminder of the structure along with the parameters. Recent advances in the field of Bayesian networks are presented in Marcot and Penman (2018).

One classical example of using dynamic Bayesian networks to model the causes and effects of emotional reactions was given by Conati and Maclaren (2009). Their diagnostic model implemented the OCC theory as theoretical model of affect, which accounts for how emotions are caused by the user's appraisal in a given context in terms of the user's goals and preferences. Their appraisal sub-network represented 6 of the 22 emotions defined in the OCC model: the pairs joy and distress, pride and shame, and admiration and reproach. As a test-bed for the model the authors used Prime Climb, an educational game designed by the EGEMS group at the University of British Columbia to help 6th and 7th grade students practice number factorization. Data of actual users' emotions during real-time interactions was collected directly from students during the interaction. An emotion-report dialog box was permanently present on the side of the Prime Climb game window that gave students the possibility to self-report on their emotional states. As a result, their predictive model of user affect based on the OCC appraisal theory of emotions could achieve reasonable accuracy on three of the four emotions tested in their study.

hierarchical Bayesian models are close to Bayesian networks in that they are also a probabilistic framework and that their graphical structure is a directed acyclic graph (Murphy, 2001). The difference lies in how they are used: Bayesian networks model dependencies of categorical variables while HBMs are rather dedicated to parameter estimation with a dedication to Bayesian statistics. Bayesian networks do not necessarily imply a commitment to Bayesian statistics. Indeed, often frequentists' methods are used to estimate the parameters of the conditional probability distributions. Bayesian networks are a useful representation for HBM, which form the foundation of applied Bayesian statistics. To the best knowledge of the author there are no studies published that use HBMs to realize student models. Therefore, building and training a HBM to realize a CogIUM to represent the learner's current cognitive state would be a contribution to the field.

3.2 Model Evaluation and Model Comparison

Naturally, what interests us most in a model is how accurate the model can predict new data. This becomes more important when there are several competing models that explain the observed data. The challenge is to evaluate a model's predictive accuracy given only the data used to fit the model, and correcting for the bias inherent in doing so. We are interested in a model's predictive accuracy for two reasons: first, to measure the performance of a model; secondly, to compare models with each other. Our goal in model comparison is not necessarily to pick the model with the lowest estimated prediction error or to average over candidate models, but at least to put different models on a common scale. This is important because models can use completely different parameterizations but predict the same measurements. (Gelman et al., 2014)

There are two typical scenarios for model comparison. First, when a model is expanded, it is natural to compare the smaller to the larger model and assess the benefits gained by expanding the model. The question then is how much complexity is necessary to fit the data. Secondly, to compare not related models with each other. In such a setting one is usually not interested in choosing one model over the other. But it can be useful to be able to compare the fit of the different models to see how the models perform when considered alone. Later, they might be combined in a larger model as special cases. (Gelman et al., 2014)

3.2.1 Posterior predictive check

The posterior predictive distribution is a probability distribution over possible values of future data $\tilde{y}$. To obtain samples from this distribution, we iterate the following (Lambert, 2018):

1. Sample $\theta^s \sim p(\theta|\mathbf{y})$, that is, sample a parameter value from the posterior distribution.
2. Sample $\tilde{y}_i \sim p(\tilde{y}|\theta^s)$, that is, sample a data value from the sampling distribution conditional on the parameter value from the previous step.

This two-stage process reflects the two sources of uncertainty that we have: the uncertainty in the parameter value θ^s from the posterior distribution $p_(\theta|\mathbf{y})$ and the uncertainty due to sampling variation from the sampling distribution $p(\tilde{y}|\theta^s)$. If the above mentioned steps are repeated a sufficient number of times, then the resulting distribution of the sampled data approaches the shape of the true posterior predictive distribution. Sampling is normally required because the high-dimensional integrals involved are often intractable.

We can use the posterior predictive distribution to discuss the quality and goodness of fit of our model. But what is meant by a *good* model? Most certainly, our model will not be unconditionally good in any aspect, but good at reproducing some aspects of the real world, and worse at others. The model was built with a particular purpose in mind and a good model is one that can account for the variation in data that is in line with this purpose. The posterior predictive distribution is used in Bayesian data analysis to test whether the model can replicate the observed patterns in the data. If the data comes from an experiment and represents subjective behavior, then the model should be able to replicate the behaviors that are most important. Specifically, the posterior predictive distribution is used to generate simulated data, which then is compared to the real observed data. These comparative tests are what constitute posterior predictive checks. The samples drawn from the posterior predictive distribution are interpreted as data samples that might be collected in the future. A graphical visualization is a great way to test te performance of a model, but may become cumbersome if many of these tests across a large number of replicates have to be conducted. A graphical visualization can only be one step in the process of model evaluation. (Lambert, 2018)

However, the replicated data y^{rep} can be used for the calculation of Bayesian p values. Let y be the observed data and θ be the vector of parameters. y^{rep} denotes the replicated data that could have been observed, wheres $\tilde{y}$ is any future observable value or vector of observable quantities. The posterior predictive distribution of y^{rep}, given our current state of knowledge, is defined as

$$p(y^{\text{rep}}|y) = \int p(y^{\text{rep}}|\theta) \cdot p(\theta|y)\, d\theta. \tag{3.1}$$

Discrepancies between the model and data can be measured by defining *test quantities*. A test quantity, or discrepancy measure, $T(y,\theta)$ is a scalar summary of parameters an data that is used as a standard when comparing data to predictive simulations. For example, the lack of fit of the data with respect to the posterior predictive distribution can be measured by tail-area probability, or p-value, of the test quantity. The Bayesian p-value is defined as the probability that the replicated data could be more extreme than the observed data, as measured by the test quantity (Gelman et al., 2014):

$$p_B = \Pr(T(y^{\text{rep}},\theta) > T(y,\theta)|y). \tag{3.2}$$

Further details about Bayesian model can be found in Gelman et al. (2014) and Lambert (2018)

3.2.2 RMSE and MAE

The most common scoring function in literature on prediction is the the squared error, which is an example of a point prediction, where a single value is reported as a prediction of the unknown future observation

(Gelman et al., 2014; Pelánek, 2015). The simplest version of the mean squared error (MSE) is given by (Gelman et al., 2014, p. 167)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - E(y_i|\theta))^2, \tag{3.3}$$

which calculates the squared error between the observed data points y_i and the predictions of the model $(y_i|\theta)$. The root mean squared error (RMSE) is simply the square root of the MSE and is often used to obtain an error that has the same scale as the predicted variable y. The MSE has the advantage of being easily computed and directly interpretable, but the disadvantage of being less appropriate for models that are far from the normal distribution (Gelman et al., 2014). The mean absolute error (MAE) is given by

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - (y_i|\theta)|, \tag{3.4}$$

but is not proper score, as mentioned by Pelánek (2015).

The point predictions do not use the full uncertainty over the unobserved data $\tilde{y}$. *Probabilistic prediction* aims at reporting inferences about $\tilde{y}$ that take the full uncertainty over $\tilde{y}$ into account. Measures of predictive accuracy for probabilistic prediction are called scoring rules. Scoring rules are often quadratic, logarithmic, or zero-one scores. (Gelman et al., 2014)

The log predictive density or log-likelihood is a local and proper scoring rule and is a standard for evaluating probabilistic predictions.

3.2.3 WAIC and LOO

The derivation of the equations presented in this section as well as the notation is based on Gelman et al. (2014) and Vehtari et al. (2016). The logarithmic score for predictions is the log predictive density, also called the log-likelihood, $log p(y|\theta)$. The log predictive density plays an important role in model comparison because of its connection to the Kullback-Leibler divergence (KL). In the limit of large sample size, the model with the lowest KL has both the highest expected log predictive density and the highest posterior probability. Therefore, the log predictive density seems to be the right measure for estimating the predictive accuracy and overall model fit.

What would be the ideal measure of a model's fit? This would be its out-of-sample predictive performance for new data produced from the true data-generating process, which we all external validation. Consider a true model f, observed data y, which is a single realization of the data set y from the distribution $f(y)$, and $\tilde{y}$ as future data or alternative data sets. The out-of-sample predictive fit for a new data point $\tilde{y}_i$ using the logarithmic score is then defined by,

$$\log p_{\text{post}}(\tilde{y}_i) = \log E_{\text{post}}(p(\tilde{y}_i|\theta)) = \log \int p(\tilde{y}|\theta) \cdot p_{\text{post}}(\theta)\, d\theta. \tag{3.5}$$

The term $p_{\text{post}}(\tilde{y}_i)$ in equation (3.5) is the predictive density for $\tilde{y}_i$ induced by the posterior distribution $p_{\text{post}}(\theta) = p(\theta|y)$.

However, as the future data $\tilde{y}_i$ are themselves unknown, we work with the expected out-of-sample log predictive density

$$\begin{aligned} \text{elpd} &= \text{expected log predictive density for a new data point} && (3.6) \\ &= E_f\left(\log p_{\text{post}}(\tilde{y}_i)\right) = \int (\log p_{\text{post}}(\tilde{y}_i)) \cdot f(\tilde{y}_i)\, d\tilde{y}. && (3.7) \end{aligned}$$

The data distribution f is in general unknown, whereas we always would have some p_{post}. In practice, the true parameter θ is also not known, so we cannot know the log predictive density $\log p(y|\theta)$. We would like to work with the posterior distribution p_{post}, and summarize the predictive accuracy of the fitted model to data by

$$\begin{aligned} \text{lppd} &= \text{log pointwise predictive density} && (3.8) \\ &= \log \prod_{i=1}^{n} p_{\text{post}}(y_i) = \sum_{i=1}^{n} \log \int p(y_i|\theta) \cdot p_{\text{post}}(\theta)\, d\theta. && (3.9) \end{aligned}$$

How can this predictive density be computed in practice? Once we have a posterior distribution, we can evaluate the expectation using draws from $p_{\text{post}}(\theta)$, the usual posterior simulations, which we label $\theta^s, s = 1, \dots, S$:

$$\text{computed lppd} = \sum_{i=1}^{n} \log\left(\frac{1}{S}\sum_{s=1}^{S} p(y_i|\theta^s)\right). \qquad (3.10)$$

The computed log pointwise predictive density of equation (3.10) is only an approximation of the true value from equation (3.8), but we usually assume that the number of simulation draws S is large enough to fully capture the true posterior distribution. However, equation (3.10) is an overestimate of (3.6) and different criteria try to correct for this bias to get reasonable estimates.

For historical reasons, measures of predictive accuracy are referred to as *information criteria* and are typically defined based on the deviance, the log predictive density of the data given a point estimate of the fitted model and multiplied by -2: $-2\log p(y|\hat{\theta})$. When using point estimates $\hat{\theta}$ and the posterior distribution p_{post} that was fitted to the data y, the out-of-sample predictions will typical be less accurate than implied by the within-sample predictive accuracy, that is, we overestimate the accuracy by relying only on the fitted data.

In addition, when different models are compared that differ in size or effective size, it is important to male some adjustment for the natural ability of a more complex and flexible model to fit data better, even if only by chance.

Watanabe-Akaike or widely available information criterion

We have seen that the within-sample predictive accuracy for existing is a naive estimate of the expected log predictive density for new data. We would like to work with the computed lppd (3.10), but, in general, overestimate (3.6). The logical next step is to adjust this estimate and correct the bias. Formulas such as AIC, DIC, and WAIC give approximately unbiased estimates of elpd by correcting the computed lppd for the numer of parameters, or the effective number of parameters, being fit. Another approach uses cross-validation, where the model is fitted to training data and its predictive accuracy is evaluated on a holdout set. Leave-one-out cross-validation (LOO-CV) requires n fits except when computational shortcuts can be used to approximate the computations.

I will focus on the criteria WAIC and LOO-CV, because these are the criteria used in the model comparison of chapter 5 Implementation (page 103).

Watanabe-Akaike or widely available information criterion (WAIC) is a fully Bayesian approach for estimating the out-of-sample expectation elpd (3.6), starting with the computed lppd (3.10) and then adding a correction for the effective number of parameters to adjust for overfitting.

There are two corrections proposed in the literature. Both are based on pointwise calculation and can be viewed as approximations to cross-validation. The first approach uses a difference

$$p_{\text{WAIC1}} = 2\sum_{u=1}^{n}\left(\log\left(E_{\text{post}}p(y_i|\theta)\right) - E_{\text{pots}}\left(\log p(y_i|\theta)\right)\right), \tag{3.11}$$

which can be computed from simulations by replacing the expectation by averages over the S posterior samples θ^s:

$$\text{computed } p_{\text{WAIC1}} = 2\sum_{u=1}^{n}\left(\log\left(\frac{1}{S}\sum_{s=1}^{S}p(y_i|\theta^s)\right) - \frac{1}{S}\sum_{s=1}^{S}\log p(y_i|\theta^s)\right). \tag{3.12}$$

The second measure uses the variance of individual terms in the log predictive density summed over the n data points:

$$p_{\text{WAIC2}} = \sum_{u=1}^{n}\text{var}_{\text{post}}\left(\log p(y_i|theta)\right). \tag{3.13}$$

Again, to calculate (3.13) we compute the posterior variance of the log predictive density for each data point y_i, that is, $V_{s=1}^{S}\log p(y_i|\theta^s)$, where $V_{s=1}^{S}$ represents the sample variance. Summing over all the data points y_i gives the effective number of parameters:

$$\text{computed } p_{\text{WAIC2}} = 2\sum_{u=1}^{n}V_{s=1}^{S}\left(\log p(y_i|\theta^s)\right). \tag{3.14}$$

Either p_{WAIC1} or p_{WAIC2} can be used to correct the bias:

$$\widehat{\text{elppd}}_{\text{WAIC}} = \text{lppd} - p_{\text{WAIC}}. \tag{3.15}$$

Gelman et al. recommend p_{WAIC2} because it is asymptotically closer to LOO-CV, which seems also to be the case for practical applications. The value reported for WAIC is usually -2 times the expression (3.15) so as to be on the deviance scale.

Compared to other information criteria like AIC and DIC, WAIC hs the desirable property of averaging over the posterior distribution rather than conditioning on a point estimate. WAIC works also with singular models and thus is especially helpful for models with hierarchical and mixture structures in which the number of parameters increases with the sample size.

Leave-one-out cross-validation

In Bayesian cross-validation, the data are repeatedly partitioned into a training set y_{train} and a holdout set y_{holdout}. The model is then fit to y_{train}, yielding a posterior distribution $p_{\text{train}}(\theta)$. Afterwards, the model is evaluated using an estimate of the log predictive density of the holdout data,

$$\log p_{\text{train}}(y_{\text{holdout}}) = \log \int p_{\text{pred}}(y_{\text{holdout}}|\theta) \cdot p_{\text{train}}(\theta)\, d\theta. \tag{3.16}$$

As usual, we can approximate the posterior distribution $p_{\text{train}}(\theta) = p(\theta|y_{\text{train}})$ by S simulation draws θ^s and calculate the log predictive density as $\log\left(\frac{1}{S}\sum_{s=1}^{S} p(y_{\text{holdout}}|\theta^s)\right)$. Leave-one-out cross-validation (LOO-CV) is the special case with n partitions in which each holdout set represents a single data points y_i. The analysis for each of the n data points yields n different inferences $p_{\text{post}(-i)}$, each summarized by S posterior simulations, θ^{is}.

The Bayesian LOO-CV estimate of out-of-sample predictive fit is

$$\text{lppd}_{\text{loo-cv}} = \sum_{i=1}^{n} \log p_{\text{post}(-i)}(y_i), \text{ calculated as } \sum_{i=1}^{n} \log\left(\frac{1}{S}\sum_{s=1}^{S} p(y_i|\theta^{is})\right). \tag{3.17}$$

The effective number of parameters can be computed as

$$p_{\text{loo-cv}} = \text{lppd} - \text{lppd}_{\text{loo-cv}}. \tag{3.18}$$

Cross-validation is like WAIC in that it requires data to be divided into disjoint, ideally conditionally independent, pieces. Regarding computational efficiency, cross-validation is the most expensive approach except in settings where shortcuts are available to approximate the distributions $p_{\text{post}(-i)}$ without having to re-fit the model each time. If no shortcuts are available, a common practice is to use k-fold cross-validation where data is partitioned in k sets. A common choice for k is 10. There exist different extensions of the

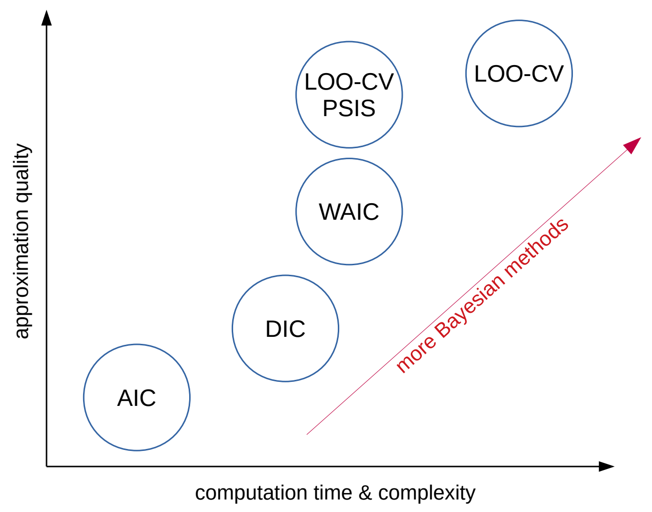


Figure 3.1.: Different measures to evaluate the out-of-sample predictive accuracy of a model as a function of computational time and complexity and approximation quality (based on Lambert, 2018, p. 235). The more Bayesian a method, the higher the approximation quality, but for the cost of a higher complexity and computation time.

basic LOO-CV that improve the efficiency of the computation, like using Pareto-smoothed importance sampling (PSIS) (Vehtari et al., 2016).

Predictive accuracy measures are useful in parallel with posterior predictive checks to see if there are important deviations of the model's predictions from the observed data. According to Lambert WAIC and LOO-CV are the best choices for predictive accuracy measures, as they are fully Bayesian (Figure 3.1). For both programming languages Python and R there exist packages that allow the computation of estimates for LOO-CV and WAIC using existing simulation draws from the posterior.

3.2.4 Bayes factor

The ratio of marginal likelihoods for two models is known as the Bayes factor, and can also be thought of as the ratio of posterior to prior odds (Shiffrin et al., 2008). The Bayes factor is part of Bayes' rule to decide between two models H_1 and H_2, given the observed data $\mathscr{D}$ (Gelman et al., 2014)

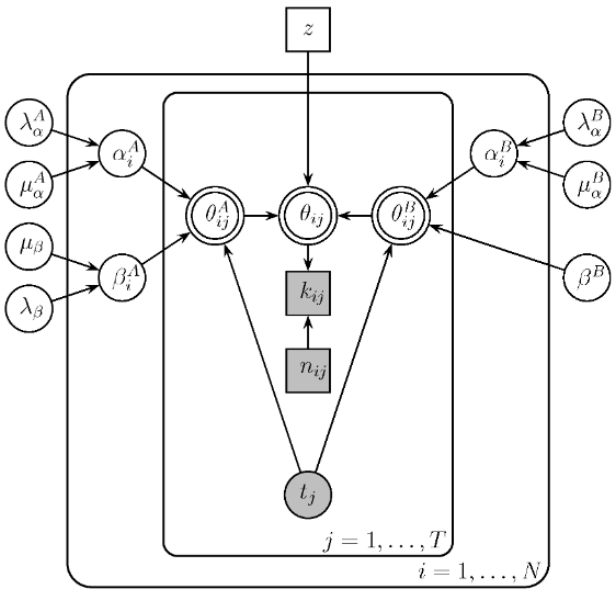
$$\frac{p(H_2|\mathscr{D})}{p(H_1|\mathscr{D})} = \underbrace{\frac{p(y|H_2)}{p(y|H_1)}}_{=:\text{ Bayes factor}} \times \frac{p(H_2)}{p(H_1)} = \frac{\int p(y|\theta_2, H_2)p(\theta_2|H_2)\,d\theta}{\int p(y|\theta_1, H_1)p(\theta_1|H_1)\,d\theta} \times \frac{p(H_2)}{p(H_1)}, \tag{3.19}$$

where the left-hand side is the ratio of the posterior probabilities for each of the models. To compute (3.19) we must calculate a term known as *Bayes factor* and the ratio of the prior preferences for each of the models.

The main problem with calculating the Bayes factor is to compute the marginal likelihood of denominator for each model, $p(\text{data}|\text{model}_i)$. There exist methods to calculate marginal likelihoods, such as Annealed Importance Sampling (Neal, 1998). Another concern is the sensitivity of the Bayes factor to the particular choice of prior distribution (Lambert, 2018). (3.19) is only defined when the marginal density of y under each model is proper. The goal in using the Bayes factor is to decide for a single model H_i or average over a discrete set using their posterior probabilities $p(H_i|y)$.

3.2.5 Hierarchical models

All the previous mentioned methods have important limitations in their ability to address the basic goals of modeling (Shiffrin et al., 2008). Predictive accuracy measures will usually give some indication of likely parameter values, and give a basis for inferring which model will predict future data better. However, they do not provide a reasoning of how and why the models succeed and fail to various degrees and do not help to drive subsequent theorizing. Shiffrin et al. suggest that hierarchical methods, and HBMs in particular, are a better wa to approach model development and evaluation in the cognitive sciences. They demonstrate the power and flexibility of HBMs for evaluating models using two worked examples. They first develop different models that are build on each other and finally combine the two most promising models in one model (Figure 3.2). The model entails the full hierarchical model as well as the simplified version and uses a latent binary variable z to choose between the models. The posterior sampling of z, counting the proportions of times it chose model one over model two, then amounts to an evaluation of the relative usefulness of each model. Shiffrin et al. (2008, p. 1281) conclude that hierarchical Bayesian methods offer "very general and powerful capabilities for developing, evaluating, and choosing between models of cognition."



$$
\begin{aligned}
z &\sim \text{Bernoulli}(1/2) \\
\mu_\alpha^A &\sim \text{Uniform}(0,1) \\
\lambda_\alpha^A &\sim \text{Gamma}(.001, .001) \\
\mu_\beta &\sim \text{Uniform}(0,1) \\
\lambda_\beta &\sim \text{Gamma}(.001, .001) \\
\alpha_i^A &\sim \text{Gaussian}(\mu_\alpha^A, \lambda_\alpha^A) && 0 < \alpha_i^A < 1 \\
\beta_i^A &\sim \text{Gaussian}(\mu_\beta, \lambda_\beta) && 0 < \beta_i^A < 1 \\
\theta_{ij}^A &= \exp\left(-\alpha_i^A t_j\right) + \beta_i^A && 0 < \theta_{ij}^A < 1 \\
\mu_\alpha^B &\sim \text{Uniform}(0,1) \\
\lambda_\alpha^B &\sim \text{Gamma}(.001, .001) \\
\beta^B &\sim \text{Uniform}(0,1) \\
\alpha_i^B &\sim \text{Gaussian}(\mu_\alpha^B, \lambda_\alpha^B) && 0 < \alpha_i^B < 1 \\
\theta_{ij}^B &= \exp\left(-\alpha_i^B t_j\right) + \beta_i^B && 0 < \theta_{ij}^B < 1 \\
\theta_{ij} &= \begin{cases} \theta_{ij}^A & \text{if } z = 0 \\ \theta_{ij}^B & \text{if } z = 1 \end{cases} \\
k_{ij} &\sim \text{Binomial}(\theta_{ij}, n_{ij})
\end{aligned}
$$

Figure 3.2.: Graphical model for comparing a full hierarchical model of retention (on the left) to a simpler version that assumes no individual differences in the β parameter (on the right) (from Shiffrin et al., 2008, p. 1272). The model uses a latent model indicator variable z to move between the models. The final posterior distribution over z can be directly interpreted as a Bayes factor.

3.3 Measuring Cognitive Load

Cognitive load as described by the Cognitive Load Theory (CLT) is a theoretical construct that describes internal processes of information processing that cannot be observed directly. However, the usage of cognitive load to inform the design of instructions and its usage as latent variable in models to predict student's performance and cognitive states requires valid and reliable instruments for assessing cognitive load.

Brünken et al. (2003) classify existing instruments along two dimensions: objectivity and causal relation. Objectivity describes whether the instrument uses subjective, self-reported data or objective observations of behavior, psychological conditions, or performance. The causal relation dimension descries whether the relationship between the phenomenon observed by the instrument and the actual attribute of interest is a direct or indirect relationship. Thus there are four areas in this two dimensional space of instruments. The area of subjective and indirect measures include self-reported invested mental effort. The area of objective and indirect measures include physiological measures, behavioral measures and learning outcome measures. The area of subjective and direct measures include self-reported stress levels and self-reported difficulty of materials. The last area of objective and direct measures include brain activity measures and dual-task performance. The authors reported that indirect, objective measures were the most common methods of investigating cognitive load effects. These are performance outcome measures and knowledge acquisition scores. Typical studies in this area compare two or more different variants of multimedia instructions of the same material, that are assumed equal with regard to their ICL, but differ with respect to ECL. It is hypothesized that learners in the condition with less ECL have higher knowledge acquisition scores. Other indirect, objective measures include time-on-task, navigation behavior, navigation errors, and orientation problems such as lost-in-hyperspace. The authors presented experimental evidence the dual-task approach as a direct and objective measure of cognitive load in multimedia learning. They used a visual secondary reaction time task for which, in accordance with the CLT, reaction times were significantly faster for th audiovisual primary task than for the visual-only primary task condition.

In the dual-task paradigm a learner is required to perform two tasks simultaneously. The assumption is that the secondary task shares working memory resources with the primary task and thus, as the primary tasks becomes more loading, performance in the secondary task drops. Dual-task measures can be conducted either as measuring accuracy and response times in an observational task that needs to be carried out during the primary task or as performance measures in a secondary task during learning. Dual-tasks measures have the advantage of being objective, but the risk of disturbing the learning process by themselves (Klepsch et al., 2017).

DeLeeuw and Mayer (2008) analyzed the sensitivity of three commonly used techniques for measuring cognitive load: response time to a secondary task during learning, effort ratings during learning, and difficulty ratings after learning. They used a secondary visual monitoring task in which learners were asked to detect a periodic color change and conducted two experiments. They found that each of the three measures was sensitive mainly to one aspect of cognitive load. Response times were most sensitive to ECL, effort ratings during learning were most sensitive to ICL, and difficulty ratings after learning were most sensitive to GCL. These findings also provide evidence for the triarchic theory of cognitive load.

The most popular scale for measuring total cognitive load is a rating scale developed by Paas (1992). The scale consists of one item with a 9-point Likert scale, ranging from very, very low mental effort (1) to very, very high mental effort (9). Typical item wordings are “I invested ... mental effort” or “my invested mental effort was ....” A problem with that scale is that it does not allow to differentiate between true variance and measurement error (Klepsch et al., 2017).

Rubio et al. (2004) evaluated three multidimensional subjective workload assessment instruments: NASA Task Load Index (TLX), the Subjective Workload Assessment Technique (SWAT), and the Workload Profile. The NASA TLX by Hart and Staveland (1988) uses six dimensions to assess mental workload: mental demand, physical demand, temporal demand, performance, effort, and frustration and 20-step bipoplar scales to obtain ratings for these dimensions. A score from 0 to 100 is obtained on each scale. A weighting procedure is used to combine the six individual scale ratings into a global score. A paired comparison task is required to be performed prior to the workload assessment. The SWAT by Reid and Nygren (1988) is a subjective rating technique that uses the three dimensions of time load, mental effort load, and psychological stress load to assess workload. The application of SWAT is done in three steps. First, an operator sorts 27 cards with all possible combinations of three levels of each of the three dimensions into the rank order that reflects their perception of increasing workload. Second, an actual rating of workload is performed. Third, each three-dimension rating is converted into numeric scores between 0 and 100 using the interval scale developed in the first step. The Workload Profile by Tsang and Velazquez (1996) asks subjects to provide the proportions of attentional resources used after they had experienced all of the tasks to be rated. The Workload Profile uses the eight workload dimensions perceptual/central processing, response selection and execution, spatial, processing, verbal processing, visual processing, auditory processing, manual output, and speech output. Subjects write in each cell on the rating sheet a number between 0 and 1 to represent the proportion of attentional resources they belief was used in a dimension for a given task. Rubio et al. found that WP was the instrument that had the highest sensitivity. NASA-TLX and SWAT had similar sensitivities, with NASA-TLX being slightly more sensitive than SWAT.

Charles and Nixon, 2019 give an overview of 58 studies that used physiological measures to measure and predict mental workload. The included physiological measures are electrocardiographic, respiratory, dermal, blood pressure, and ocular.

Leppink et al. (2013) developed a new instrument to measure all three components ICL, ECL, and GCL of the CLT in complex knowledge domains. Their questionnaire consists of 10 items, which were tested in the domain of statistics. The questionnaire includes three items on ICL, three items on ECL, and four items on GCL. However, the could not find a positive correlation between items that are supposed to measure GCL and learning outcomes.

Based on the previous attempts, Klepsch et al. (2017) developed and evaluated two self-report measures that treat the different aspects of cognitive load in different ways. The questionnaire that measures ECL and ICL evaluate the inherent complexity and the design of the learning material as it was perceived by the learner. Their revised questionnaire from the second study showed that all three scales for ICL, ECL, and GCL had satisfying reliability and validity scores. The developed questionnaire is not specific and only needs adoption based on the learning material. The authors state that their questionnaire is easy to apply and fit to each content, especially for short interventions. For longer interventions and complex learning material, the authors recommend to apply the questionnaire multiple times.

4 Concept

This chapter presents the main conceptual ideas for realizing cognitive intelligent user model both with cognitive architectures and hierarchical Bayesian models. First, a general discussion is held on possible interactions between a learner and an adaptive system that is based on a cognitive user model. Afterwards, interaction patterns for the serious game Lost Earth are analyzed and presented in a systematic way. The next section presents the observable variables that were derived from the interaction patterns. The main contribution of this chapter us to be found in last section, in which the concepts for realizing cognitive user models are presented. It will be explained why the use of Soar was discontinued and why HBMs were a better tool for the purposes of this thesis.

4.1 Interaction Between Learner and Cognitive User Model

What is the role of a cognitive user model? The learner model in the four-process adaptive cycle of (Shute and Zapata-Rivera, 2012) builds a connection between the captured user data and the presented learning material that is suitable for the learner. The learner model should allow for a dynamic assessment of the learner's current state. A cognitive user or learner model is a model that can make statements about the learner's cognitive state, that is, statements about their mental actions and processes that deal with knowledge acquisition and understanding. In this section, I will describe the interaction between the user who plays a serious game and the adaptive system that uses a cognitive user model which tries to infer the user's current cognitive state. For the remainder of this thesis, I will use the acronym CogIUM to refer to the learner model or cognitive user model, respectively. Please be aware that CogIUM is also the name of the Python package developed for this thesis. Here, CogIUM only refers to a general and abstract cognitive user model and not to a particular implementation. In addition, I will use the words player, learner, and user interchangeably, because in the context of educational serious games and e-learning, a user of such a system is always a player and, at the same time, a learner.

One general purpose of a CogIUM is to derive the right time for the system to intervene or react to the user with adaptive measures (Figure 4.1 on page 71). Given any educational serious game, the player is always confronted with a certain kind of task or challenge they have to complete. At any given point in time the player is in a certain state determined by the player's current state and the state of the game. From their current state they can choose between several actions that are offered by the game to advance the current game level. Some of these actions lead to states that are closer to a desired goal state, normally the end of the current game level, while others will turn out to be dead ends or, worse, might lead to a failure of the current task. Another possibility is that certain actions will lead to undesirable emotional states for the player like an ongoing experience of frustration or boredom. The CogIUM can be thought of as a mirror image of the learner's situation. The model knows about the learner's current position in the state space, it knows about the learner's possible actions and where these actions will lead them and it knows about the learner's current cognitive state. The cognitive state of the learner is represented in the CogIUM by a set of cognitive variables. The model's knowledge about the learner is updated iteratively with every user interaction. The model's knowledge about the learner can be leveraged to guide the player through the problem space towards a goal state while avoiding states that are detrimental for the player.

There are several possibilities how an adaptive system based on a CogIUM can react to user interactions:

Monitoring The model can register the interaction and learn about how the interaction is correlated with changes in the cognitive variables. Did the interaction lead to a change in the affective state? Did the interaction lead to an unfavorable state which could have been avoided by another, better, option? For example the player might have repeatedly tried to execute a certain game functionality but failed due to unfulfilled requirements, which they are not aware of and that increases the player's frustration.

Feedback The system can give feedback before the interaction takes place. This requires the model to infer what the player is intended to do next. In this scenario the model might be able to warn a player to not execute an action the system believes to be imminent and harmful for the player's progress or learning success. For example the player might run out of resources or time and the system can help the player in completing the current task with the remaining resources or time available.

The system can give feedback after the interaction took place. This could be feedback for an interaction that the model thought impossible given the learner's estimated skill level or feedback for an interaction that led to an unfavorable state. Encouraging feedback can motivate the player to go on with the next challenge. Other types of feedback might help the learner to better internalize a concept and to avoid the interaction the next time. For example the player might have failed the mission because they exceeded the permitted time or resources for that mission. In hindsight, the system can give feedback to the player's performance and help identify actions that the player did not take but should have or did take but should have not.

The system can give feedback on player's command. In this case the systems behavior is reactive. When asked for assistance by the player, the model analyzes all knowledge about the learner's current and previous states and tries to find an answer to the player's request with maximum benefit for the player. For example the player might ask how to solve a particular problem or what the next steps are. The system can answer this question with its domain knowledge and knowledge about an ideal or expert solution as well as with its knowledge about the player.

Adaptation The system can adapt game features to meet the player's needs. Based on the previous interactions and the inferred cognitive state of the player, the model might signal that some of the cognitive variables are about to exceed or have already exceeded a threshold. For example the model might infer from the user's interactions that the user's level of attention is too low to proceed with the game. Then the system might suggest a pause or insert a rest phase before the next challenge.

Role Assignment The system can assign the player to a certain role or group and adjust the game play in accordance to the needs of the assigned group. This requires the model to have data about several different users and detect similarities between them. Or the model has a build-in distinction of players. For example the model might have learned to distinct players according to different learning styles. When the model has gathered enough information about a new player to assign them to a learning style, the system can adjust the game play and present the learning material in a way that is most appropriate for learners with this learning style.

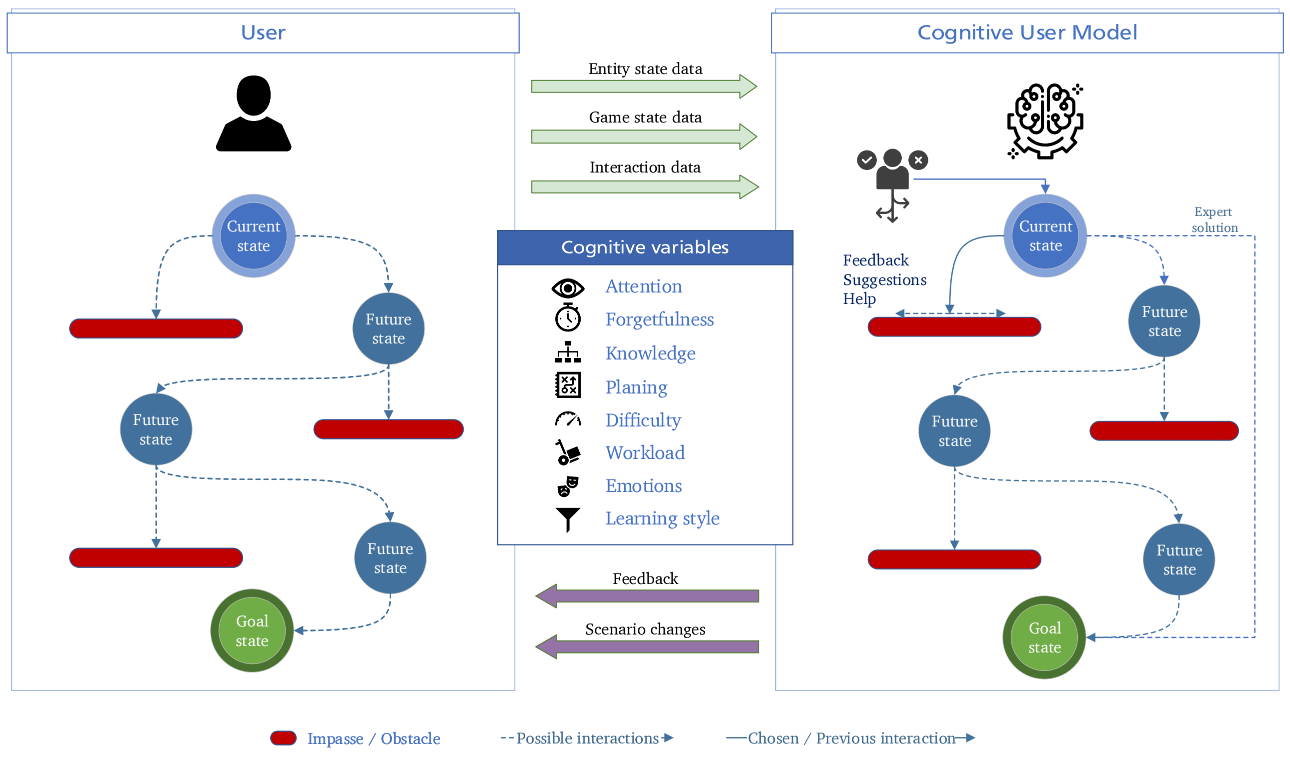


Figure 4.1.: Interaction between a user and an adaptive system that is based on a cognitive user model. Given the user is in a certain state, they have a range of actions to choose from (dashed arrow). Some actions lead to undesirable states (red bar) while others lead closer to the goal state (green node). The CogIUM can be seen as a mirror image of the player's situation. The CogIUM has domain knowledge, knowledge about an ideal or expert solution and knowledge about the user's current cognitive state represented by a set of cognitive variables. The CogIUM has access to entity state data, game state data, interaction data, and the history of user's previous actions. After a user has executed an action (solid arrow), the adaptive system can react to it, for example with feedback, suggestions, or help, and bring the user back on a path that leads to the goal state. Own graphic, based on Sottilare and Gilbert (2011).

This overview of possible interactions between a user and an adaptive system that is based on a cognitive user model provides a framework for the search of possible approaches to realize CogIUMs. It is not to be expected to find an already existing user model or to build a new user model that can infer all listed cognitive variables or that supports all listed interaction types. But it should, in principle, allow for an adaptive system to choose adaptive measures at the right time based on users' interactions. The next section will have a closer look at the educational serious game Lost Earth, which is the digital game the analysis of this work is based on. I will analyze and characterize the interaction patterns necessary for modeling the game with Soar 9 and derive observable variables that are necessary for modeling the game with HBMs.

4.2 Interaction Patterns for Lost Earth

Lost Earth 2307[1] is a serious game based on the principles of game-based learning. The game offers a fictional world and story while integrating learning objectives into the gameplay. In Lost Earth the player follows the character Alex, who is part of a rebel organization in the year 2307. The goal of the game is to free all human colonies from a vicious cult, that took over control after a self-induced disaster on Earth. The main mean of transportation and center of the game is the Ark (Figure 4.2 on the following page), a space ship that travels throughout the galaxy. From here the player can start new missions and begin the liberation of the colonies.

The learning domain of Lost Earth is about image interpretation and supports image interpreters in gaining a basic knowledge in interpreting and analyzing aerial and satellite imagery, in understanding the process of Reconnaissance-Cycle, and the differences of visual, infrared, and radar sensors.

Lost Earths has two types of missions: reconnaissance and deployment. In reconnaissance missions the player has to analyze original imagery according to the problem and task of the mission giver, to annotate and align imagery, to deliver reports, and answer multiple choice questions (Figure 4.5 on page 75). With progress in gameplay the player has to learn about considering the influences of weather and technology. Deployment missions deal with the deployment and control of sensors and illustrate advantages and disadvantages of sensors and platforms. The gameplay is turn-based and uses a genre combination of the 4X-strategy and adventure. 4X stands for explore, expand, exploit and exterminate (Figure 4.3 on page 74).

The game is currently ported from the Havoc game engine to the Unity game engine. Simultaneously, a game engine adapter for the ELAI framework is developed to collect user data as xAPI statements from the game. An early prototype of the port was used as educational serious game for this thesis. Therefore, when I write about Lost Earth, without any number, I refer to the prototype version that was used for the analysis.

Every mission in Lost Earth follows the same scheme: first, acquiring the mission along with a briefing (Figure 4.4 on page 74), second, choosing an appropriate optical sensor, third applying the sensor under consideration of the weather conditions (Figures 4.6 and 4.7 on page 75 and on page 76), and fourth, processing the acquired image data (Figure 4.5 on page 75). Every successful mission ends with a detailed mission report and mission score (Figure 4.8 on page 76). The scenarios focus on this sequence of steps.

I analyzed the basic user interactions relevant for advancing the mission and summarized the core game play in an activity diagram (Figure 4.9 on page 77). The diagram also shows important interactions that might reveal some information about the user's cognitive state. Did the player understand the mission requirements or did they miss something? Did the player read the briefing, was it helpful or rather confusing? Did the learner understand the influence of the weather on mission success and type of sensor? Did they adjust the number of delay rounds according to the information of the weather console? If not, what was missed? When the mission failed, was it due a time limit or due to inappropriate weather

[1] https://www.iosb.fraunhofer.de/servlet/is/58015/

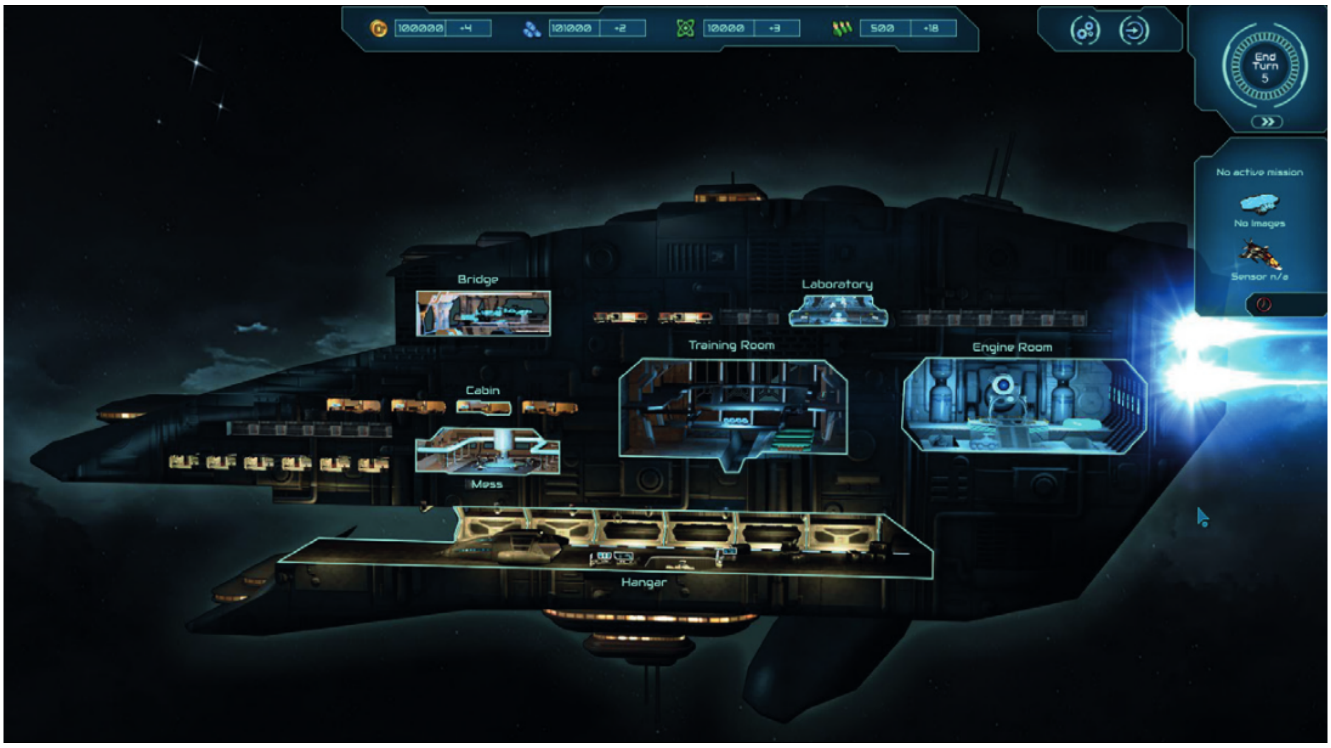
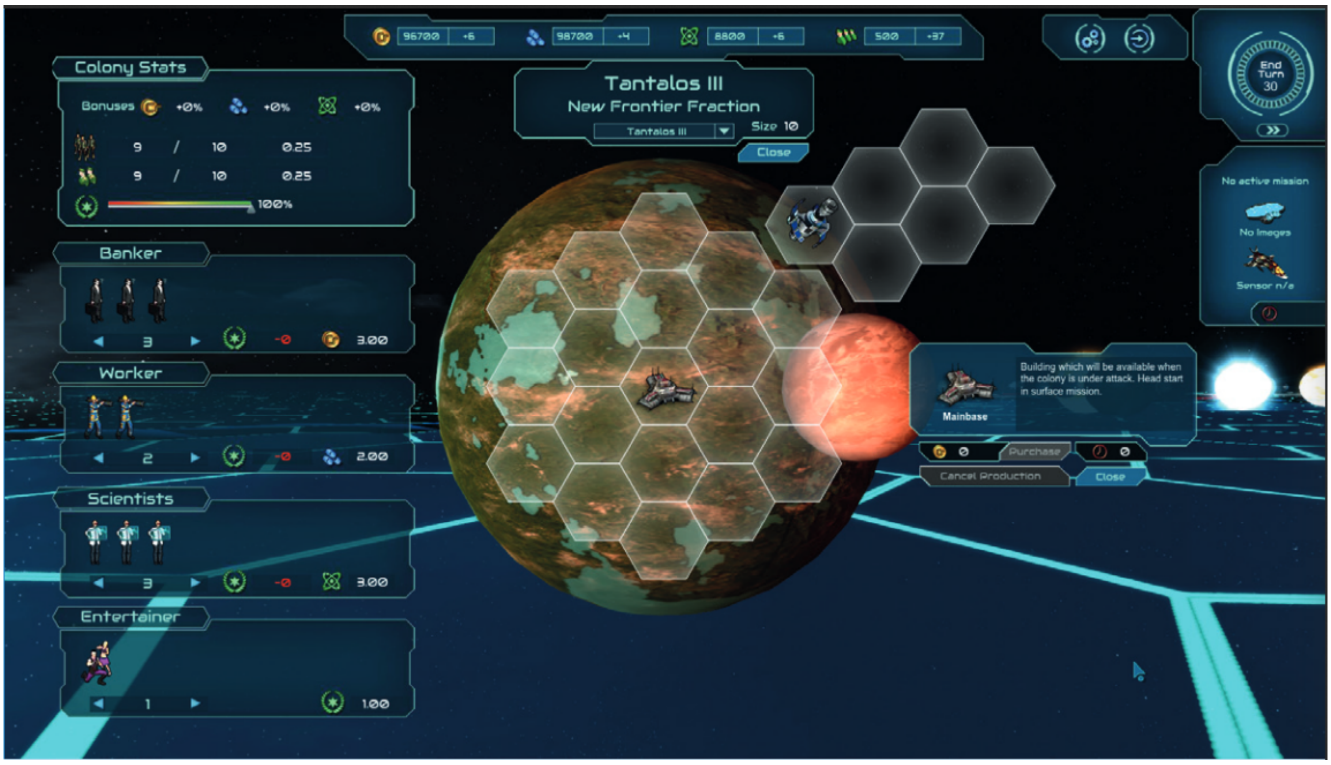


Figure 4.2.: Side face of the Ark. From here the player can visit the different decks of the Ark.

conditions for the selected sensor? Did the learner know how to fill out the report, were they able to correctly use the external application ViLand?

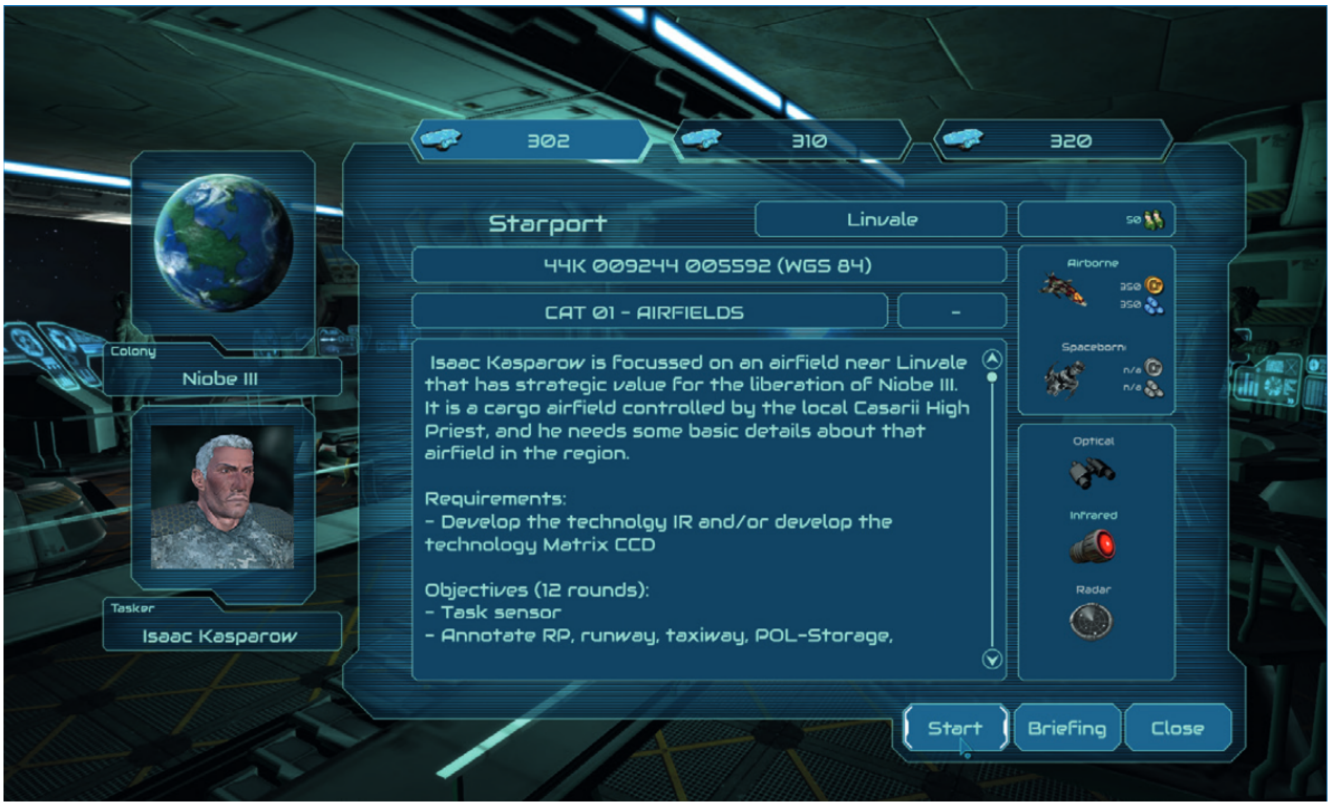
Besides these questions directly coupled with certain user interactions, I assumed that the user's cognitive state directly affects the user interactions. If the experienced difficulty of the mission is too high, a player have more failed attempts. If there is too much information, maybe in addition poorly presented, the player will need more time to process all information and gather the relevant information, which might lead to an increase in the number of opened dialogues. Thus I tried to come up with a set of observable variables that could be reported by xAPI statements. In general, xAPI statements can report any user action that is done by the user, with additional information provided in the context field of the statement (see Section 2.2 on page 43). With the final set of observable variables I tried to focus on generalization rather than specificity. The chosen variables should be as broadly applicable as possible to allow for transferring the work of this thesis to other domains and games. This is the reason for not considering atomic interactions like a single mouse click or key stroke.

The next sections presents the set of observable variables that can be captured by xAPI statements from the game Lost Earth.



Building which will be available when the colony is under attack. Head start in surface mission.

Figure 4.3.: 4X-strategy part of the game. In Lost Earth 2307, the player can explore the galaxy, expand their influence by liberating colonies, exploit and manage resources, and compete against opponents.



44K 009244 005592 (WGS 84)

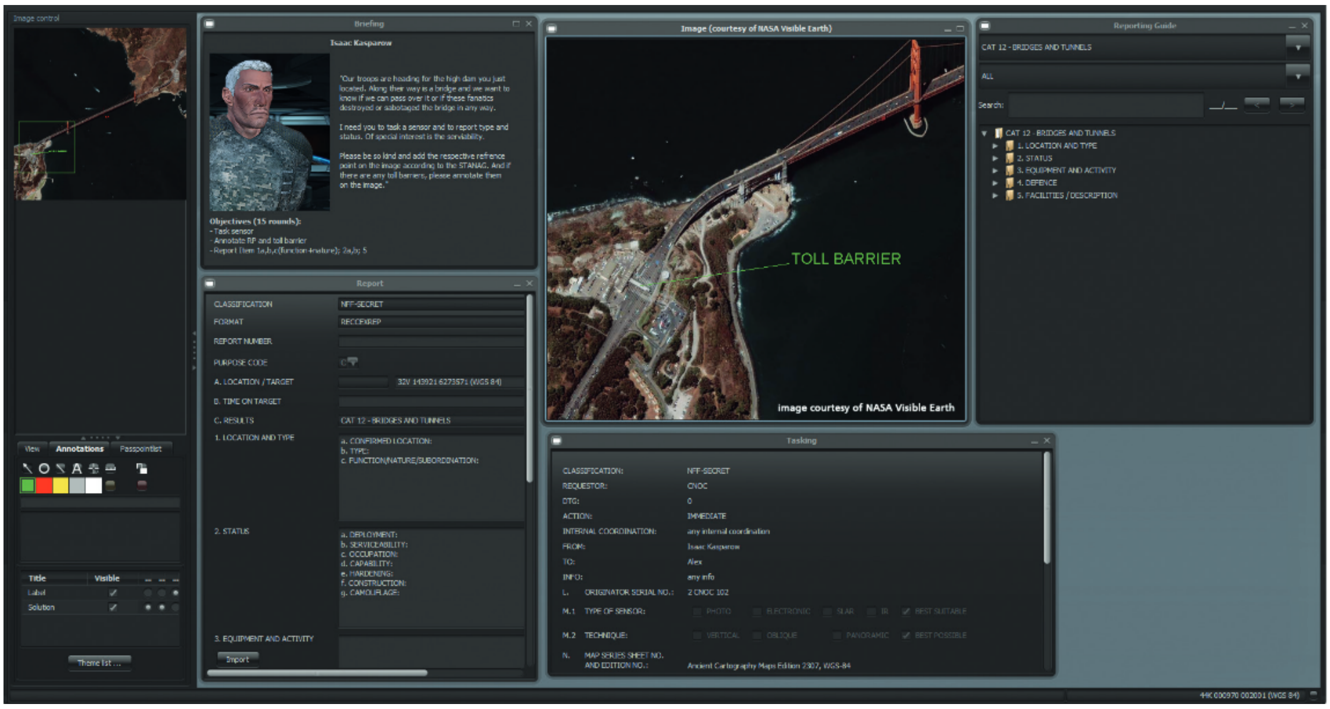
CAT 01 - AIRFIELDS

Isaac Kasparow is focussed on an airfield near Linvale that has strategic value for the liberation of Niobe III. It is a cargo airfield controlled by the local Casarii High Priest, and he needs some basic details about that airfield in the region.

Requirements:
- Develop the technolgy IR and/or develop the technology Matrix CCD

Objectives (12 rounds):
- Task sensor
- Annotate RP, runway, taxiway, POL-Storage,

Figure 4.4.: Mission list panel. List currently available missions with a short description, requirements of resources and constraints. The player can get further instructions by reading the briefing.



Isaac Kasparow

"Our troops are heading for the high dam you just located. Along their way is a bridge and we want to know if we can pass over it or if these fanatics destroyed or sabotaged the bridge in any way.

I need you to task a sensor and to report type and status. Of special interest is the serviability.

Please be so kind and add the respective refrence point on the image according to the STANAG. And if there are any toll barriers, please annotate them on the image."

Objectives (15 rounds):
- Task sensor
- Annotate RP and toll barrier
- Report Item 1a,b,c(function+nature); 2a,b; 5

CLASSIFICATION	NFF-SECRET
FORMAT	RECCEXREP
REPORT NUMBER	
PURPOSE CODE	C
A. LOCATION / TARGET	32V 143921 6273571 (WGS 84)
B. TIME ON TARGET	
C. RESULTS	CAT 12 - BRIDGES AND TUNNELS
1. LOCATION AND TYPE	a. CONFIRMED LOCATION: b. TYPE: c. FUNCTION/NATURE/SUBORDINATION:
2. STATUS	a. DEPLOYMENT: b. SERVICEABILITY: c. OCCUPATION: d. CAPABILITY: e. HARDENING: f. CONSTRUCTION: g. CAMOUFLAGE:
3. EQUIPMENT AND ACTIVITY	

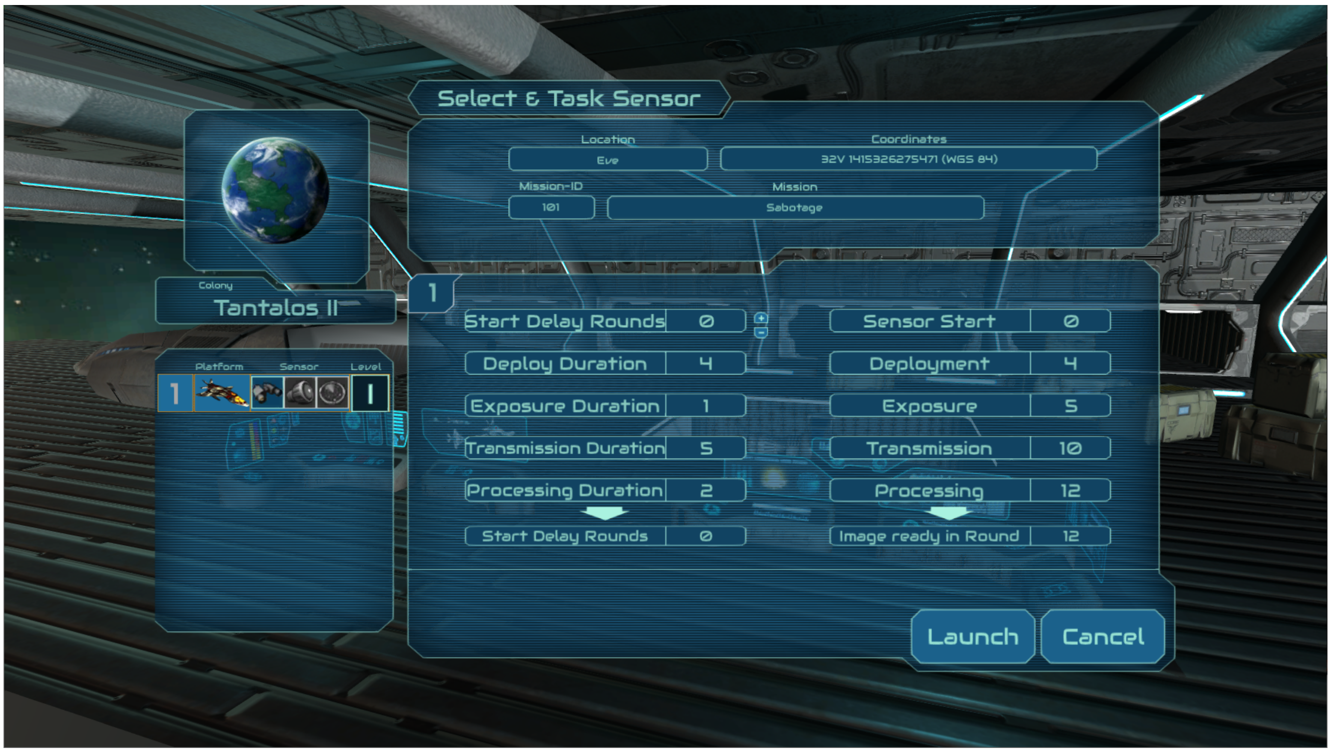
CAT 12 - BRIDGES AND TUNNELS

ALL

- CAT 12 - BRIDGES AND TUNNELS
  - 1. LOCATION AND TYPE
  - 2. STATUS
  - 3. EQUIPMENT AND ACTIVITY
  - 4. DEFENCE
  - 5. FACILITIES / DESCRIPTION

CLASSIFICATION:	NFF-SECRET
REQUESTOR:	CNOC
DTG:	0
ACTION:	IMMEDIATE
INTERNAL COORDINATION:	any internal coordination
FROM:	Isaac Kasparow
TO:	Alex
INFO:	any info
L. ORIGINATOR SERIAL NO.:	2 CNOC 102
M.1 TYPE OF SENSOR:	PHOTO, ELECTRONIC, SLAR, IR, [✓ BEST SUITABLE]
M.2 TECHNIQUE:	VERTICAL, OBLIQUE, PANORAMIC, [✓ BEST POSSIBLE]
N. MAP SERIES SHEET NO. AND EDITION NO.:	Ancient Cartography Maps Edition 2307, WGS-84

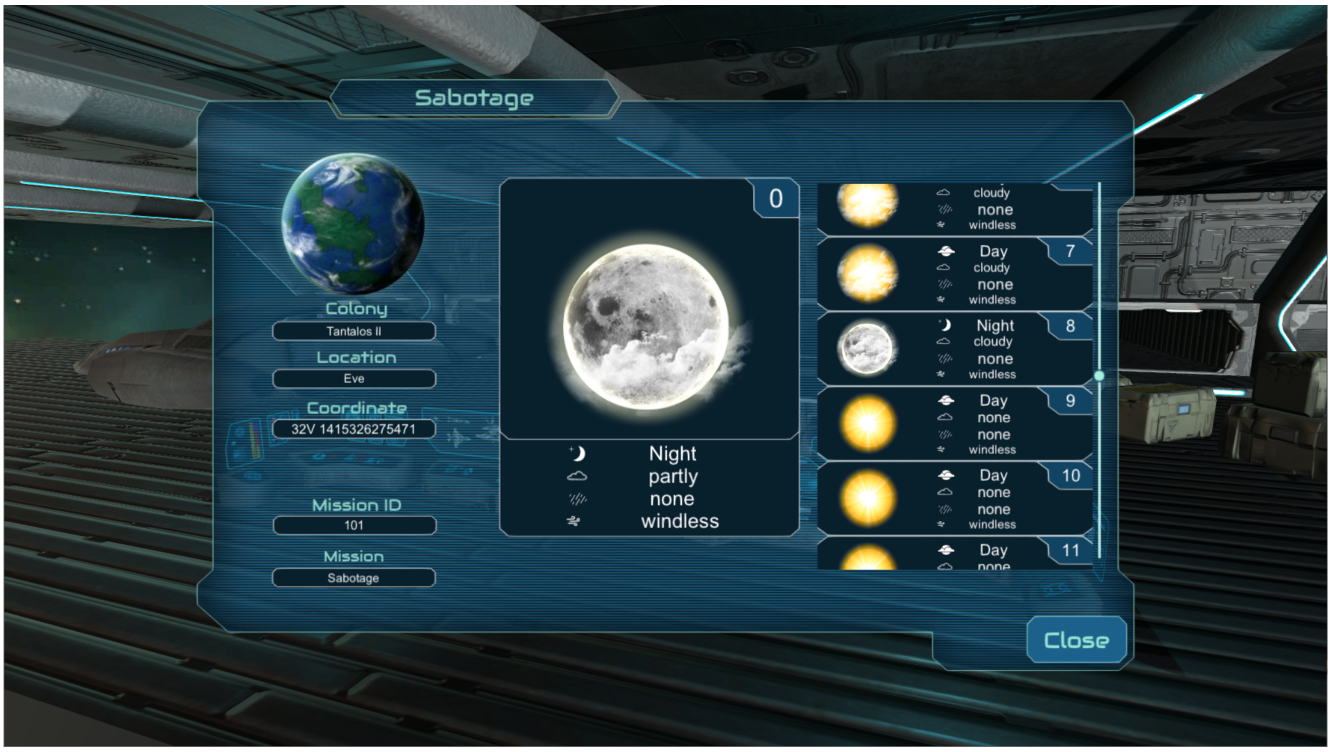
Figure 4.5.: Reconnaissance mission. ViLand, a training system for imagery, is used by the player to analyze imagery and create the reports



Location	Eve
Coordinates	32V 1415326275471 (WGS 84)
Mission-ID	101
Mission	Sabotage

Start Delay Rounds	0	Sensor Start	0
Deploy Duration	4	Deployment	4
Exposure Duration	1	Exposure	5
Transmission Duration	5	Transmission	10
Processing Duration	2	Processing	12
Start Delay Rounds	0	Image ready in Round	12

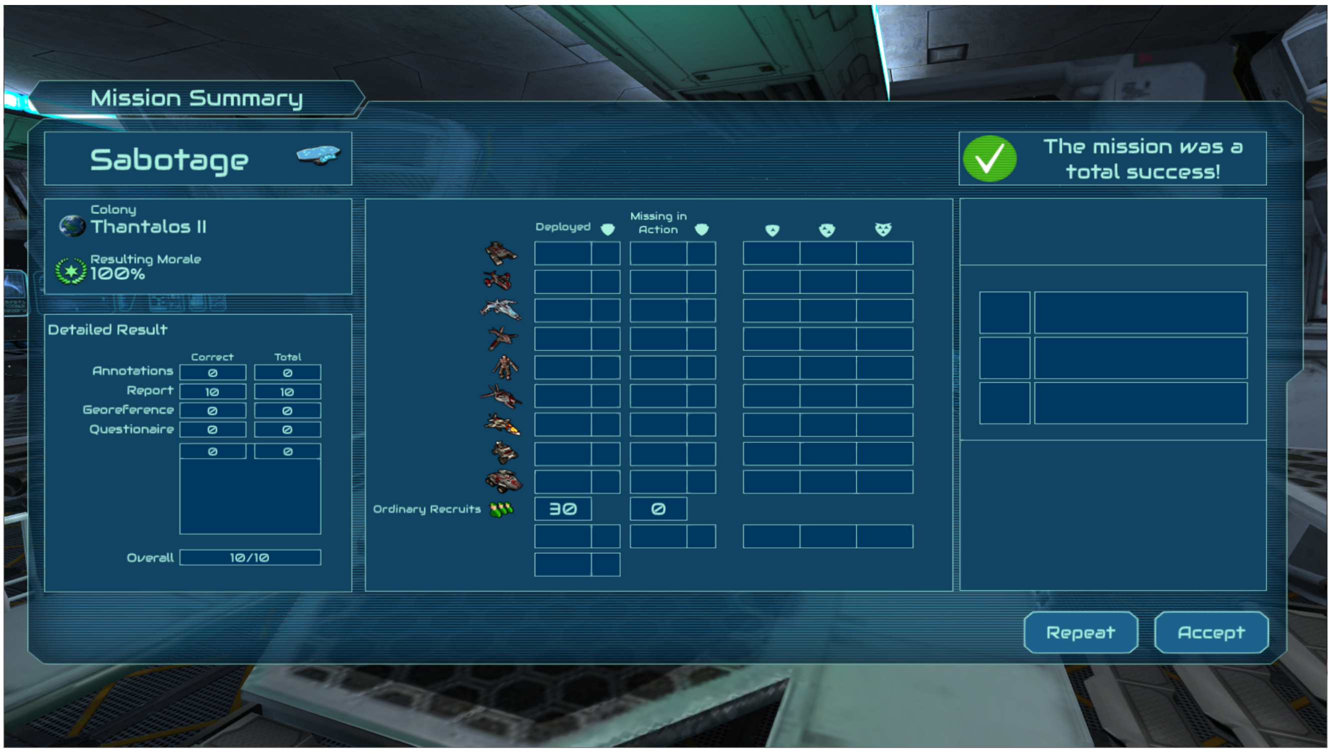
Figure 4.6.: Deployment of sensor and platform. After choosing an appropriate sensor and platform, the player can launch the platform. They can decide whether to start the platform with or without a start delay to account for weather conditions.



Colony: Tantalos II
Location: Eve
Coordinate: 32V 1415326275471
Mission ID: 101
Mission: Sabotage

0	Night	partly	none	windless
		cloudy	none	windless
7	Day	cloudy	none	windless
8	Night	cloudy	none	windless
9	Day	none	none	windless
10	Day	none	none	windless
11	Day	none		

Figure 4.7.: Weather console. The console informs the player about the weather forecast on the target planet.



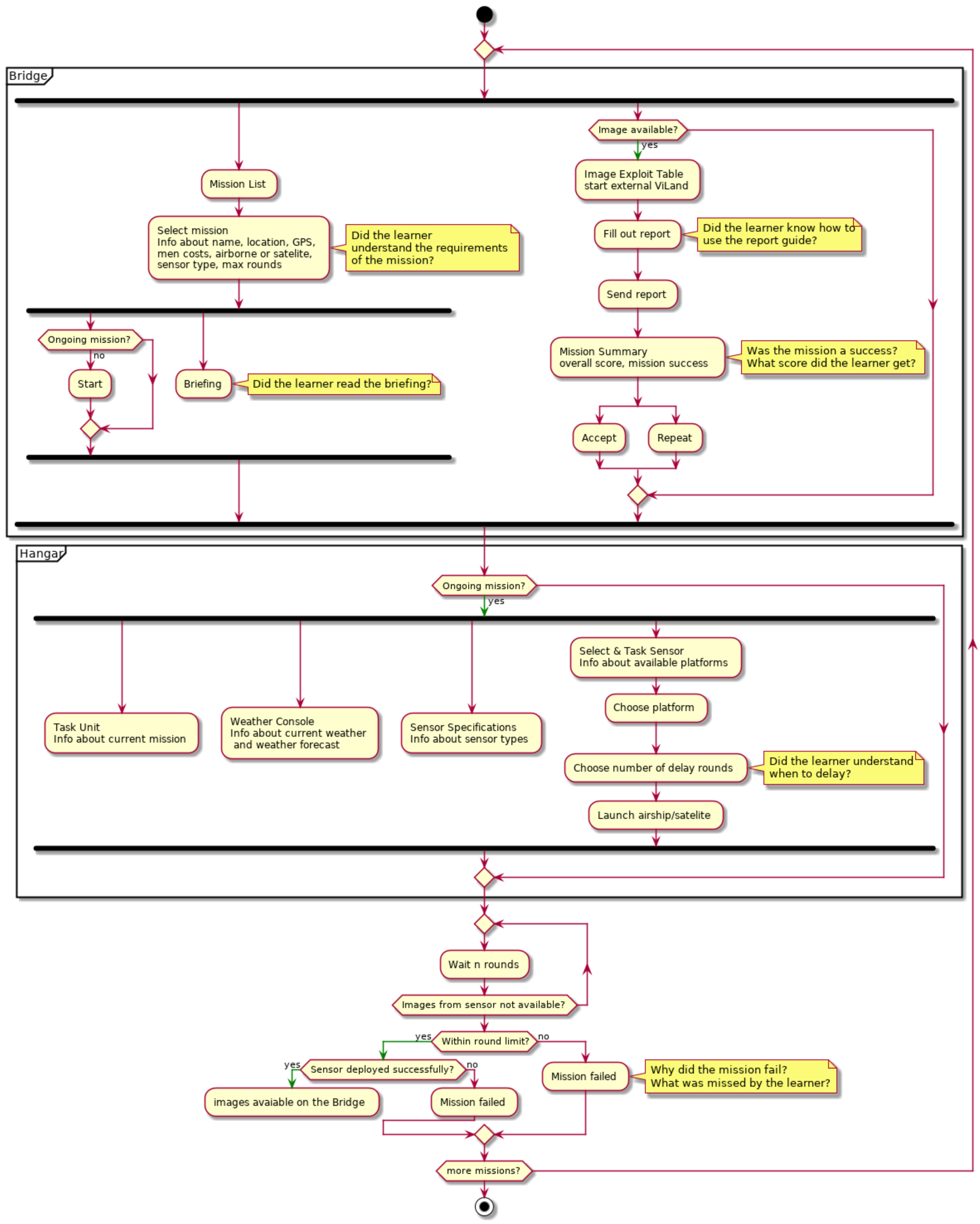
Colony: Thantalos II
Resulting Morale: 100%

Detailed Result

	Correct	Total
Annotations	0	0
Report	10	10
Georeference	0	0
Questionaire	0	0
	0	0
Overall	10/10	

	Deployed	Missing in Action
Ordinary Recruits	30	0

Figure 4.8.: Mission report and summary. The mission success depends on the quality of the filed report. The player's performance is measured as overall score which is ratio of correctly reported items. This score influences the moral of the deployed troops.



Bridge
Mission List
Select mission
Info about name, location, GPS,
men costs, airborne or satelite,
sensor type, max rounds
Did the learner
understand the requirements
of the mission?
Ongoing mission?
no
Start
Briefing
Did the learner read the briefing?
Image available?
yes
Image Exploit Table
start external ViLand
Fill out report
Did the learner know how to
use the report guide?
Send report
Mission Summary
overall score, mission success
Was the mission a success?
What score did the learner get?
Accept
Repeat
Hangar
Ongoing mission?
yes
Task Unit
Info about current mission
Weather Console
Info about current weather
and weather forecast
Sensor Specifications
Info about sensor types
Select & Task Sensor
Info about available platforms
Choose platform
Choose number of delay rounds
Did the learner understand
when to delay?
Launch airship/satelite
Wait n rounds
Images from sensor not available?
yes
Within round limit?
no
yes
Sensor deployed successfully?
no
images avaiable on the Bridge
Mission failed
Mission failed
Why did the mission fail?
What was missed by the learner?
more missions?

Figure 4.9.: Basic activity diagram for Lost Earth. The player can either be on the bridge or in the hangar. Most actions are only available during a mission. When a mission failes, or after the player received the mission summary, they have a chance to retry.

4.2.1 Identify the data – first step of Bayesian data analysis

Three categories of observable variables were identified: general performance measures, domain-specific measures, and game-specific measures (Table 4.1). The main focus lies on the first two categories, which will be used in the Bayesian approach as observable variables the descriptive model tries to explain.

Table 4.1.: A set of observable variables that can be captured by xAPI statements from the game Lost Earth. A description of the variables is given in the text.

Name	Level	Variable	Type	Unit	Domain
task success	performance	k	binary	-	$\{0, 1\}$
mission score	performance	s	discrete	-	$[0, \dots, max_{\text{items}}]$
mission time	performance	t	continuous	minutes or seconds	$\mathbb{R}_{\geq 0}$
required rounds	domain	n_{rnd}	discrete	-	$[1, \dots, max_{\text{rounds}}]$
required hints	domain	n_{hnt}	discrete	-	$[0, \dots ($
location changes	domain	n_{loc}	discrete	-	$[0, \dots ($
dialogues	domain	n_{dia}	discrete	-	$[min_{\text{dialogues}}, \dots ($
detours	game	n_{det}	discrete	-	$[0, \dots ($

General performance measures are mostly domain-independent and can be measured in any application. `Task success` k describes the success of a player for a given task and is either true or false. In Lost Earth, such a task is the deployment of a sensor in the target system, which can fail if the player has not correctly considered the weather conditions or has exceeded the number of rounds available for this mission. `Mission score` s is the overall score of the player for the given mission, but it applicable to any game and any learning task, because it is a simple accuracy measure that reflects the number of errors the player made. In Lost Earth this number lies between zero and the maximal number of items the player had to report for a given mission. The score from Lost Earth can be normalized by dividing the overall score by the maximum score for this mission, so it becomes comparable across missions and across different applications. `Mission time` t is the amount of time it took the player to finish the mission, but is applicable again to any domain or learning task by measuring the time the learner needs to finish a given task. Mission score and mission time can be captured when the mission report and the mission summary are generated by the game. Task success can be captured when the result of the sensor deployment is generated by the game. Each of these three are specific triggered game events and therefore easily captured by xAPI statements.

Domain measures are domain-dependent but applicable to any game of that domain that supports this feature. For example all strategy games use resources of some kind and all turn-based strategy games have a measure of rounds. `Required rounds` n_{rnd} describes the number of rounds the player needed to finish the mission. Because this number is dependent on the number of deployment delay rounds the player has chosen in the select and task sensor panel, required rounds is calculated by the total number of rounds minus the chosen deployment delay rounds. Whenever the player exceeds the maximum number of rounds assigned to this mission the mission fails automatically. Although the calculation is specific for Lost Earth, an analogous variable can easily be measured in similar games, where it might be the number of days, the number of turns, the number of steps, and so on. `Required hints` n_{hnt} is the number of hints the player has requested during the mission. Hints hold information that is explicitly designed to help the player but has to be optional for the player. In Lost Earth this could be the mission briefing, which is optional but provides the player with further instructions and background information. More traditional, hints are explicitly requested by the player and guide the player to the next reasonable step. Hints are normally limited in their amount or repeat the same information when requested repeatedly until the player has overcome the current challenge. Alternatively, any optional game manual can be understood as a general hint explaining game play functionality. `Location changes` n_{loc} is the number of locations the player has visited and/or the number of changes between the locations in the game. For Lost Earth, this is the number of times the player visits the bridge or the hangar. Of course, instead of one variable this could be easily extended to a variable for each location, for which it would represent the frequency of player visits. This variable is applicable to all games and applications that allow the player to change "locations", where a location can mean a real in-game location or just a different window in a window-based application. Finally, `dialogues` n_{dia} is the number of dialogues that were opened by the player. Like with `location changes` this can be more fine-grained and extended to a different variable for each dialogue in the game, where it would allow to track the players frequency and time resolution for a specific dialogue. For example this variable would measure how often the player has opened the weather console because this information is vital in choosing the right deployment delay round for a specific sensor. Most, if not all, of these variables require additional events to be captured as xAPI statements.

Game measures are the most specific of all observable variable because they only apply to a particular game and are normally not transferable to other games or applications without a reinterpretation. Because the focus of this thesis lies on general frameworks and cognitive user models, I have decided not to use observable variables of this level, but instead base the descriptive models on variables of the more generally applicable levels. One example of a game measure variable might be `detours` which captures the number of detours the player has taken during the mission compared to an ideal solution where the player knows exactly what the next steps are and where to go. This variable can only be game-dependent because what counts as a detour has to be defined with respect to the particular game, the available locations and interaction possibilities of the player and the current type of mission. Another problem of this variable is how to interpret a detour. Maybe the player knows exactly where to go next, but first wants to visit side characters to further advance personal stories or wants to check the latest news from the game universe. In general, the more specific a measure gets, the more careful we have to be with its interpretation.

I want to state explicitly that the list of presented observable variables is neither exhaustive nor final. There might be many other variables that are equally or even better suited to capture player interactions in Lost Earth, or any game. My intention was to present a hierarchy of levels the observable variables can be assigned to and to derive observable variables useful for the realization of a cognitive user model.

To work with Bayesian models, it is required to define the relevant observable data, the involved measurement scales of the data and the definition of variables that are to be predicted and variables that are predictors. This was done in this chapter by defining a set of observable variables along with their type and domain. These observational variables will be the target variables in the descriptive models presented in section 2.1.2 Hierarchical Bayesian modeling (page 31).

4.3 Realizing Cognitive User Models

In this section I present the main concepts that I developed to realize cognitive user models with the help of either cognitive architectures or hierarchical Bayesian models. Regarding the aim of this thesis to contribute to applied research and not fundamental research—what are the expectations and requirements for the use of existing approaches? I considered the following requirements given in alphabetically order for convenience:

Actively developed	The software or library should be actively developed, ideally on a publicly accessible online repository.
Appropriateness	The software or library should support the goal of this thesis to infer the learner's current cognitive state. It does not have to support all cognitive variables but must provide information that can be used by an adaptive system to decide when to adapt. The selection of an adaptive measure is not part of the cognitive user model.
Availability	The software or library should be publicly available and free to use.
Extensibility	The software or library should provide support for own modifications or extensions if necessary.
Extensive support	The software or library should provide extensive support in form of (online) manuals, demonstrations, tutorials, and a supportive community.
Generality	The software or library should allow for general solutions and not only provide solutions for a very narrow set of problems. It should be game-independent as well as domain-independent.
Intelligence	The software or library should not require that all intelligence be brought in by the modeler but instead show some kind of intelligent behavior that goes beyond what was defined in the model.
Interoperability	The software or library should operate on xAPI statements and provide output that can be easily used within the ELAI framework to derive adaptive measures. There should be no use of proprietary formats for input or output.
Lightweight	The software or library should be lightweight to facilitate the use as an external controller in the ELAI framework that hosts the intelligence.
Low dependencies	The software or library should be a stand-alone program and not be dependent on external requirements.
Proved and tested	The software or library should not be experimental but well-proven, ideally with publicly available research studies.

Simplicity The software or library should be simple and easy to learn for the modeler.

Both Soar 9 and PyMC3 are software packages or frameworks, respectively, that fulfill many of these requirements, but have different strengths and weaknesses that I will address in section 7.1 Assessment of Requirements (page 153) of the final chapter 7 Conclusion and Recommendations. The following two sections provide the details of how to realize cognitive user models with each of the chosen frameworks.

4.3.1 With cognitive architectures

The latest version of Soar 9, 9.6.0, was used. The architecture itself along with all additional material is available from the website[2]. Soar 9.6.0 can be downloaded as a multi-platform version with or without additional tutorials. Soar runs on different platforms, including Linus, Unix, Mac OS X, and Windows, both on 32-bit and 64-bit versions. After the download, Soar can be started with a graphical user interface via the `SoarJavaDebugger` file or using a command line interface via the `Soar_CLI` file.

The Soar program can be started with the Soar Debugger script (Figure 4.10 on the next page). The Soar program is used to load and run Soar agents that use if-then rules, called productions, to solve a given problem. The Soar Debugger is a powerful tool for debugging Soar agents: it allows to run Soar step-wise and halt the program at any point in the Soar Cycle and inspect its state.

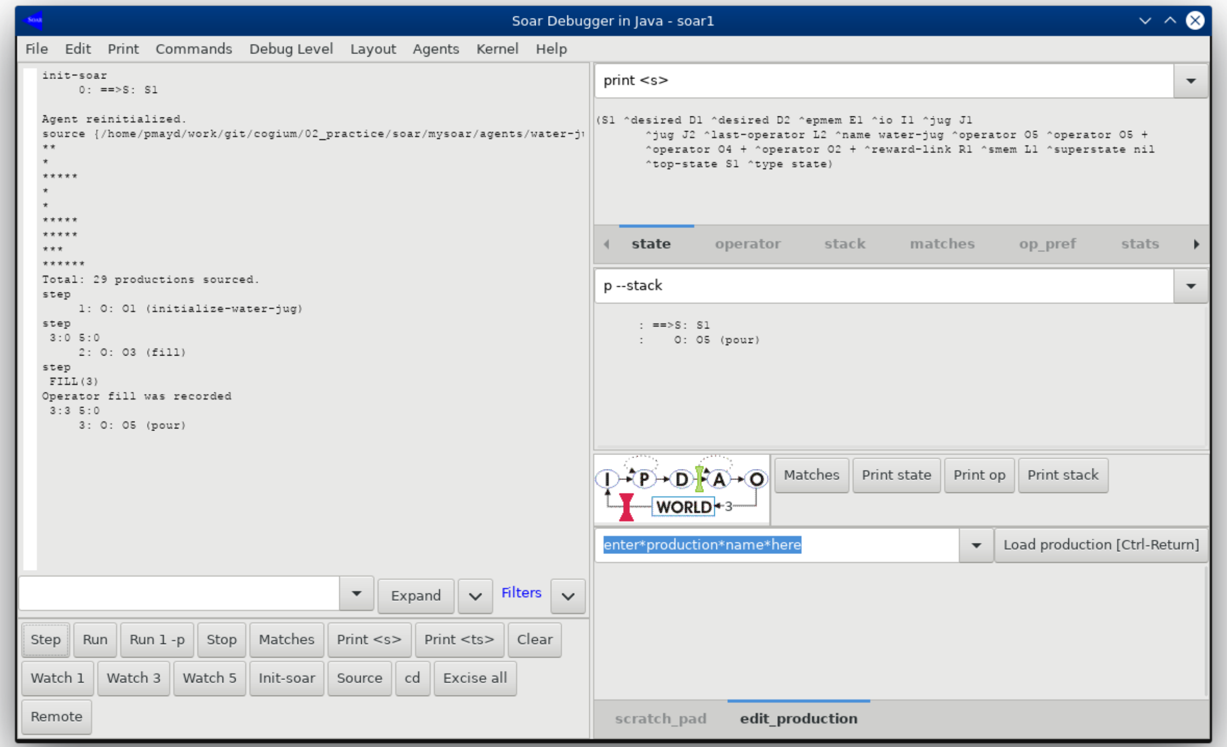
Everything in Soar is accomplished by productions stored in the production memory. Productions are either read in from a file, that is, entered by a user, or generated by chunking, which is one of Soar's learning mechanisms. Each production has three required components: a name, a set of conditions, also called the left-hand side, and a set of actions, also called the right-hand side. Syntactically, each production consists of the symbol `sp`, followed by: an opening curly brace, the production's name, the production's conditions, the symbol `-->`, the production's actions and a closing curly brace:

```
sp {production-name
    ‘‘Documentation string’’
    :type
    CONDITIONS
    -->
    ACTIONS
    }
```

An example production, named `blocks-world*propose*move-block` is shown in Listing 4.1 on the following page. This production proposes a new operator (`+`) with the name `move-block` that takes `<thing1>` and puts it to destination `<thing2>`, whenever the conditions are met, which check whether the block can be moved safely to the destination. The Soar syntax is fully described in the Soar User's Manual by Laird, Congdon, et al. (2017).

To write Soar programs any text editor can be used, but Soar comes with `VisualSoar`, an editor with internal support for Soar programs (Figure 4.11 on page 83).

[2] https://soar.eecs.umich.edu/


Soar Debugger in Java - soar1
File Edit Print Commands Debug Level Layout Agents Kernel Help
print <s>
state operator stack matches op_pref stats
p --stack
I P D A O
WORLD 3
Matches Print state Print op Print stack
enter*production*name*here
Load production [Ctrl-Return]
Expand Filters
Step Run Run 1 -p Stop Matches Print <s> Print <ts> Clear
Watch 1 Watch 3 Watch 5 Init-soar Source cd Excise all
Remote
scratch_pad edit_production

```
init-soar
     0: ==>S: S1

Agent reinitialized.
source {/home/pmayd/work/git/cogium/02_practice/soar/mysoar/agents/water-j
**
*
*****
*
*
*****
*****
***
******
Total: 29 productions sourced.
step
     1: O: O1 (initialize-water-jug)
step
 3:0 5:0
     2: O: O3 (fill)
step
 FILL(3)
Operator fill was recorded
 3:3 5:0
     3: O: O5 (pour)
```

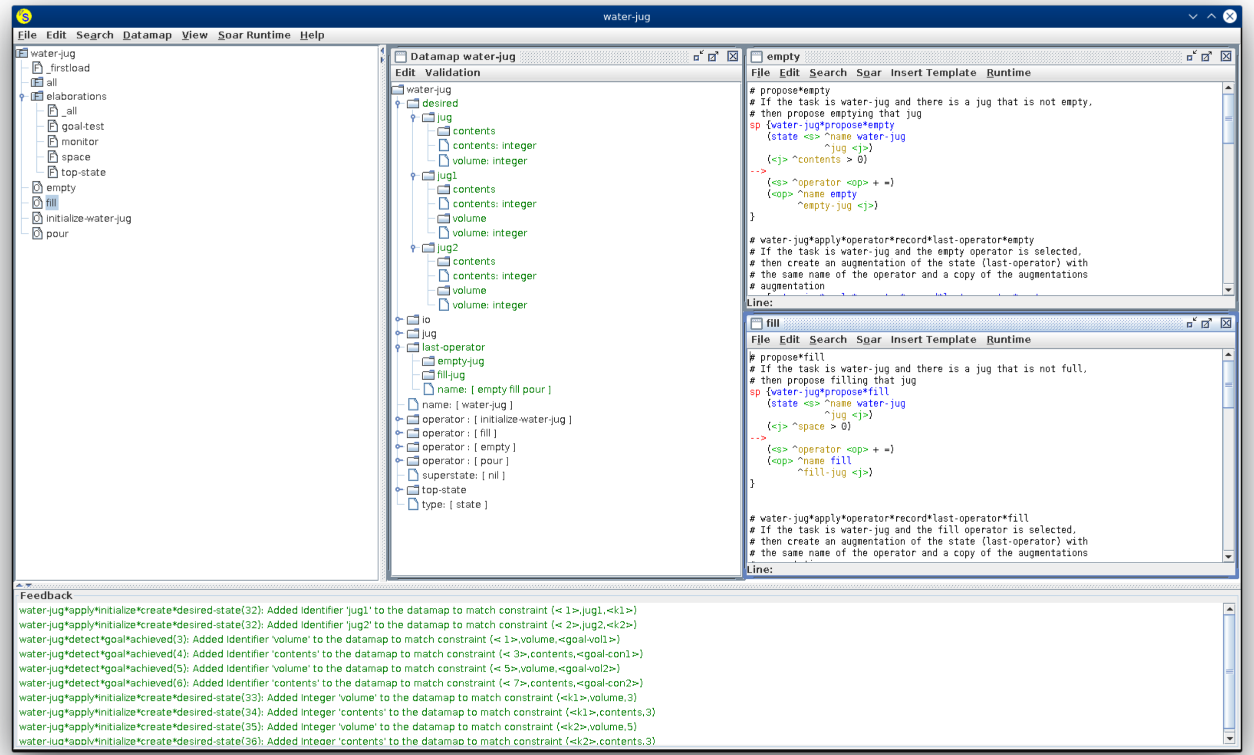
```
(S1 ^desired D1 ^desired D2 ^epmem E1 ^io I1 ^jug J1
       ^jug J2 ^last-operator L2 ^name water-jug ^operator O5 ^operator O5 +
       ^operator O4 + ^operator O2 + ^reward-link R1 ^smem L1 ^superstate nil
       ^top-state S1 ^type state)
```

```
      : ==>S: S1
      :    O: O5 (pour)
```

Figure 4.10.: Soar Debugger. Soar with a graphical user interface. The large window on the left is the interaction window with debugging information and output of print statements. Below the interaction window is the Command Box for user commands. The windows on the right display information about the state of the Soar agent and the Soar Cycle widget. In this example a simple agent for the Water Jug problem was loaded and executed for three runs.

Listing 4.1: An example production from the example blocks-world task.

```
sp {blocks-world*propose*move-block
   (state <s> ^problem-space blocks
   ^thing <thing1> {<> <thing1> <thing2>}
   ^ontop <ontop>)
   (<thing1> ^type block ^clear yes)
   (<thing2> ^clear yes)
   (<ontop> ^top-block <thing1>
   ^bottom-block <> <thing2>)
   -->
   (<s> ^operator <o> +)
   (<o> ^name move-block
   ^moving-block <thing1>
   ^destination <thing2>)}
```



```
# propose*empty
# If the task is water-jug and there is a jug that is not empty,
# then propose emptying that jug
sp {water-jug*propose*empty
   (state <s> ^name water-jug
              ^jug <j>)
   (<j> ^contents > 0)
-->
   (<s> ^operator <op> + =)
   (<op> ^name empty
         ^empty-jug <j>)
}

# water-jug*apply*operator*record*last-operator*empty
# If the task is water-jug and the empty operator is selected,
# then create an augmentation of the state (last-operator) with
# the same name of the operator and a copy of the augmentations
# augmentation
```

```
# propose*fill
# If the task is water-jug and there is a jug that is not full,
# then propose filling that jug
sp {water-jug*propose*fill
   (state <s> ^name water-jug
              ^jug <j>)
   (<j> ^space > 0)
-->
   (<s> ^operator <op> + =)
   (<op> ^name fill
         ^fill-jug <j>)
}


# water-jug*apply*operator*record*last-operator*fill
# If the task is water-jug and the fill operator is selected,
# then create an augmentation of the state (last-operator) with
# the same name of the operator and a copy of the augmentations
```

```
water-jug*apply*initialize*create*desired-state(32): Added Identifier 'jug1' to the datamap to match constraint (< 1>,jug1,<k1>)
water-jug*apply*initialize*create*desired-state(32): Added Identifier 'jug2' to the datamap to match constraint (< 2>,jug2,<k2>)
water-jug*detect*goal*achieved(3): Added Identifier 'volume' to the datamap to match constraint (< 1>,volume,<goal-vol1>)
water-jug*detect*goal*achieved(4): Added Identifier 'contents' to the datamap to match constraint (< 3>,contents,<goal-con1>)
water-jug*detect*goal*achieved(5): Added Identifier 'volume' to the datamap to match constraint (< 5>,volume,<goal-vol2>)
water-jug*detect*goal*achieved(6): Added Identifier 'contents' to the datamap to match constraint (< 7>,contents,<goal-con2>)
water-jug*apply*initialize*create*desired-state(33): Added Integer 'volume' to the datamap to match constraint (<k1>,volume,3)
water-jug*apply*initialize*create*desired-state(34): Added Integer 'contents' to the datamap to match constraint (<k1>,contents,3)
water-jug*apply*initialize*create*desired-state(35): Added Integer 'volume' to the datamap to match constraint (<k2>,volume,5)
water-jug*apply*initialize*create*desired-state(36): Added Integer 'contents' to the datamap to match constraint (<k2>,contents,3)
```

Figure 4.11.: The VisualSoar editor fo Soar programs. The example shows the project structure for the Water Jug problem from the tutorial. The window on the left is the Operator window and shows the project structure in form of a tree. The Datamap window (middle) describes the hierarchical working memory structure. VisualSoar can run tests against the Datamap structure to detect spelling mistakes and other problems with production rules. VisualSoar opens the files that contain the production rules in separate windows (right).

Listing 4.2: The initialization application rule for the Water Jug proble. The rule adds the name to the state and creates two jugs with volumes of 5 and 3 and contents 0.

```
sp {water-jug*apply*initialize-water-jug
   (state <s> ^operator <o>)
   (<o> ^name initialize-water-jug)
   -->
   (<s> ^name water-jug
   ^jug <j1>
   ^jug <j2>)
   (<j1> ^volume 5
   ^contents 0)
   (<j2> ^volume 3
   ^contents 0)}
```

For each problem that a Soar agent is going to solve there has to be a state representation in working memory that tells Soar what attributes can be modified. Besides the definition of the production rules, this is the main task for the modeler: to come up with a useful and complete state representation. Each program begins with an initialization in which the initial state and initial WMEs are created. This is done by one initialization application rule which fires at the begin of a Soar program. For example in the Water Jug problem, there are two jugs and each jug has a volume that can be filled and the amount of water it currently holds. The initialization production rule for this problem is given in Listing 4.2.

To understand how to work with Soar, how to write Soar programs and how to use Soar for the purposes of this thesis, I have worked through the first three out of eight tutorials provided by Soar, which alone were 150 pages in total. At this point I had to decide whether Soar was the right approach for realizing CogIUMs. I came to the conclusion, that it is not.

From what I have presented so far about the working mechanisms of Soar it should be clear that working with Soar 9 means to define the initial state the agent starts in and possible WMEs it can manipulate along with a set of production rules that elaborate states, propose operators or define the application of operators. To realize a CogIUM with Soar 9 would have meant for me to model the game with this methods. A Soar program would create an agent that starts in a given initial state that represents a state from Lost Earth. Then, the agent would explore the problem space to find a path from the initial state to the defined goal state, which would be the successful deployment of a sensor and platform. Because ViLand is an external application and not part of the game Lost Earth, I would not write a Soar program for the imagery interpretation part but only for the interactions that took place directly in Lost Earth. Because I worked with an early prototype of Lost Earth, the set of possible interactions was still limited and it would have been possible to write a Soar program for this part of the game, as was shown in the activity diagram (Figure 4.9 on page 77). I would have to write a production rule for each interaction in Lost Earth and define how this action changes the state. For example opening the weather console would create new semantic knowledge about the weather forecast, which could be stored in the semantic long-term memory of the agent via the `^store` command. Or, when the player selects the number of deployment delay rounds this would change an attribute like `^delay-rounds` of the current state to reflect the player's choice.

Although this was a possible way to go, it would have taken a great amount of time to model all necessary operators and production rules in Soar. There are tools to help with this, like the Game Description Language[3], which describes the state of a game as a series of facts, and the game mechanics as logical rules, but resources were outdated and the tool was not available[4]. However, there was a more serious problem.

The main problem with Soar 9 as the tool for a CogIUM is how to come from the generated output to the learner's cognitive state. Soar 9 has no internal variables that represent the cognitive state of the Soar 9 agent. Soar is called a cognitive architecture because it has built-in assumptions about how human cognition works but it delivers no output that allows a direct inference about the agent's cognitive state. Soar 9 supports four learning mechanisms, chunking, semantic learning, episodic learning and reinforcement learning, all of which could have been used for this thesis. But again, the question remains how to use these mechanisms to learn something about the internal cognitive state of the agent. I have summarized my considerations regarding Soar 9 as a CogIUM and have given exemplary outputs that I think are possible with Soar 9 (Figure 4.1 on page 71). Each interaction of the user will be provided to the Soar program as input by xAPI statements. Soar uses this input to create the agent and the initial state S_0. Then the Soar agent runs the Soar program and explores the problem space until it finds the goal state. The output of Soar will be the explored problem space and the found path from the initial state to the goal state along with everything the agent has learned during the process. The output can be used in several ways:

- Soar can provide an overview of all valid next actions along with an evaluation of each action's effect towards reaching the goal state. The player can be informed about possible next actions or just the best next action given their current position in the game.
- Soar can detect dead ends or harmful states. The player can be warned about actions that might lead to dead ends or, in hindsight, can be informed about alternative actions that would have led to the goal state.
- Soar can provide a path from the initial state to the goal state and compute the distance to the goal state. If repeated runs of Soar compute the same distance, this would indicate that the player does not get closer to the goal state, or even increases their distance. The player could be informed about this observation and be asked if they like assistance to progress.
- Soar can provide alternative paths to the goal state, if they exist. The player can be informed about these alternative paths to further strengthen their mastery of the domain or to show alternative solutions to foster problem understanding and solution.
- Soar can provide a history of previous states via episodic memory, which represents an agent's stream of experience. Soar can detect if the agent was in the same state before or has executed an operator under the same conditions. The player can be informed about going in circles or about what they have done the last time in a similar situation.

To sum up, there are ways to use Soar 9 for an adaptive system to decide when to adapt and influence the user in choosing their next action, but there is no direct way to asses the user's current cognitive state. That is not what Soar is intended to do. To infer the user's cognitive state one has to use proxy data like

[3] http://games.stanford.edu/games/gdl.html

[4] https://soar.eecs.umich.edu/articles/downloads/domains/180-general-game-player-translator

the number of decisions and impasses required to come to a solution. Sweller (1988, p. 265) already mentioned that “a production system is not specifically designed to measure cognitive load, [but] there are several aspects of production systems which could provide suitable measures.” He lists the number of statements in working memory, the number of productions and the number of conditions that need to match statements in working memory as possible correlates. I corresponded with John Laird, one of the inventors of Soar, and he confirmed that, at the moment, Soar does not provide such information as there a no background calculations of cognitive load, or attention monitoring. He pointed at work done by Soar Tech where ITS were built that monitor people doing tasks, but I did not pursue this direction further.

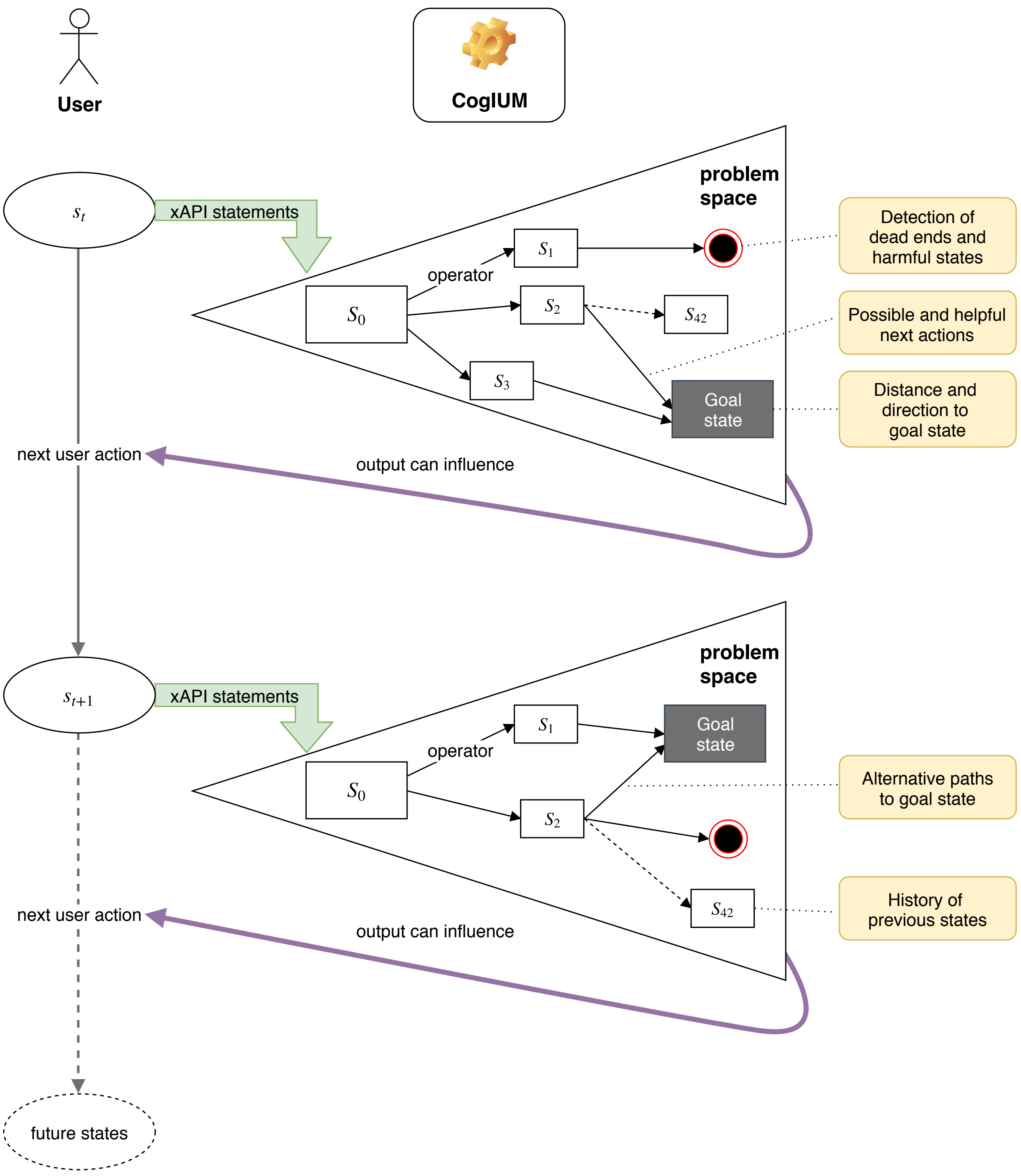


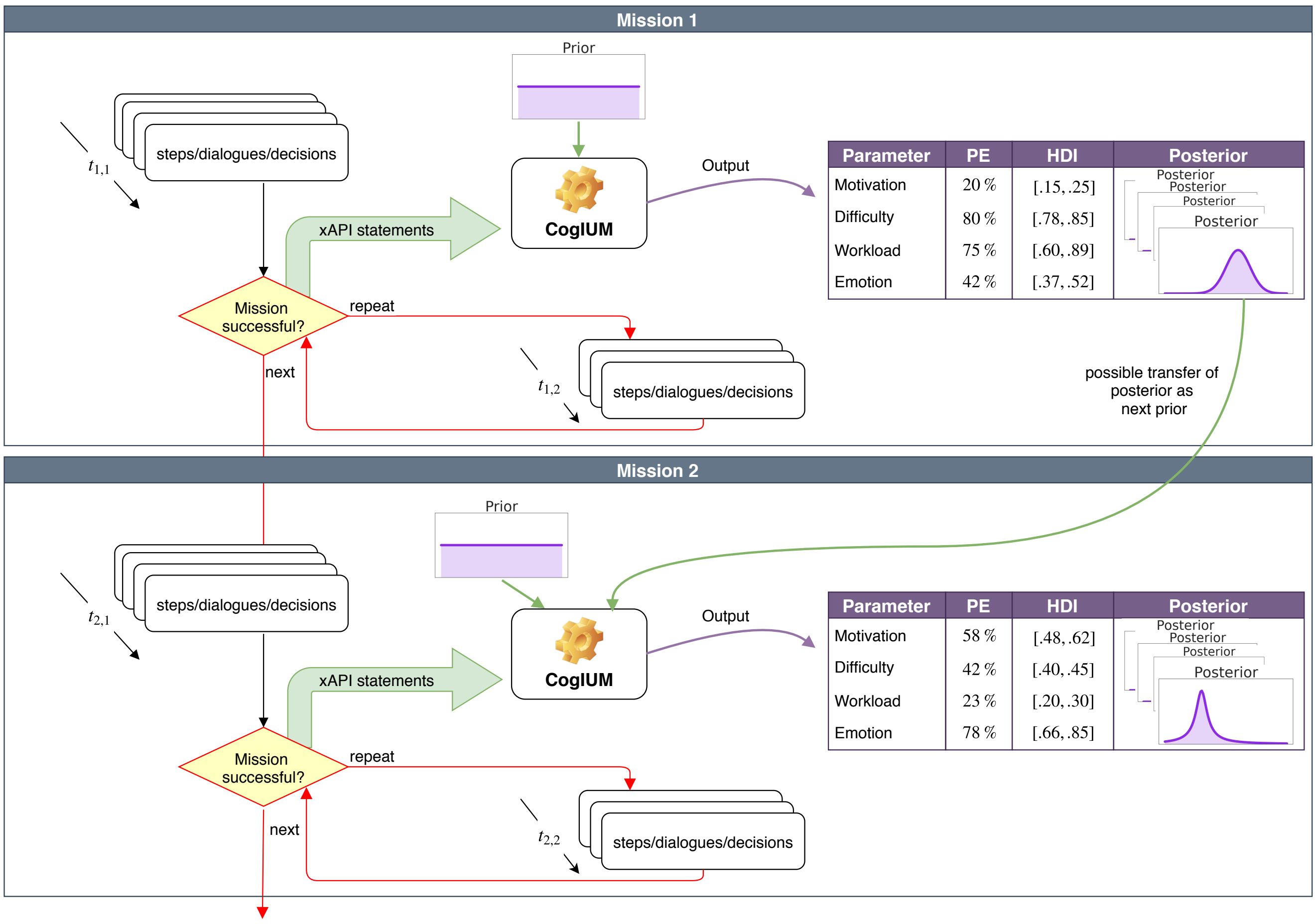
Figure 4.12.: Realizing cognitive user models with Soar 9. Each action of the user is captured by xAPI statements and provided to Soar 9 as input. Soar 9 has an internal model of the game and conducts a search through the problem space to find a path from the user's current position (initial state S_0) to the goal state (shaded rectangle). The problem space changes as the user moves on in the game. The information produced by Soar 9 as output can be used as input for an adaptive system.

4.3.2 With (hierarchical) Bayesian models – second and third step of Bayesian data analysis

After the insight that Soar 9 is not suited to meet the research goals of this thesis, and the decision to not longer follow this approach, another approach had to be found. Instead of a very powerful framework like Soar 9, which demands a lot of domain and task modeling effort but cannot directly offer inferences about the cognitive state of the learner, I decided to choose a different approach that is like the direct opposite of the first: to build a probabilistic statistical model of the data that only specifies what is really needed for the task of this thesis. The Bayesian modeling approach allows for maximal control and flexibility as the model can be as general or as complex as needed. Bayesian models are directly built in a way to infer the state of latent, non-observable variables from observable variables.

Because I ended the last section with an overview about how Soar 9 can be used to realize CogIUMs, I want to give the same line of reasoning for HBMs (Figure 4.13 on the next page). In comparison to the procedure that I have presented for Soar 9, HBMs will have to work with the collected data at the end of a mission, because all observations the model is going to explain have to be first collected by the game, which happens only at the end of a mission for variables like the mission time. After the player has finished a mission, either successful or with failure, the collected data is send to the CogIUM in form of xAPI statements. The intelligence of the CogIUM is realized by a HBM which takes predefined prior distributions for each model parameter as well as the observable data as input and calculates via Bayesian inference the posterior distributions for all model parameter. The model parameters are variables that can be directly interpreted as cognitive variables like motivation or the experienced difficulty. Because the model produces posterior distributions, which are probability densities, the model gives directly interpretable probabilities as output. It is straight forward to derive point estimates like mean and mode values as well as HDIs from the posterior distributions. Therefore, at the end of each mission, the model can infer the learner's current cognitive state and make statements about their current value with an estimate of uncertainty. Because the posterior is a probability density, it can be reused in another run of Bayesian inference as the prior, replacing the old, non-informative prior with the latest belief according to the data.

According to the five steps of Bayesian data analysis that were presented in section 2.1.2 Bayesian data analysis (page 34), to realize a CogIUM with HBMs I needed to define a descriptive model for the relevant data, as defined in the section 4.2.1 Identify the data (page 78), with meaningful parameters, and to specify the prior distributions of the parameters. By choosing the Cognitive Load Theory (CLT) as a central aspect of the cognitive state of the learner, the components of this theory had to play a key role in the descriptive model as latent variables. As was mentioned in section 2.3.1 Cognitive Load Theory (page 48), cognitive load consists of three main components: intrinsic cognitive load (ICL), extraneous cognitive load (ECL) and germane cognitive load (GCL) (Figure 2.15 on page 51). ECL is the only variable that is truly independent of the learner and only depends on the instructional design of the material. ICL is independent of the learner, if the level of prior knowledge is constant. Otherwise, ICL changes with the level of prior knowledge. GCL, on the other side, is independent of the learning material and only dependent on the learner's characteristics. The motivation of the learner further complicates the relationship, because motivation has an influence of the amount of resources allocated in working memory to deal with the ICL that was imposed by the task. As stated by the CLT, a learner's performance will



Parameter	PE	HDI	Posterior
Motivation	20 %	[.15, .25]	
Difficulty	80 %	[.78, .85]	
Workload	75 %	[.60, .89]	
Emotion	42 %	[.37, .52]	

Parameter	PE	HDI	Posterior
Motivation	58 %	[.48, .62]	
Difficulty	42 %	[.40, .45]	
Workload	23 %	[.20, .30]	
Emotion	78 %	[.66, .85]	

Figure 4.13.: Realizing cognitive user models with hierarchical Bayesian models. During each mission the player executes a series of actions, which takes time t. At the end of each mission all user interactions are send to the CogIUM in form of xAPI statements. The CogIUM is given prior distributions for each model parameter and observable data as input and produces via Bayesian inference posterior distributions for each model parameter. This posterior distribution is the base for the computation of point estimates (PE) like mean or mode values and highest density intervals (HDI). The posterior of one mission can also serve as the prior for another mission.

decrease when either ICL, ECL or the combination of both exceeds the working memory capacity of the learner (Figure 2.16 on page 52). All things considered, I had enough information to specify a first draft of the CogIUM.

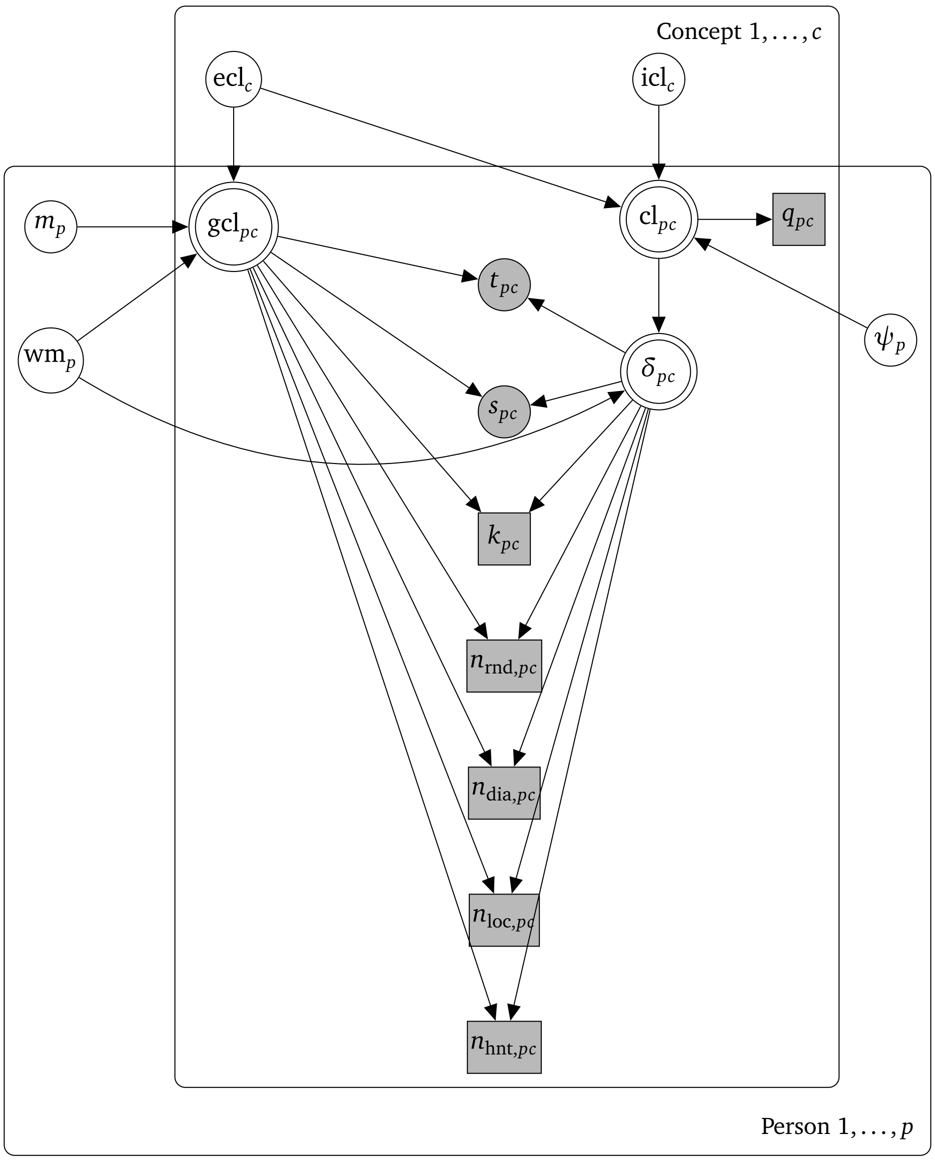
First draft and core design

The first draft of a descriptive model based on the CLT and the described observable variables is very close to the model of CLT (Figure 4.14 on the following page). It is not a fully descriptive model, because the probability distributions for the variables are yet to be defined. But it already specifies the equations to compute the deterministic variables. The model defines variables for two different groups: personal variables that differ between subjects and are indexed with a p and conceptual variables that differ between concepts and are indexed with a c. 'Concept' in this context means a learning concept that the learner should acquire. In the context of a digital educational game this can be any concept implemented in the game. For this thesis, I have assumed that each mission of the serious game Lost Earth 2307 can be seen as one such concept. Thus, for the context of this analysis, the term concept is identical with mission, but it has not to be. The model is flexible enough to handle as much fine-grained concepts as necessary as long as the observable data fits to the model. For Lost Earth 2307, observable data is gathered at the end of a mission and therefore a concept is identical to a mission. In addition to the group variables, there are also variables that differ between subjects as well as between concepts and are indexed by both pc. All three deterministic variables belong to this category. Table 4.2 gives a detailed overview of all variables in the basic model along with their attributes and domains.

Table 4.2.: Main variables of CogIUM and their domains.

Variable	Name	Group	Index	Matrix Dimension	Domain
ecl	ECL	Concept	c	$c \times 1$	$\mathbb{R}_{\geq 0}$
icl	ICL	Concept	c	$c \times 1$	$\mathbb{R}_{\geq 1}$
m	Motivation	Person	p	$p \times 1$	$[0, 1]$
wm	Working Memory Capacity	Person	p	$p \times 1$	7 ± 2
ψ	Prior Knowledge	Person	p	$p \times 1$	$[0, 1]$
gcl	GCL	Concept & Person	pc	$p \times c$	$[0, 1]$
cl	Total Cognitive Load	Concept & Person	pc	$p \times c$	$\mathbb{R}_{\geq 0}$
δ	Free Working Memory Capacity	Concept & Person	pc	$p \times c$	$\mathbb{R}_{\leq 1}$

ECL ecl_c and ICL icl_c are the two conceptual variables that differ between concepts. Because these components of the CLT are determined by the number of interactive elements of the learning material, they are best represented by integer values. As there cannot exist a negative number of interactive elements, both variables are positive. ecl_c is assumed to have the possibility to be 0, that is, ECL can be entirely eliminated, when the instructional material imposes no additional load. icl_c, however, has to be greater or equal 1, because there has to be at least one interactive element so that there is something



Deterministic variables

$$\mathrm{cl}_{pc} \leftarrow (1-\psi_p)\cdot \mathrm{icl}_c + \mathrm{ecl}_c$$

$$\mathrm{gcl}_{pc} \leftarrow \begin{cases} 0 \text{ if } m_p \cdot \mathrm{wm}_p - \mathrm{ecl}_c < 0 \\ \dfrac{m_p \cdot \mathrm{wm}_p - \mathrm{ecl}_c}{\mathrm{wm}_p} \text{ else} \end{cases}$$

$$\delta_{pc} \leftarrow \frac{\mathrm{wm}_p - cl_{pc}}{\mathrm{wm}_p}$$

Figure 4.14.: First draft of the CogIUM. Plates indicate variables that belong to a group. Personal variables with index p are variables that differ between subjects but not concepts. Conceptual variables with index c, on the other side, differ between concepts but not subjects. In the intersection of both plates are variables with an index pc that are both personal and conceptual and thus differ between subjects as well as between concepts. The notation for HBMs was introduced in Table 2.2 on page 32.

for the learner to learn. The greater the value for either icl_c or ecl_c the higher the number of interactive elements in the learning material or the instructional design.

Motivation m_p, working memory capacity wm_p, and prior knowledge ψ_p are the personal variables that differ between subjects. Motivation influences the amount of working memory dedicated to deal with the intrinsic cognitive load, so motivation will determine gcl_{pc}. As the role of motivation is to determine the amount of free working memory capacity, I decided to model motivation as a multiplicative factor between 0 and 1. Thus, a motivational value of $m_p = 0$ means that there is no working memory capacity used at all for learning. If, on the other side, the motivational value is $m_p = 1$, then all working memory capacity is used for learning. The working memory capacity wm_p is the same kind of variable as ecl_c and icl_c, which means that it is modeled as a discrete number of elements the learner can hold simultaneously in their working memory. This is because both types of cognitive load impose a cognitive load to the working memory. For example if ecl_c and icl_c both are half the number of the working memory capacity, then their sum is equal to the whole working memory capacity and the working memory is charged to capacity. The role of prior knowledge ψ_p is quite similar to the role of motivation, as it determines the influence of icl_c. If a person has a lot of prior knowledge about a concept or task, then the ICL that is imposed by the learning material should be much smaller for this particular person because a lot of concepts are already known, familiar and organized into more effective chunks or schemata. When, on the other side, a person has no prior knowledge at all about a concept or task, then any information is new and everything has to be learned from scratch, there are no familiar concepts and no connections to related concepts. Therefore, I chose to model prior knowledge in the same way as motivation, that is, as a multiplicative factor between 0 and 1. If the value of prior knowledge is $\psi_p = 1$, then everything about the concept is already known and icl_c will have no contribution to the total cognitive load cl_{pc}. If, on the other side, the value of prior knowledge is $\psi_p = 0$, then nothing is known about the concept and the icl_c value will fully contribute to the total cognitive load cl_{pc}.

GCL gcl_{pc}, total cognitive load cl_{pc}, and free working memory capacity δ_{pc} are the variables that differ both between subjects and concepts. This is because they are determined by both personal and conceptual variables. gcl_{pc} is modeled according to the CLT as the amount of working memory that is dedicated to handle the ICL. The total amount of working memory capacity, that is available, is given by multiplying the motivational factor with the working memory capacity, $m_p \cdot \mathrm{wm}_p$. From this initial capacity, the value for ecl_c is subtracted because the learner has to deal with ECL to understand what the instructions are about, so gcl_{pc} is computed by $m_p \cdot \mathrm{wm}_p - \mathrm{ecl}_c$. gcl_{pc} is limited to positive values because negative values are neither plausible nor interpretable. To allow a comparison between different applications, gcl_{pc} is normalized by the working memory capacity wm_p, because this variable can differ between subjects. Thus, the final equation for gcl_{pc} is

$$\mathrm{gcl}_{pc} = \begin{cases} 0 & \text{if } m_p \cdot \mathrm{wm}_p - \mathrm{ecl}_c < 0 \\ \dfrac{m_p \cdot \mathrm{wm}_p - \mathrm{ecl}_c}{\mathrm{wm}_p} & \text{else} \end{cases} . \tag{4.1}$$

Due to the normalization, gcl_{pc} is always between 0 and 1. A value for gcl_{pc} of 1 means that there is no ecl_c and all working memory capacity is dedicated to learning. A value for gcl_{pc} of 0 can mean one of two things, or a combination: either the motivation is so small, that there was not enough initial working memory capacity, or the value for ECL is higher than the working memory capacity available to deal with ICL. The total cognitive load cl_{pc} is the sum of icl_c and ecl_c. However, as we have said and as is indicated by the index pc, cl_{pc} is not only a conceptual variable but also a personal variable, because the cognitive load imposed by ICL is reduced by the prior knowledge of the subject. Thus, cl_{pc} is computed as

$$\mathrm{cl}_{pc} = (1 - \psi_p) \cdot \mathrm{icl}_c + \mathrm{ecl}_c. \tag{4.2}$$

The range of the values for cl_{pc} has a lower limit of 0 and no strict upper limit. The free working memory capacity δ_{pc} is the working memory capacity left after subtracting the total cognitive load cl_{pc} from the initial working memory capacity. Again, to obtain comparable results across different applications, δ_{pc} is normalized by the working memory wm_p,

$$\delta_{pc} = \frac{\mathrm{wm}_p - \mathrm{cl}_{pc}}{\mathrm{wm}_p}. \tag{4.3}$$

The range of values for δ_{pc} has an upper limit of 1 and no strict lower limit. δ_{pc} is the only deterministic variable which can be both positive and negative and where the sign has an actual meaning:

$\boldsymbol{\delta_p} > 0$ When δ_{pc} is positive, this means that there are more working memory capacities available than the learning task requires. This can be the case if the total cognitive load cl_{pc} is rather small or if a high ICL is reduced by a lot of prior knowledge or if the working memory capacities wm_p for this person are especially high. A positive value for δ_{pc} is favorable. However, if the value for δ_{pc} is near 1, this might indicate a boring task as there are no real demands imposed upon the learner by the task.

$\boldsymbol{\delta_p} = 0$ Whenever the working memory capacity wm_p equals the total cognitive load cl_{pc} imposed by the task, the free working memory capacity equals 0. On one side, this means that the learner is operating at their limit. On the other side, this might be the right condition for the learner to experience a state of flow or to enter the ZPD.

$\boldsymbol{\delta_p} < 0$ When δ_{pc} is negative, the total cognitive load cl_{pc} imposed by the task exceeds the learner's available working memory resources wm_p. Is this the case then the task consists of too many interactive elements that need to be considered at the same time, or the instructional design imposes a too high ECL.

The presented design is shared among all models. It is based on theoretical findings and experimental evidence from research on CLT. Different models differ only in the way they use the deterministic variables as the cause for the observations and in the number of layers used to model the relationship between variables. All observations also share the indices pc because, in the model, they are caused by the deterministic variables. So for each subject and each concept there exists a value for each of the observable variables. The different models analyzed in this thesis differ mainly in the way how they model the relationship between the observations and the latent variables.

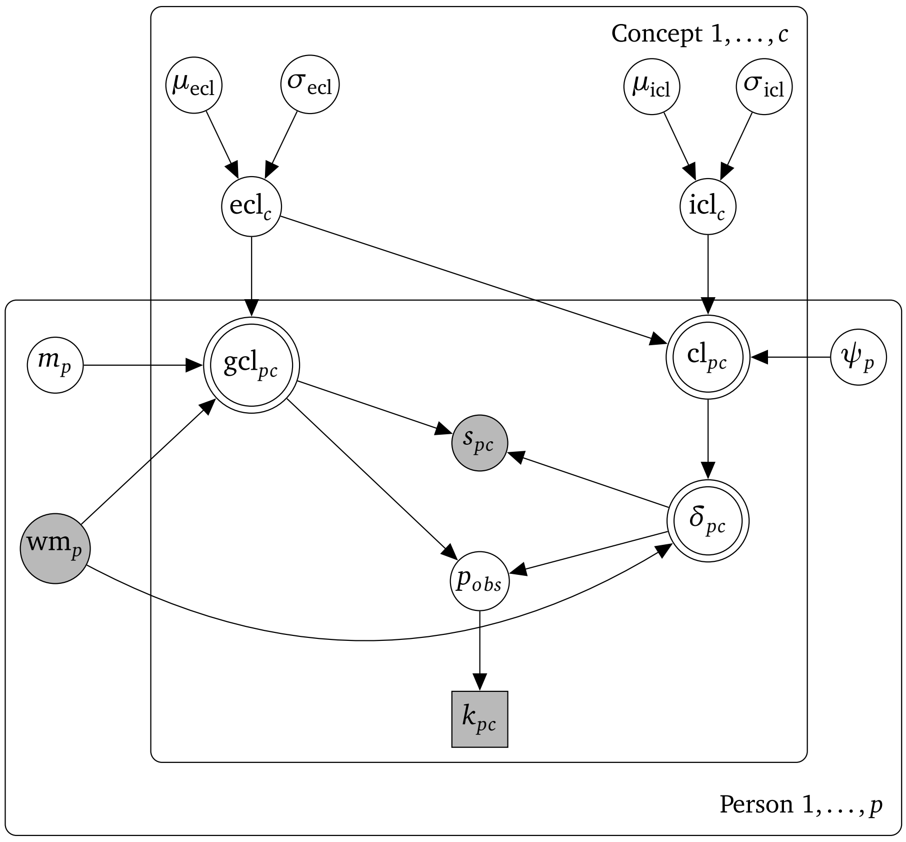
Due to time limitations it was not possible to find build and train a model that describes all observable variables. With the first three models that I am going to present in the next couple of paragraphs I focused on modeling the cause for the two observable parameters task success k_{pc} and mission score s_{pc}. After having succeeded in finding an appropriate model, I went on to model a third observable variable mission time t_{pc}. With the fourth model being able to successfully model three observable variables I validated the model on a wide range of observable data sets to assess the model's performance. I will discuss possible model extensions in section 5.4 How to Extend the Model Further (page 144) and in section 7.2 Open Questions and Future Work (page 156).

There is an important exception for all models regarding the latent variable for working memory capacity wm_p. This variable was originally thought to be latent and the model should have had the possibility to find an appropriate value for this variable for each subject and model individual differences regarding the performance of the working memory. Originally, the variable wm_p was modeled as normally distributed around a mean value of 7 with a standard deviation of 1 to reflect the knowledge about the general limitations of the working memory (Miller, 1956; Toh, 2005). However, during the implementation I realized the model used this variable to explain individual differences by setting wm_p to values near zero or far above 7. Besides the initial prior probability distribution centered around 7, the model was not constraint in the choice of wm_p values. So I decided to set this variable in all models to a value of 7 and treat it like a kind-of-observed variable. Of course, it is possible to turn the variable again into a latent variable and, For example use a more restricted range of allowed values. That is the reason for the node wm_p to be shaded in the following graphical models.

Model $\mathscr{M}_1$ – original CogIUM for two observations

The first fully specified descriptive model is called CogiumOrig2Obs, or $\mathscr{M}_1$ for shorter reference in this thesis, and is implemented in the class with the same name in the CogIUM package (Figure 4.15 on the following page). $\mathscr{M}_1$ models the two observable variables task success k_{pc} and mission score s_{pc}—hence the last part of the name: "2Obs".

The mission score s_{pc} is modeled as a beta distribution that is governed by the two parameters a and b. The beta distribution is a natural choice for a variable that should be interpreted as a success probability and is limited between 0 and 1. The reason for choosing $a = 0.5 + \mathrm{gcl}_{pc}$ and $b = 1 - \delta - pc$ is to reflect a credible behavior for values of either gcl_{pc} or δ_{pc} near 1 or 0 (Figure 4.16 on page 96). When gcl_{pc} is near zero, in most cases the probability for a low mission score s_{pc} is higher than for a high mission score. However, this has not to be the case for the combination of a low gcl_{pc} value and a high δ_{pc} value. The logic behind this is that when δ_{pc} is near 1, the learning task imposes so little cognitive load that not much gcl_{pc} is needed to deal with it. On the other side, if gcl_{pc} is high, the probability of a high mission score is in most cases higher than for a low mission score. Again, the only exception from this behavior is when δ_{pc} is near -1, because this indicates a cognitive load imposed by the learning task that is much higher than the available working memory resources. In this case, even a very high gcl_{pc} is not enough for successful learning because the learning material is to complex and consists of too many interactive elements or the instructional design is too bad.



Priors

$$m_p \sim \text{Beta}(3,1)$$
$$\psi_p \sim \text{Beta}(2,2)$$
$$\mu_{\text{icl}} \sim \text{Gamma}\,(2,0.5)$$
$$\mu_{\text{ecl}} \sim \text{Gamma}\,(2,0.5)$$
$$\sigma_{\text{icl}} \sim \text{HalfCauchy}\,(3)$$
$$\sigma_{\text{ecl}} \sim \text{HalfCauchy}\,(3)$$

Intermediate variables

$$\text{icl}_c \sim \text{Normal}\,(\mu_{\text{icl}},\sigma_{\text{icl}})_{\mathbb{I}(1,\infty)}$$
$$\text{ecl}_c \sim \text{Normal}\,(\mu_{\text{ecl}},\sigma_{\text{ecl}})_{\mathbb{I}(0,\infty)}$$
$$p_{obs} \sim \text{Beta}\left(1+\text{gcl}_{pc}, 1-\delta_{pc}\right)$$

Deterministic variables

$$\text{wm}_p = 7$$
$$\text{cl}_{pc} \leftarrow (1-\psi_p)\cdot\text{icl}_c + \text{ecl}_c$$
$$\text{gcl}_{pc} \leftarrow \begin{cases} 0, \text{ if } m_p\cdot\text{wm}_p - \text{ecl}_c < 0 \\ \dfrac{m_p\cdot\text{wm}_p - \text{ecl}_c}{\text{wm}_p}, \text{ else} \end{cases}$$
$$\delta_{pc} \leftarrow \frac{\text{wm}_p - cl_{pc}}{\text{wm}_p}$$

Observable variables

$$s_{pc} \sim \text{Beta}\left(0.5+\text{gcl}_{pc}, 1-\delta_{pc}\right)$$
$$k_{pc} \sim \text{Bern}\,(p_{obs})$$

Figure 4.15.: Graphical model of $\mathscr{M}_1$, as implemented in the CogIUM package. $\mathscr{M}_1$ models the cause for two observable variables: task success k_{pc} and mission score s_{pc}. s_{pc} is modeled as normally distributed and depends on both gcl$_{pc}$ and δ_{pc}. k_{pc} is modeled with a Bernoulli distribution with success probability p_{obs}. The notation for (hierarchical) Bayesian models was introduced in Table 2.2 on page 32.

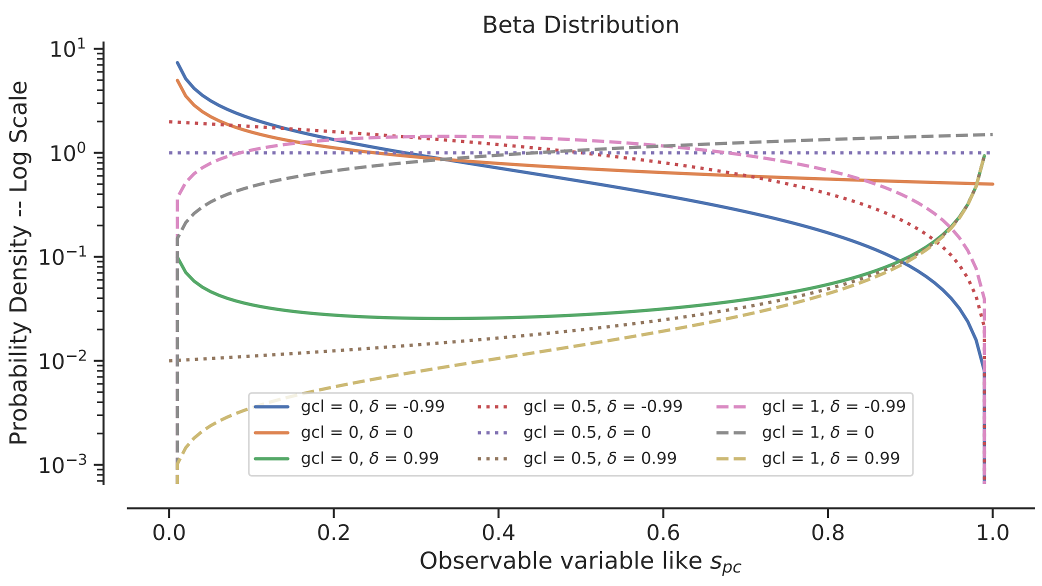


Figure 4.16.: Beta distribution for different values of gcl_{pc} and δ_{pc}. Line style is the same for curves with the same gcl_{pc} value. y-axis has logarithmic scale.

The task success k_{pc} is modeled as a Bernoulli distribution with success probability p_{obs}. The main advantage of the Bernoulli distribution is that it produces only binary outcomes, either success or failure, or 1 and 0, which is exactly the domain of the variable task success. The success probability p_{obs} is again modeled as a beta distribution. Very similar to the mission score s_{pc}, gcl_{pc} influences the a parameter and δ_{pc} influences the b parameter of the beta distribution. Here I chose the equations $a = 1 + \text{gcl}_{pc}$ and $b = 1 - \delta_{pc}$ because I wanted k_{pc} to have a chance of 50 % to be either 1 or 0 when $\text{gcl}_{pc} = 0$ and $\delta_{pc} = 0$, that is, when the learner dedicates no resources to the task and the learning task demands all available resources. A positive gcl_{pc} increases the success probability as well as a positive δ_{pc}. A negative gcl_{pc} decreases the success probability as well as a negative δ_{pc}.

The conceptual variables ICL icl_c and ECL ecl_c are modeled as normally distributed with a mean value μ_{icl} and a standard deviation σ_{icl}. Originally, both were intended to be modeled as discrete variables, but Bayesian inference for discrete variables is not as robust as with continuous variables and during the inference PyMC3 would have to use different MCMC samplers. This is due to the fact that key assumptions of MCMC convergence diagnostics are violated for discrete variables, or require a larger sample size (Deonovic and Smith, 2017). for Therefore, I decided to only use continuous distributions for latent variables. For interpretation, one can always treat a continuous variable like a discrete variable by discretization. Because icl_c and ecl_c both represent the interactive elements of the learning material, one discretization would be to round to the nearest integer value. Because a normal distribution is not limited in its codomain, the subscripts $\mathbb{I}[1, \infty]$ and $\mathbb{I}[0, \infty]$ mean that the values of the normal distribution are limited to the given range. The priors for μ_{icl} and μ_{ecl} should be positive and non-informative, so a gamma distribution with parameters 2 and 0.5 was chosen[5]. This choice slightly favors smaller values over higher values but does not rule out higher values. Likewise, the associated standard deviations σ_{icl} and σ_{ecl} should be positive and non-informative, so a half-Cauchy distribution with parameter 3 was

[5] See `https://docs.pymc.io/api/distributions/continuous.html` for a definition of probability distributions along with examples for different parameter values for all distributions used in the Bayesian models.

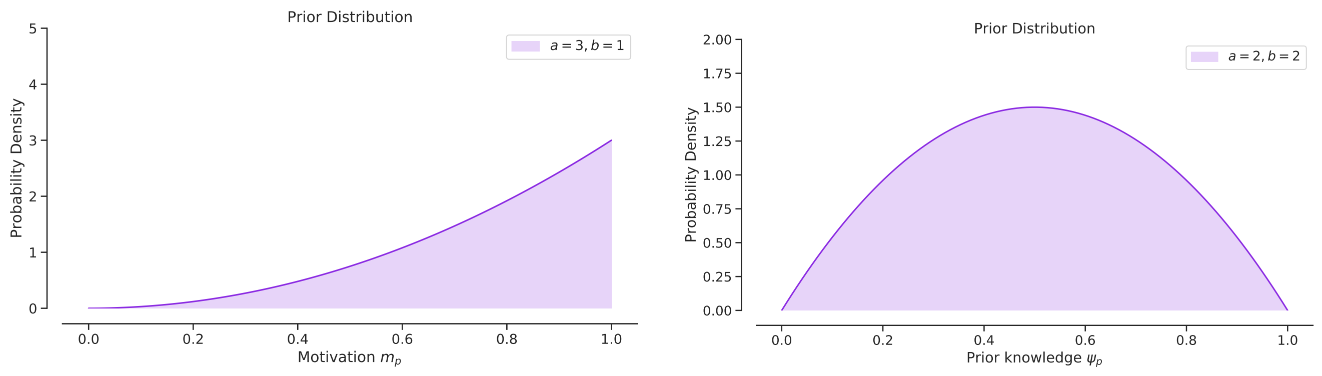


Figure 4.17.: Shape of beta distribution for (left) motivation m_p and (right) prior knowledge ψ_p.

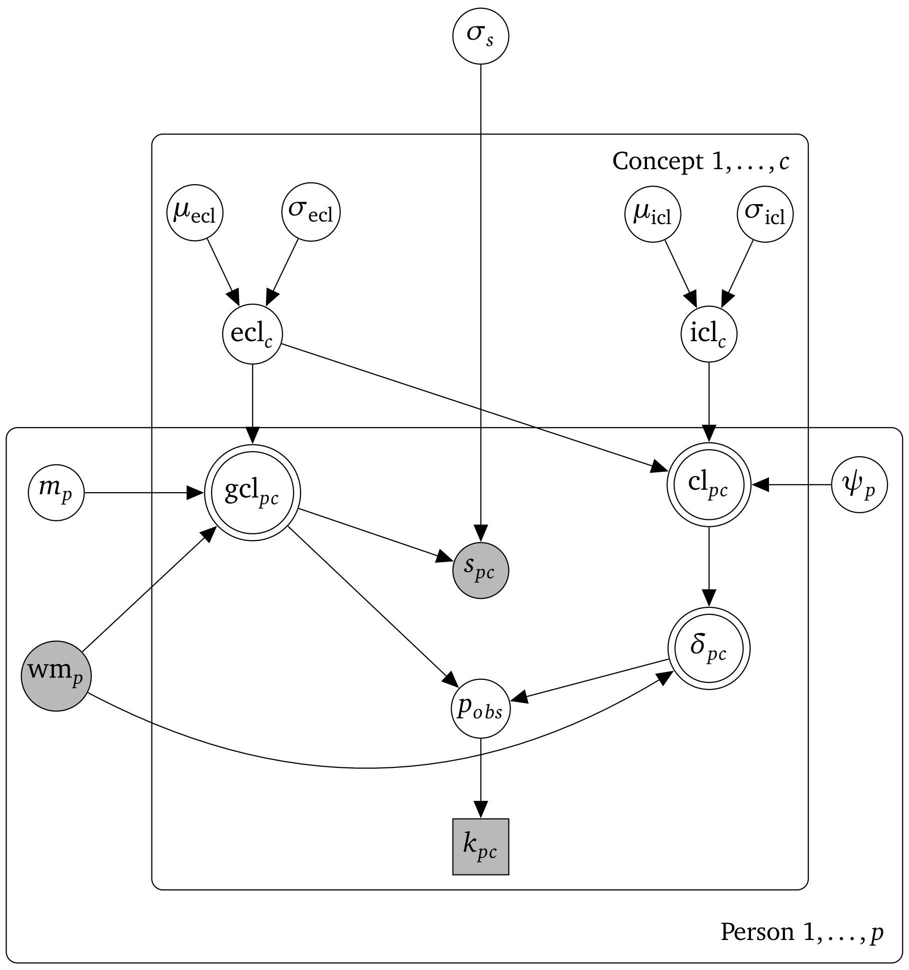
chosen. Again, this slightly favors smaller values over higher values, but the higher the β parameter of the half-Cauchy distribution, the more the distribution resembles a uniform distribution. The choice of priors follows in general the recommendations of Gelman (2006) for priors in hierarchical models. It is important to understand that both mean and standard deviation are group variables and that is the reason why they are placed within the concept plate in the graphical model. This means that each concept c has its own values for μ_{icl}, μ_{ecl} and $\sigma_{icl}, \sigma_{ecl}$, which are then used to draw values for icl_c and ecl_c. There are no global variables for the conceptual variables so each concept is completely independent of other concepts in the model. To change this, one could introduce global hyper-parameters for the group variables μ and σ.

The last variables that require an explanation are the personal variables for motivation m_p and for prior knowledge ψ_p. Both are modeled as group variables and drawn from a beta distribution with parameters $a = 3$ and $b = 1$ for motivation and with $a = b = 2$ for prior knowledge (Figure 4.17). The choice of any prior distribution is a commitment to certain assumptions and should be accepted by a skeptical scientific audience. For motivation I made the assumption that it is more likely for a person to be a priori highly motivated than to be less motivated. This reflects the findings from research about motivation in players of digital educational games. For prior knowledge I made the assumption that its value is a priori centered around the value 0.5 and that it is more and more unlikely that a person knows anything ($\psi_p = 1$), or nothing ($\psi_p = 0$) about a concept. Because they are group variables of the person, they are placed within the person plate in the graphical model. They differ between subjects but not between concepts. However, because there are no global variables, m_p and ψ_p are independent between subjects.

Model $\mathscr{M}_2$ – improved CogIUM for two observations

The second fully specified descriptive model is called CogiumImproved2Obs, or $\mathscr{M}_2$ for shorter reference in this thesis, and is implemented in the class with the same name in the CogIUM package (Figure 4.18 on the next page). $\mathscr{M}_2$ models the two observable variables task success k_{pc} and mission score s_{pc}—hence the last part of the name: “2Obs”.

Model $\mathscr{M}_2$ differs from $\mathscr{M}_1$ only in the way how the mission score s_{pc} is modeled. Instead of a beta distribution, s_{pc} is now modeled as normally distributed, with a mean value determined by gcl_{pc} and a global standard deviation σ_s. In this model, δ_{pc} has no influence on the value of s_{pc}. Instead, s_{pc} is much more dependent from gcl_{pc} and is centered around its value, which can be interpreted as that the value of gcl_{pc} is a direct forecast for the value of s_{pc}. A global standard deviation σ_s means that all subjects share this standard deviation over all concepts, so s_{pc} deviates from its center value gcl_{pc} always about the same value σ_s.



Priors

$$\begin{aligned}
m_p &\sim \mathrm{Beta}(3,1)\\
\psi_p &\sim \mathrm{Beta}(2,2)\\
\mu_{\mathrm{icl}} &\sim \mathrm{Gamma}(2,0.5)\\
\mu_{\mathrm{ecl}} &\sim \mathrm{Gamma}(2,0.5)\\
\sigma_{\mathrm{icl}} &\sim \mathrm{HalfCauchy}(3)\\
\sigma_{\mathrm{ecl}} &\sim \mathrm{HalfCauchy}(3)\\
\sigma_s &\sim \mathrm{HalfCauchy}(1)
\end{aligned}$$

Intermediate variables

$$\begin{aligned}
\mathrm{icl}_c &\sim \mathrm{Normal}(\mu_{\mathrm{icl}},\sigma_{\mathrm{icl}})_{\mathbb{I}(1,\infty)}\\
\mathrm{ecl}_c &\sim \mathrm{Normal}(\mu_{\mathrm{ecl}},\sigma_{\mathrm{ecl}})_{\mathbb{I}(0,\infty)}\\
p_{obs} &\sim \mathrm{Beta}\left(1+\mathrm{gcl}_{pc},1-\delta_{pc}\right)
\end{aligned}$$

Deterministic variables

$$\begin{aligned}
\mathrm{wm}_p &= 7\\
\mathrm{cl}_{pc} &\leftarrow (1-\psi_p)\cdot\mathrm{icl}_c+\mathrm{ecl}_c\\
\mathrm{gcl}_{pc} &\leftarrow \begin{cases} 0, \text{ if } m_p\cdot\mathrm{wm}_p-\mathrm{ecl}_c<0\\ \dfrac{m_p\cdot\mathrm{wm}_p-\mathrm{ecl}_c}{\mathrm{wm}_p}, \text{ else}\end{cases}\\
\delta_{pc} &\leftarrow \frac{\mathrm{wm}_p-cl_{pc}}{\mathrm{wm}_p}
\end{aligned}$$

Observable variables

$$\begin{aligned}
s_{pc} &\sim \mathrm{Normal}\left(\mathrm{gcl}_{pc},\sigma_s\right)\\
k_{pc} &\sim \mathrm{Bern}(p_{obs})
\end{aligned}$$

Figure 4.18.: Graphical model of $\mathscr{M}_2$, as implemented in the CogIUM package. $\mathscr{M}_2$ models the cause for two observable variables: task success k_{pc} and mission score s_{pc}. In comparison to $\mathscr{M}_1$, $\mathscr{M}_2$ models s_{pc} with a normal distribution that is only determined by gcl_{pc} and no longer by δ_{pc}. The notation for HBMs was introduced in Table 2.2 on page 32.

Model $\mathscr{M}_3$ – hierarchical CogIUM for two observations

The third fully specified descriptive model is called CogiumImprovedHierarchical2Obs, or $\mathscr{M}_3$ for shorter reference in this thesis, and is implemented in the class with the same name in the CogIUM package (Figure 4.19 on page 101). $\mathscr{M}_3$ models the two observable variables task success k_{pc} and mission score s_{pc}—hence the last part of the name: "2Obs".

Model $\mathscr{M}_3$ differs from $\mathscr{M}_2$ and $\mathscr{M}_1$ only in the way how the mission score s_{pc} is modeled. Instead of a beta distribution, s_{pc} is now modeled as normally distributed, like in $\mathscr{M}_2$, but with a mean value determined by both gcl_{pc} and δ_{pc} and a global standard deviation σ_s. In this model, δ_{pc} has again an influence on the value of s_{pc}. The mean value of the normal distribution is computed as a sum of gcl_{pc} and δ_{pc}. δ_{pc} is mitigated by a multiplicative factor ν_s, which governs the percentage of how much δ_{pc} is added to gcl_{pc}. ν_s is a personal group variable, meaning that each subject has their own value for ν_s. However, ν_s is also hierarchically modeled, because all ν_s are dependent on the global distributions for μ_ν and σ_ν. This introduces shrinkage between the individual values for ν_s because different ν_s variables are linked together by the global variables. A global standard deviation σ_s means that all subjects share this standard deviation over all concepts, so s_{pc} deviates from its center value $\mathrm{gcl}_{pc} + \nu_s \cdot \delta_{pc}$ always about the same value σ_s.

Model $\mathscr{M}_4$ – hierarchical CogIUM for three observations

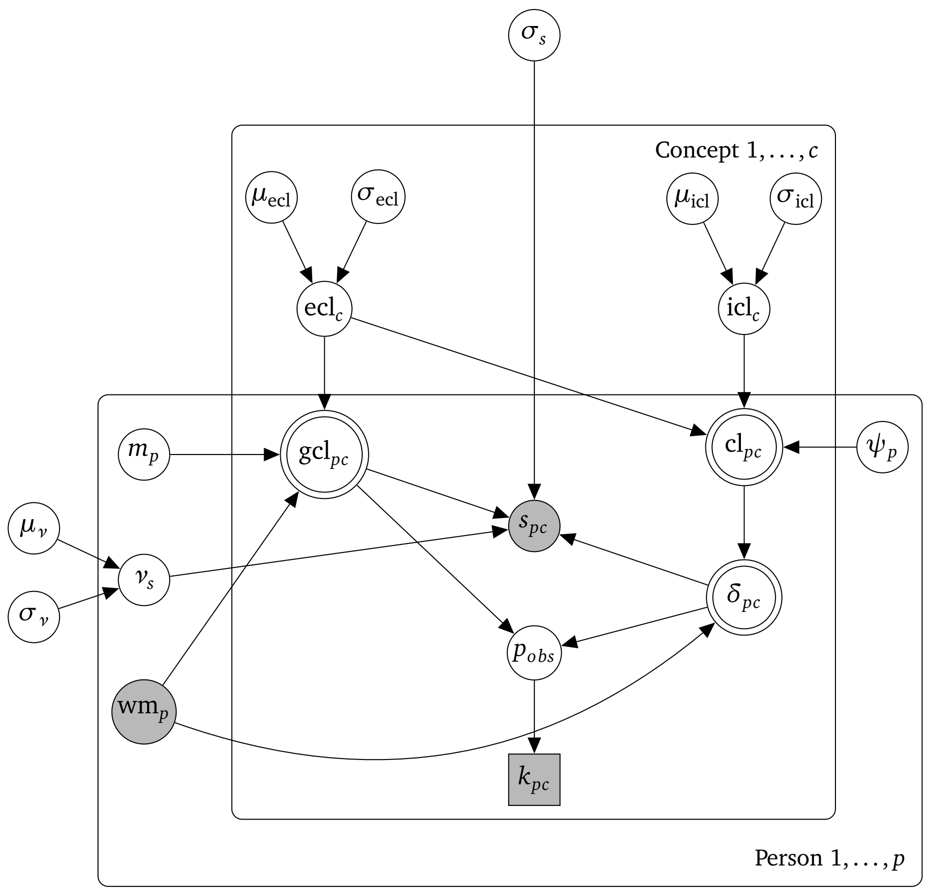
The fourth fully specified descriptive model is called CogiumImprovedHierarchical3Obs, or $\mathscr{M}_4$ for shorter reference in this thesis, and is implemented in the class with the same name in the CogIUM package (Figure 4.20 on page 102). $\mathscr{M}_4$ models the three observable variables task success k_{pc}, mission score s_{pc}, and mission time t_{pc}—hence the last part of the name: "3Obs".

Model $\mathscr{M}_4$ extends $\mathscr{M}_3$ by modeling the cause of the mission time t_{pc}. The mission time is modeled as a linear combination of three distinct parts: a minimal time t_{min}, a contribution from gcl_{pc}, and a contribution from δ_{pc}. The minimal time is modeled as a concept group variable because I assumed each concept to have a minimal time that is required to complete or learn the concept independent of the learner's effort. For the game Lost Earth 2307, For example this is the case because each mission requires a minimal number of steps to successfully complete the mission like accepting the mission, deploying an appropriate sensor in the hangar, and processing the raw image back on the bridge. The time required for these actions cannot be decreased further. Thus, every concept has a minimal time required to complete this concept.

This minimal time can be increased further either by gcl_{pc} or δ_{pc}. I assumed no increase in mission time when the learner invests all resources in learning, that is when gcl_{pc} is 1. The less resources are invested in learning, the more the time required to finish the concept should increase. As gcl_{pc} is limited to the range of $[0,1]$, a multiplicative factor α_t is needed to obtain values that are valid for the mission time, which is measured in minutes. α_t is limited to positive values so that the logic of the influence of gcl_{pc} is not turned around should α_t become negative. The minimal mission time is also not increased when δ_{pc} is zero. If δ_{pc} is positive, that is, there are free working memory resources, then the time actually decreases. This contradicts the assumption that the minimal mission time cannot be decreased, but as δ_{pc}

is allowed to have positive and negative values, it is hard to find a good parametrization that can deal with both positive and negative values and still be interpretable. I decided to implement the model in this way to have a valid interpretation of the core conceptual variables gcl_{pc} and δ_{pc}. If δ_{pc} becomes negative, that is, there are no free working memory resource, the minimal time increased, as the learner has to deal with a higher task demand than they can cope with. Again, β_t is limited to positive values so that the logic of the influence of δ_{pc} is not turned around. α_t and β_t are personal group variables, which means that each subject has their own values for these variables. Like ν_s before, α_t and β_t are modeled hierarchically and are dependent on global parameters σ_α and σ_β. All subjects share these global variables, but draw their own values from the global distributions so that each subject can have their own α_t and β_t values.

When compared with s_{pc} and k_{pc}, t_{pc} is the variable with the most complex model. k_{pc} only depends on the deterministic variables gcl_{pc} and δ_{pc} and not on any other variable. s_{pc} also depends on the deterministic variables, and, in addition, on global and personal group variables. Finally, t_{pc} depends on deterministic variables, on both personal and conceptual group variables, and on global variables. Table A.1 on page 168 gives a detailed overview about all models, the total number of model parameters and all model variables according to their hierarchical level.



Priors

$$m_p \sim \text{Beta}(3,1)$$
$$\psi_p \sim \text{Beta}(2,2)$$
$$\mu_{\text{icl}} \sim \text{Gamma}(2,0.5)$$
$$\mu_{\text{ecl}} \sim \text{Gamma}(2,0.5)$$
$$\mu_\nu \sim \text{Normal}(0.5,0.01)$$
$$\sigma_{\text{icl}} \sim \text{HalfCauchy}(3)$$
$$\sigma_{\text{ecl}} \sim \text{HalfCauchy}(3)$$
$$\sigma_s \sim \text{HalfCauchy}(1)$$
$$\sigma_\nu \sim \text{HalfCauchy}(1)$$

Intermediate variables

$$\text{icl}_c \sim \text{Normal}(\mu_{\text{icl}},\sigma_{\text{icl}})_{\mathbb{I}(1,\infty)}$$
$$\text{ecl}_c \sim \text{Normal}(\mu_{\text{ecl}},\sigma_{\text{ecl}})_{\mathbb{I}(0,\infty)}$$
$$p_{obs} \sim \text{Beta}\left(1+\text{gcl}_{pc},1-\delta_{pc}\right)$$
$$\nu_s \sim \text{Normal}(\mu_\nu,\sigma_\nu)_{\mathbb{I}(0,1)}$$

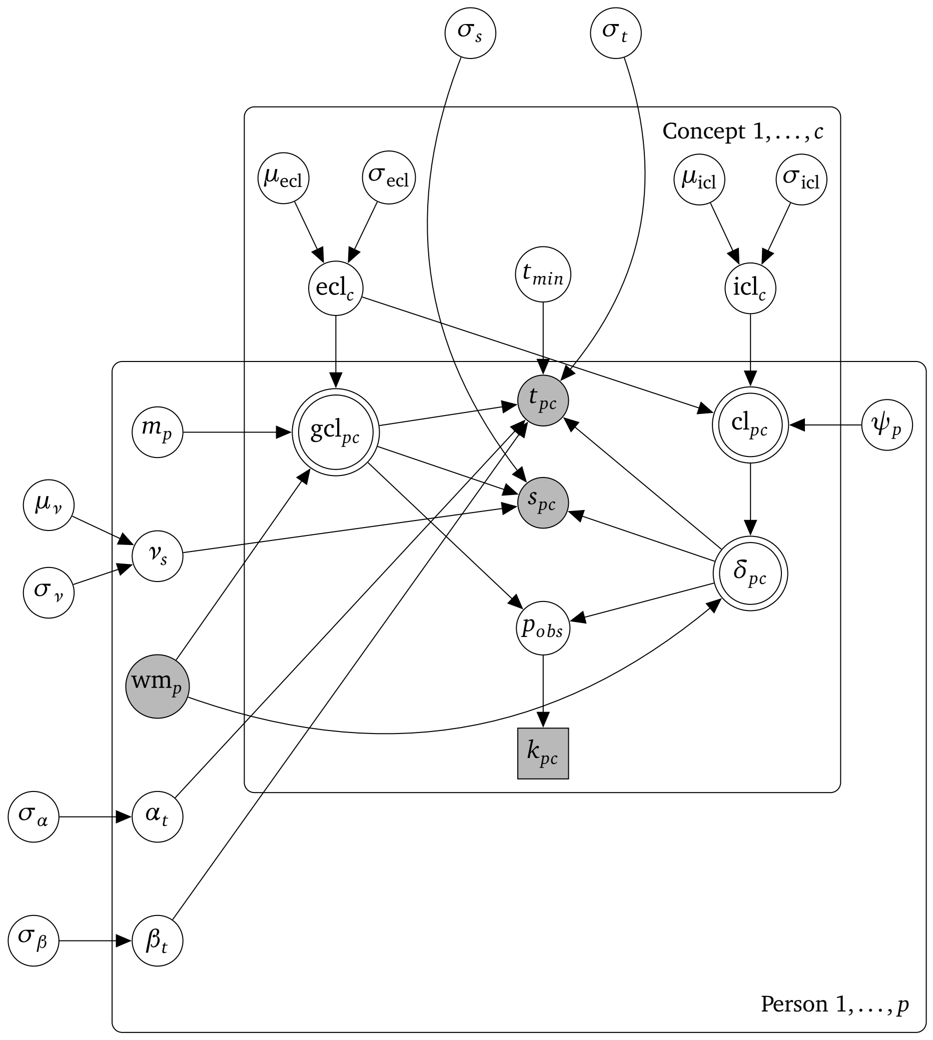
Deterministic variables

$$\text{wm}_p = 7$$
$$\text{cl}_{pc} \leftarrow (1-\psi_p)\cdot\text{icl}_c + \text{ecl}_c$$
$$\text{gcl}_{pc} \leftarrow \begin{cases} 0, \text{ if } m_p\cdot\text{wm}_p - \text{ecl}_c < 0 \\ \dfrac{m_p\cdot\text{wm}_p - \text{ecl}_c}{\text{wm}_p}, \text{ else} \end{cases}$$
$$\delta_{pc} \leftarrow \frac{\text{wm}_p - cl_{pc}}{\text{wm}_p}$$

Observable variables

$$s_{pc} \sim \text{Normal}\left(\text{gcl}_{pc} + \nu_s\cdot\delta_{pc},\sigma_s\right)$$
$$k_{pc} \sim \text{Bern}(p_{obs})$$

Figure 4.19.: Graphical model of $\mathscr{M}_3$, as implemented in the CogIUM package. $\mathscr{M}_3$ models the cause for two observable variables: task success k_{pc} and mission score s_{pc}. $\mathscr{M}_3$ is based on $\mathscr{M}_2$, but extends the normal distribution for s_{pc} to be also dependent on δ_{pc} with a multiplicative factor ν_s, which is hierarchically modeled. The notation for HBMs was introduced in Table 2.2 on page 32.



Priors

$$
\begin{aligned}
m_p &\sim \text{Beta}(3,1)\\
\psi_p &\sim \text{Beta}(2,2)\\
t_{min} &\sim \text{Gamma}(2,0.5)\\
\mu_{\text{icl}} &\sim \text{Gamma}(2,0.5)\\
\mu_{\text{ecl}} &\sim \text{Gamma}(2,0.5)\\
\mu_\nu &\sim \text{Normal}(0.5,0.01)\\
\sigma_{\text{icl}} &\sim \text{HalfCauchy}(3)\\
\sigma_{\text{ecl}} &\sim \text{HalfCauchy}(3)\\
\sigma_s &\sim \text{HalfCauchy}(1)\\
\sigma_t &\sim \text{HalfCauchy}(1)\\
\sigma_\nu &\sim \text{HalfCauchy}(1)\\
\sigma_\alpha &\sim \text{HalfCauchy}(1)\\
\sigma_\beta &\sim \text{HalfCauchy}(1)
\end{aligned}
$$

Intermediate variables

$$
\begin{aligned}
\text{icl}_c &\sim \text{Normal}(\mu_{\text{icl}},\sigma_{\text{icl}})_{\mathbb{I}(1,\infty)}\\
\text{ecl}_c &\sim \text{Normal}(\mu_{\text{ecl}},\sigma_{\text{ecl}})_{\mathbb{I}(0,\infty)}\\
p_{obs} &\sim \text{Beta}\left(1+\text{gcl}_{pc},1-\delta_{pc}\right)\\
\nu_s &\sim \text{Normal}(\mu_\nu,\sigma_\nu)_{\mathbb{I}(0,1)}\\
\alpha_t &\sim \text{HalfNormal}(\sigma_\alpha)\\
\beta_t &\sim \text{HalfNormal}\left(\sigma_\beta\right)
\end{aligned}
$$

Deterministic variables

$$
\begin{aligned}
\text{wm}_p &= 7\\
\text{cl}_{pc} &\leftarrow (1-\psi_p)\cdot\text{icl}_c+\text{ecl}_c\\
\text{gcl}_{pc} &\leftarrow \begin{cases} 0, \text{ if } m_p\cdot\text{wm}_p-\text{ecl}_c<0\\ \dfrac{m_p\cdot\text{wm}_p-\text{ecl}_c}{\text{wm}_p}, \text{ else}\end{cases}\\
\delta_{pc} &\leftarrow \frac{\text{wm}_p-cl_{pc}}{\text{wm}_p}
\end{aligned}
$$

Observable variables

$$
\begin{aligned}
s_{pc} &\sim \text{Normal}\left(\text{gcl}_{pc}+\nu_s\cdot\delta_{pc},\ \sigma_s\right)\\
k_{pc} &\sim \text{Bern}(p_{obs})\\
t_{pc} &\sim \text{Normal}(t_{min}+\alpha_t\cdot(1-\text{gcl}_{pc})-\\
&\qquad \beta_t\cdot\delta_{pc},\ \sigma_t)
\end{aligned}
$$

Figure 4.20.: Graphical model of $\mathscr{M}_4$, as implemented in the CogIUM package. $\mathscr{M}_4$ models the cause for three observable variables: task success k_{pc}, mission score s_{pc}, and mission time t_{pc}. $\mathscr{M}_4$ is based on $\mathscr{M}_3$, but extends the model by incorporating t_{pc} as additional observable variable, modeled as normally distributed with hierarchically modeled parameters. The notation for HBMs was introduced in Table 2.2 on page 32.

5 Implementation

This chapter deals with the implementation of the cognitive intelligent user model based on the concepts of the previous chapter. Because hierarchical Bayesian models were chosen as the modeling tool, a software library was necessary that allows for computing Bayesian inference and implementing Bayesian models. Such a framework is the probabilistic programming library PyMC3 for Python, which is presented in the first chapter. The second chapter provides an in-depth description of the developed Python package CogIUM, which enables the user to build, train and validate cognitive user models based on HBMs. The validation chapter takes up the most space and shows in detail the model comparison and the model extension process. To better understand this chapter, knowledge about HBMs, Bayesian data analysis, the MCMC procedure, predictive accuracy metrics and the CLT are helpful, all of which was provided in previous sections.

5.1 Bayesian Inference with PyMC3 – Fourth Step of Bayesian Data Analysis

Probabilistic programming allow the formulation of inference problems and the computation of their solutions (Meent et al., 2018). Probabilistic programming aims at offering a tool chain that supports supervised, unsupervised, and semi-supervised inference. Meent et al. provide an in-depth introduction to probabilistic programming that covers the basics from language design to evaluator implementation.

Today, there exist multiple software packages that allow probabilistic programming and computation. The most common are a) Stan[1]—for which many interfaces to popular programming languages exist, for example RStan for R and PyStan for Python, b) JAGS[2]—mainly designed to work closely with R, and c) PyMC3[3]—the current version and successor of PyMC, implemented in Python. I decided to work with PyMC3 because Python, as a programming language, is very common, widespread, and natively supported by most operating systems, because PyMC3 is actively developed, very good supported, and provides a lot of in-depth guides and helpful material. Besides, choosing a Python package allowed the implementation to stay in a complete Python-only ecosystem without third-party dependencies because the whole model logic and all helper functions are written purely in Python.

PyMC3 is a probabilistic programming package for Python that allows for building and fitting Bayesian models using a variety of numerical methods, such as MCMC and variational inference (Salvatier et al., 2016). In addition, PyMC3 offers functionality for summarizing the output of the inference process and for model diagnostics. PyMC3 makes Bayesian modeling as simple as possible, allowing the user to focus on the scientific problem, rather than on the numerical methods. PyMC3 offers a large suite of well-documented statistical distributions to choose from. It uses Theano[4], a Python library that allows for defining, optimizing, and evaluation of mathematical expressions, as the computational backend,

1 https://mc-stan.org/
2 http://mcmc-jags.sourceforge.net/
3 https://docs.pymc.io/
4 http://deeplearning.net/software/theano/

which provides fast expression evaluation, automatic gradient calculation, and GPU computing. PyMC3 is extensible, so that the user can incorporate custom step methods and unusual probability distributions as the need arises.

The development on PyMC began in 2003 and lasted until 2006 with the aim to making MCMC more accessible to applied scientists. David Huard, Anand Patil, and Chris Fonnesbeck began the development on the next version, PyMC2, in 2006 and development went on until 2013. With the help of John Salvatier, PyMC3 was first released 2015. The latest release of PyMC3 is version 3.0 from January 2017. For an introduction to PyMC3 see Salvatier et al. (2016).

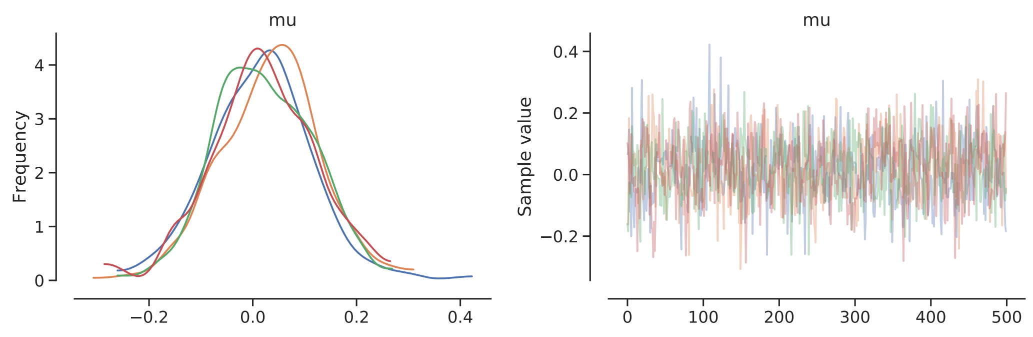
Example: A Simple PyMC3 Model

Working with PyMC3 is really easy and straight forward (Listing 5.1 on page 107). In this example we try to fit a uni-variate normal distribution to 100 observations drawn from the standard normal distribution. Therefore, our model can be stated as $y \sim \mathcal{N}(\mu, \sigma)$. We set the standard deviation to $\sigma = 1$, leaving the mean value μ as the only unknown latent variable here. Latent because μ is not directly observable from the data and has to be inferred somehow.

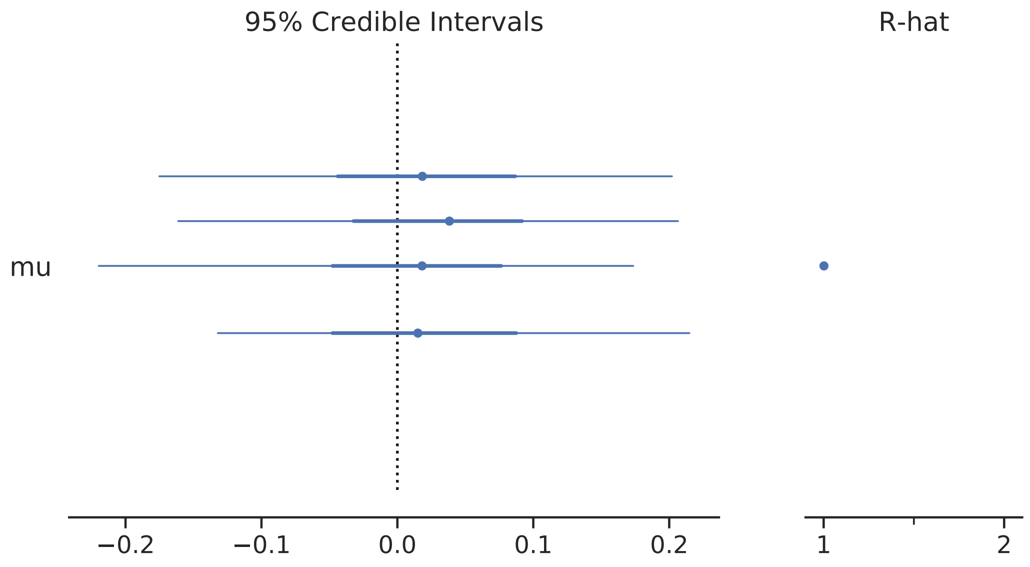
The package itself is loaded into any Python program with the `import pymc3 as pm` command (line 2), that imports the module and make it available via the 'pm'-alias. A model is specified in PyMC3 by the use of the `Model` class. Usually, the creation of a new model and the specification of the model happen simultaneously as port of a `with` context (line 4). To build our model, we use the `Normal` distribution of the PyMC3 package and specify the standard deviation with 1 and the mean value with some random variable `mu` (line 6). In addition, we pass the observations to the `Normal` class so that PyMC3 knows that this line is the likelihood of our model. All that is left to do is to specify the latent variable `mu`, which we model as samples from a normal distribution with mean value 0 and standard deviation 1 (line 5). This is the prior of our model for the parameter `mu`. This means that `mu` can, in principle, take any value from the realm of the real numbers because the normal distribution is not restricted to any interval, but that most of the mass of its probability distribution is initially centered around the value 0. Thus, our prior believe is that the value of `mu` will be around this value. That is all we have to do to define the model.

Bayesian inference is than performed to approximate the posterior distribution. In this case we use MCMC sampling (line 8). On default, PyMC3 uses the NUTS sampler for continuous variables, a very efficient sampler even for complex models. There are more default settings like the number of samples, the number of samples for the tuning phase, the number of chains, and so on. The result of the Bayesian inference is stored in the `trace` variable, which represents a PyMC3 `MultiTrace` object. The sampling process for this example took around one second for 1000 samples and four chains, that are 4000 samples in total, without any warnings.

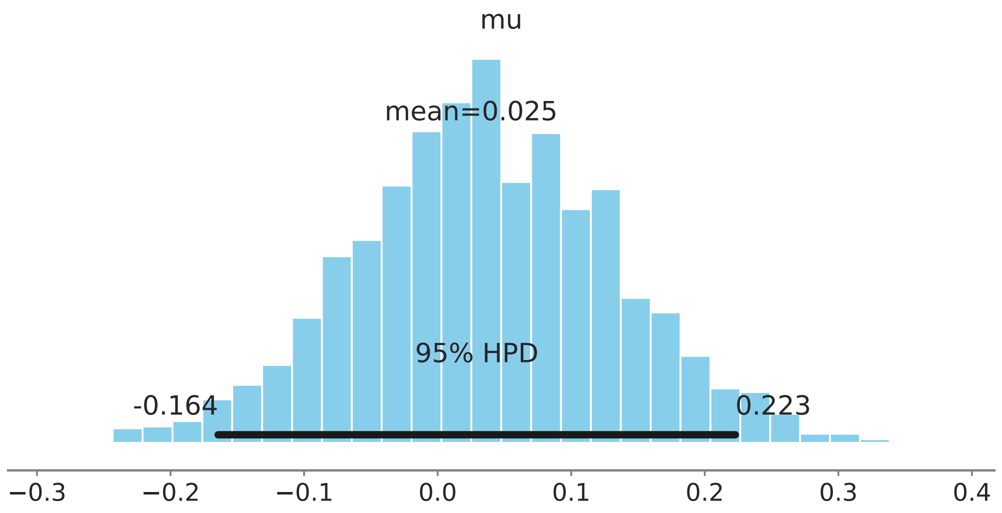
Having completed the inference, we are usually interested in the quality of the posterior's approximation and the shape of the posterior. The quality of the approximation can be visually analyzed by the `pm.traceplot(<trace>)` function (line 10), which outputs for each parameter of the model the posterior distribution and the traceplot of all chains (Figure 5.1a on the following page). The traceplot shows no signs for diverging chains and the posterior distributions seem plausible. Another way to check for diverging chains is to look at the so-called forestplot, created by the `pm.forestplot(<trace>)` function (line 12) (Figure 5.1b on the next page). A value for $\hat{R} = 1$ indicates convergence of the chains, as do the overlapping credible intervals. The summary statistics of the trace for each model parameter can be conveniently printed out with the help of the `pm.summary(<trace>)` function (line 11). The final result of the Bayesian inference is the posterior distribution, which can be plotted by calling the `pm.plot_posterior(<trace>)` function (line 13) (Figure 5.1c on the following page). To further process the samples gained during the inference process, one has access to all samples by calling `trace.get_values()`, which accepts the parameter name as the first argument.



(a) Posterior distribution and traceplot as outputted by `pm.traceplot()`. (Left) Kernel density estimation of the model parameter `mu` for each of the four chains. (Right) Traceplot for each of the four chains. All chains have converged to the same region of the parameter space and the posterior approximations are similar.



(b) Forestplot as outputted by `pm.forestplot()`. (Left) The 95 % credible intervals for all four chains. (Right) The Gelman-Rubin $\hat{R}$ statistic for the model parameter `mu`.



(c) Posterior distribution as outputted by `pm.plot_posterior()`. The posterior distribution is visualized as histogram over the samples from the MCMC sampler. Either the median or the mean value are shown as point estimate. The 95 % HDI is shown with both its end points.

Figure 5.1.: Output for the PyMC3 example of Section 5.1 on the previous page, generated by Listing 5.1 on the following page.

Listing 5.1: A simple, but fully functional example with PyMC3

```python
import numpy as np
import pymc3 as pm

with pm.Model() as model:
   mu = pm.Normal('mu', mu=0, sd=1)
   obs = pm.Normal('obs', mu=mu, sd=1, observed=np.random.randn(100))

   trace = pm.sample(1000, tune=500)

pm.traceplot(trace)
pm.summary(trace)
pm.forestplot(trace)
pm.plot_posterior(trace)
```

5.2 The CogIUM Python Package

A natural way in Python to structure code is with the help of custom modules and packages. A module[5] is nothing more than a simple Python file containing definitions and statements. The filename is the module name with the file ending `.py`. A module can contain executable statements and function definitions. The statements are executed only the first time the module is imported via an **`import`** statement. Functions are not directly executed but instead being imported explicitly either with the **`from`** `<module>` **`import`** `<function1>[, <function2>]` statement or as a bulk import via **`from`** `<module>` **`import`** `*`, though the last statement is considered bad practice. Thus, a custom module behaves exactly like a a module from the Python standard library and working with custom modules is no different than working with a standard module.

However, modules are not suited for larger projects when code is spread over multiple module files. In this case, a namespace is desirable that unifies all module files under a shared namespace. This is possible with Python packages[6], which are a collection of modules that share a namespace. In a Python package modules are addressed by "dotted module names". For example, the module name `A.B` designates a submodule `B` in a package named `A`. Submodule in this context means that the module is part of a Python package, not that it is a module under another module, which is not possible in Python. What is possible, however, is a nested package structure with subpackages under the main package. To recognize a package, Python scans all directories for a `__init__.py` file. This file can be empty as well as contain executable initialization code for the package or set the `__all__` variable. The `__all__` variable allows a package author to define which submodules or subpackages are loaded when the user writes **`from`** `<module>` **`import`** `*`.

The Python package I created for this thesis is named *cogium*, following the convention of Python package names written only in all-lowercase, and encapsulates all necessary files to reproduce the Bayesian data analysis conducted as part of this thesis. The name of the package stands for

[5] https://docs.python.org/3/tutorial/modules.html
[6] https://docs.python.org/3/tutorial/modules.html#packages

cognitive intelligent user model (CogIUM) and I will use CogIUM throughout the rest of this work to reference the package and the models it contains.

The structure of the CogIUM package is as follows:

```
.
├── cogium
│   ├── model
│   │   ├── BaseModel.py
│   │   ├── CogiumImproved2Obs.py
│   │   ├── CogiumImprovedHierarchical2Obs.py
│   │   ├── CogiumImprovedHierarchical3Obs.py
│   │   ├── CogiumOrig2Obs.py
│   │   ├── __init__.py
│   ├── __init__.py
│   ├── generator.py
│   ├── plots.py
│   └── utils.py
├── db
│   ├── models
│   │   ├── CogiumOrig2Obs
│   │   ├── Improved2Obs
│   │   ├── ImprovedHierarchical2Obs
│   │   └── ImprovedHierarchical3Obs
│   └── traces
│        ├── CogiumOrig2Obs
│        ├── Improved2Obs
│        ├── ImprovedHierarchical2Obs
│        └── ImprovedHierarchical3Obs
├── docs
├── nb
├── output
│   ├── ImprovedHierarchical3Obs
│   └── ModelComparison
├── MANIFEST.in
├── README.md
└── setup.py
```

cogium The directory `cogium` contains the actual Python package with a subpackage `cogium.model` that holds the model definitions as class files, the submodule `generator.py` for the data generating functionality, the submodule `plots.py` for the plotting functions, and the submodule `utils.py` for minor helping functions used in the other submodules.

db The directory `db` contains the stored model files for a particular data set as well as the associated traces from the PyMC3 runs. Each call of `BaseModel::save()` creates a new file in db/models/<model>/. Each call of the sample() method with parameter save_trace=True will create a folder in db/traces/<model>/.

docs The directory docs contains the setup and auto-generated files from the Sphinx Documentation Generator[7].

nb The directory nb contains all Jupyter Notebooks[8] developed during this thesis to run the simulations.

output The directory output contains all the figures produced by the plotting functions, separated by each model and data set.

setup.py The files MANIFEST.in, README.md and setup.py belong to a complete package setup for easier distribution and installation[9]. The setup.py defines the meta data of the package and lists all dependencies.

With this setup, local installation is quite simple. All that is needed is Python in the version 3.6 or higher and the Python package installer pip. The package can be installed from the command line with `$ pip install . -e`. The option -e installs the package with a symlink, so that changes to the source files will be immediately available. pip will install all required dependencies and the CogIUM package. After the installation you can verify a successful installation by running `import cogium` in a Python console, from a Python script or a Jupyter Notebook. The CogIUM package should be locally available to you no matter your current working location.

5.2.1 The data generator

The submodule generator.py of the CogIUM package provides a single function generate_observations(). This function allows the generation of expected observations gained from a learner's interaction with the game Lost Earth 2307. As described in section 4.2.1 on page 78, observable variables differ in domain and numerical type. Furthermore, as was discussed in section 4.3.2 on page 88, the model not only operates on a single observation gained from a single learner in a single mission, but supports multiple observations from different learners and from multiple missions. Thus, generate_observations() uses the NumPy data type ndarray to return a three dimensional array. The first dimension is the number of subjects, the second dimension is the number of concepts/missions and the third dimension is the number of observations. The third dimension suffices because all observations are modeled as scalar values. Further, the data generator supports the split of subjects into distinct groups. To simulate different groups, the data generator uses a Python dictionary (Listing 5.2 on page 111) as mapping with the four group levels 'good', 'average', 'bad', and 'random' as keys and the data generator functions as values. The user can define the number of groups, as well as the kind of the group and the number of subjects for this group.

The task success k_{pc} is a binary variable of either 0 or 1. In the 'good' group, a subject has always a task success of 1. In the 'bad' group, a subject has always a task success of 0. In both the 'average' and 'random' group, the subject has a random chance of a task success of 0 or 1.

[7] http://www.sphinx-doc.org/en/master/
[8] https://jupyter.org/
[9] https://setuptools.readthedocs.io/en/latest/setuptools.html

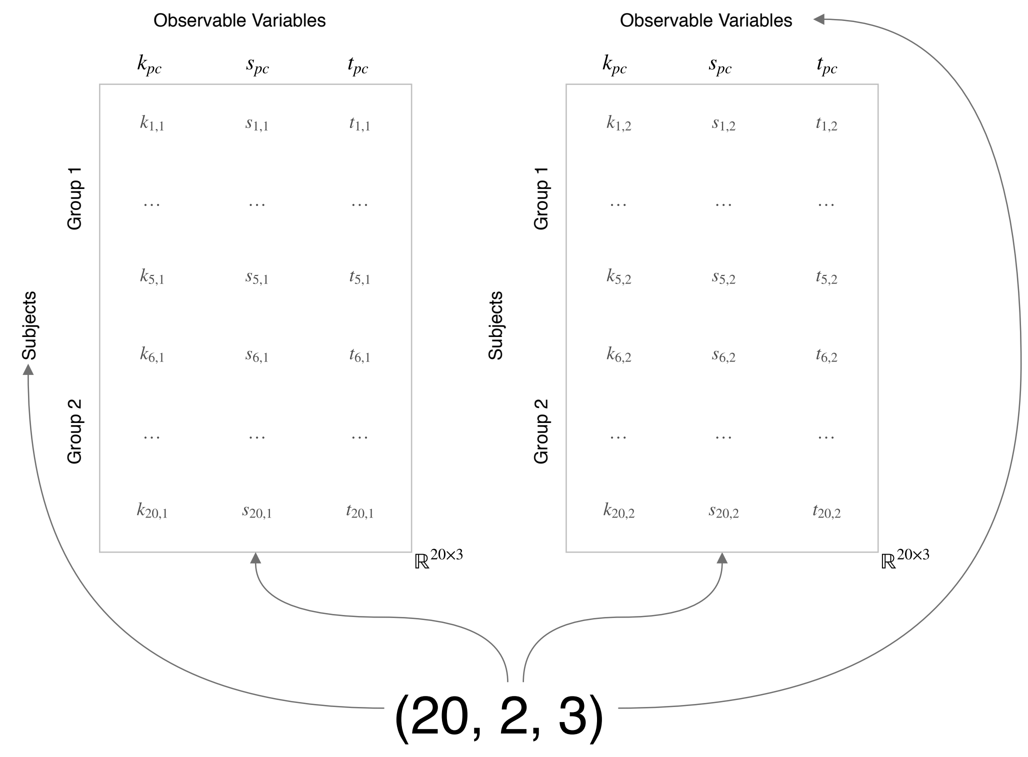

Observable Variables
k_{pc}
s_{pc}
t_{pc}
Observable Variables
k_{pc}
s_{pc}
t_{pc}
Subjects
Group 1
Group 2
$k_{1,1}$ $s_{1,1}$ $t_{1,1}$
$k_{5,1}$ $s_{5,1}$ $t_{5,1}$
$k_{6,1}$ $s_{6,1}$ $t_{6,1}$
$k_{20,1}$ $s_{20,1}$ $t_{20,1}$
Subjects
Group 1
Group 2
$k_{1,2}$ $s_{1,2}$ $t_{1,2}$
$k_{5,2}$ $s_{5,2}$ $t_{5,2}$
$k_{6,2}$ $s_{6,2}$ $t_{6,2}$
$k_{20,2}$ $s_{20,2}$ $t_{20,2}$
$\mathbb{R}^{20\times3}$
$\mathbb{R}^{20\times3}$
(20, 2, 3)

Figure 5.2.: Explanation of the NumPy shape and data returned by the `generate_observations()` function (Listing 5.3 on the next page). The function was called for 20 subjects, 2 concepts and 3 observable variables. Subjects were divided into 2 groups, with 5 and 15 members, respectively. The left table holds all observations for the first concept, the right table all observations for the second concept. The indices of each variable indicates the subject and the concept.

The mission score s_{pc} is modeled of as a continuous variable with values between the interval $[0,1]$, but in practice can only have integer values between $0,1,\dots,10$ and is divided by the maximum score of 10 to obtain a number that is limited to the range of $[0,1]$. In the 'good' group, a subject has a mission score randomly drawn from the set $\{7,8,9,10\}$. In the 'average' group, a subject has a mission score randomly drawn from the set $\{3,4,5,6,7\}$. In the 'bad' group, a subject has a mission score randomly drawn from the set $\{0,1,2,3\}$. In the 'random' group, a subject has a mission score randomly drawn from the complete range of possible values.

Finally, the mission time t_{pc} is modeled as a continuous variable with a lower limit of 5 and no upper limit. The time for each subject is generated by first randomly choosing a base time and than adding to this base time a random number from a exponential distribution with $\lambda = 2$. In the 'good' group, a subject has a mission time with a base line drawn from the range $[5,10]$. In the 'average' group, a subject has a mission time with a base line drawn from the range $[10,15]$. In the 'bad' group, a subject has a mission time with a base line drawn from the range $[15,20]$. In the 'random' group, a subject has a time mission with a baseline drawn from the range $[5,35]$.

For each concept the number of groups are given as a list of strings, indicating the group type, for example `['good', 'bad']`. How the subjects are split into the number of groups is specified as a list of integers, indicating the number of subjects for the associated group, for example `[10,20]`. Listing 5.3 on the following page shows an example for the generation of observable data for 20 subjects, 2 concepts, and 3 variables. For both concepts, the 20 subjects where divided into two groups with 5 and 15 subjects, respectively. In the first concept, the first group type was ‘good’, and the second group type was ‘bad’. The group type of the first group changed in the second concept to ‘average’, with the second group type remaining ‘bad’. This call returns a NumPy array of shape $(20, 2, 3)$ and 120 values (Figure 5.2 on the previous page). The result is simplified when there is only one concept, one subject or one observable variable, as the associated dimension can be omitted and the resulting data structure is a matrix with only two remaining dimensions instead of three. With the help of the data generator function all possible combinations of observations can be created. To add a new observable variable, one has to extend the mapping dictionary with a new key-value-pair that tells the function how to generate data for this new observable variable.

Listing 5.2: The mapping of the `generate_observations()` function between group type and observation value

```
mapping: Dict[str, Dict[str, Callable[[], float]]] = {
   'k_pc': {
      'good': lambda: 1.,
      'average': lambda: np.round(np.random.rand()),
      'bad': lambda: 0.,
      'random': lambda: np.random.choice([0,1])
   },
   's_pc': {
      'good': lambda: np.random.randint(7, 11) / 10,
      'average': lambda: np.random.randint(3, 8) / 10,
      'bad': lambda: np.random.randint(0, 4) / 10,
      'random': lambda: np.random.randint(0, 11) / 10
   },
   't_pc': {
      'good': lambda: np.random.randint(5, 10) + np.random.exponential(2),
      'average': lambda: np.random.randint(10, 15) + np.random.exponential(2),
      'bad': lambda: np.random.randint(15, 20) + np.random.exponential(2),
      'random': lambda: np.random.randint(5,35)
   }
}
```

Listing 5.3: An example call of the `generate_observations()` function

```
from cogium.generator import generate_observations

generate_observations(
   20, 2, ['k_pc', 's_pc', 't_pc'], [['good', 'bad'], ['average', 'bad']], [[5, 15], [5, 15]]
)
```

5.2.2 The model subpackage

The subpackage `model` of the CogIUM package provides an abstract base class `BaseModel`. The base class delivers all the functionality for a PyMC3 model like sampling, debug and draw possibilities, as well as save and load methods for persistent storage of the model along with its traces. However, the base class does not implement any statistical model yet. This happens in a subclass, which implements a particular statistical model in its construction method. Each subclass expects as arguments the observable data and the working memory capacity for this model. Because the observable data is part of the statistical model in PyMC3, it is not possible to only instantiate a single model for all kinds of observable data, at least not if the observations differ in their dimensionality. Therefore, a new model has to be instantiated whenever the data changes. When a model is saved to the disk, the data for this model is hold in the `self.data` attribute.

The CogIUM package provides four pre-defined models: CogiumOrig2Obs, Improved2Obs, ImprovedHierarchical2Obs, and ImprovedHierarchical3Obs. The first three models were used to demonstrate model comparison and model selection and to find the best model to describe the first two observable variables task success k_{pc} and mission score s_{pc}. ImprovedHierarchical2Obs is the best of the tree models according to different model comparison metrics. The final model ImprovedHierarchical3Obs is based on ImprovedHierarchical2Obs and extends it by modeling the observable variables mission time t_{pc} (Listing B.1 on page 175).

A model can either be built from scratch or loaded from a stored model in the `db` directory. Building a new model is rather simple: all that is needed is to pass the observable data in the right format to the constructor of the model class (Listing 5.4 on the next page). Because each model has a particular likelihood function it can only accept observable data that fits to the number of observations that is entailed in the model. However, if the number of observable variables is correct, the other dimensions of the data can be changed arbitrarily, that is, the number of subjects, the number of groups and how the subjects are split into groups as well as the number of concepts. Once a model has been built, the model of an instance is always accessible via the `model` attribute. To fit a model to the observable data, one has to call the `sample()` method of the model. This method is a wrapper around the PyMC4 `pm.sample()` method, which saves the `MultiTrace` object returned by the MCMC sampler to the disk in a directory `db/traces/<model>/Y-m-d_H-M-S`. After having fitted the model, the model itself can be saved to the disk, too, preserving the Theano graph of the model as well as the trace object from the sample method. This is done by calling the `save()` method of a model, which expects a filename. The `save()` method will save the model as pickle file to disk under `db/models/<model>/<filename>` with file ending '.pkl'.

To meet the requirements that the models can deal with observable data of any dimensionality, that is with arbitrary numbers of subjects and concepts, a lot of time went into the implementation of the deterministic variables. The deterministic variables are indexed with pc, which stands for both subject and concept. The shape of these variables has to match the number of subjects as well as the number of concepts if both dimensions are greater than one. However, all personal variables are vectors of $\mathbb{R}^{p\times 1}$ as well as all conceptual variables are vectors of $\mathbb{R}^{c\times 1}$, whereas the deterministic variables had to be of $\mathbb{R}^{p\times c}$, so each row is one subject and each column is the value for a concept. According to the equations for the deterministic variables, personal and conceptual variables are combined to obtain the pc variables, which means that vectors of different lengths had to be combined. This was solved by using the outer product of two vectors, also called a tensor product, which is available in Theano as `theano.tensor.outer(u,v)` for two vector variables u, v. The outer product is defined as $\boldsymbol{u} \otimes \boldsymbol{v} = \boldsymbol{u}\boldsymbol{v}^T$. The outer product of two vectors u, v with dimensions p and c will result in a $p \times c$ matrix, which is exactly what is needed here.

Listing 5.4: An example setup of all four models of the CogIUM package.

```python
from cogium.model import *
from cogium.generator import generate_observations

wm_capacity = 7
# setup for model 1-3
data = generate_observations(20, 1, ['k_pc', 's_pc'], [['good']], [[20]])

model1 = CogiumOrig2Obs(data, wm_capacity)
model2 = CogiumImproved2Obs(data, wm_capacity)
model3 = CogiumImprovedHierarchical2Obs(data, wm_capacity)

#setup for model 4
data = generate_observations(20, 1, ['k_pc', 's_pc', 't_pc'], [['good']], [[20]])
```

5.2.3 The plots

A lot of effort and time went into the plotting functions. They allow for analyzing the results of the MCMC sampler, of the posterior distribution for different traces, subjects and concepts as well as for different groups, of the posterior predictive distribution and for comparing different models with respect to the RMSE and MAE error. All plots can be highly customized via optional arguments and can be saved to disk as PDF files. Most plotting functions accept either a trace object or a list of trace objects, allowing for an easier model comparison.

I will summarize quickly each plot function and give one example of the produced plot, although it is not possible to show all combinations of parameters because the plots are sophisticated and support many different parameter combinations like plots for one trace object or comparison plots for a list of trace objects, all realized with the same plot function.

plot_trace_summary This function accepts a trace object or a list of trace objects and produces three histograms for each trace object or model: the ESS value, the value of the potential scale reduction $\hat{R}$ and the MCMC standard error value (for an exemplary output see Figure B.1 on page 170). These statistics can be plotted for a single or multiple variables, passed as a list of variable names to the function. If the list is empty, then the statistics will be computed over all variables contained in the trace object. With the help of this function the quality of the MCMC approximation of the true posterior distribution can be estimated and visually analyzed. To directly inspect the trace plots for single variables, use the PyMC3 `traceplot()` function.

plot_posterior This functions accepts a trace object or a list of trace objects, a variable name, and the row id, which is the id of the subject for both personal and personal plus conceptual variables and

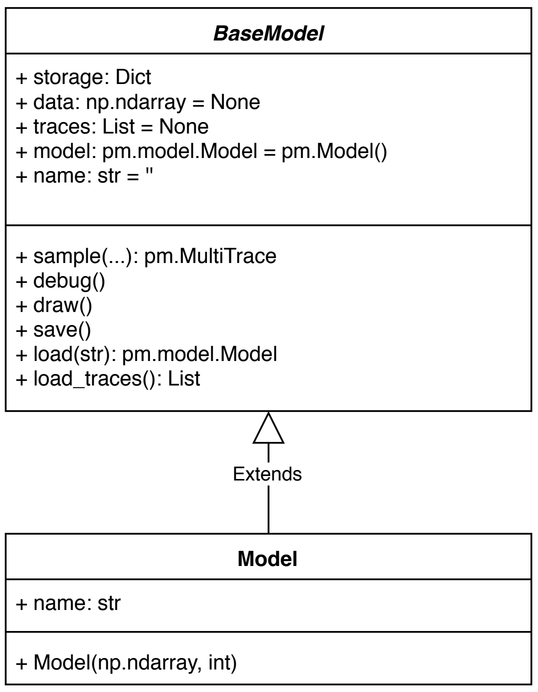


Figure 5.3.: UML class diagram for the model classes in the CogIUM package. All concrete models are implemented as subclasses of `BaseModel` which provides all functionality. Each subclass defines the statistical model with PyMC3 within the `__init__()` method, the constructor in Python. `np` is the alias for the NumPy package, `pm` is the alias for the PyMC3 package.

the id of the concept for conceptual variables, and produces a histogram for each row id along with the 95,% HDI and, if specified, the prior distribution for easier comparison of prior and posterior distribution (for an exemplary output see Figure B.2 on page 170). This function allows for visually analyzing the posterior distribution for each model parameter and subject along with the model parameters' credible intervals.

show_group_differences This functions accepts a trace object, a variable name, and a list of cutpoints where subjects were split into groups (for an exemplary output see Figure B.4 on page 172). If there was only one group in the data, then the list of cutpoints is left empty. This function is more sophisticated than other plotting functions and produces two outputs. First, one plot that uses box plots to show the values of the posterior distribution for each subject and, in addition, marks the group boundaries so that differences between subjects of different groups can be visually analyzed. Secondly, a more condensed plot of the group differences by aggregating the variable in question over all subjects of one group and only plotting the mean values for this group over all samples. This function helps in analyzing the posterior distribution for inter-individual differences and whether the model was able to model group differences represented in the observable data by different group types. However, this function can only plot group differences for a given concept id, so to compare different concepts, the function has to be called for each concept id.

show_concept_differences This function accepts a trace object and a variable name and, like the second plot of `show_group_differences()`, plots the distribution of the mean values over all subjects for each concept into one plot (for an exemplary output see Figure B.3 on page 171). Thus, group differences are no longer visible and combined into one mean over all subjects of one sample of the trace object. However, if the value for the variable differ systematically for all subjects between conditions, this plot helps identify the effect of the condition. This function is only suitable for variables with index pc and helps in analyzing the model's performance for different concepts contained in the observable data.

plot_posterior_predictive This function accepts an observable data array, a posterior predictive sample trace object, a variable name, a list of cutpoints, and the observable variable id (for an exemplary output see Figure B.5 on page 173). This function is more sophisticated than other plotting functions and produces to outputs to conduct a visual posterior predictive check. The first plot is based on the `show_group_differences()` function, but now based on the model's posterior predictive distribution and extended by marking the true value for each subject. The second plot is a comparison of the observable data with the predictions of the model. The better the model the closer the predictions to the observations. For a better comparison the single predictions are condensed to a mean prediction curve that can be compared with the observable curve. With both plots it is possible to analyze how close the model's predictions, gained from the posterior predictive distribution, are in comparison to the original observable data.

plot_predictive_error This function accepts a list of models, a list of posterior predictive trace objects, the observable data, the observable variable ids, and a list of measures and produces for each model and given observable variable the prediction error calculated by the given measure (for an exemplary output see Figure B.6 on page 174). The function calculates the model's prediction error for each observable variable with the given measure, supported are RMSE for continuous variables and MAE for categorical variables. The function allows the comparison of different models based on their prediction error but does not account for model complexity.

5.3 Validation – fifth step of Bayesian data analysis

I presented the concept of different hierarchical Bayesian models analyzed and compared in this thesis in section 4.3.2 With (hierarchical) Bayesian models (page 88) as well as their basic implementation in section 5.2 The CogIUM Python Package (page 107). This section is dedicated to the model comparison and selection process that led to the best model, to the process of extending the model to new observable variables, and to an extended model validation of the best found model. In the discussion of this section I will recapitulate the process of model selection and model validation, address open or unsolved questions and talk about the difference between model validation and model evaluation.

5.3.1 Model comparison – a case study

In this section I will describe the process from model $\mathscr{M}_1$ over model $\mathscr{M}_2$ to model $\mathscr{M}_3$, the results that led to the modifications between the models and how the models performed against each other. To keep the presentation of the material as as compact as possible, I will present only a few number of key figures that allow for a visual analysis of the three models' performances. All figures in this section will present the findings of all three models to allow for a direct comparison. However, $\mathscr{M}_1$ was the first model I

Table 5.1.: Overview of the data sets used in the model comparison process.

Data Set	N	N_p	N_c	N_g	Group Types	Splits
$\mathscr{D}_1$	200	100	1	2	['good', 'bad']	$[50, 50]$
$\mathscr{D}_2$	100	50	1	3	['good', 'average', 'bad']	$[10, 30, 10]$
$\mathscr{D}_3$	200	50	2	2	['good', 'average'], ['average', 'bad']	$[25, 25], [25, 25]$
$\mathscr{D}_4$[10]	200	50	2	2	['good', 'bad'], ['bad', 'good']	$[25, 25], [25, 25]$
$\mathscr{D}_5$	180	30	3	1	['good'], ['average'], ['bad']	$[30], [30], [30]$

N: total number of observations, N_p: number of subjects, N_c: number of concepts, N_g number of groups per concept.

analyzed and based on its inadequacy to capture the structure of the observable variable mission score s_{pc}, model $\mathscr{M}_2$ was designed. As intended, model $\mathscr{M}_2$ showed an improved performance for s_{pc}, but was still not capable of capturing all the structure of the observable variable. That is why model $\mathscr{M}_3$ was designed as a hierarchical model for explaining s_{pc}.

The comparison of the three models was conducted with five different data sets (Table 5.1). I will mainly talk about and present figures for data set $\mathscr{D}_2$, a data set with 50 subjects, one concept, and three different groups, with two groups of size 10 and one group of size 30. The results for all five data sets are reported at the end of this section.

Performance of $\mathscr{M}_1$

The first model implemented in this thesis was the original model, which is called `CogiumOrig2Obs` in the CogIUM package, and named $\mathscr{M}_1$ throughout this thesis. The model details can be revisited in section 4.3.2 Model $\mathscr{M}_1$ – original CogIUM for two observations (page 94) and Table A.1 on page 168.

$\mathscr{M}_1$ models k_{pc} and s_{pc} as a cause of gcl_{pc} and δ_{pc}, but uses no further variables. $\mathscr{M}_1$ is the simplest of the three models. But the model is able to capture the structure of the observable variable task success k_{pc}, as the results from the posterior predictive check show (Figures 5.9 and 5.11 on page 125 and on page 127). It is not a 100 % match, but the model predicts the right value for k_{pc} for the first group (k_{pc} always 1) in roughly 80 % of all simulations, for the third group (k_{pc} always 0) in roughly 60 % of all simulations, and for the second group (k_{pc} either 0 or 1) in either 80 % or 60 % of all simulations. So it seems that model is better in explaining values for k_{pc} that are equal to 1 than for values that are equal to 0, because the accuracy is always 20 % less. However, $\mathscr{M}_1$ is not able to capture the structure of the observable variable mission score s_{pc} at all (Figures 5.10 and 5.11 on page 126 and on page 127). The distribution of values for s_{pc} from the model's simulation are more or less identical for all subjects, which is consolidated by averaging over all 1000 samples of the posterior predictive distribution: the averaged distribution has a single peak and cannot reproduce the structure in the observable data. In addition, the standard deviation or IQR in the box plots is rather high and leads to a distribution for each subject that covers the whole range of possible values for s_{pc}. I concluded from these findings, that $\mathscr{M}_1$ is good enough in capturing the essence of k_{pc}, but not for s_{pc}. Further analysis of $\mathscr{M}_1$ revealed that it did not make use of all model parameters. Prior knowledge ψ_p and motivation m_p were nearly identical for all

[10] Data set $\mathscr{D}_4$ is contradictory on purpose, as explained in section 5.3.1 Final model comparison and model selection (page 128).

subjects and groups (Figures 5.4 to 5.6 on pages 121–120). There were small differences in GCL gcl_{pc}, but nearly none in free working memory capacity δ_{pc} (Figures 5.4, 5.7 and 5.8 on pages 123–120).

Performance of $\mathscr{M}_2$

Therefore, I decided to reparameterize the observable variable s_{pc} in $\mathscr{M}_2$. I assumed the main problem in $\mathscr{M}_1$ with s_{pc} was the choice of the beta distribution, because there is no separate parameter for a standard deviation. In addition, the beta distribution is quite flat for small parameter values of a, b, which is the case in $\mathscr{M}_1$ because gcl_{pc} and δ_{pc}, which govern the values of a, b, are both limited to a small range. As a result of these considerations, a second model $\mathscr{M}_2$, which is called `CogiumImproved2Obs` in the CogIUM package, was implemented and tested, which tries to tackle some of the aforementioned problems of $\mathscr{M}_1$. The model details can be revisited in section 4.3.2 Model $\mathscr{M}_1$ – original CogIUM for two observations (page 94) and Table A.1 on page 168.

Changing the distribution for s_{pc} from a beta distribution to a normal distribution allows s_{pc} to be centered around a mean value with a standard deviation, which seems a more appropriate choice to reduce the variance in the model's predictions. As a drawback, instead of the beta distribution, which was naturally limited to values between the range of zero and one, the normal distribution has no such limitations. However, because s_{pc} is an observable variable and thus part of the model's likelihood, it is the task of the Bayesian inference to find parameter values that will restrict the normal distribution to a plausible range in accordance with the observations. If this is not possible, then the model is not well specified. Regarding the model's performance, the first thing that can be noticed is that $\mathscr{M}_2$ is now worse than $\mathscr{M}_1$ with regard to k_{pc} (Figures 5.9 and 5.11 on page 125 and on page 127). The accuracy of the predicted values for k_{pc} in the first group is reduced from roughly 80 % to 60 %. However, $\mathscr{M}_2$ is visually significantly better in predicting s_{pc} (Figures 5.10 and 5.11 on page 126 and on page 127). The individual box plots are closer to or even contain the true value and the average of all simulated predictions resembles closer the distribution of the observations. Regarding the model's performance with respect to different groups, the posterior predictive check allows the conclusion that $\mathscr{M}_2$ is able to capture the structure of s_{pc} for moderate values – all box plots for the subjects of the "average" group contain the true value –, but fails to capture the structure of s_{pc} for extreme values near the limit of the domain – most box plots for subjects of the "good" as well as the "bad" group do not contain the true value. And when we have a look at the IQR or standard deviation, respectively, of the predicted values, then we notice that the IQR are still relatively wide, covering nearly the whole range of possible values. The better performance of $\mathscr{M}_2$ can be explained by the reparametrization which led to a greater inter-individual and inter-group difference in the model parameter motivation m_p, but not for prior knowledge ψ_p (Figures 5.4 to 5.5 on pages 121–120). Because m_p has an influence on gcl_{pc}, $\mathscr{M}_2$ also shows inter-individual as well as inter-group differences in gcl_{pc}, but like $\mathscr{M}_1$ still not for δ_{pc}—which means, that this model parameter is still not used, or needed, by the model to explain the observations (Figures 5.4, 5.7 and 5.8 on pages 123–120). However, it is noticeable that the width of the posterior distribution for δ_{pc} is much larger for all individuals. This is due to the fact that in $\mathscr{M}_2$, the posterior distribution for icl_c has a 95 % HDI ranging from 8.35 to 41.46 with a mean value of 22.76, which is much higher than in model $\mathscr{M}_1$. This can be interpreted as follows: $\mathscr{M}_2$ tries to explain the differences seen in the observable variables k_{pc} and s_{pc} from data set $\mathscr{D}_2$ by setting the difficulty of the learning material, represented by icl_c, much higher than $\mathscr{M}_1$ did. This automatically increases the total cognitive load cl_{pc}, which is the sum of ECL and ICL. By adjusting the motivation m_p for individuals of different groups, $\mathscr{M}_2$ is able to explain the observations in modeling subjects from the “good” group with a high motivation, subjects from the “average” group with an average motivation and subjects from the “bad” group with a low motivation, because motivation influences the value of gcl_{pc}, which is the mean value for s_{pc} in model $\mathscr{M}_2$.

So there are a few issues left with model $\mathscr{M}_2$: the nearly identical values of δ_{pc} across all subjects, the implausible high value for icl_c, which improves the model’s performance but is not very likely to reflect the real world, the worse performance regarding k_{pc}, and the variance of the model’s predictions that are still too high across all subjects.

Performance of $\mathscr{M}_3$

To further improve the performance of $\mathscr{M}_2$ regarding the observable variable s_{pc}, I decided to model s_{pc} hierarchically. In $\mathscr{M}_2$, s_{pc} is modeled with a mean value that directly and completely depends on gcl_{pc}. To improve the model’s flexibility, I chose a new parametrization where s_{pc} remains normally distributed, but depends on both the values of gcl_{pc} and δ_{pc}. In addition, the impact of δ_{pc} is governed by a multiplicative factor ν_s, which is modeled as personal group variable. With this modification the new model is allowed to alter the mean value of the normal distribution that generates s_{pc}, given by gcl_{pc}, according to the value of δ_{pc}. A positive value of δ_{pc} further increases s_{pc}, counterbalancing lower values of gcl_{pc}. A negative value of δ_{pc} has the opposite effect and decreases s_{pc}, negating the effects of a higher value for gcl_{pc}. As a result of these considerations, a third model $\mathscr{M}_3$, which is called `CogiumImprovedHierarchical2Obs` in the CogIUM package, was implemented and tested, which tries to tackle some of the aforementioned problems of $\mathscr{M}_2$. The model details can be revisited in section 4.3.2 Model $\mathscr{M}_1$ – original CogIUM for two observations (page 94) and Table A.1 on page 168.

Regarding the observable variable k_{pc}, model $\mathscr{M}_3$ is back to the performance level of $\mathscr{M}_1$, in fact, $\mathscr{M}_3$ is in some cases even above the 80 % accuracy for the first group (Figures 5.9 and 5.11 on page 125 and on page 127). A real improvement was achieved in the model’s performance regarding s_{pc}. The model is not only able to reproduce the structure of s_{pc} seen in the observable data for every subject, but also has a reduced variance for its predictions (Figures 5.10 and 5.11 on page 126 and on page 127). The average distribution of the model’s predictions is very close to the distribution of the observable variable and matches all structural characteristics. The reparametrization from $\mathscr{M}_2$ to $\mathscr{M}_3$ led to further differences in how the model latent variables were used to explain the observations. Like in $\mathscr{M}_2$ motivation differs between individuals and between groups (Figures 5.4 and 5.5 on page 121 and on page 120). As a result of the motivational differences, we see similar differences in the values for gcl_{pc} (Figures 5.4 and 5.7 on page 123 and on page 120). However, for the first time prior knowledge ψ_p also differs between individuals, and most notably, between groups (Figures 5.4 and 5.6 on page 122 and on page 120). As prior knowledge influences the cognitive load imposed by icl_c, as a direct consequence the total load cl_{pc} and with it the value for free working memory δ_{pc} is changed, too (Figures 5.4 and 5.8 on page 124 and on the next page). Model $\mathscr{M}_3$ is the only model that shows these differences in prior knowledge and free working memory capacity. The posterior distribution for icl_c has a 95 % HDI ranging from

5.05 to 13.69 with a mean value of 9.43, which is considerably smaller than the value in $\mathscr{M}_2$. So the behavior of $\mathscr{M}_3$ can be interpreted as follows: The model uses both motivation m_p and prior knowledge ψ_p to accommodate the model's predictions with the observations. Subjects in the "good" group are characterized by a high value for motivation as well as a larger prior knowledge. This results in more free working memory capacities δ_{pc} as well as a greater amount of working memory resources gcl_{pc} dedicated to learning. The opposite is true for subjects of the "bad" group, where low values for m_p and ψ_p result in lower values for gcl_{pc} and δ_{pc}, which reduces overall performance.

All relevant figures that were referenced in the previous text are listed on the next pages. For a better readability they are presented en bloc. All figures have the same structure and are explained further in the appendix. The next section follows after the last figure of this block.

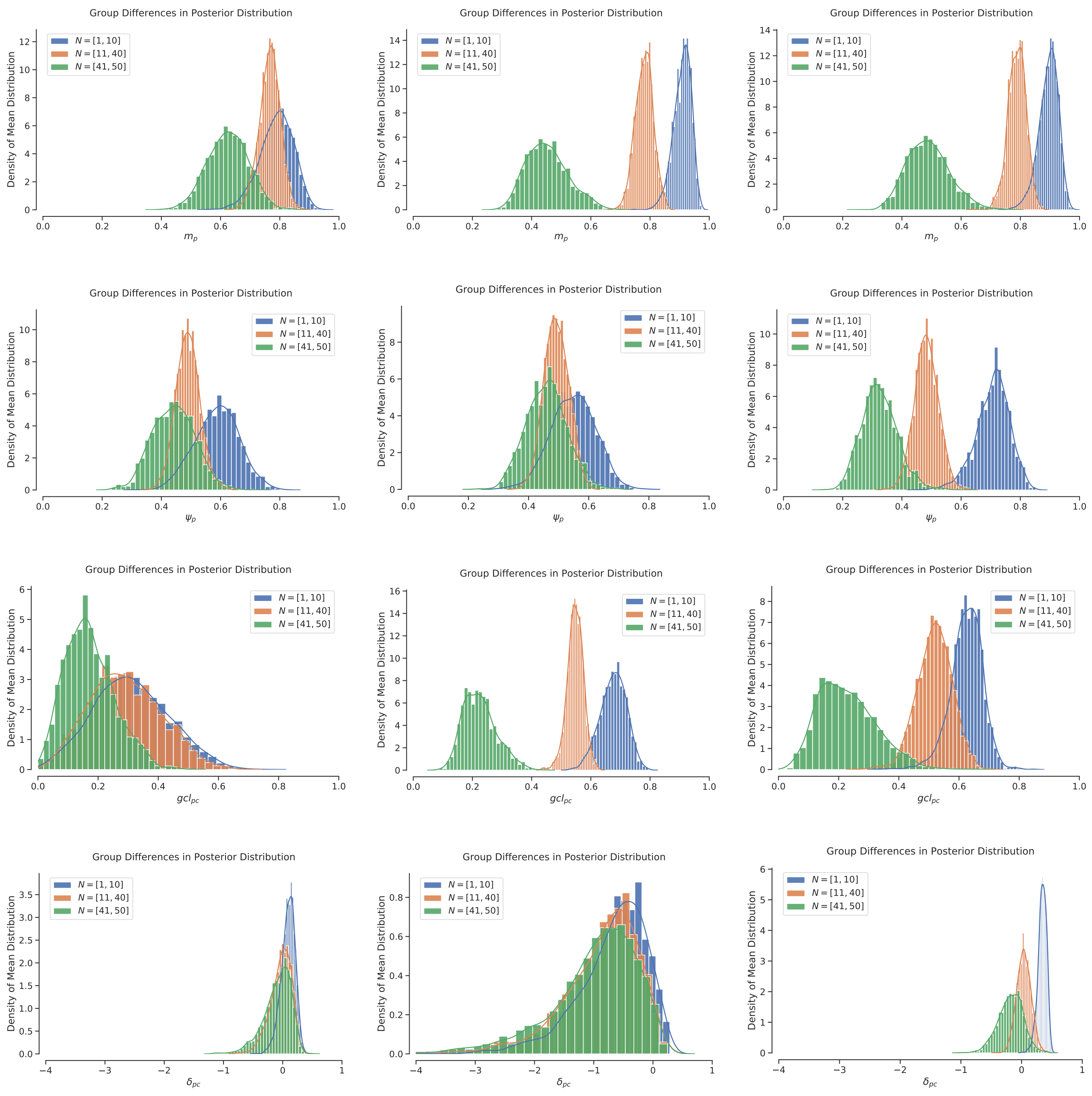


Figure 5.4.: Group differences in the posterior distribution for four variables and three models, based on data set $\mathscr{D}_2$. Shown are the distributions of the group means. (Top) Motivation m_p. (Second top) Prior knowledge ψ_p. (Second bottom) GCL gcl_{pc}. (Bottom) Free working memory capacity δ_{pc}. (Left) $\mathscr{M}_1$. (Middle) $\mathscr{M}_2$. (Right) $\mathscr{M}_3$. For further explanations see Figure B.4 on page 172.

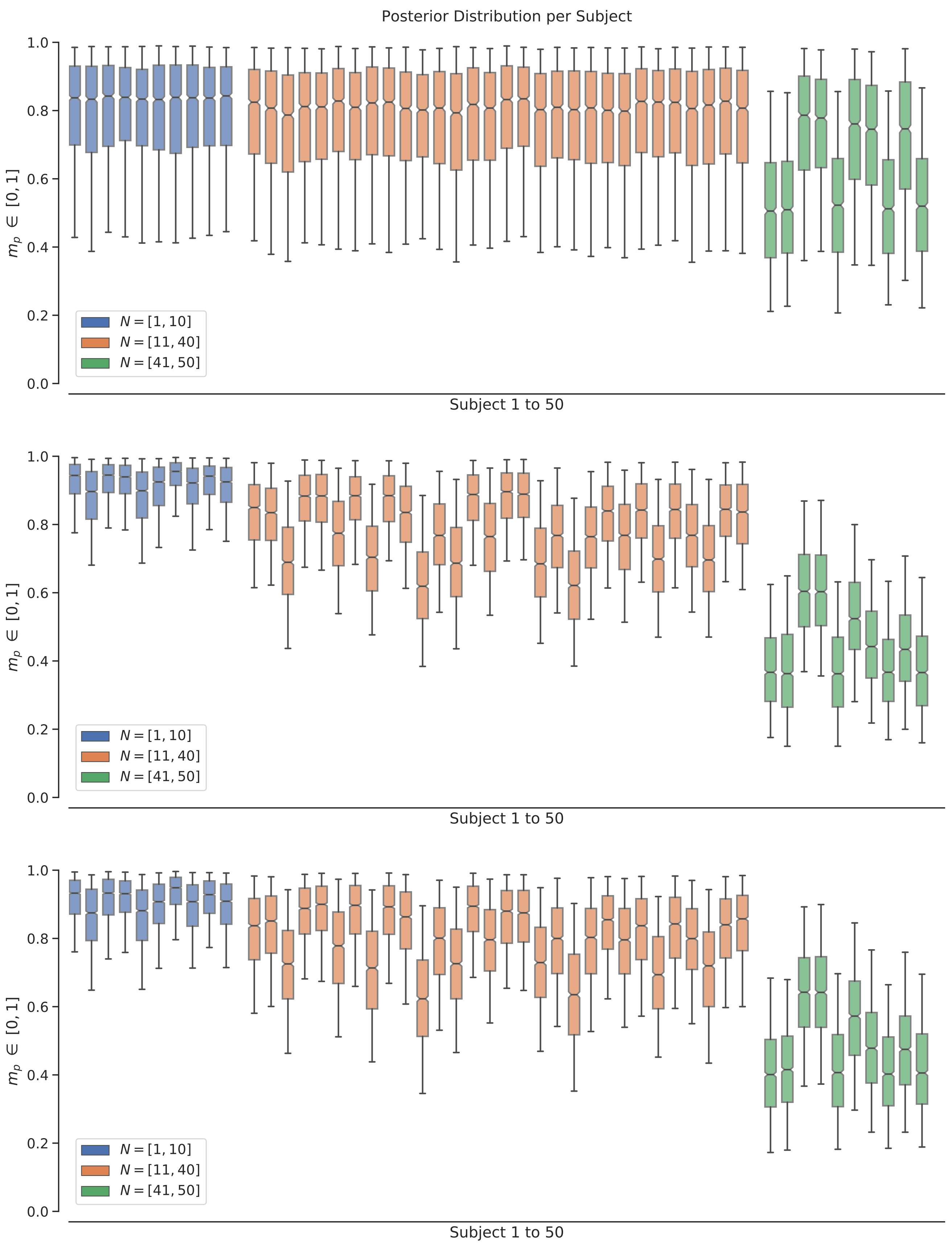


Figure 5.5.: Individual differences in the posterior distribution for personal variable motivation m_p, based on data set $\mathscr{D}_2$. The posterior distribution is shown as box plot for each subject, colored according to the subject's group membership. (Top) $\mathscr{M}_1$. (Middle) $\mathscr{M}_2$. (Bottom) $\mathscr{M}_3$. For further explanations see Figure B.4 on page 172.

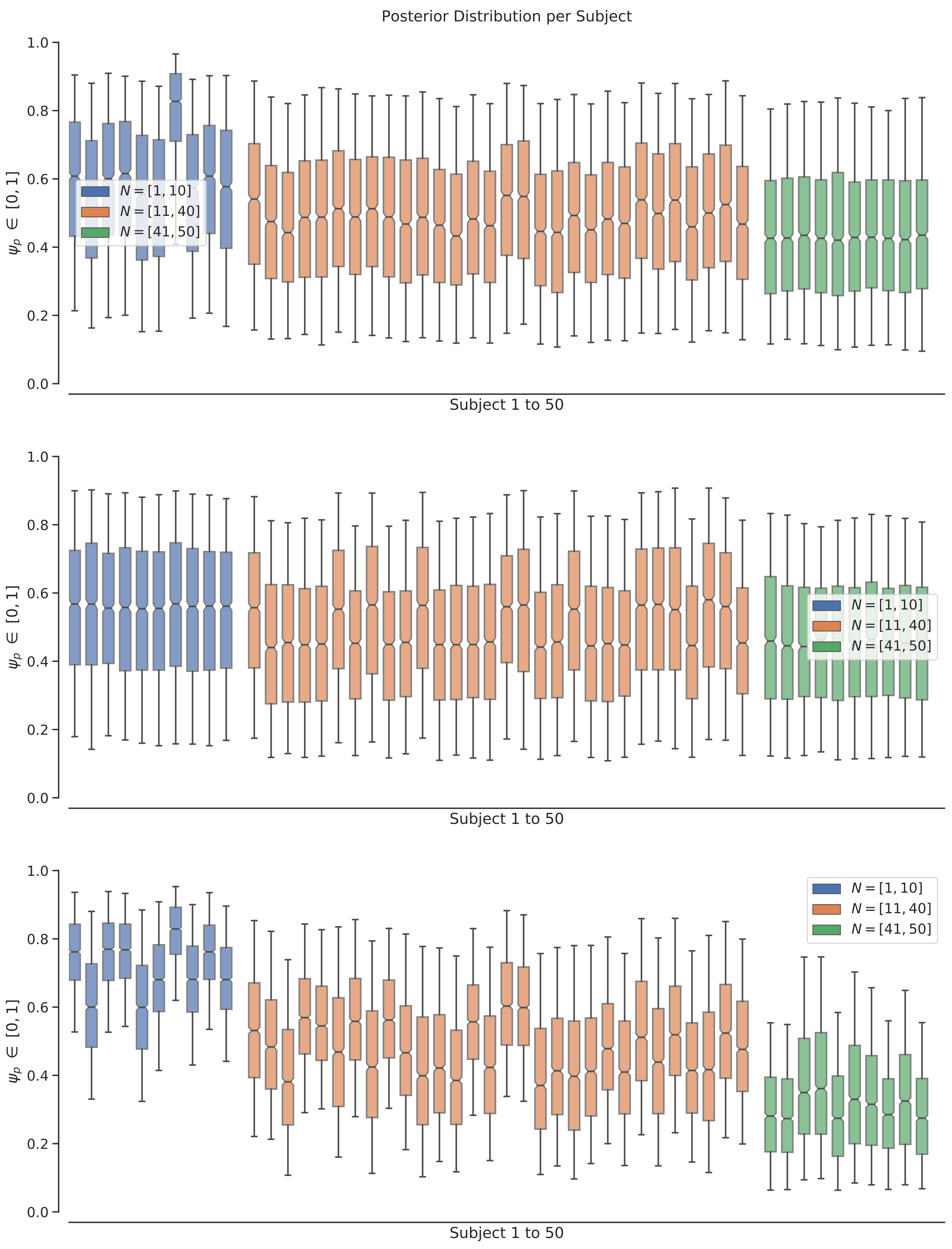


Figure 5.6.: Individual differences in the posterior distribution for personal variable prior knowledge ψ_p, based on data set $\mathscr{D}_2$. The posterior distribution is shown as box plot for each subject, colored according to the subject's group membership. (Top) $\mathscr{M}_1$. (Middle) $\mathscr{M}_2$. (Bottom) $\mathscr{M}_3$. For further explanations see Figure B.4 on page 172.

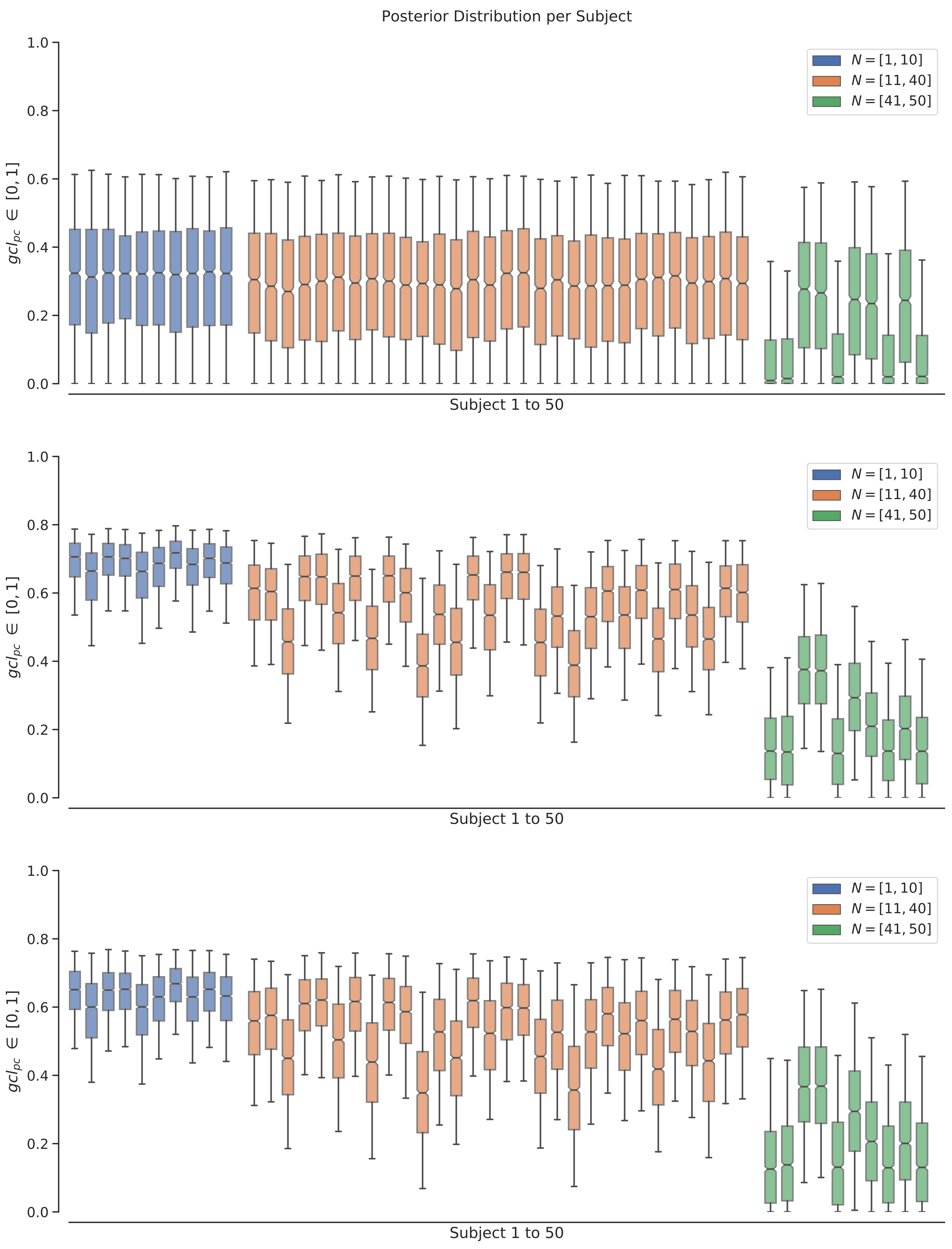


Figure 5.7.: Individual differences in the posterior distribution for deterministic variable GCL gcl_{pc}, based on data set $\mathscr{D}_2$. The posterior distribution is shown as box plot for each subject, colored according to the subject's group membership. (Top) $\mathscr{M}_1$. (Middle) $\mathscr{M}_2$. (Bottom) $\mathscr{M}_3$. For further explanations see Figure B.4 on page 172.

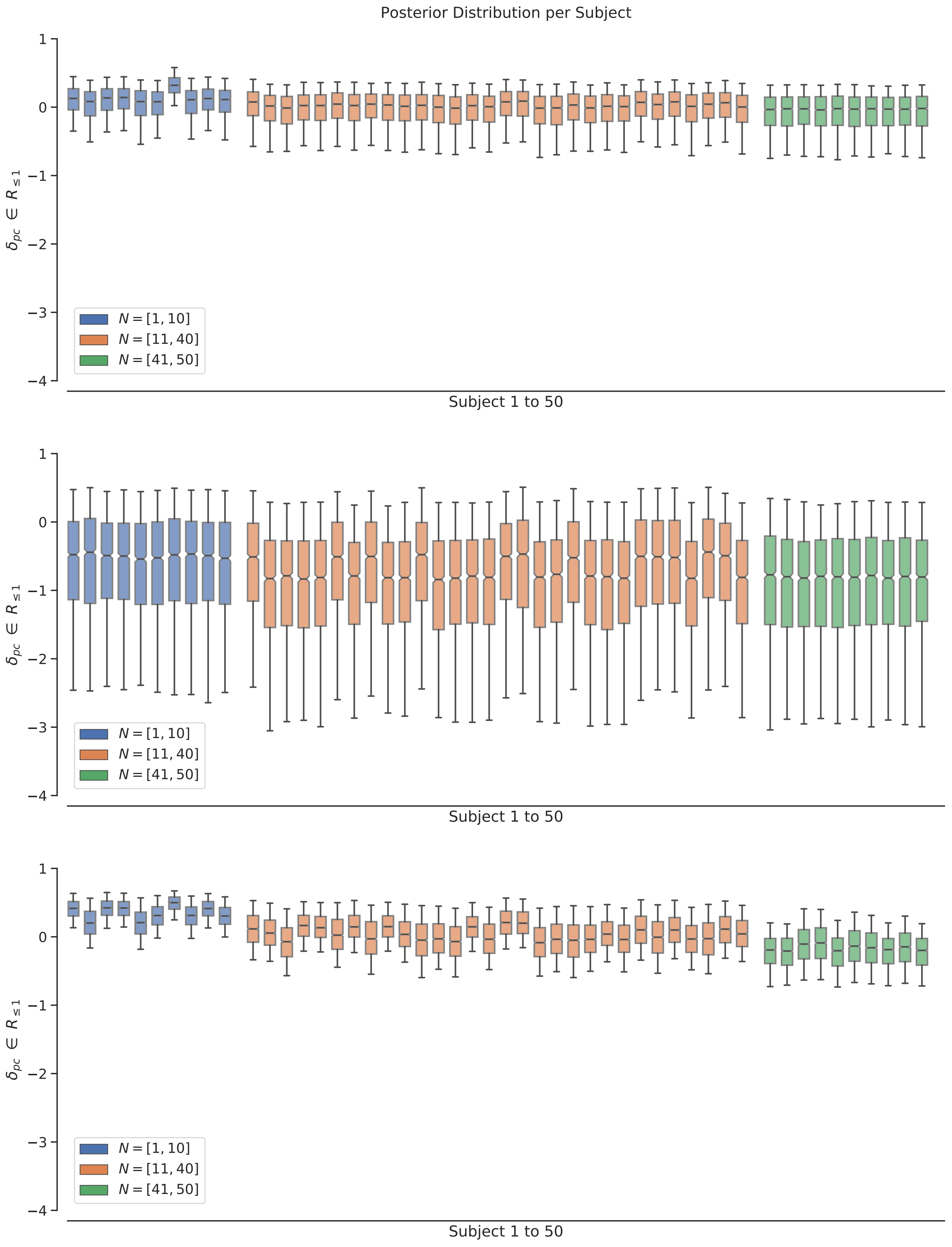


Figure 5.8.: Individual differences in the posterior distribution for deterministic variable free working memory capacity δ_{pc}, based on data set $\mathscr{D}_2$. The posterior distribution is shown as box plot for each subject, colored according to the subject's group membership. (Top) $\mathscr{M}_1$. (Middle) $\mathscr{M}_2$. (Bottom) $\mathscr{M}_3$. For further explanations see Figure B.4 on page 172.

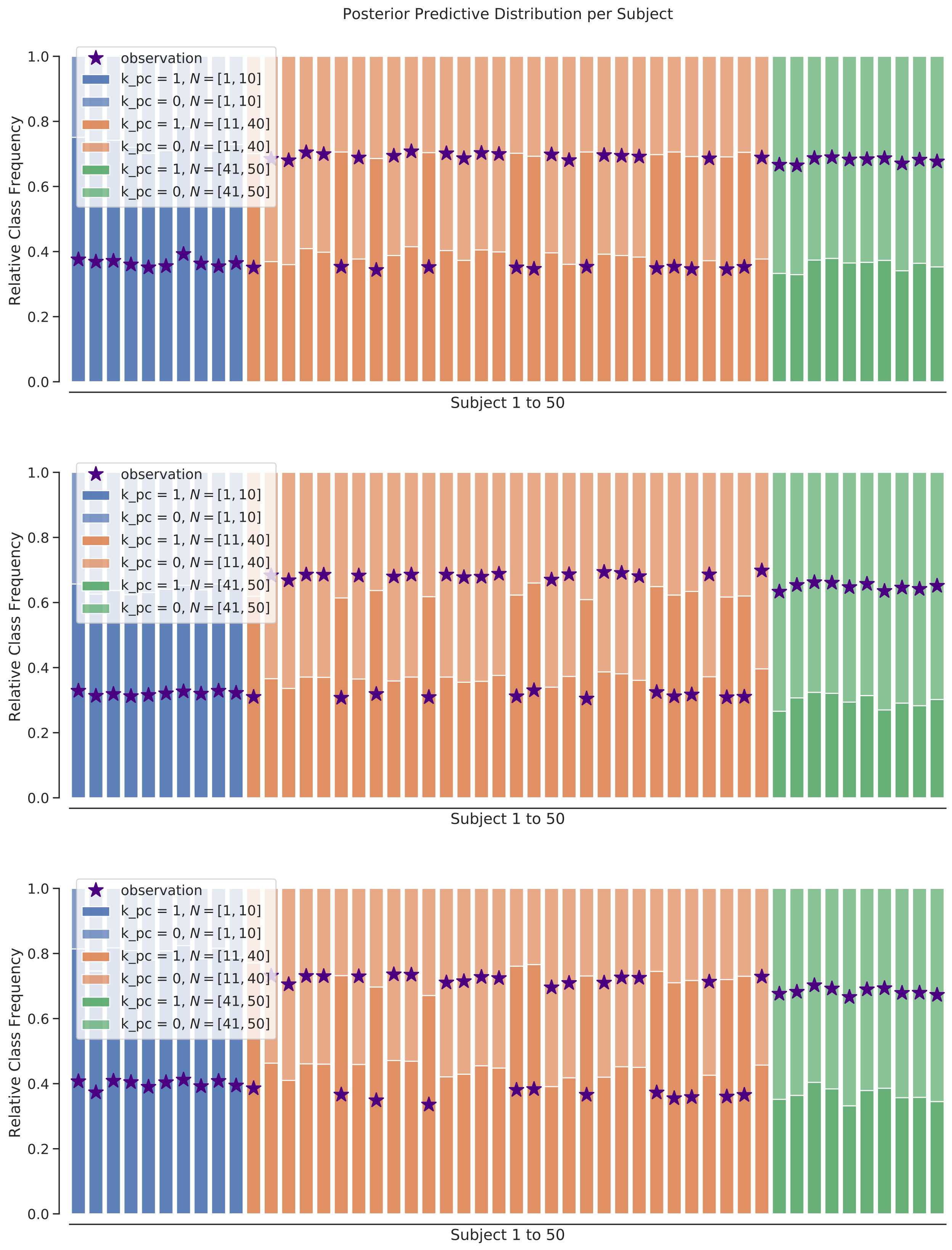


Figure 5.9.: Posterior predictive check for observable variable task success k_{pc}, based on data set $\mathscr{D}_2$. Each row shows the model's predictions for each subject. (Top) $\mathscr{M}_1$. (Middle) $\mathscr{M}_2$. (Bottom) $\mathscr{M}_3$. For each subject 1000 samples were drawn from the model's posterior predictive distribution. Because k_{pc} is a discrete variable, the class frequencies are shown for each subject. The true value is marked. For further explanations see Figure B.5 on page 173.

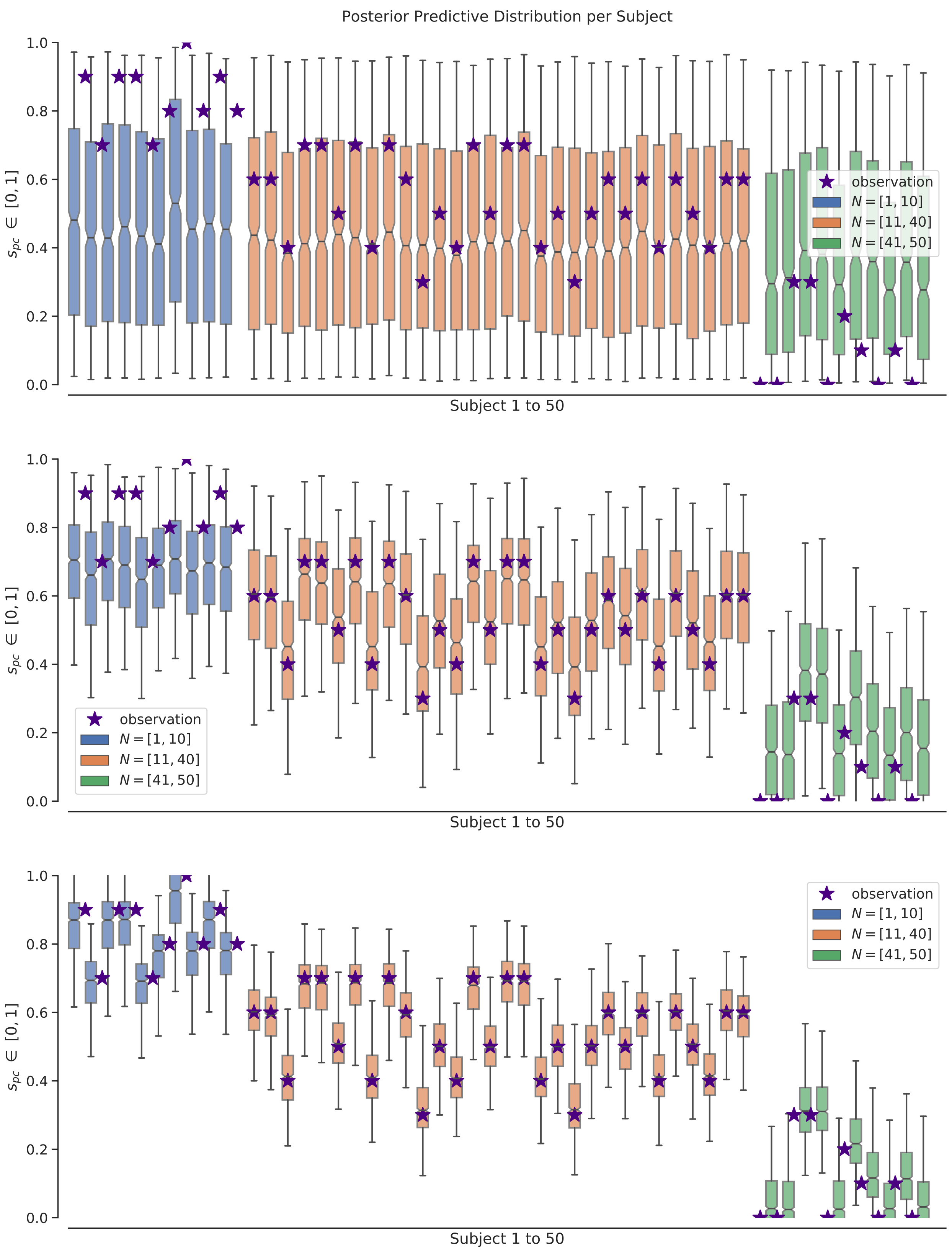


Figure 5.10.: Posterior predictive check for observable variable mission score s_{pc}, based on data set $\mathscr{D}_2$. Each row shows the model's predictions for each subject. (Top) $\mathscr{M}_1$. (Middle) $\mathscr{M}_2$. (Bottom) $\mathscr{M}_3$. For each subject 1000 samples were drawn from the model's posterior predictive distribution. Because k_{pc} is a discrete variable, the class frequencies instead of a box plots are shown for each subject. The true value is marked. For further explanations see Figure B.5 on page 173.

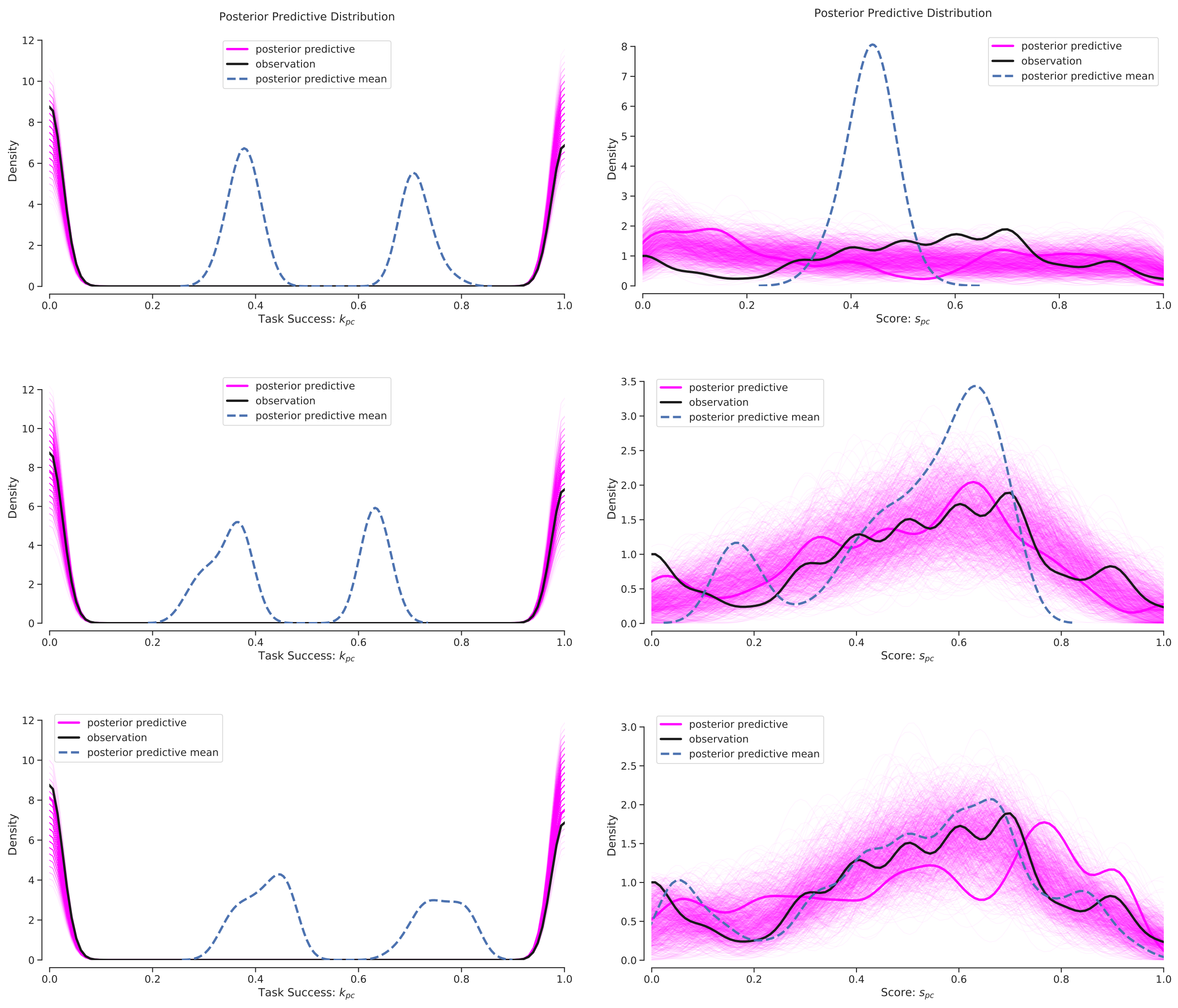


Figure 5.11.: Posterior predictive check for two observable variables and three models, based on data set $\mathscr{D}_2$. Each row shows the posterior predictive check for one model. (Top) $\mathscr{M}_1$. (Middle) $\mathscr{M}_2$. (Bottom) $\mathscr{M}_3$. (Left) Task success k_{pc}. (Right) Mission score s_{pc}. For each subject 1000 samples were drawn from the model's posterior predictive distribution. Each subplot shows the distribution of single samples as well as the model's average prediction. For further explanations see Figure B.5 on page 173.

In the three previous sections I discussed in detail the performances of all three models $\mathscr{M}_1, \mathscr{M}_2$, and $\mathscr{M}_3$ with respect to the data set $\mathscr{D}_2$. However, the analyses was mainly based on a visual inspection of the posterior predictive distribution in comparison with the observable data. Besides, $\mathscr{D}_2$ was not the only data set that was analyzed during this phase of the implementation, but instead five data sets were tested, as mentioned in Table 5.1 on page 116.

I will continue with the presentation of three common and appropriate metrics for model comparison and model selection: root mean squared error (RMSE), Watanabe-Akaike or widely available information criterion (WAIC) and leave-one-out cross-validation (LOO-CV). They were already introduced in section 3.2 Model Evaluation and Model Comparison (page 58), so that I will limit the presentation to the results of the calculations and their implications for model selection.

I begin with the presentation of mean absolute error (MAE) values for k_{pc} and RMSE values for s_{pc} (Figure 5.12 on the next page). To compute RMSE and MAE, I first computed the single prediction errors for 1000 samples drawn from the model's posterior predictive distribution. Afterwards, I took the mean of the MAE and RMSE values of the single samples to obtain the average value for the model. Model $\mathscr{M}_1$ has the lowest MAE for k_{pc} in one of five cases, for data set $\mathscr{D}_4$. Model $\mathscr{M}_2$ has the lowest MAE for k_{pc} in one of five cases, for data set $\mathscr{D}_5$. In all other cases, Model $\mathscr{M}_3$ has the lowest MAE for k_{pc}. $\mathscr{M}_1$ has the lowest RMSE for s_{pc} in one of five cases, for data set $\mathscr{D}_4$. $\mathscr{M}_2$ never has the lowest RMSE for s_{pc}, so model $\mathscr{M}_3$ has the lowest RMSE for s_{pc} in all other four cases. In general, differences in MAE are quite small, but the differences in RMSE are up to one order of magnitude.

As was discussed in section 3.2, RMSE does not use the full information available with the posterior distribution. WAIC and LOO-CV are fully Bayesian approaches to estimate the within-sample predictive accuracy of a model. When comparing multiple models we prefer the model with the highest predictive accuracy, which refers to the lowest values for WAIC and LOO-CV. The computed values for WAIC and LOO-CV are reported for all five data sets and all three models (detailed overview of the results from the PyMC3 functions `pm.waic()` and `pm.loo()` are given in Table 5.2 on page 131, a graphical summary is given in Figure 5.13 on the next page). Models are sorted ascending so that the best model according to the given predictive accuracy metric is listed first. p_{Value} describes the effective number of parameter for both metrics and can be interpreted as a measure of the model's complexity. Thus, WAIC and LOO-CV naturally account for the model's complexity. If a best model, that is, a model with the lowest value according to WAIC or LOO-CV, has a sufficiently large distance from the other two models, its Akaike weight is zero and it is supposed to describe the data best according to the reported metrics. If, however, the models are not clearly separated, as measured by d_{WAIC} and $d_{\text{LOO-CV}}$, then the Akaike weights can be interpreted as the probability that model $\mathscr{M}_i$ is the best model with regard to the given metric (Wagenmakers and Farrell, 2004). Model $\mathscr{M}_1$ is the best model according to both WAIC and LOO-CV in one of five cases, for data set $\mathscr{D}_4$. Model $\mathscr{M}_2$ is the best model according to both WAIC and LOO-CV in one of five cases, for data set $\mathscr{D}_5$. Model $\mathscr{M}_3$ is the best model according to both WAIC and LOO-CV for three out of five cases, for the data sets $\mathscr{D}_1, \mathscr{D}_2$, and $\mathscr{D}_3$. The order of the models is for both WAIC and LOO-CV the same in four of five cases, only for data set $\mathscr{D}_2$ the order of $\mathscr{M}_1$ and $\mathscr{M}_2$ differs between the two metrics.

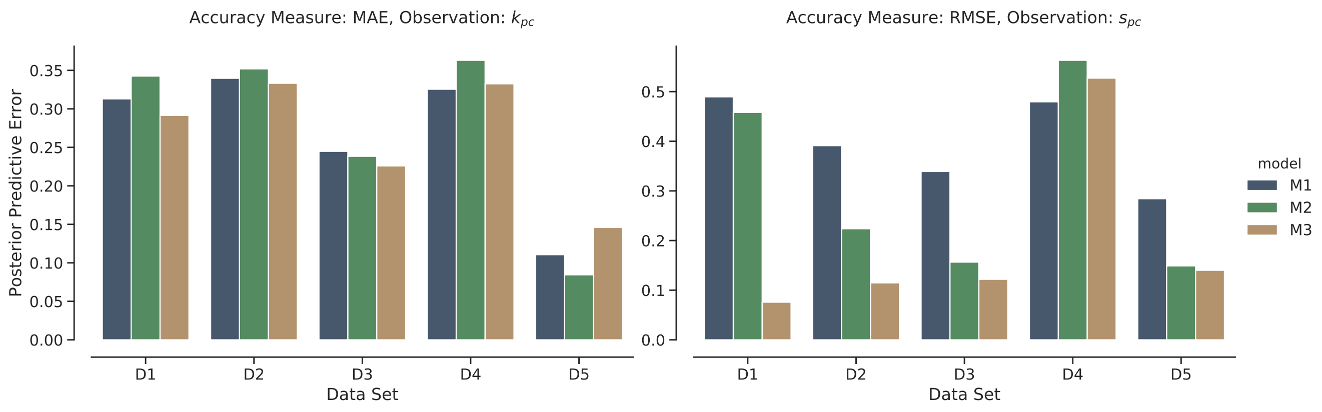


Figure 5.12.: Posterior predictive error for all three models and for all five data sets. (Left) MAE for task success k_{pc}. (Right) RMSE for mission score s_{pc}. Smaller values are better.

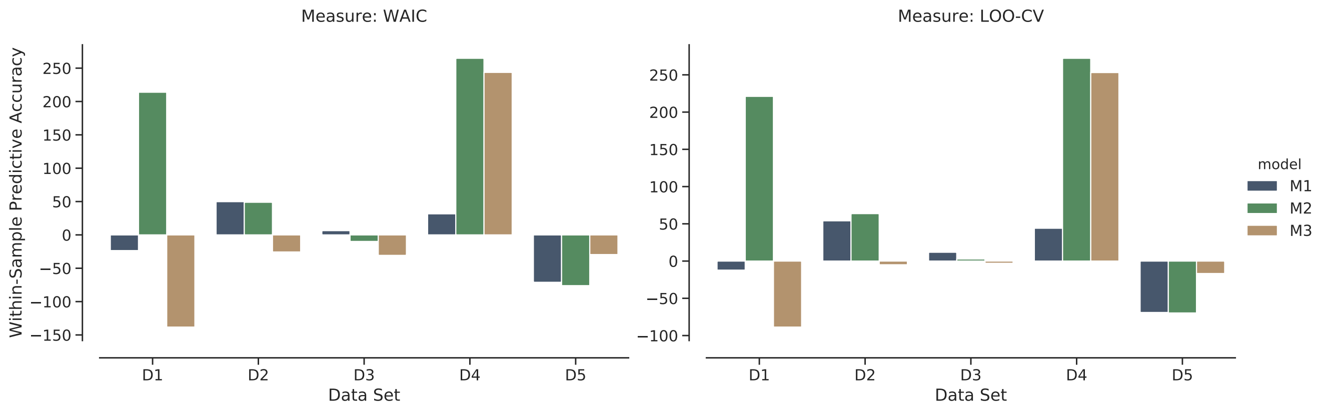


Figure 5.13.: Within-sample predictive accuracy for all three models and for all five data sets. (Left) WAIC. (Right) LOO-CV. Smaller values are better.

With the results from both the posterior predictive error measures and the predictive accuracy measures what can be said about the question of model selection? Which model should be preferred? The RMSE measures are in favor for $\mathscr{M}_3$ for four of five data sets, most prominently for data set $\mathscr{D}_1$ with 100 subjects, two groups, and only one concept. The better predictive performance of $\mathscr{M}_3$ is still noticeable for data set $\mathscr{D}_2$, but becomes closer and closer to model $\mathscr{M}_2$ for data sets with more concepts, like in $\mathscr{D}_3, \mathscr{D}_4$, and $\mathscr{D}_5$. This relationship between the models is further substantiated with the results of WAIC and LOO-CV. When there is only one concept, like in the data sets $\mathscr{D}_1$ and $\mathscr{D}_2$, model $\mathscr{M}_3$ is clearly favored with Akaike weights between 0.78 and 0.99. Only for the data sets $\mathscr{D}_4$ and $\mathscr{D}_5$ the model is the least plausible and has an Akaike weight of zero. $\mathscr{D}_4$ is a special case because it is contradictory on purpose: the data set contains two concepts and two groups, but the "good" group in the first concept becomes the "bad" group in the second concept and vice versa. None of the models has the mechanics to explain such data, they are confronted with the task to explain observations that are "impossible" for them to re-create. This is because personal variables do not differ between concepts. The only two ways for a model to explain the observation that a "good" subject becomes a "bad" subject is by assuming that either the task has become more difficult (increase icl_c, ecl_c, or both), or that the subject has become less motivated or has less prior knowledge about the material (decrease m_p, ψ_p or both). However, increasing the difficulty of the learning material for concept two to explain the change of a "good" subject into a "bad" subject increases difficulty for all subjects, and, because personal variables are fixed, cannot explain the simultaneous change of a "bad" subject into a "good" subject, when difficulty was increased. Likewise, decreasing a subject's motivation or prior knowledge to explain the change of a "good" subject into a "bad" subject applies to all concepts, because, again, all personal variables are fixed for a subject. So the subject has to become worse in all concepts, not worse in one and better in another. This is a very important consideration and can only be overcome by allowing personal variables to *differ* between concepts, which means that personal variables have to become variables with an index pc. I will come back to this when talking about model extensions in section 5.3.2 Model extension – a case study (page 132). For the purpose of this section, data set $\mathscr{D}_4$ is intended to be hard for the more specialized models $\mathscr{M}_2$ and $\mathscr{M}_3$, and it is in accordance with the theoretical assumptions that the simplest model $\mathscr{M}_1$ has the best predictive accuracy for this data set. Therefore, only data set $\mathscr{D}_5$ poses a problem for selecting model $\mathscr{M}_3$ as the best model, because this data set has three concepts with only one group, but represents plausible observable data. The reason why I decided to select the model $\mathscr{M}_3$ as the best model of the three models discussed so far is that the main focus of the reparametrizations that led to the models $\mathscr{M}_2$ and $\mathscr{M}_3$ was to improve the predictive performance with respect to s_{pc} based on data set $\mathscr{D}_2$. Therefore, the focus was on supporting individual differences in the personal variable and not in the conceptual variable. With this in mind, $\mathscr{M}_3$ is the best model. To overcome the problems that arise with data set $\mathscr{D}_5$, further adaptations of the model are needed, mainly to allow personal variables to change over time, that is, differ between concepts.

Table 5.2.: Within-sample predictive accuracy measures WAIC and LOO-CV for different models and data sets. Models are sorted ascending by criterion value, so best model is listed first. For an explanation of the table header see[11]. For a graphical version see Figure 5.13 on page 129.

Data Set	Model	Criterion	Value	p_{Value}	d_{Value}	w_{Akaike}	SE	d_{SE}	Warning
$\mathscr{D}_1$	$\mathscr{M}_3$	WAIC	−138.22	100.86	0	0.85	23.70	0	1
	$\mathscr{M}_1$		−23.53	45.93	114.69	0.15	32.96	25.64	1
	$\mathscr{M}_2$		213.91	45.50	352.13	0	6.16	21.28	1
	$\mathscr{M}_3$	LOO-CV	−88.64	125.65	0	0.78	20.66	0	1
	$\mathscr{M}_1$		−12.17	51.61	76.47	0.22	33.03	25.28	1
	$\mathscr{M}_2$		221.15	49.12	309.79	0	6.37	18.32	1
$\mathscr{D}_2$	$\mathscr{M}_3$	WAIC	−25.60	57.95	0	0.99	16.14	0	1
	$\mathscr{M}_2$		48.94	31.75	74.54	0	10.31	9.15	1
	$\mathscr{M}_1$		49.85	20.61	75.46	0.01	14.85	15.50	1
	$\mathscr{M}_3$	LOO-CV	−4.97	68.26	0	0.94	14.95	0	1
	$\mathscr{M}_1$		54.06	22.71	59.03	0.06	14.57	15	1
	$\mathscr{M}_2$		63.72	39.14	68.69	0	10.71	9.58	1
$\mathscr{D}_3$	$\mathscr{M}_3$	WAIC	−30.70	65.29	0	0.75	22.13	0	1
	$\mathscr{M}_2$		−9.81	51.13	20.89	0.01	20.50	10.90	1
	$\mathscr{M}_1$		6.40	31.36	37.10	0.24	29.21	27.26	1
	$\mathscr{M}_3$	LOO-CV	−2.99	79.14	0	0.44	22.47	0	1
	$\mathscr{M}_2$		2.77	57.42	5.76	0.29	20.86	10.18	1
	$\mathscr{M}_1$		11.84	34.08	14.83	0.28	28.93	26.99	1
$\mathscr{D}_4$	$\mathscr{M}_1$	WAIC	31.62	50.09	0	1	28.57	0	1
	$\mathscr{M}_3$		243.66	48.39	212.04	0	7.54	30.98	1
	$\mathscr{M}_2$		264.80	41.95	233.17	0	6.08	31.10	0
	$\mathscr{M}_1$	LOO-CV	44.08	56.32	0	1	28.61	0	1
	$\mathscr{M}_3$		253.10	53.11	209.02	0	7.97	30.89	1
	$\mathscr{M}_2$		272.25	45.67	228.17	0	6.40	30.87	1
$\mathscr{D}_5$	$\mathscr{M}_2$	WAIC	−76.09	30.98	0	0.74	17.07	0	1
	$\mathscr{M}_1$		−71.07	13.60	5.02	0.26	35.50	34.65	0
	$\mathscr{M}_3$		−29.48	46.03	46.61	0	19.39	7.47	1
	$\mathscr{M}_2$	LOO-CV	−69.68	34.19	0	0.73	17.27	0	1
	$\mathscr{M}_1$		−69.02	14.62	0.66	0.27	35.53	34.80	1
	$\mathscr{M}_3$		−16.71	52.41	52.97	0	19.59	7.75	1

[11] `https://docs.pymc.io/notebooks/model_comparison.html`. `pWAIC` and `pLOO` are both reported in p_{Value}. `dWAIC` and `dLOO` are both reported in d_{Value}. `weight` is renamed to w_{Akaike}. `dSE` is renamed to d_{SE}. `var_warn` and `shape_warn` are both reported in `Warning`.

5.3.2 Model extension – a case study

The best model according to the model comparison and model selection process, described in the previous section, is model $\mathscr{M}_3$. As was explained earlier, "best" is not to be understood in absolute terms but has many limitations and constraints. But for the given predictive accuracy metrics, the observable variables task success and mission score, and not too many concepts, $\mathscr{M}_3$ is the best model out of the three analyzed models.

However, $\mathscr{M}_3$ does not support all observable variables yet. In this section I will describe the process of extending $\mathscr{M}_3$ to incorporate a model for the mission time. Due to time limitations it was not possible to implement a model that supports all observable variables that were identified in section 4.2.1 Identify the data (page 78). This is part of the future work.

As described earlier, mission time t_{pc} is a continuous variable and, unlike s_{pc}, can take on any positive value. t_{pc} is assumed to be measured in minutes. I assumed t_{pc} to be composed of a minimal mission time, fixed for a given concept, and two additive components that are dependent on gcl_{pc} and δ_{pc}. The influence of the deterministic variables is mitigated by two multiplicative factors α_t and β_t, which are modeled as personal group variables. This is very similar to the way the original moel for s_{pc} was extended to a hierarchical model in $\mathscr{M}_3$. As a result of these considerations, a fourth model $\mathscr{M}_4$, which is called `CogiumImprovedHierarchical3Obs` in the CogIUM package, was implemented and tested. The model details can be revisited in section 4.3.2 Model $\mathscr{M}_4$ – hierarchical CogIUM for three observations (page 99) and Table A.1 on page 168.

Data set $\mathscr{D}_6$, which is identical to $\mathscr{D}_2$ but with values for t_{pc}, will be the data set on which the posterior predictive check is based on. I will present the same plots for $\mathscr{M}_4$ as in the previous sections for the other three models to allow for comparison. Inter-individual and inter-group differences are shown for motivation m_p, prior knowledge ψ_p, GCL gcl_{pc}, and free working memory capacity δ_{pc} (Figures 5.14 and 5.15 on the next page and on page 134). The posterior predictive check is shown for the three observable variables task success k_{pc}, mission score s_{pc}, and mission time t_{pc} (Figures 5.16 and 5.17 on page 135 and on page 136).

The posterior predictive check shows clearly that $\mathscr{M}_4$ is able to capture the structure of the observable variable t_{pc}, which was the designated goal for this model. The true values for each subject are contained in the central 50 % range of the model's simulated data and the average distribution of all model's predictions matches the distribution of the observation. $\mathscr{M}_4$ is capable of capturing the essence of the observable variable t_{pc} while holding the performance of its predecessor with regard to the other two observable variables s_{pc} and k_{pc}. Furthermore, $\mathscr{M}_4$ assumes group differences as well as individual differences for motivation, prior knowledge, GCL, and free working memory capacity, showing different group mean distributions for all these variables.

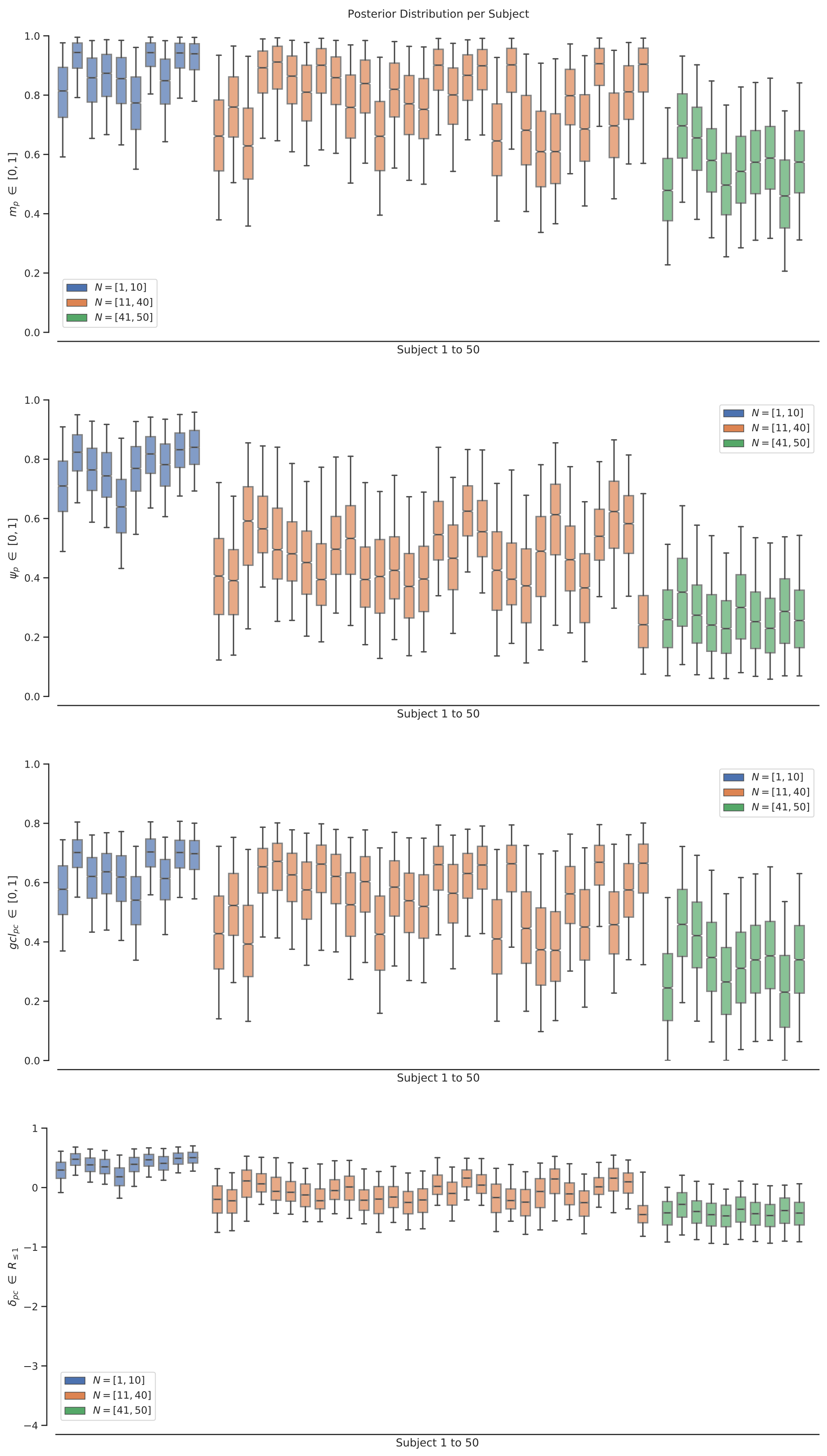


Figure 5.14.: Individual differences in the posterior distribution of $\mathscr{M}_4$, based on $\mathscr{D}_6$. (Top) Motivation m_p. (Second top) Prior knowledge ψ_p. (Second bottom) GCL gcl$_{pc}$. (Bottom) Free working memory capacity δ_{pc}. For further explanations see Figure B.4 on page 172.

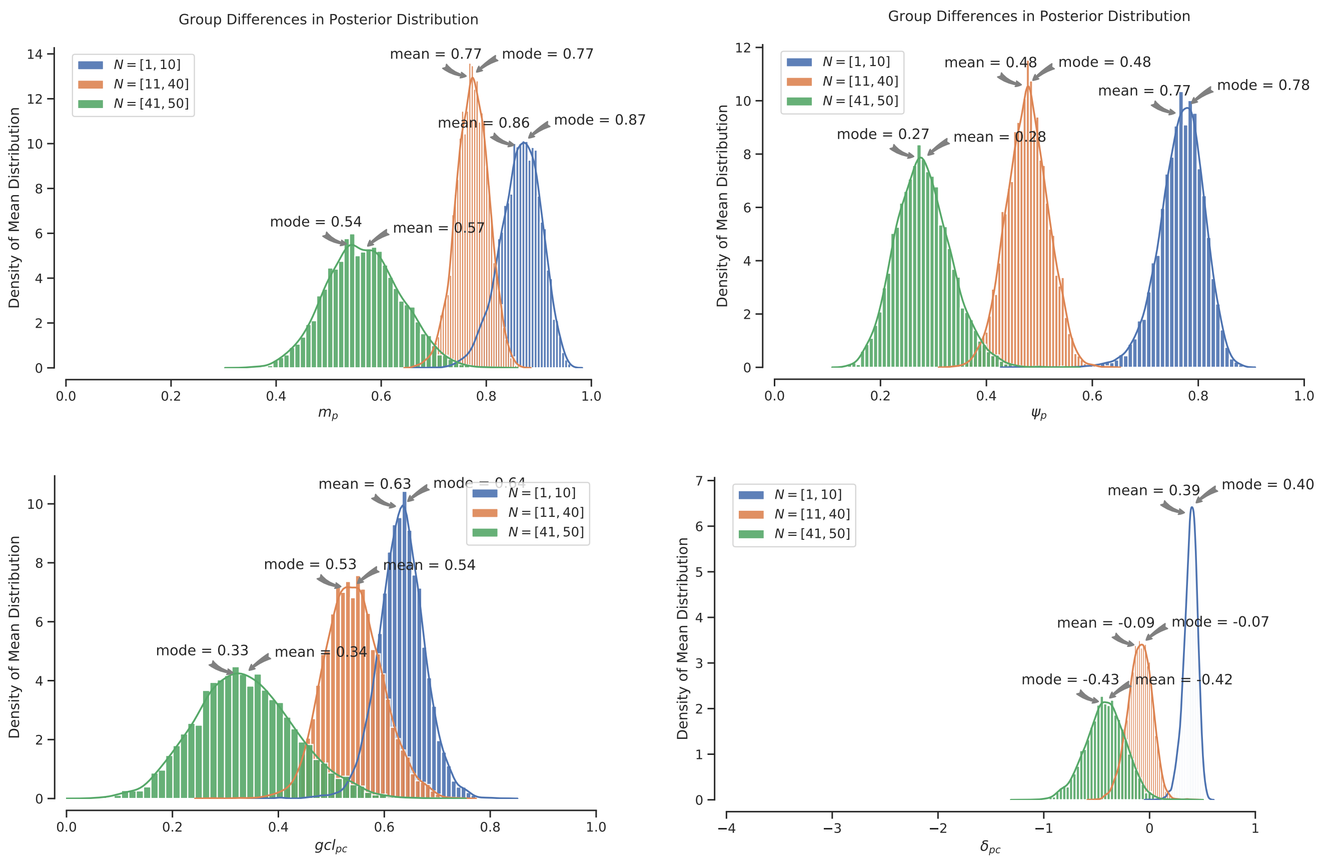


Figure 5.15.: Group differences in the posterior distribution of $\mathscr{M}_4$, based on $\mathscr{D}_6$. (Top left) Motivation m_p. (Top right) Prior knowledge ψ_p. (Bottom left) GCL gcl$_{pc}$. (Bottom right) Free working memory capacity δ_{pc}. For further explanations see Figure B.4 on page 172.

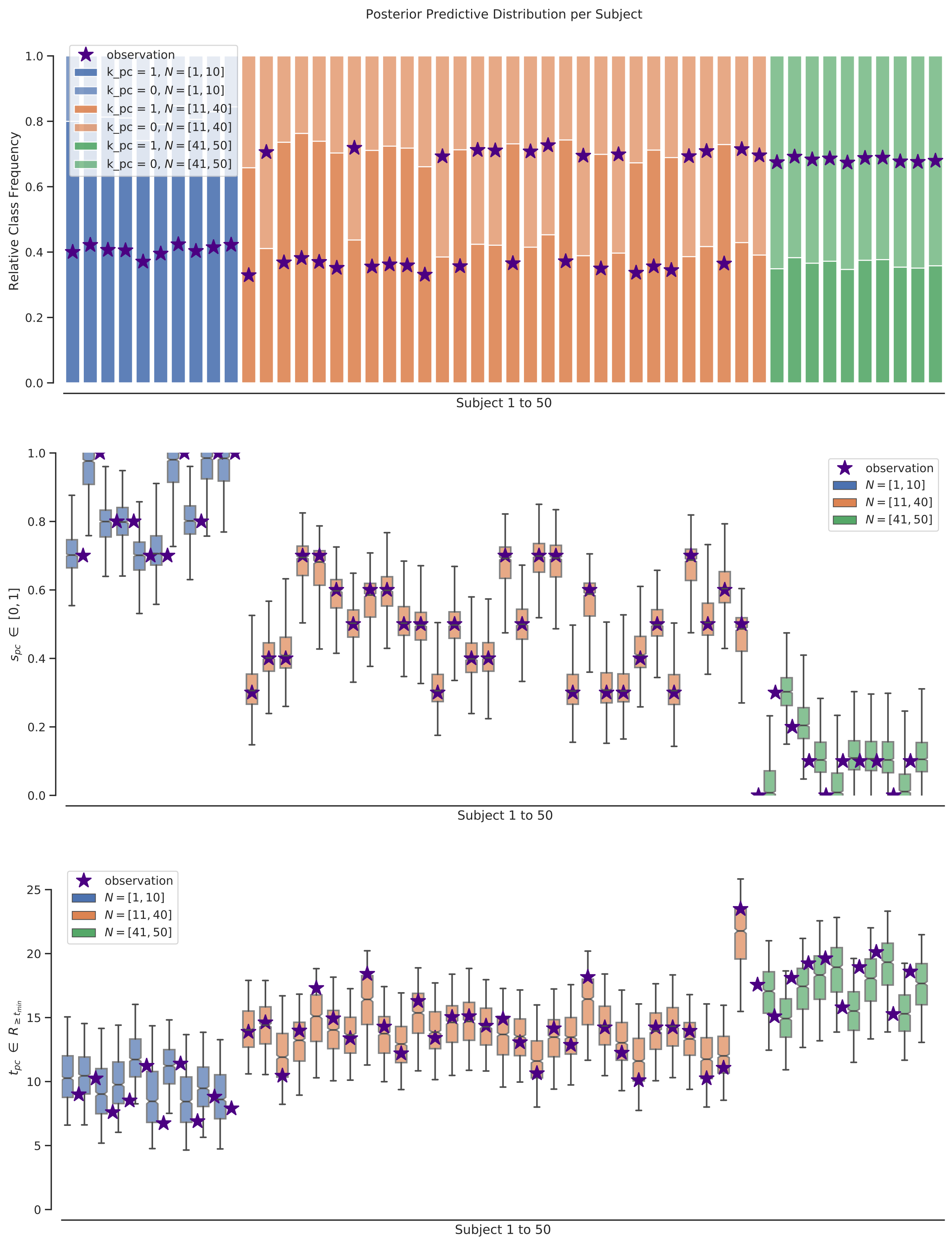


Figure 5.16.: Posterior predictive check of three observable variables for model $\mathscr{M}_4$, based on data set $\mathscr{D}_6$. (Top) Task success k_{pc}. (Middle) Mission score s_{pc}. (Bottom) Mission time t_{pc}. For each subject 1000 samples were drawn from the model's posterior predictive distribution. Because k_{pc} is a continuous variable, class frequencies are shown for each subject. The true value is marked. For further explanations see Figure B.5 on page 173.

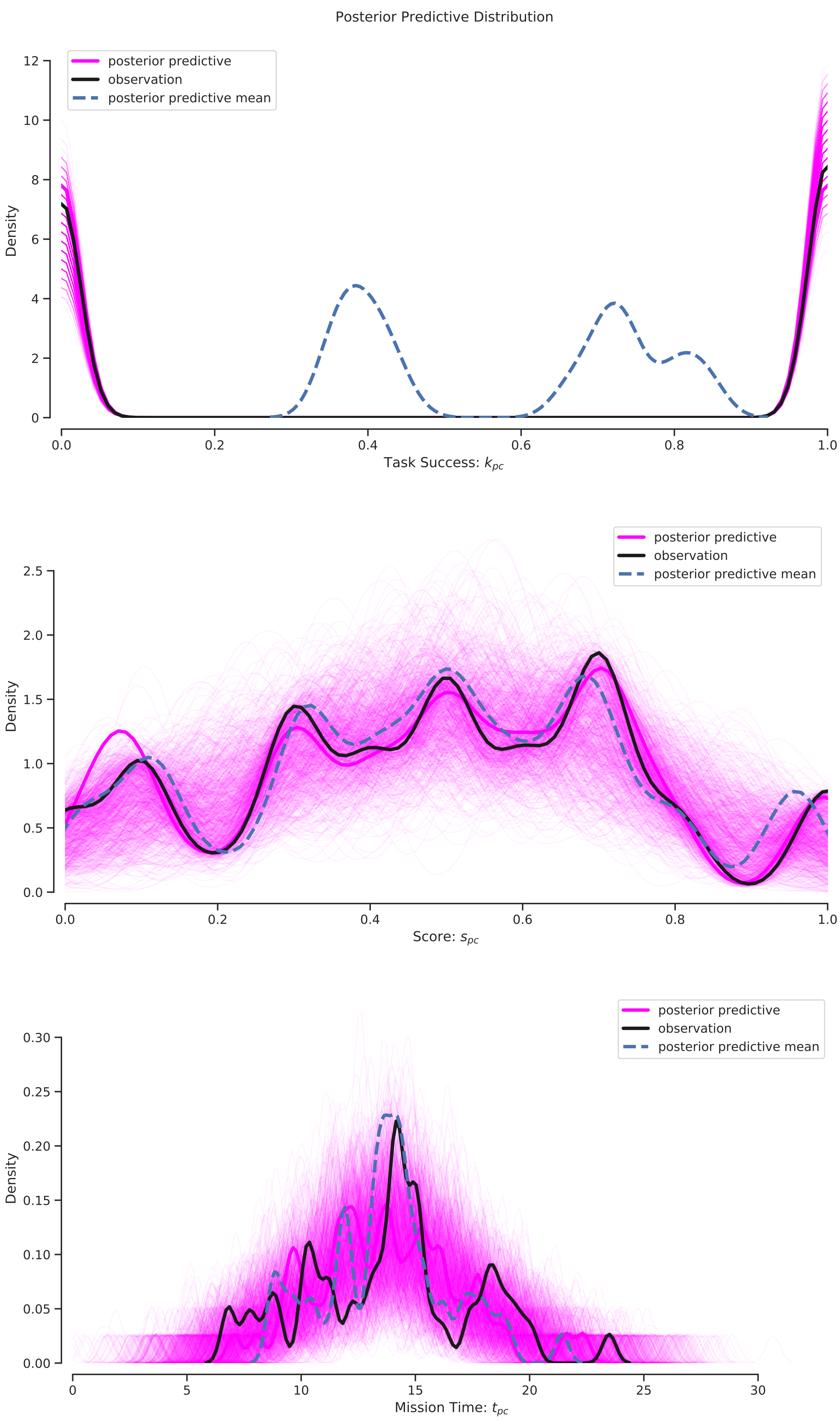


Figure 5.17.: Posterior predictive check for three observable variables for model $\mathscr{M}_4$, based on data set $\mathscr{D}_2$. (Top) Task success k_{pc}. (Middle) Mission score s_{pc}. (Bottom) Mission time t_{pc}. For each subject 1000 samples were drawn from the model's posterior predictive distribution. Each subplot shows the distribution of single samples as well as the model's average prediction. For further explanations see Figure B.5 on page 173.

5.3.3 Extended model validation

The previous section explained the process of extending model $\mathscr{M}_3$ to incorporate the observable variable mission time t_{pc}. The model that came out of this process is $\mathscr{M}_4$, the final model of this thesis.

Like with the models before, $\mathscr{M}_4$ was not only tested on data set $\mathscr{D}_6$, but on a variety of data sets representing different characteristics of plausible observable data (Table 5.3 on the next page). All data sets contain the three observable variables task success k_{pc}, mission score s_{pc} and mission time t_{pc}.

Reporting results for $\mathscr{M}_4$ is more complicated than with the model comparison before. The reported WAIC and LOO-CV values of the model comparison section were calculated for different models but the same data set. However, in this section I have only one model with different data sets. But the data sets $\mathscr{D}_7$ to $\mathscr{D}_{14}$ are all set up to have 90 observations. They only vary in the number of subjects, number of groups, number of concepts or in all of the three. $\mathscr{D}_7$, for example, has 30 subjects, all in a single group of type "good", whereas $\mathscr{D}_{14}$ has 3 different concepts, and for each concept 10 subjects in one group, but each concept has a different group type. Both data sets consists of 90 observations in total. Because the predictive accuracy measures WAIC and LOO-CV do not know about the internal structure of the observable data, I can compare at least the values of WAIC and LOO-CV that were measured based on data sets with equal number of observations. Because it is only one model that I discuss in this section, differences in the predictive accuracy mean that the model's performance differ with the data sets.

The model's predictive accuracy for different data sets based on the criteria WAIC and LOO-CV are reported in Table 5.3 on the following page. All columns that deal with differences between models are obsolete: d_{Valu}, w_{Akaike}, d_{SE}. Data sets are sorted ascending by criterion WAIC, so the data set with the best model's performance is listed first. WAIC was preferred over LOO-CV because it is a fully Bayesian criterion. Besides, LOO-CV and WAIC produce the same order with exception for data sets $\mathscr{D}_{10}$ and $\mathscr{D}_{13}$, which would be swapped if sorted by LOO-CV. According to the results there seem to be three groups of data sets: group one with $\mathscr{D}_{11}, \mathscr{D}_7, \mathscr{D}_{12}$, group two with $\mathscr{D}_{10}, \mathscr{D}_{13}, \mathscr{D}_9$, and group three with $\mathscr{D}_{14}, \mathscr{D}_8$.

The best results are obtained for highly structured data sets, as would be expected, like $\mathscr{D}_7$, which only contains one group of type "good", or $\mathscr{D}_{11}$, which is nearly identical because 29 subjects are in one group of type "good" and only a single subject is in a group of type "bad". Normally, one would expect the model to perform better on $\mathscr{D}_7$ than on $\mathscr{D}_{11}$, because the model has not enough information to model the single subject in the "bad" group. But the differences in the results of the predictive accuracy measures are not deterministic, as each data set was generated with randomness, as was explained in section 5.2.1 The data generator (page 109). To mitigate the randomness and get more robust results, one would have to repeat the analysis with multiple data sets of the same composition and average the model's performance on each data set. The good performance of $\mathscr{M}_4$ on $\mathscr{D}_{12}$ is surprising, given that this data set consists of three distinct groups, each consisting of 10 subjects, but it seems that the number of subjects suffices for the model to learn the structure that is entailed in the observable data.

The next group of data sets is characterized by two groups, one being in two of three cases dominant. Interestingly, the model performs nearly identical on $\mathscr{D}_{10}$ and $\mathscr{D}_{13}$, although they seem to be quite different. The first data set consists of two imbalanced groups: a "good" group of 25 subjects and only 5 subjects

[12] `https://docs.pymc.io/notebooks/model_comparison.html`. `pWAIC` and `pLOO` are both reported in p_{Value}. `var_warn` and `shape_warn` are both reported in `Warning`.

Table 5.3.: Overview of the data sets used in the model validation process of $\mathscr{M}_4$.

Data Set	N	N_p	N_c	N_g	Group Types	Splits
$\mathscr{D}_6$	150	50	1	3	[‘good’, ‘average’, ‘bad’]	$[10, 30, 10]$
$\mathscr{D}_7$	90	30	1	1	[‘good’]	$[30]$
$\mathscr{D}_8$	90	30	1	1	[‘random’]	$[30]$
$\mathscr{D}_9$	90	30	1	2	[‘good’, ‘bad’]	$[15, 15]$
$\mathscr{D}_{10}$	90	30	1	2	[‘good’, ‘bad’]	$[25, 5]$
$\mathscr{D}_{11}$	90	30	1	2	[‘good’, ‘bad’]	$[29, 1]$
$\mathscr{D}_{12}$	90	30	1	3	[‘good’, ‘average’, ‘bad’]	$[10, 10, 10]$
$\mathscr{D}_{13}$	90	30	1	3	[‘average’, ‘good’, ‘bad’]	$[20, 5, 5]$
$\mathscr{D}_{14}$	90	10	3	1	[‘bad’], [‘good’], [‘average’]	$[10], [10], [10]$

N: total number of observations, N_p: number of subjects, N_c: number of concepts, N_g number of groups per concept.

Table 5.4.: Predictive accuracy WAIC and LOO-CV values for model $\mathscr{M}_4$ and eight data sets. Results come from the PyMC3 functions `pm.waic()` and `pm.loo()`. Data sets are sorted ascending by criterion WAIC, so data set with best model performance is listed first. For an explanation of the table header see[12].

Data Set	Criterion	Value	p_{Value}	SE	Warning
$\mathscr{D}_{11}$	WAIC	−2.32	60.43	17.77	1
	LOO-CV	25.45	74.31	18.42	1
$\mathscr{D}_7$	WAIC	8.72	56.82	29.00	1
	LOO-CV	31.62	68.27	29.43	1
$\mathscr{D}_{12}$	WAIC	16.10	81.26	20.45	1
	LOO-CV	42.32	94.38	20.69	1
$\mathscr{D}_{10}$	WAIC	57.98	66.27	19.59	1
	LOO-CV	87.94	81.25	20.75	1
$\mathscr{D}_{13}$	WAIC	59.81	68.43	23.21	1
	LOO-CV	86.76	81.91	23.44	1
$\mathscr{D}_9$	WAIC	70.31	77.40	22.23	1
	LOO-CV	92.41	88.44	22.32	1
$\mathscr{D}_{14}$	WAIC	150.28	27.19	26.03	1
	LOO-CV	157.98	31.04	26.09	1
$\mathscr{D}_8$	WAIC	169.90	70.24	15.57	1
	LOO-CV	188.65	79.61	17.15	1

in the “bad” group. The second data set consists of three imbalanced groups, an “average” group of 20 subjects, and two groups, with 5 subjects each. A higher number of subjects helps the model in two ways: First it makes inference on global variables more stable because all subjects and concepts share these global variables, secondly, it improves inference on conceptual latent variables because these are fixed for a given concept.

The last group of data sets either consists of data sets with multiple concepts or less structured data. $\mathscr{D}_{14}$ consists of three concepts, each concept with a group of different group type. Although 30 subjects are more than enough for the model to learn the structure of the data, as was seen with $\mathscr{D}_8$, on which the model had the second best performance, the combination of three different concepts the model has to explain simultaneously is quite challenging. I already explained the nature of this challenge in the section about model comparison and model selection. As long as the model is not designed to vary some of the personal variables between concepts, it has not enough flexibility to model the observable variable alone with the latent variables of the conceptual group. $\mathscr{D}_8$ shows another problem the model can encounter: unstructured data. In comparison to the group types “good”, “bad”, and “average”, which are created by the data generator of the CogIUM Package to follow a certain performance logic and produce data that reflects these performance levels, the “random” group type has not such logic and randomly produces valid but not plausible observable data, at least not plausible in the sense that we would expect to find clusters of individuals that behave similarly when sharing similar characteristics.

A final word on the number of observations the model need to make accurate predictions. This number is, in fact, quite low. For only a single subject, the model’s predictions are already in the right area for k_{pc} and t_{pc}, only the distribution of the predictions for s_{pc} span the whole range of possible values for this variable. However, already with 5 subjects, predictions improve up to the point where they are spot on (Figure 5.18 on the next page). Bayesian methods are already a good way to deal with so-called cold start problems, that is, problems where we have not enough data at the beginning to run a data-driven application, because the uncertainty in the predictions is a inherent output of the Bayesian inference. Our model can be run on a data set with a single observation, but predictions will be quite cruel and the prior assumptions will dominate the inference.

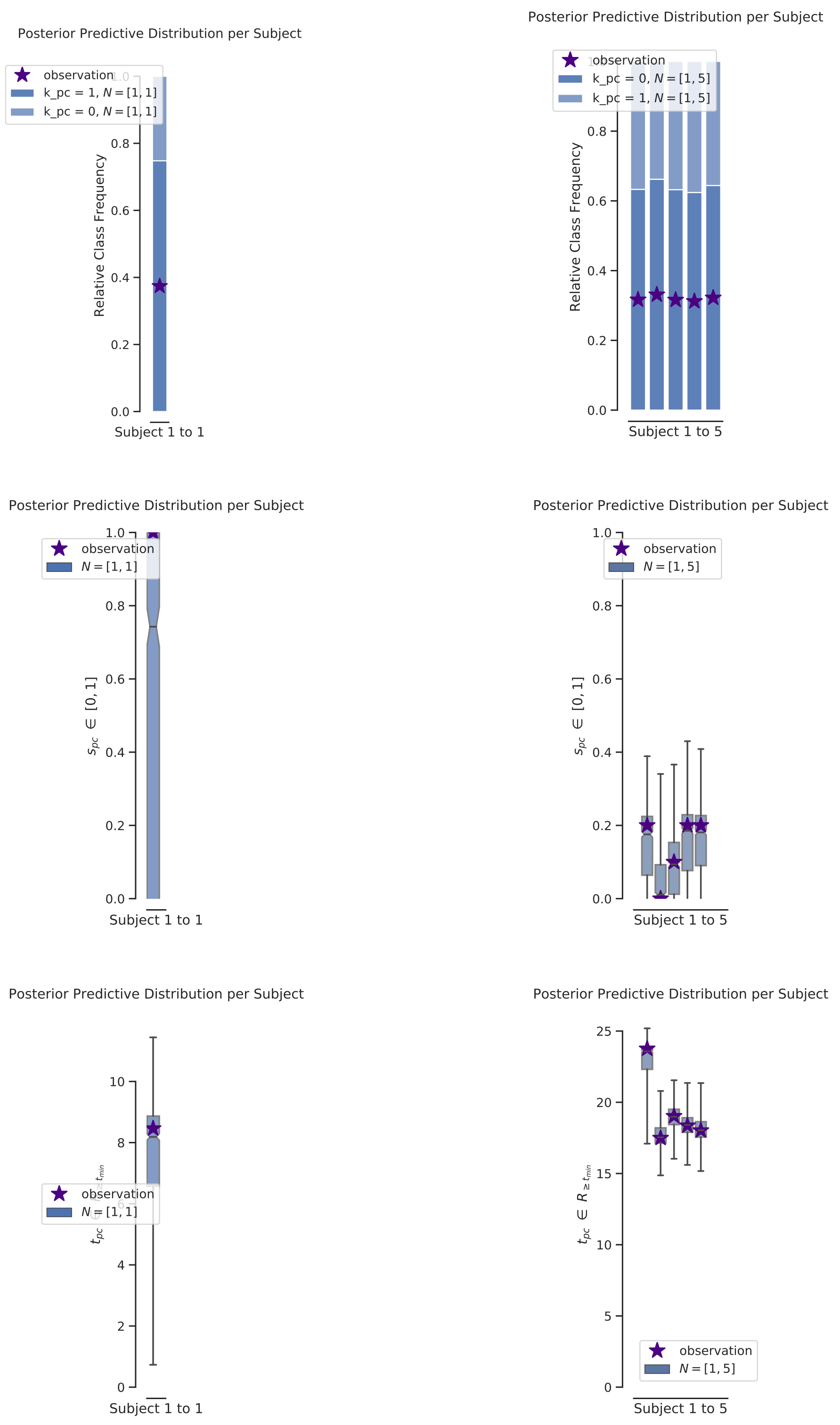


Figure 5.18.: Influence of sample size on model performance for $\mathscr{M}_4$. Compared are results for (top) k_{pc}, (middle) s_{pc}, and (bottom) t_{pc}. Each subplot shows the model's performance on a data set with (left) a single "good" subject versus a data set with (right) 5 "bad" subjects. Note the differences in the 95 % confidence intervals represented by the whiskers.

5.3.4 Discussion

In the previous sections, I presented the model comparison and model selection process and showed how to evolve from the original model $\mathscr{M}_1$ over an alternative model $\mathscr{M}_2$ to a hierarchical model $\mathscr{M}_3$ with the aim to enable the model to capture the main tendencies in the observed data. Model comparison was done for five different data sets and three different metrics. After deciding to stay with model $\mathscr{M}_3$ as the best model for the data, I derived the final model $\mathscr{M}_4$ by adding the ability to explain a third observed variable to the model. I evaluated the validity of the model and the model's predictive accuracy on nine different data sets. The final model was able to predict all individual observations with a satisfactory accuracy as well as to capture group differences that were present in the observed data. The model works best for one concept and can successfully explain three continuous observed variables. More than one concept becomes challenging for the final model and the model's performance decreases with an increased number of concepts. However, this behavior was expected and the logical next step in extending the model is to give the model the ability to vary personal variables between concepts.

What are the outputs from the realized CogIUM? As I have chosen to realize the CogIUM with the help of HBMs, the main output from Bayesian inference is a posterior distribution over all model parameters. The gained posterior is represented by a trace object that holds the samples for each MCMC chain drawn by an MCMC sampler. The posterior distribution describes the model's belief about the distribution of values for all model parameters. For example, if we want to know about the learner's current level of motivation or GCL, the model can provide us with a complete answer inferred from the observed data. One form the answer can be given in are point estimates like the mean or mode value – also called the maximum a posteriori estimation (MAP) – of the posterior distribution. A second and much better way in which the answer can be given is in form of HDIs. An HDI contains the parameter values of highest probability and spans the x % most probable values. With a 95 % HDI we would obtain the 95 % most credible values. In this sense the HDI directly provides us with the model's uncertainty about a particular model parameter. When an adaptive system's response is based on the output of the CogIUM, which is the overall purpose because the inferred values should allow us to determine the best point in time when to adapt, then it is of utmost importance to consider and have an estimate of the uncertainty of the models output.

HDIs can be used for further analyses and to gain additional insights. If we are interested in whether some parameter value of the learner has exceeded a threshold or not, we can use the HDI to answer this question in combination with a region of practical equivalence (ROPE) around this threshold. For example, we might define that a learner's level of motivation value should not be below 0.2 and so we define the region $[0, 0.2]$ as a ROPE. If the 95 % HDI falls entirely outside the ROPE, that is, all values in the HDI are larger than the end point of the ROPE, then we reject the ROPE'd value, which means we do not believe that the learner's current level of motivation is below 0.2. More on this topic can be found in Kruschke and Liddell (2018b).

Another two pieces of information provided by the posterior distribution joint posterior distributions and posterior predictive distributions. The joint posterior distribution is the probability distribution for the simultaneous occurrence of at least two variables and carries more information than the marginal posterior distributions (Wagenmakers and Farrell, 2004). The joint posterior is in general not statistically independent and cannot be gained by just the product of the two marginals. If we are interested in the relationship between two ore more cognitive variables, we might, for example, look at the joint posterior distribution of prior knowledge and motivation or of free working memory capacity and motivation to see if the model provides evidence for any relationship between these variables. The posterior predictive distribution is a distribution over simulated data. It summarizes the model's belief about the distribution of future data and can be used for validation purposes or to compute the probability of new data points. After these remarks I come back to the model comparison process.

Besides the success in building and training the models, the model comparison process was not without difficulties. Although the PyMC3 library offers methods to calculate the two metrics WAIC and LOO-CV, both of them threw warnings for most of the models and data sets (column `Warning` in Tables 5.2 and 5.4 on page 131 and on page 138). As explained in Vehtari et al. (2016), the computation of WAIC should give a warning in the case any of the terms $V_{s=1}^{S} \log p(y_i|\theta^s)$ exceeds the value 0.4. In this case the number of effective parameters $\hat{p}_{\text{waic}}$ is unreliable. Likewise, the user should be warned if the estimated shape parameter $\hat{k}$ of the generalized Pareto distribution exceeds the value 0.7 for the computation of LOO-CV. Vehtari et al. suggest to directly sample from the posterior for the problematic i or use K-fold cross-validation. Because PyMC3 does not inform the user about how many samples in the computation of WAIC and LOO-CV were problematic, I tried to implement K-fold cross-validation. How to do K-fold cross-validation is also stated in Vehtari et al. I was able to implement a version of K-fold cross-validation that produced results comparable to PyMC3's WAIC and LOO-CV values for models with only global model parameters θ. However, for hierarchical models it is not clear from the equations alone how to compute $p(y_i|\theta^{k,s})$ because $\boldsymbol{\theta}$ contains the parameter for different groups. As the observations are modeled individually in the CogIUM models, $\boldsymbol{\theta}$ contains a variable for each concept and each person for the likelihood functions. For example in a 10-fold cross-validation with 50 subjects, I have to calculate the probability $p(y_i|\theta^{k,s})$ for a holdout of 5 observations and a posterior that was obtained from a training set with 45 observations. But because the model calculates an individual θ for each of the 45 subjects, I did not know how to aggregate $\boldsymbol{\theta}$ for the holdout data. I tried taking the mean of all $p(y_i|\theta^{s,k})$ for all groups in $\boldsymbol{\theta}$ as well as the maximum, so that the probability for the holdout is calculated as if the new data point would belong to the best group seen in the training, but both approaches did not produce results that were comparable to the WAIC and LOO-CV values of the PyMC3 implementation. Therefore, it is an open question how to handle model comparison if there are warnings during the computation of the predictive accuracy metrics.

Another metric I wanted to use are Bayes factors. However, the computation of Bayes factors require the calculation of the model marginal likelihood. One way to obtain the marginal likelihood of a model when using the probabilistic programming library PyMC3 is to use Sequential Monte Carlo sampler[13]. However, I was not able to set up the SMC sampler to work with my models because PyMC3 always raised warning about incompatible shapes. Thus, another open question is how to obtain marginal likelihoods and how to compute Bayes factors for the given models.

What can be said about the computational efficiency of the implemented models? Regarding data set $\mathscr{D}_2$ with 50 subjects, one concept, three groups and two observed variables, computation time did differ quite a lot between the four models. For only two chains, the computation of 1000 samples with

[13] https://docs.pymc.io/notebooks/Bayes_factor.html

500 samples as tuning phase, 6 cores and a target accepting rate of 0.95, model $\mathscr{M}_1$ took 1 minute and 31 seconds, model $\mathscr{M}_2$ took 31 seconds, and model $\mathscr{M}_3$ took 9 minutes. Model $\mathscr{M}_4$ took 16 minutes for a data set with 50 subjects, one concept, three groups and three observed variables. For practical applications, there must be a balance between the power of the model and the computation time the model fitting requires. Given the fact that the best model only explained three of the eight presented observable variables, it is clear that more work has to be done to obtain an efficient model that can be used in a real scenario where it suggests, based on its inference, the best time point of when to adapt to the learner.

One approach to improve the model's performance and computation time is to update priors[14]. This approach goes beyond the scope of this thesis, but PyMC3 offers a way to do this. The main idea is to use the posterior distribution of a previous run as the prior distribution of the current run. This has many advantages. We do not need to store the whole data set and we do not need to train the model again on all previous data just because a new subject was added. Computation time will benefit because, instead of non-informative priors, the MCMC sampler can start directly in regions with high probability mass, as long as the new data is not completely different from the old one. PyMC3 offers an `Interpolated` class object that takes the trace object from a sampling run as argument and returns a kernel density estimation of the posterior that can be used within a PyMC3 model as a probability distribution like any other probability distribution. With this extension, the model becomes independent of previous data because everything the model has learned from the data is encoded in the posterior distribution that allows any kind of inference.

Regarding the reliability of the computed results, the quality depends on the number of samples during the MCMC process. Bayesian inference is deterministic when done analytically, that is, when the posterior is calculated analytically. This, however, is not possible for real-world problems and so we have to rely on computer simulations to draw samples from the posterior distribution to gain an approximation of this distribution. It lies in the nature of the sampling process that the quality of the approximation is directly determined by the quality of the samples, which, in return, depends on the quality of the MCMC algorithm. Different algorithms will produce different results for finite number of samples, and all real-world computations are finite. The Hamiltonian Monte Carlo (HMC) algorithm used in PyMC3 is the No-U-Turn Sampler (NUTS), first introduced by Hoffman and Gelman (2011). NUTS proved efficient for high dimensional, complex hierarchical models and shows adequate convergence with significantly lesser number of iterations and no hand tuning at all (Chong and Lam, 2017; Monnahan and Kristensen, 2018). Nevertheless, other MCMC algorithms might change the results presented in this thesis. In addition to the choice of an MCMC algorithm, the number of samples is an important aspect for the quality of the approximation. As was discussed in section 2.1.2 Markov Chain Monte Carlo (page 37), Kruschke (2015) recommends an ESS of around 10,000 to ensure a good approximation of the posterior. Such a high number was not feasible for the simulations of this work. The simulation runs for the model comparison based on data set $\mathscr{D}_2$ had ESSs between 1000 and 4000. This does not mean that the results presented here are not accurate as a high ESS only guarantees a good quality for estimating rare events and the limits of HDIs. However, the overall shape of the posterior as well as its mean and mode values can be accurately computed with fewer samples and a smaller ESS. That said, it is important to check if a larger sample size changes the findings presented here.

[14] `https://docs.pymc.io/notebooks/updating_priors.html`

5.4 How to Extend the Model Further

Based on the previous discussion and the results of the model validation for model $\mathscr{M}_4$, I have already mentioned the two most prominent shortcomings that have to be addressed to build a complete CogIUM: the ability to accurately describe differences in the performance of subjects over multiple concepts and the ability to model all observable variables. I will shortly present a few ideas regarding these shortcomings.

The latter is easier to solve but was not done in this work solely due to time restrictions. I have already explained the basic procedure of extending the model to additional observable variables in section 5.3.2 Model extension – a case study (page 132). It is necessary to pick a probability distribution that best represents the shape of the observable variable and define its parameters in dependence of the model's latent variables. Regarding the set of observable variables described in Table 4.1 on page 78, the next four candidates would all be discrete variables, most of them limited to the range of positive numbers only. In this case it can be decided to either pick a discrete probability distribution that directly supports the target type of the observable variable, or to pick a continuous probability distribution and use some kind of post-processing to transpose the parameter values from the continuous range to a discrete range. But discretizing is always possible (Norets and Pelenis, 2018). The main problem with both discrete and continuous probability distributions in the same model is the issue of mixing other MCMC samplers with NUTS, as NUTS works only for continuous variables. The difficulty for NUTS arises from changes in discrete parameters that affect the continuous distribution's geometry so that adaptation may be inappropriate for the Hamiltonian Monte Carlo sampling. Gelman et al. (2014) provide a section about this problem but it is an ongoing research question. If a discrete distribution should be used, PyMC3 provides a large set of suitable distributions[15]. For the derived observable variables of Lost Earth, distributions like the binomial distribution, the beta-binomial distribution, or the Poisson distribution are appropriate choices because their domain is $\mathbb{N}_0$. The binomial distribution is especially suited for the variable required rounds n_{rnd} because it supports an upper limit given by the maximum number of rounds allowed for this mission.

The second shortcoming, the problem of static personal variables over different concepts, is harder to solve because it requires the structure of the model and the relationship between the latent variables to be changed. Personal latent variables that are fixed for a given subject can remain unchanged, for example a person's capacity of working memory. However, it is reasonable and mandatory that latent variables representing psychological states instead of traits do change with concepts, for example motivation which changes over time. The same is true for latent variables that are directly associated with a particular concept like prior knowledge. To make this possible, the simplest change in the model's structure is to change motivation and prior knowledge from group variables of one dimension to fully individual variables of two dimensions, so both become variables with an index pc. This would mean that there are values for each subject and for each concept, instead of only for each subject. Besides the simplicity of this change, it has to be ensured that variables with a temporal structure are represented correctly. This is the case for motivation which I assume will not change dramatically between consecutive missions but only slightly. A simple change in the model from a group variable to an individual variable for each data point does not respect this constraint and allows motivation to change arbitrarily between concepts.

[15] `https://docs.pymc.io/api/distributions/discrete.html`

Therefore, dynamic hierarchical models, that is, hierarchical models that consider time variation for the parameters through dynamic linear models, might be a more appropriate approach (Paez, 2013). This clearly remains an open research question.

A final word on incorporating additional cognitive variables and cognitive theories. It should not be harder than the integration of the CLT. For example if it is the intention to extend the model to incorporate a latent variable that can be interpreted as a learner's flow experience, it is necessary to define which components make up the construct of flow, how the single components can be described by a probability distribution, and how these latent variables contribute to the explanation of the observations. Besides the growing model complexity, I see no practical issues with extending the model to additional cognitive theories or variables. On the contrary, it is one of the strengths of the Bayesian approach to allow such extensions with ease.

6 Application Example

In this chapter I give an application example of how to use the CogIUM Python package. In addition, I will show the main output provided by the package.

I want to analyze the following scenario. I have collected data from 10 subjects playing a single mission of the game Lost Earth. The data consists of three observed variables: task success, mission score and mission time. The mission's difficulty was low and all subjects showed a good performance.

First I generate the observed data for the first mission.

```
Np = 10, Nc = 1
obs = ['k_pc', 's_pc', 't_pc'], groups = [['good']], splits=[[10]]

data = generator.generate_observations(nr_subjects=Np, nr_concepts=Nc, variables=obs, groups=groups,
    splits=splits)
```

The generated data is returned as a NumPy array with shape $(10, 1, 3)$ for 10 subjects, one concept and three observations. Because the data is generated for a good performing group, task success is always 1, mission score is always between 0.7 and 1.0 and the mission time is between 5 and 10 plus a random number.

```
array([[[ 1.    , 1.      , 7.24401135]],
[[ 1.     , 0.7     , 7.36746463]],
[[ 1.     , 1.      , 6.08471188]],
[[ 1.     , 1.      , 8.32547848]],
[[ 1.     , 1.      , 9.74794714]],
[[ 1.     , 0.7     , 12.19119183]],
[[ 1.     , 0.9     , 9.84261509]],
[[ 1.     , 0.8     , 11.54478263]],
[[ 1.     , 0.9     , 10.03550018]],
[[ 1.     , 0.9     , 8.34281885]]])
```

The next steps are building the model and fitting the model to the observed data. I choose the final model $\mathscr{M}_4$ as model for this example and draw 1000 samples from the posterior distribution after a tuning phase of 500 samples.

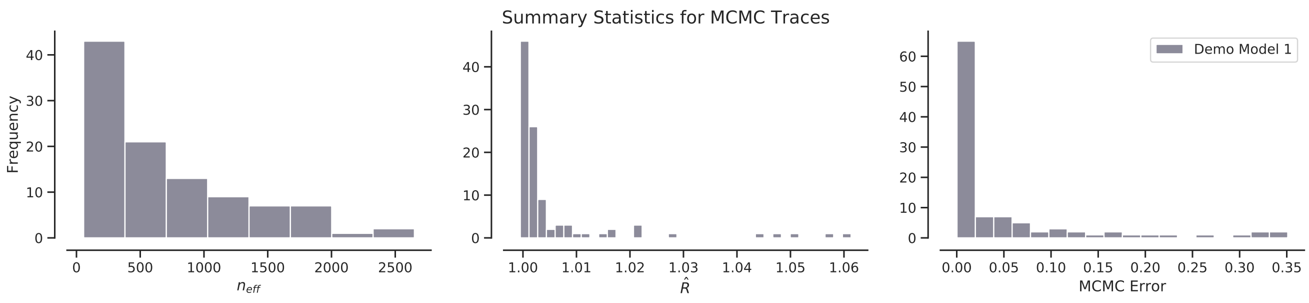
```
model_filename = 'data(10-1-3)_groups((good))_split((10))_desc(demoApp)'
model = CogiumImprovedHierarchical3Obs.load(model_filename)
trace = model.sample(nr_samples=1000, nr_tune=500, nr_chains=2, nr_cores=6, target_accept=0.95)
model.save(model_filename)
```

Unfortunately there are warnings during the MCMC sampling process. I am informed that the gelman-rubin statistic is larger than 1.05 for some parameters, which means that the MCMC chains have not converged yet.

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 6 jobs)
NUTS: [p_obs, t_min, ecl_c, icl_c, sigma_ecl, sigma_icl, mu_ecl, mu_icl, beta_t, alpha_t, nu_s, psi_p, m_p, mu_nu, sigma_beta, sigma_alpha, sigma_nu, sigma_t, sigma_s]
Sampling 2 chains: 100%|██████████| 3000/3000 [08:24<00:00,  4.03draws/s]
There were 138 divergences after tuning. Increase `target_accept` or reparameterize.
The acceptance probability does not match the target. It is 0.8943630725468116, but should be close to 0.95. Try to increase the number of tuning steps.
The chain reached the maximum tree depth. Increase max_treedepth, increase target_accept or reparameterize.
There were 86 divergences after tuning. Increase `target_accept` or reparameterize.
The gelman-rubin statistic is larger than 1.05 for some parameters. This indicates slight problems during sampling.
The estimated number of effective samples is smaller than 200 for some parameters.
```

Output of the first call of the `pm.sample()` method. The warnings indicate divergence of the chains.

I further analyze the quality of the MCMC chains with the help of the `plot_trace_summary()` function to get the effective sample size (ESS), the potential scale reduction $\hat{R}$ and the MCMC error. The ESS is quite small and a few $\hat{R}$ values are indeed in the region of 1.05.



Summary statistics for the MCMC chains of the first run.

Based on this information I decide to rerun the sampling with a larger tuning phase of 1000 samples.

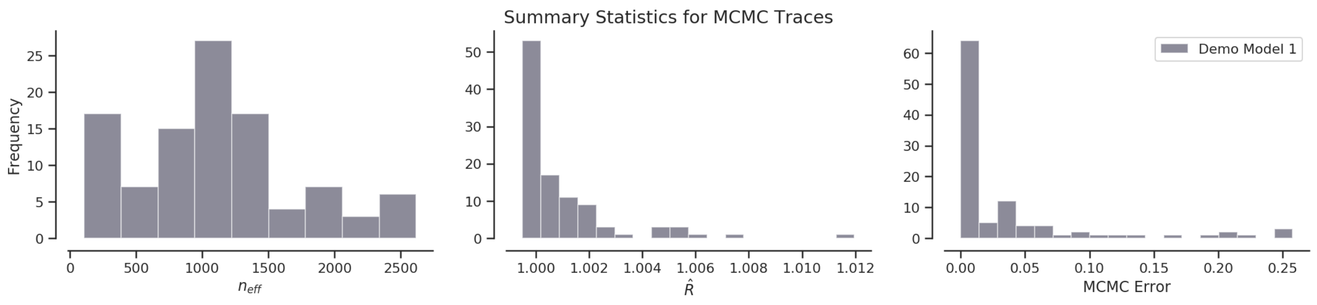
```python
model_filename = 'data(10-1-3)_groups((good))_split((10))_desc(demoApp)'
model = CogiumImprovedHierarchical3Obs.load(model_filename)

trace = model.sample(nr_samples=1000, nr_tune=1000, nr_chains=2, nr_cores=6, target_accept=0.95)
model.save(model_filename)
```

This time, there are still warnings but the warning with regard to the gelman-rubin statistic is gone. Interestingly, the run time was even shorter than for the first run with less samples.

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 6 jobs)
NUTS: [p_obs, t_min, ecl_c, icl_c, sigma_ecl, sigma_icl, mu_ecl, mu_icl, beta_t, alpha_t, nu_s, psi_p, m_p, mu_nu, sigma_beta, sigma_alpha, sigma_nu, sigma_t, sigma_s]
Sampling 2 chains: 100%|██████████| 4000/4000 [05:10<00:00,  5.13draws/s]
There were 61 divergences after tuning. Increase `target_accept` or reparameterize.
There were 64 divergences after tuning. Increase `target_accept` or reparameterize.
The estimated number of effective samples is smaller than 200 for some parameters.
```

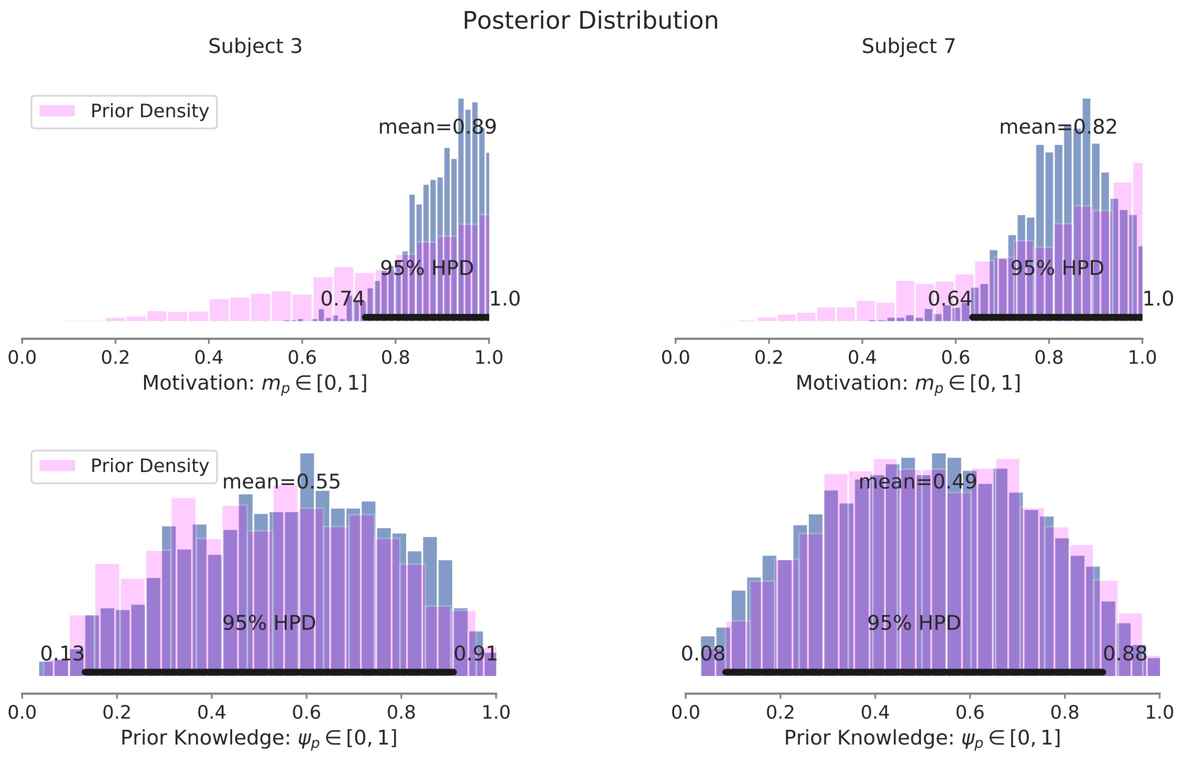
Output of the second call of the `pm.sample()` method. This time warnings do not indicate a divergence of the chains, only of single samples.

The output of the `plot_trace_summary()` function confirms that the quality of the MCMC chains has improved. The ESS is higher and there are less MCMC errors.



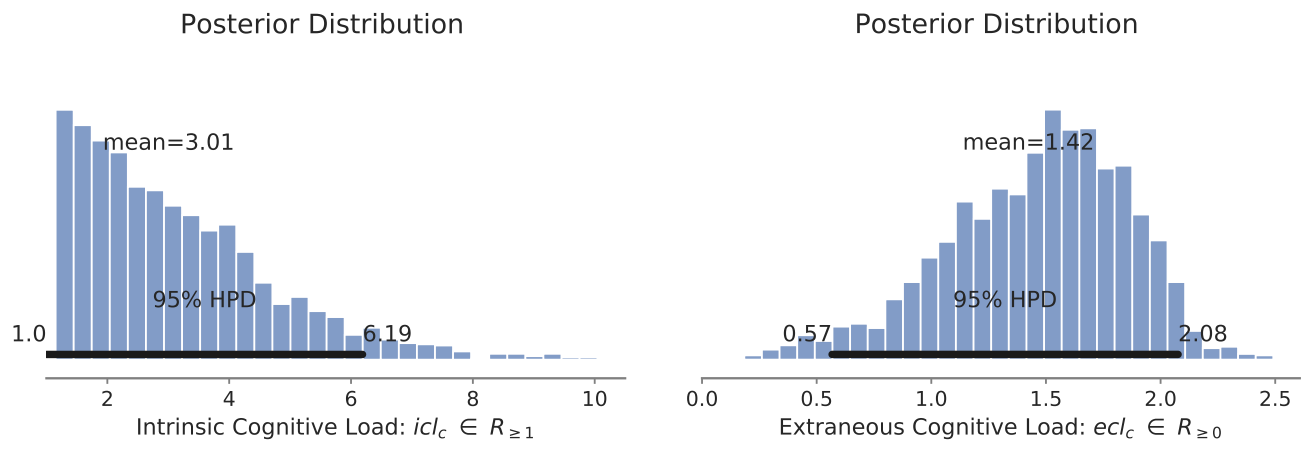
Summary statistics for the MCMC chains of the second run.

Now I begin to analyze the posterior distributions, the main output of the Bayesian inference. I first look at the posterior for motivation and prior knowledge for two of the ten subjects. Because all subjects have equal performance I do not expect to find significant differences in their posterior distributions. I can see that the posterior distribution for prior knowledge is not very different from the prior distribution which indicates that the model did not update that variable. However, the posterior distribution for motivation differs from the prior distribution so the model did update its prior assumptions based on the observed data. But, as expected, motivation does not differ between subjects. The 95 % HDI tells me that 95 % of the most credible values are between 0.24 and 1.0 for motivation.



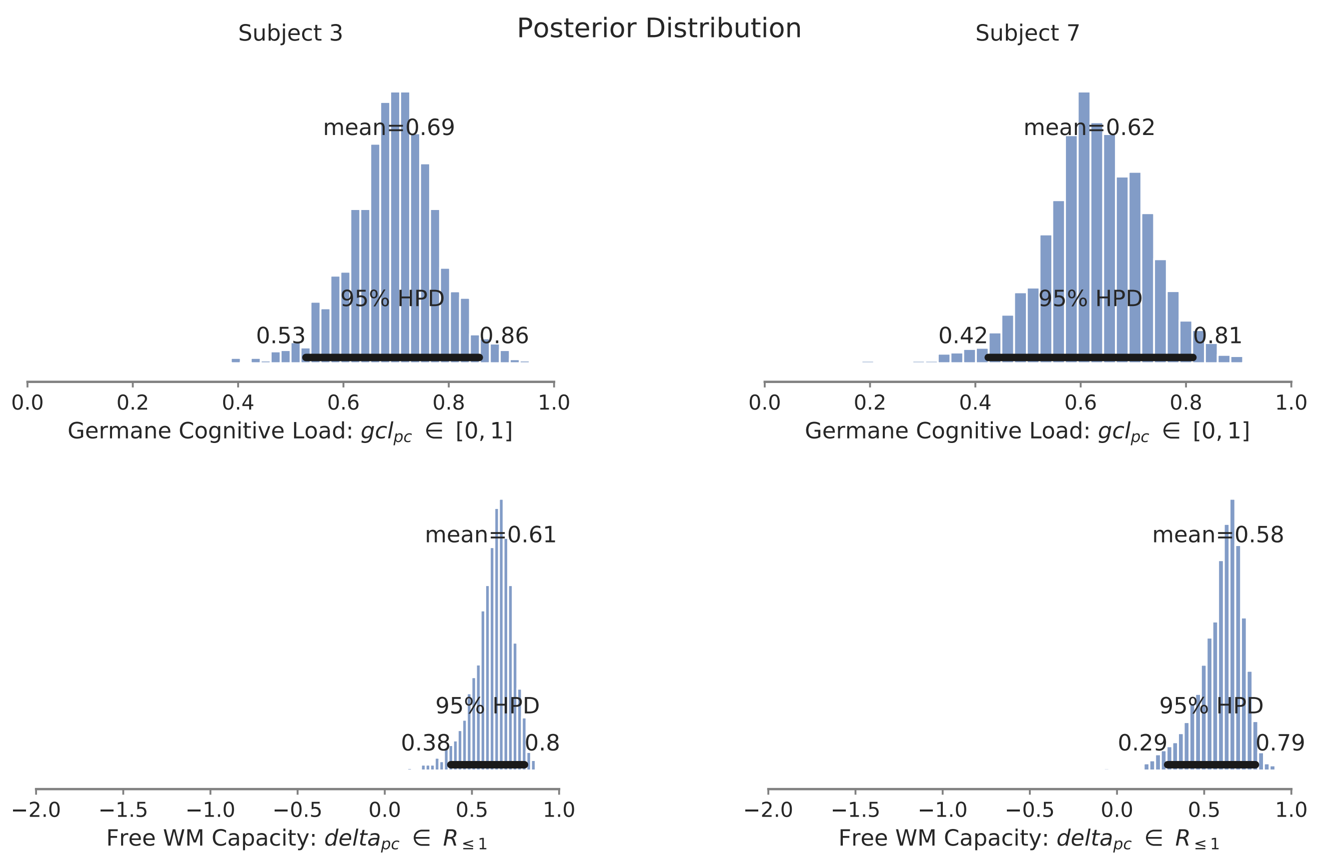
Posterior distribution for motivation and prior knowledge.

Next I have a look at the posterior distributions for the cognitive load variables ICL and ECL. I can see that the model estimates ICL to be between 1 and 6.19 with a mean value of 3.01 and ECL to be between 0.57 and 2.08. So the model estimates ICL to be higher than ECL. However, the value for ICL is relative low compared to a person's working memory capacity of 7 so this seems to confirm that the mission was easy.



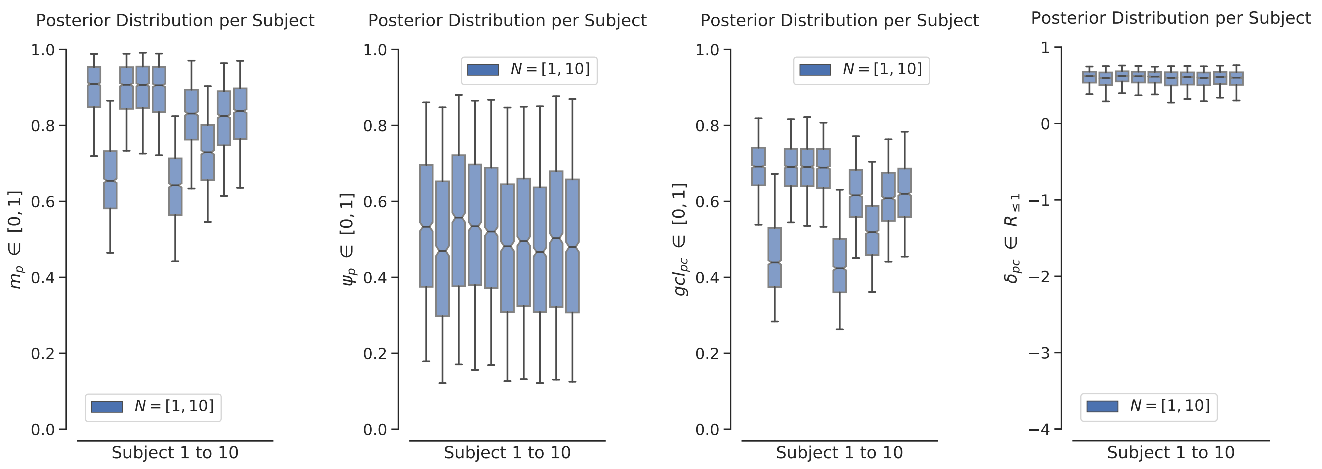
Posterior distribution for ICL and ECL.

This is further confirmed by checking the posterior distributions of GCL and free working memory capacity. GCL is estimated to be between 0.53 and 0.86 which is rather high and means that the person dedicates much of their resources to learning. The free working memory capacity is also high, which means that the working memory resources of that person are not depleted.



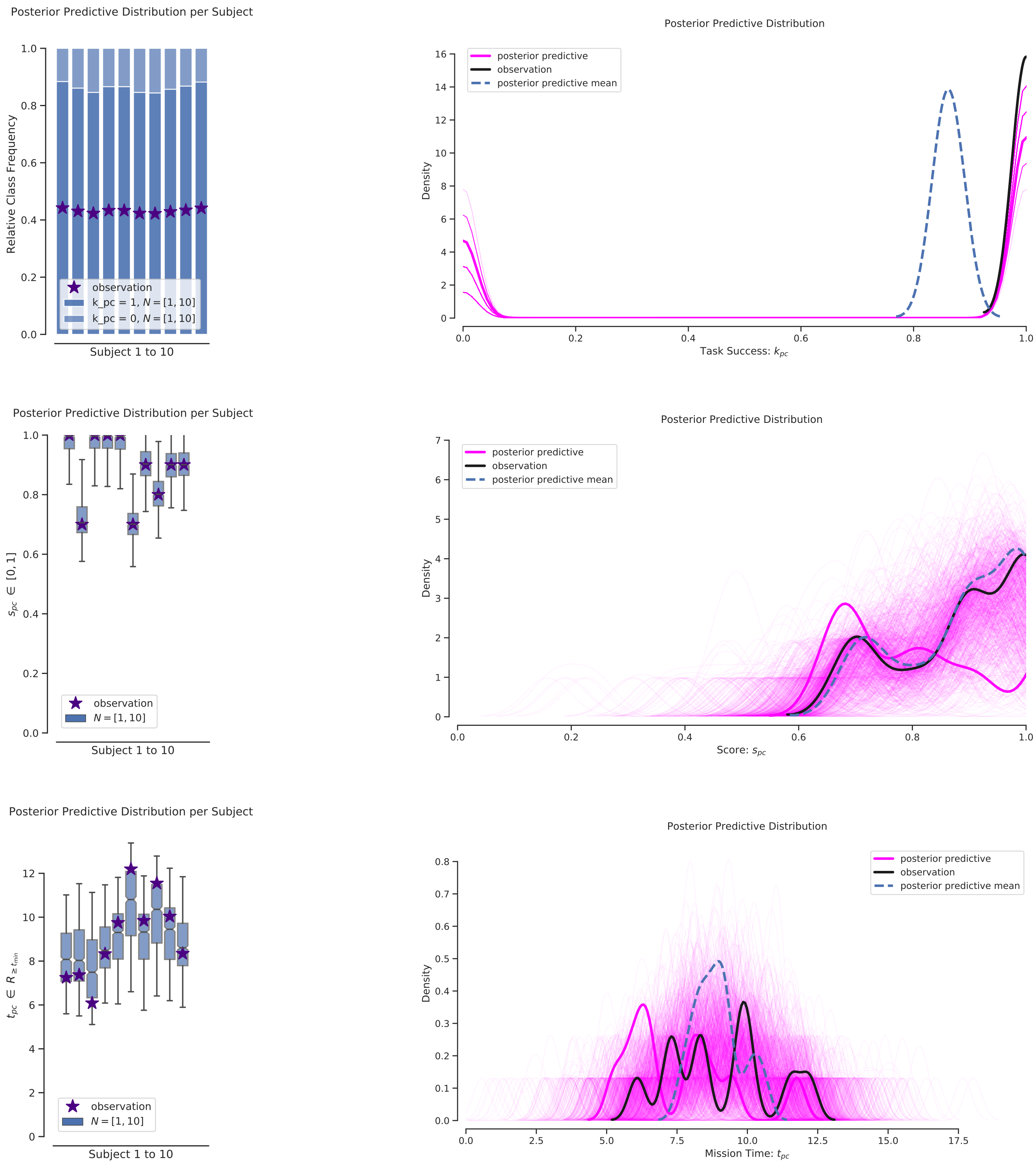
Posterior distribution for GCL and free working memory capacity.

A look at the output of the `show_group_differences()` function can confirm that the model has detected individual differences. However, the model has primarily changed motivation between different subjects, which also influences GCL because it is a deterministic variable dependent on motivation. Prior knowledge is more or less constant over all subjects and as a direct consequence so is free working memory capacity.



Individual differences in the posterior distribution for motivation, prior knowledge, GCL and free working memory capacity.

The final step of the analysis is to look at the posterior predictive distribution and to check if the model was able to accurately capture the structure of the observed data. The posterior predictive check tells me that the model successfully captured the structure of the data. It produces the right predictions for all subjects and all observable variables. However, it can be seen that the predictions are more accurate for mission score than for mission time, which is confirmed by comparing the model's average prediction distribution with the distribution of the observed data.



Posterior predictive check for task success, mission score, and mission time.

7 Conclusion and Recommendations

Adaptive educational serious games strive for the right balance between enjoyment and challenge: to motivate the learner and offer incentives and a playful experience while, at the same time, confronting the player with problems that facilitate their domain mastery and go beyond the current learner's skill level. To achieve this balance, a dynamic assessments of the learner's current cognitive state is mandatory. Only when adaptive measures are applied at the right time to meet the learner's current needs, it is possible for the learner to experience a state of flow and to remain in this state. The question of when to adapt is still an open research question for adaptive educational games.

The central component of an adaptive system for educational serious games is the student model that represents the learner's current cognitive state. If the student model is accurate and appropriate for a given learner and learning task, the model's estimate of the learner's current cognitive state can be used to derive and suggest adaptive measures to help navigating the player through the game on a smooth and engaging path. The realization of such a cognitive architectureas the main topic of this thesis.

The field of computational cognitive science offers a rich toolbox for approaches that can be used to build and train student models. Two approaches were chosen and tested for their applicability and practicality. Soar, a hybrid cognitive architecture, was used as the first approach to realize cognitive user models. Soar conducts a search through problem space to find a path from an initial state to a defined goal state by applying suitable operators to the current state. The second approach made use of Bayesian methods, especially hierarchical Bayesian models (HBMs). In Bayesian data analyses descriptive models are used that captures the essential structure of the observations with the help of probability distributions. Bayesian inference then allows to infer the posterior distribution of the model parameters based on prior knowledge about the model parameters and the probability of the actual observable data. To work with Soar, it is required to model the game in the Soar language, to define an initial state with all relevant attributes, and to specify the operators that represent the possible interactions. To work with Bayesian methods, it is required to define the relevant observable data, the model's latent variables that are the targets of the Bayesian inference, and the relationship between the latent and the observable variables, that is, the model's structure.

The digital educational serious game used in this work was Lost Earth, developed at the IAS of the Fraunhofer IOSB. The game is currently ported from the Havoc game engine to the Unity game engine and is only available as an early prototype with a subset of the original game's content. The first step of this thesis was an in-depth analysis of the interaction patterns found in the game. The results of this analysis were an activity diagram for the missions in Lost Earth and a set of observable variables that can be captured from the users' interactions with the game by Experience API statements. Three categories of observable variables were identified: general performance measures, domain-specific measures, and game-specific measures. For each observable variable its domain and numerical type was specified.

At this point of the thesis, I decided to no longer continue with the cognitive architecture approach. It seemed not possible to realize a cognitive user model with Soar 9 given the thesis' time constraints and the fact that Soar is primarily a problem solver and does not provide a learner's cognitive state as output. However, possible first starting points were mentioned and considerations were made as to how Soar could be used as a basis for adaptive systems.

Instead, HBMs provided a direct way to implement a cognitive user model that is based on the interactions between the learner and a serious game. The second approach required the development of a descriptive model. No source for a comparable procedure could be found in the literature that made use of a Bayesian model for digital educational serious games for inferring the learner's current cognitive state based on learner interactions. Therefore, the descriptive model had to be build from the ground. The concept of cognitive load and its constituting components were ideal candidates for the model's central latent variables. The relationship between the characteristics of the learning material and the characteristics of the learner, as explained by the Cognitive Load Theory (CLT), was the main source of inspiration for the model's structure. The main contribution of this thesis is the specification of four HBMs that explain the occurrence of observed data with the help of latent variables that are partly interpretable as cognitive variables. The models accept observations from multiple users and multiple missions.

The practical use of HBMs was shown in concrete examples by means of a model comparison and model selection process and the extension of existing models. It was demonstrated how to estimate a model's predictive accuracy based on the model's simulations given as the posterior predictive distribution. The results obtained were used to improve the original model until its predictive accuracy was consistent with the observed data. A model comparison ensured that the developed model was indeed to be preferred over the predecessor models. Likewise, it was shown how to extend the model with the ability to explain additional observable variables. Particular attention was paid to explaining the benefits of hierarchical modeling. The final model allows for inferring the learner's current cognitive state with respect to their level of motivation, prior knowledge of a concept, and invested working memory resources for learning. In addition, the model makes statements about the cognitive demands imposed by the learning material composed of the natural intrinsic complexity of the material and the extraneous additional cognitive load. Finally, free working memory capacity as the difference between the available working memory resources and the cognitive load imposed by the learning task can be interpreted as a measure of difficulty.

However, it has to be noted that the presented results are based on simulated data and not on empirical data. Moreover, the model's latent variables are based on cognitive theories, but the verification that the variables have a real equivalence has yet to be done. It is not clear whether differences in the model's personal variables correctly describe real world differences between subjects. All of this must be examined by a detailed model evaluation in the form of a user study still to be carried out.

7.1 Assessment of Requirements

Section 4.3 Realizing Cognitive User Models (page 80) listed 12 requirements that should be fulfilled by an ideal approach for realizing cognitive intelligent user models. This section provides a short comparison of the two approaches based on the degree to which they fulfilled the requirements. Because the use of Soar was discontinued in favor of HBMs, the evaluation is based on my knowledge about Soar and the experiences made during the conceptual phase of this thesis.

In my opinion, PyMC3 fulfills all of the 12 requirements, whereas Soar 9 only meets 7 out of 12 requirements (Table 7.1 on the next page).

Table 7.1.: Compliance with the requirements for the two frameworks Soar and PyMC3. A check mark (✓) marks the fulfillment of a requirement. If both approaches meet a requirement it is not implied they do so equally well.

	Approach	
Requirement	Soar 9	PyMC3
Actively developed	✓	✓
Appropriateness		✓
Availability	✓	✓
Extensibility		✓
Extensive support		✓
Generality	✓	✓
Intelligence	✓	✓
Interoperability		✓
Lightweight	✓	✓
Low dependencies	✓	✓
Proved and tested	✓	✓
Simplicity		✓
$\sum$	7	12

Regarding availability, both frameworks are publicly available and have hosted their implementation as public projects on GitHub[1]. Both are available for all major platforms. Soar can be downloaded as a stand-alone multi-platform version. PyMC3 is a Python library and can be easily installed with common package managers for Python like `pip` or `conda`.

Regarding being actively developed, both are still supported and actively developed, but PyMC3 has a much faster development cycle and therefore more recent releases. The latest stable version of Soar is Soar 9.6.0 from June 2017 and the last activity on GitHub dates back to December 2018. In comparison, the latest stable version of PyMC3 is version 3.6 and was released in December 2018, while the library is actively developed on GitHub on a daily basis.

Regarding being proved and tested, both frameworks have a substantial amount of literature supporting their application. Publications related to Soar are directly available via the official website[2] and Kotseruba and Tsotsos (2018) give an overview of the many applications of Soar. The case for PyMC3 is different because PyMC3 does not contribute new algorithms or architectures but implements algorithms for performing Bayesian inference accessible for a broad audience. PyMC3 was introduced to the scientific community by Salvatier et al. (2016) and offers a substantial library of examples with solutions to standard problems of the field[3].

Regarding extensive support, PyMC3 offers much more resources than Soar does. PyMC3 provides the user with a quick start, multiple in-depth guides, tutorials in form of Jupyter Notebooks that provide executable code, many examples how to use PyMC3 for specific use cases, and a documented API. Soar, instead, offers little online resources but mainly static documents provided as PDF files. The main documents provided by Soar are the Soar User's Manual from November 2017 with a total page count

[1] `https://github.com/SoarGroup/Soar` and `https://github.com/pymc-devs/pymc3`
[2] `https://soar.eecs.umich.edu/Soar-RelatedResearch`
[3] `https://docs.pymc.io/nb_examples/index.html`

of 313 and a tutorial divided into eight parts. Additional resources including developed agents, agent development tools, and domains are provided via the website. In my opinion, beginning with Soar takes a lot of time and requires at least a little understanding of the architecture and the Soar syntax to be productive, whereas PyMC3 offers a much smoother and more motivating beginning that leads to fast first results.

Regarding generality, both frameworks can rightfully claim to be general frameworks. Soar is per design a framework for general artificial intelligence and can be applied to any domain and any problem as long as the problem can be described with the tools provided by Soar. Likewise, PyMC3 is a framework for probabilistic programming that offers MCMC algorithms to solve Bayesian inference for a wide range of possible models. PyMC3 does not limit the user in the implementation of models as long as the model can be expressed with the provided probability distributions.

Regarding extensibility, PyMC3 offers the usage of custom distributions whenever the built-in probability distributions do not suffice. To my knowledge, Soar does not provide any possibilities to extend the framework.

Regarding low dependencies, both frameworks do not require third party libraries. Soar does not even require Java if used as a command line tool. PyMC3 only requires other Python packages that are automatically installed during the installation routine.

Regarding interoperability, PyMC3 has the clear advantage of working on and producing exclusively numerical data. xAPI statements can be easily processed once a parser has been written to extract the numerical values of the observed variables from the xAPI statements, which is straight forward thanks to the specification of the API. The output of PyMC3 is numerical data that can be easily accessed by an adaptive system. Working with Soar requires to extract the attributes for the initial state from the xAPI statements, which might be equally simple once a parser has been written. However, the output is much harder to process because Soar does not provide a standard output. The output has to be generated by the user via the `print` or `output` command or other diagnostic commands provided by Soar. This requires much more effort in comparison to the output provided by PyMC3 and is not easily transferable to other applications.

Regarding being lightweight, both frameworks are self-contained and relatively small, only a few tens of megabytes. Soar 9 has the advantage of a command line interface, while PyMC3 needs an up-to-date Python environment. Thus, Soar 9 can be seen as more lightweight than PyMC3.

Regarding appropriateness, this requirement was the main reason to discontinue the use of Soar as I could not see an easy way to implement a CogIUM with Soar. I have presented a few starting points, but in the end Soar 9 does not provide output that represents cognitive variables. In contrast to this, HBMs implemented with PyMC3 can be built to directly model desired cognitive variables and explain observed data and the output is easily interpretable due to the nature of probability distributions.

Regarding simplicity, PyMC3 requires a lot less understanding from the user than Soar 9 does. It is true that one has to know how to build (hierarchical) Bayesian models to work with PyMC3, but simple applications consist of only a few lines of code and the user needs no knowledge at all about MCMC algorithms to create output with PyMC3, although this knowledge becomes important when the MCMC algorithm get stuck. Working with Soar requires to learn the syntax for Soar programs and defining all operators before the agent is able to show meaningful behavior. I do not consider Soar to be simple and even simple problems like the Water Jug require a considerable amount of effort from the user. This is the price Soar has to pay for its generality.

Regarding intelligence, the frameworks cannot be compared. Soar is a framework that is built to allow general artificial intelligence and consequently offers mechanisms to elicit intelligent behavior. Soar supports four major learning mechanisms: chunking, semantic, episodic, and reinforcement learning (Lehman et al., 2006). These mechanisms can be used to generate all of the different representations of knowledge in Soar. PyMC3 is not a framework for realizing agents and thus has no learning mechanisms. However, a lot of magic happens in the background of PyMC3 to choose the best MCMC algorithm and to find the best starting point for the MCMC chains. And it offers the possibility to use the previous posterior distribution as the prior distribution of the current run, so that insights from previous inferences carry over to new data. However, I would in any case award Soar the greater ability to intelligence.

In summary, both frameworks analyzed in this thesis show a great potential for realizing cognitive intelligent user models, but PyMC3 has a clear lead with regard to practicality and the stated research goals.

Chrysafiadi and Virvou (2013, p. 4716) state that "in order to construct a student model, it has to be considered what information and data about a student should be gathered, how it will update in order to keep it up-to-date, and how it will be used in order to provide adaptation." Hierarchical Bayesian models provided a practical and useful answer to all three points and are a real promising approach for the realization of cognitive intelligent user models.

7.2 Open Questions and Future Work

At the end of this thesis I would like to discuss the starting points for future work that have emerged in the course of this work.

The most urgent issue is the incomplete model evaluation. While the predictive accuracy of the model has been extensively validated, there was no validation of latent model variables with respect to their correspondence to the cognitive state of the learner. The only way to evaluate the model's correspondence to the learner's cognitive state is by means of a user study. This was originally intended to be part of the thesis, but could not be put into practice because dealing with Soar took much more time than anticipated. The idea was already present in the first draft of the CogIUM (Figure 4.14 on page 91). The idea is that total cognitive load cl_{pc} is the cause for an observable variable q_{pc} which represents scores for total cognitive load obtained by subjective questionnaires as was discussed in section 3.3 Measuring Cognitive Load (page 66). This can be extended to all components of the CLT, because there are questionnaires that can reliably measure ICL, ECL, and GCL (Klepsch et al., 2017; Leppink et al., 2013). In this case, a user study would be conducted with N participants playing Lost Earth, probably different missions with varying difficulty, and the questionnaires administered at the end of each mission. The acquired observational data would contain the observed variables from Lost Earth as well as the subjective measures of cognitive load. The model would then be fitted to the data and a posterior predictive check would show if the model was able to learn the structure in the observed data and whether the assumed values for the latent variables that represent the components of the CLT are comparable to the learner's subjective impression of the experienced cognitive load. However, this procedure requires that these additional observations are always present, which is normally not the case as measuring cognitive load is not part of the game play of Lost Earth.

An alternative is a user study conducted in the same manner as described above, but in which the subjective measures of cognitive load are compared with what the model assumes about the learner's current cognitive load. This means that the subjective measures are not presented to the model but used as an external source for validation. For example if a learner plays two different missions and rates the experienced cognitive load significantly different, the model should show the same tendency in the personal variables for this subject when fitted to the subject's observed data, but without the subjective measures. Likewise, if several subjects play the same two different missions and discordantly rate one mission as more difficult than the other, the model should show the same tendency in the conceptual variables for the two concepts when fitted to the data. It is not important that there is a one-to-one correspondence between the values of the model and the subjective measures of the questionnaire as the model's latent variables use a different scale and are designed to work with the rest of the model variables and not to represent the scores of a questionnaire. However, the tendency must be the same if the model is expected to accurately represent the learner's current cognitive load. Chang et al. (2017) conducted a study to examine the differences on flow experiences and different kinds of cognitive load between participants engaged in game-based learning and non-game-based learning. They used a modified version of a scale for measuring flow experience and an own cognitive load scale based on existing scales in literature. Participants first took a prior knowledge test on the domain of the learning objective. Afterwards, they completed a guidance orientation phase in which the learning objective was explained. The instructional experiment with the questionnaires administered at the end followed. Study design and the provided questionnaires might be useful starting points for future work that intends to conduct the above outlined user studies. Irrespective of the method chosen, however, the evaluation of the model for real user data must be carried out at a later stage.

A second unanswered research question is how cognitive architectures can be used to realize cognitive intelligent user models. I have presented first starting points for Soar in section 4.3.1 With cognitive architectures (page 81), but future work should not be limited in the choice of a suitable cognitive architecture. In fact, there are so many different cognitive architectures that it is very well possible to find another already available architecture that is better suited for inferring the learner's current cognitive state. The overview of the last 40 years of research on cognitive architectures by Kotseruba and Tsotsos (2018) can be a valuable starting point for finding better suited architectures. Such a candidate might be the cognitive architecture CLARION (Sun et al., 2005). CLARION is an integrative architecture with a number of distinct subsystems and a unique dual representational structure in each subsystem—implicit versus explicit representations. The most interesting part about CLARION are the motivational subsystem and the meta-cognitive subsystem. The motivational subsystem provides underlying motivation for perception, action, and cognition, in terms of providing impetus and feedback. The meta-cognitive subsystem monitors, controls, and regulates cognitive processes with the aim of improving cognitive performance. Sun (2007, p. 16) states that "CLARION is grounded in psychological research, is reasonably compact, and matches a range of empirical data." Since Sun is the architecture's inventor, the praise should be taken with caution. Nevertheless, the specifications of CLARION give the impression that this architecture is much more suited for realizing CogIUMs than Soar was. Hélie and Sun

(2014, p. 54) confirm that CLARION “can learn without much prior knowledge, such as initial productions necessary in ACT-R or Soar.” However, they also note that “CLARION has not yet been applied to some of the most complex tasks where ACT-R and Soar have been successful, so more research is required to test whether efficient strategies would automatically emerge from Clarion, and how they would math human results.”

As a final thought, it should be mentioned that there is no reason to limit the choice of appropriate methods to one tool only. Pavlik Jr. et al. (2013) mention in their review that many mature ITS use a blend of student models. They differentiate between three levels at which pedagogy occurs: the inner loop, the outer loop, and the curriculum loop. They give an example of a model that uses a version of BKT student model for the outer loop to solve problems, but within problems the student model is more like a constraint-based model with inner loop pedagogical responses to student actions disconnected from the outer loop BKT student model. In another example a constraint-based student model is used for the the inner loop to tutor problem solving, a knowledge space domain model for the outer loop to select problems, and a classification of students into basic skills unit for the curriculum branching loop. Thomson and Lebiere (2013) demonstrate how a functional cognitive architecture can constrain Bayesian inference by tying neurally-consistent mechanisms into Bayesian-compatible sub-symbolic activation. Llargues Asensio et al. (2014) combine combine the two cognitive architectures CERA-CRANIUM and Soar to CCBotSOAR, an advanced controller for believable agents. Soar is used in CERA-CRANIUM to give the system learning and memory based decision-making mechanisms based on the bot’s experiences. So there are many possibilities to combine the strengths of the two approaches presented in this thesis and mitigate some of their weaknesses.

Bibliography

Activity Streams 2.0 (2017). *W3C Recommendation 23 May 2017*. W3C.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* Oxford University Press.

Baker, R. S. (2016). Stupid Tutoring Systems, Intelligent Humans. In: *International Journal of Artificial Intelligence in Education* 26 (2), pp. 600–614. DOI: 10.1007/s40593-016-0105-0.

Bakhouyi, Abdellah, Dehbi, Rachid, Lti, Mohamed Talea, and Hajoui, Omar (2017). "Evolution of standardization and interoperability on E-learning systems: An overview". In: *2017 16th International Conference on Information Technology Based Higher Education and Training (ITHET)*. IEEE. DOI: 10.1109/ITHET.2017.8067789.

Bell, B. (2015). "One-Size-Fits-Some: ITS Genres and What They (Should) Tell Us About Authoring Tools. Authoring Tools and Expert Modeling Techniques". In: *Design Recommendations for Intelligent Tutoring Systems. Authoring Tools and Expert Modeling Techniques*. Ed. by Sottilare, Robert A., Graesser, Arthur C., Hu, Xiangen, and Brawner, Keith. Vol. 3. U.S. Army Research Laboratory. Chap. 3.

Berdun, Franco D. and Armentano, Marcelo G. (2018). Modeling Users Collaborative Behavior with a Serious Game. In: *IEEE Transactions on Games*, pp. 1–1. DOI: 10.1109/TG.2018.2794419.

Betancourt, M. (2017). A Conceptual Introduction to Hamiltonian Monte Carlo. In: *arXiv e-prints*, arXiv:1701.02434.

Blanco, A. del, Serrano, A., Freire, M., Martinez-Ortiz, I., and Fernandez-Manjon, B. (2013). "E-Learning standards and learning analytics. Can data collection be improved by using standard data models?" In: *2013 IEEE Global Engineering Education Conference (EDUCON)*. IEEE. DOI: 10.1109/EduCon.2013.6530268.

Boyle, E. A. et al. (2016). An update to the systematic literature review of empirical evidence of the impacts and outcomes of computer games and serious games. In: *Computers & Education* 94, pp. 178–192. DOI: 10.1016/J.COMPEDU.2015.11.003.

Brooks, S. P. and Gelman, A. (1998). General Methods for Monitoring Convergence of Iterative Simulations. In: *JCGS* 7 (4), pp. 434–455. DOI: 10.1080/10618600.1998.10474787.

Brünken, R., Plass, J. L., and Leutner, D. (2003). Direct Measurement of Cognitive Load in Multimedia Learning. In: *Educational Psychologist* 38 (1), pp. 53–61. DOI: 10.1207/S15326985EP3801_7.

Chang, C.-C., Liang, C., Chou, P.-N., and Lin, G.-Y. (2017). Is game-based learning better in flow experience and various types of cognitive load than non-game-based learning? Perspective from multimedia and media richness. In: *Computers in Human Behavior* 71, pp. 218–227. DOI: 10.1016/j.chb.2017.01.031.

Charles, R. L. and Nixon, J. (2019). Measuring mental workload using physiological measures: A systematic review. In: *Applied Ergonomics* 74 (September 2016), pp. 221–232. DOI: 10.1016/j.apergo.2018.08.028.

Chater, N., Tenenbaum, J. B., and Yuille, A. (2006). Probabilistic models of cognition: Conceptual foundations. In: *Trends in Cognitive Sciences* 10 (7), pp. 287–291. DOI: 10.1016/j.tics.2006.05.007.

Chong, A. and Lam, K. P. (2017). "A Comparison of MCMC Algorithms for theBayesian Calibration of Building Energy Models". In: *Proceedings of the 15th IBPSA Conference*, pp. 1319–1328. DOI: 10.26868/25222708.2017.336.

Chown, E. (2004). *Cognitive Modeling*. Ed. by Tucker, A. DOI: 10.1016/S0898-1221(03)90060-5.

Chrysafiadi, K. and Virvou, M. (2013). Student modeling approaches: A literature review for the last decade. In: *Expert Syst. Appl.* 40 (11), pp. 4715–4729. DOI: 10.1016/j.eswa.2013.02.007.

Conati, C. and Maclaren, H. (2009). Empirically building and evaluating a probabilistic model of user affect. In: *User Modeling and User-Adapted Interaction* 19 (3), pp. 267–303. DOI: 10.1007/s11257-009-9062-8.

De Schutter, B. (2011). Never Too Old to Play: The Appeal of Digital Games to an Older Audience. In: *Games and Culture* 6 (2), pp. 155–170. DOI: 10.1177/1555412010364978.

DeLeeuw, K. E. and Mayer, R. E. (2008). A comparison of three measures of cognitive load: Evidence for separable measures of intrinsic, extraneous, and germane load. In: *Journal of Educational Psychology* 100 (1), pp. 223–234. DOI: 10.1037/0022-0663.100.1.223.

Deonovic, Benjamin E. and Smith, Brian J. (2017). Convergence diagnostics for MCMC draws of a categorical variable. In: *arXiv e-prints*, arXiv:1706.04919.

Dermeval, D., Paiva, R., Bittencourt, I. I., Vassileva, J., and Borges, D. (2017). Authoring Tools for Designing Intelligent Tutoring Systems: a Systematic Review of the Literature. In: *International Journal of Artificial Intelligence in Education* 28 (3), pp. 336–384. DOI: 10.1007/s40593-017-0157-9.

Farrell, S. and Lewandowsky, S. (2018). *Computational Modeling of Cognition and Behavior*. Cambridge University Press. DOI: 10.1017/cbo9781316272503.

Feldman, J., Monteserin, A., and Amandi, A. (2014). Detecting students perception style by using games. In: *Computers & Education* 71, pp. 14–22. DOI: 10.1016/j.compedu.2013.09.007.

Friedenberg, J. and Silverman, G. (2006). *Cognitive Science: An Introduction To The Study Of Mind (Pb)*. Sage Publications Inc.

Gelman, A. (2006). Prior distributions for variance parameters in hierarchical models (comment on article by Browne and Draper). In: *Bayesian Analysis* 1 (3), pp. 515–534. DOI: 10.1214/06-BA117A.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2014). *Bayesian Data Analysis*. third. Chapman and Hall/CRC.

Griffiths, T. L., Chater, N., Kemp, C., Perfors, A., and Tenenbaum, J. B. (2010). Probabilistic models of cognition: exploring representations and inductive biases. In: *Trends in Cognitive Sciences* 14 (8), pp. 357–364. DOI: 10.1016/j.tics.2010.05.004.

Gudivada, V. N. (2016). "Cognitive Computing : Concepts, Architectures, Systems, and Applications". In: *Cognitive Computing: Theory and Applications*. Ed. by Gudivada, V., Raghavan, V., Govindaraju, V., and Rao, C. R. Vol. 35, pp. 3–38.

Hart, S. G. and Staveland, L. E. (1988). *Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research*. Elsevier, pp. 139–183. DOI: 10.1016/S0166-4115(08)62386-9.

Hélie, S. and Sun, R. (2014). Autonomous learning in psychologically-oriented cognitive architectures: A survey. In: *New Ideas in Psychology* 34, pp. 37–55. DOI: 10.1016/j.newideapsych.2014.03.002.

Hoffman, M. D. and Gelman, A. (2011). The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. In:

Kickmeier-Rust, M. D. and Albert, D. (2012). Educationally adaptive: Balancing serious games. In: *International Journal of Computer Science in Sport* 11 (1), pp. 15–28.

Kickmeier-Rust, M. D., Mattheiss, E., Steiner, C., and Albert, D. (2011). A Psycho-Pedagogical Framework for Multi-Adaptive Educational Games. In: *International Journal of Game-Based Learning* 1 (1), pp. 45–58. DOI: 10.4018/ijgbl.2011010104.

Klepsch, M., Schmitz, F., and Seufert, T. (2017). Development and Validation of Two Instruments Measuring Intrinsic, Extraneous, and Germane Cognitive Load. In: *Front. Psychol.* 8. DOI: 10.3389/fpsyg.2017.01997.

Kotseruba, I. and Tsotsos, J. K. (2018). 40 years of cognitive architectures: core cognitive abilities and practical applications. In: *Artificial Intelligence Review*. DOI: 10.1007/s10462-018-9646-y.

Kruschke, J. K. (2010a). Bayesian data analysis. In: *Wiley Interdiscip. Rev. Cognit. Sci.* 1 (5), pp. 658–676. DOI: 10.1002/wcs.72.

Kruschke, J. K. (2010b). What to believe: Bayesian methods for data analysis. In: *Trends in Cognitive Sciences* 14 (7), pp. 293–300. DOI: 10.1016/j.tics.2010.05.001.

Kruschke, J. K. (2015). *Doing Bayesian Data Analysis*. Elsevier LTD, Oxford.

Kruschke, J. K. and Liddell, T. M. (2018a). Bayesian data analysis for newcomers. In: *Psychonomic Bulletin & Review* 25 (1), pp. 155–177. DOI: 10.3758/s13423-017-1272-1.

Kruschke, J. K. and Liddell, T. M. (2018b). The Bayesian New Statistics: Hypothesis testing, estimation, meta-analysis, and power analysis from a Bayesian perspective. In: *Psychonomic Bulletin & Review* 25 (1), pp. 178–206. DOI: 10.3758/s13423-016-1221-4.

Kulik, J. A. and Fletcher, J. D. (2016). Effectiveness of Intelligent Tutoring Systems. In: *Review of Educational Research* 86 (1), pp. 42–78. DOI: 10.3102/0034654315581420.

Kurup, L. D., Joshi, A., and Shekhokar, N. (2016). "A review on student modeling approaches in ITS". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 2513–2517.

Laird, J. E. (2012). The Soar Cognitive Architecture. In: *AISB Quarterly* 134 (1), pp. 1–4.

Laird, J. E., Congdon, C. B.s, Assanie, M., Derbinsky, N., and Xu, J. (2017). *The Soar User's Manual. Version 9.6.0*. Draft. Division of Computer Science and EngineeringUniversity of Michigan.

Laird, J. E., Gluck, K., et al. (2017). Interactive Task Learning. In: *IEEE Intelligent Systems* 32 (4), pp. 6–21. DOI: 10.1109/mis.2017.3121552.

Laird, J. E., Lebiere, C., and Rosenbloom, P. S. (2017). A Standard Model of the Mind: Toward a Common Computational Framework across Artificial Intelligence, Cognitive Science, Neuroscience, and Robotics. In: *AI Magazine* 38 (4), p. 13. DOI: 10.1609/aimag.v38i4.2744.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. In: *Artif. Intell.* 33 (1), pp. 1–64. DOI: 10.1016/0004-3702(87)90050-6.

Lambert, B. (2018). *A Student's Guide to Bayesian Statistics*. SAGE Publications Ltd.

Langley, P. (2012). The Cognitive Systems Paradigm. In: *Advances in Cognitive Systems* 1, pp. 3–13.

Lee, M. D. and Wagenmakers, E.-J. (2014). *Bayesian Cognitive Modeling*. Cambridge University Pr.

Lehman, J. F., Laird, J. E., and Rosenbloom, P. S. (1998). "A Gentle Introduction to Soar: An Architecture for Human Cognition. Methods, Models, and Conceptual Issues". In: *An Invitation to Cognitive Science*. Ed. by Scarborough, Don, Sternberg, Saul, and Osherson, Daniel N. Second. Vol. 4. Chap. 6.

Lehman, J. F., Laird, J. E., and Rosenbloom, P. S. (2006). *A Gentle Introduction to Soar, An Architecture for Human Cognition: 2006 Update*.

Leppink, J., Paas, F., Van der Vleuten, C. P. M., Van Gog, T., and Van Merriënboer, J. J. G. (2013). Development of an instrument for measuring different types of cognitive load. In: *Behav. Res. Methods* 45 (4), pp. 1058–1072. DOI: 10.3758/s13428-013-0334-1.

Lieto, A., Bhatt, M., Oltramari, A., and Vernon, D. (2018). The role of cognitive architectures in general artificial intelligence. In: *Cognitive Systems Research* 48, pp. 1–3. DOI: 10.1016/j.cogsys.2017.08.003.

Lieto, A. and Radicioni, D. P. (2016). From human to artificial cognition and back: New perspectives on cognitively inspired AI systems. In: *Cognit. Syst. Res.* 39, pp. 1–3. DOI: 10.1016/j.cogsys.2016.02.002.

Lin, T., Kinshuk, D., Patel, A., and Hong, H. (2003). "Cognitive Trait Model for Persistent Student Modelling". In: *Proceedings of EdMedia + Innovate Learning 2003*. Ed. by Lassner, David and McNaught, Carmel. Honolulu, Hawaii, USA: Association for the Advancement of Computing in Education (AACE), pp. 2144–2147.

Llargues Asensio, J. M., Peralta, J., Arrabales, R., Bedia, M. G., Cortez, P., and Peña, A. L. (2014). Artificial Intelligence approaches for the generation and assessment of believable human-like behaviour in virtual characters. In: *Expert Syst. Appl.* 41 (16), pp. 7281–7290. DOI: 10.1016/j.eswa.2014.05.004.

Marcot, B. G. and Penman, T. D. (2018). Advances in Bayesian network modelling: Integration of modelling technologies. In: *Environmental Modelling & Software*. DOI: 10.1016/j.envsoft.2018.09.016.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. New York, NY, USA: W. H. Freeman.

McClelland, J. L. (2009). The Place of Modeling in Cognitive Science. In: *Topics in Cognitive Science* 1 (1), pp. 11–38. DOI: 10.1111/j.1756-8765.2008.01003.x.

Meent, J.-W. van de, Paige, B., Yang, H., and Wood, F. (2018). An Introduction to Probabilistic Programming. In: *arXiv e-prints*, arXiv:1809.10756v1.

Merriënboer, J. J. G. van and Sweller, J. (2005). Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions. In: *Educational Psychology Review* 17 (2), pp. 147–177. DOI: 10.1007/s10648-005-3951-0.

Millán, E., Loboda, T., and Pérez-de-la-Cruz, J. L. (2010). Bayesian networks for student model engineering. In: *Computers & Education* 55 (4), pp. 1663–1683. DOI: 10.1016/j.compedu.2010.07.010.

Miller, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. In: *Psychological Review* 63 (2), pp. 81–97. DOI: 10.1037/h0043158.

Monnahan, C. C. and Kristensen, K. (2018). No-U-turn sampling for fast Bayesian inference in ADMB and TMB: Introducing the adnuts and tmbstan R packages. In: *PLOS ONE* 13 (5). Ed. by Deng, Yong, e0197954. DOI: 10.1371/journal.pone.0197954.

Murphy, K. P. (2001). *An Introduction to Graphical Models*. Tech. rep.

Nakamura, J. and Csikszentmihalyi, M. (2009). "Flow Theory and Research". In: *The Oxford Handbook of Positive Psychology*. Ed. by Snyder, C. R. and Lopez, S. J. 2nd ed. Oxford University Press. Chap. 18. DOI: 10.1093/oxfordhb/9780195187243.013.0018.

Neal, R. M. (1998). Annealed Importance Sampling. In:

Newell, A. (1994). *Unified Theories of Cognition*. Harvard University Press.

Newell, A. and Simon, H. A. (1976). Computer Science As Empirical Inquiry: Symbols and Search. In: *Commun. ACM* 19 (3), pp. 113–126. DOI: 10.1145/360018.360022.

Norets, Andriy and Pelenis, Justinas (2018). Adaptive Bayesian Estimation of Mixed Discrete-Continuous Distributions under Smoothness and Sparsity. In:

Oppermann, Reinhard, ed. (1994). *Adaptive User Support: Ergonomic Design of Manually and Automatically Adaptable Software*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc.

Paas, F. G. (1992). Training strategies for attaining transfer of problem-solving skill in statistics: A cognitive-load approach. In: *Journal of Educational Psychology* 84 (4), pp. 429–434. DOI: 10.1037/0022-0663.84.4.429.

Paez, M. (2013). "Hierarchical Dynamic Models". In: *The SAGE Handbook of Multilevel Modeling*. Ed. by Scott, M. A., Simonoff, J. S., and Marx, B. D. SAGE Publications Ltd, pp. 335–356. DOI: 10.4135/9781446247600.n19.

Papamitsiou, Z. and Economides, A. A. (2014). Learning Analytics and Educational Data Mining in Practice: A Systematic Literature Review of Empirical Evidence. In: *Journal of Educational Technology & Society* 17 (4), pp. 49–64.

Parasuraman, R., Sheridan, T. B., and Wickens, C. D. (2000). A model for types and levels of human interaction with automation. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 30 (3), pp. 286–297. DOI: 10.1109/3468.844354.

Pavlik Jr., P. I., Brawner, K., Olney, A., and Mitrovic, A. (2013). "A Review of Student Models Used in Intelligent Tutoring Systems". In: *Design Recommendations for Intelligent Tutoring Systems. Learner Modeling*. Ed. by Sottilare, Robert, Graesser, Arthurt, Hu, Xiangen, and Holden, Heather. Vol. 1. U.S. Army Research Laboratory. Chap. 5, pp. 39–68.

Pelánek, R. (2015). Metrics for Evaluation of Student Models. In: *Journal of Educational Data Mining* 7 (2), pp. 1–19.

Pellegrini, A. D. (2009a). *The Role of Play in Human Development*. OXFORD UNIV PR.

Pellegrini, A. D. (2009b). Research and Policy on Children's Play. In: *Child Development Perspectives* 3 (2), pp. 131–136. DOI: 10.1111/j.1750-8606.2009.00092.x.

Plass, J. L., Homer, B. D., and Kinzer, C. K. (2015). Foundations of Game-Based Learning. In: *Educational Psychologist* 50 (4), pp. 258–283. DOI: 10.1080/00461520.2015.1122533.

Qian, M. and Clark, K. R. (2016). Game-based Learning and 21st century skills: A review of recent research. In: *Computers in Human Behavior* 63, pp. 50–58. DOI: 10.1016/j.chb.2016.05.023.

Rabelo, T., Lama, M., Amorim, R. R., and Vidal, J. C. (2015). "SmartLAK: A big data architecture for supporting learning analytics services". In: *2015 IEEE Frontiers in Education Conference (FIE)*. IEEE. DOI: 10.1109/FIE.2015.7344147.

Rabelo, T., Lama, M., Vidal, J. C., and Amorim, R. (2017). "Comparative study of xAPI validation tools". In: *2017 IEEE Frontiers in Education Conference (FIE)*. IEEE. DOI: 10.1109/FIE.2017.8190729.

Reid, G. B. and Nygren, T. E. (1988). "The Subjective Workload Assessment Technique: A Scaling Procedure for Measuring Mental Workload". In: *Advances in Psychology*. Elsevier, pp. 185–218. DOI: https://doi.org/10.1016/S0166-4115(08)62387-0.

Romero, C. and Ventura, S. (2012). Data mining in education. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 3 (1), pp. 12–27. DOI: 10.1002/widm.1075.

Rubio, S., Díaz, E., Martín, J., and Puente, J. M. (2004). Evaluation of Subjective Mental Workload: A Comparison of SWAT, NASA-TLX, and Workload Profile Methods. In: *Applied Psychology* 53 (1), pp. 61–86. DOI: 10.1111/j.1464-0597.2004.00161.x.

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016). Probabilistic programming in Python using PyMC3. In: *PeerJ Computer Science* 2, e55. DOI: 10.7717/peerj-cs.55.

Serrano-Laguna, A., Martı´nez-Ortiz, I., Haag, J., Regan, D., Johnson, A., and Fernández-Manjón, B. (2017). Applying standards to systematize learning analytics in serious games. In: *Computer Standards & Interfaces* 50, pp. 116–123. DOI: 10.1016/j.csi.2016.09.014.

Sh (2007). "The Physical Symbol System Hypothesis: Status and Prospects. Essays Dedicated to the 50th Anniversary of Artificial Intelligence". In: *50 Years of Artificial Intelligence*. Ed. by Lungarella, M., Iida, F., Bongard, J., and Pfeifer, R. Springer Berlin Heidelberg, pp. 9–17. DOI: 10.1007/978-3-540-77296-5_2.

Shi, Z., Hu, H., and Shi, Z. (2007). "A bayesian computational cognitive model". In: *Proceedings of the 3rd International Conference on Neural-Symbolic Learning and Reasoning-Volume 230*. Citeseer, pp. 46–51.

Shiffrin, R., Lee, M., Kim, W., and Wagenmakers, E.-J. (2008). A Survey of Model Evaluation Approaches With a Tutorial on Hierarchical Bayesian Methods. In: *Cognitive Science: A Multidisciplinary Journal* 32 (8), pp. 1248–1284. DOI: 10.1080/03640210802414826.

Shute, V. J. and Zapata-Rivera, D. (2012). "Adaptive Educational Systems". In: *Adaptive Technologies for Training and Education*. Ed. by Durlach, P. J. and Lesgold, A. M. Cambridge University Press. Chap. 1, pp. 7–27. DOI: 10.1017/CBO9781139049580.004.

Simon, H. A. (1957). *Models of man: social and rational; mathematical essays on rational human behavior in society setting*. eng. 275. Wiley.

Sottilare, R. A. and Gilbert, S. (2011). Considerations for adaptive tutoring within serious games: authoring cognitive models and game interfaces. In: p. 10.

Squire, K. (2005). Changing the Game: What Happens When VideoGames Enter the Classroom? In: *Innovate: Journal of Online Education* 1 (5) (6).

Streicher, A. and Roller, W. (2017). "Interoperable Adaptivity and Learning Analytics for Serious Games in Image Interpretation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10474 LNCS, pp. 598–601. DOI: 10.1007/978-3-319-66610-5_71.

Streicher, A., Roller, W., and Biegemeier, C. (2017). "Application of Adaptive Game-based Learning in Image Interpretation". In: *11th European Conference on Game-Based Learning ECGBL 2017*. Graz: Academic Conferences and Publishing International Limited, pp. 975–978.

Streicher, A. and Smeddinck, J. D. (2016). "Personalized and Adaptive Serious Games". In: *Entertainment Computing and Serious Games*. Ed. by Dörner, R., Göbel, S., Kickmeier-Rust, M., Masuch, M., and Zweig, K. Cham: Springer International Publishing, pp. 332–377. DOI: 10.1007/978-3-319-46152-6_14.

Streicher, A., Szentes, D., and Gundermann, A. (2016). "Game-Based Training for Complex Multi-institutional Exercises of Joint Forces". In: *Adaptive and Adaptable Learning*. Ed. by Verbert, K., Sharples, M., and Klobučar, T. Springer International Publishing, pp. 497–502. DOI: 10.1007/978-3-319-45153-4_49.

Sun, R. (2004). Desiderata for cognitive architectures. In: *Philosophical Psychology* 17 (3), pp. 341–373. DOI: 10.1080/0951508042000286721.

Sun, R. (2007). The importance of cognitive architectures: an analysis based on CLARION. In: *J. Exp. Theor. Artif. Intell.* 19 (2), pp. 159–193. DOI: 10.1080/09528130701191560.

Sun, R., ed. (2008). *The Cambridge Handbook of Computational Psychology*. Cambridge Handbooks in Psychology. Cambridge University Press. DOI: 10.1017/CBO9780511816772.

Sun, R. (2009). Theoretical status of computational cognitive modeling. In: *Cognit. Syst. Res.* 10 (2), pp. 124–140. DOI: 10.1016/j.cogsys.2008.07.002.

Sun, R., Slusarz, P., and Terry, C. (2005). The Interaction of the Explicit and the Implicit in Skill Learning: A Dual-Process Approach. In: *Psychological Review* 112 (1), pp. 159–192. DOI: 10.1037/0033-295x.112.1.159.

Sweller, J. (1988). Cognitive Load During Problem Solving: Effects on Learning. In: *Cognitive Science* 12 (2), pp. 257–285. DOI: 10.1207/s15516709cog1202_4.

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. In: *Learning and Instruction* 4 (4), pp. 295–312. DOI: 10.1016/0959-4752(94)90003-5.

Sweller, J. (2010). Element Interactivity and Intrinsic, Extraneous, and Germane Cognitive Load. In: *Educational Psychology Review* 22 (2), pp. 123–138. DOI: 10.1007/s10648-010-9128-5.

Sweller, J., Merrienboer, J. J. G. van, and Paas, F. G. W. C. (1998). Cognitive Architecture and Instructional Design. In: *Educational Psychology Review* 10 (3), pp. 251–296. DOI: 10.1023/A:1022193728205.

Taatgen, N. and Anderson, J. R. (2010). The Past, Present, and Future of Cognitive Architectures. In: *Topics in Cognitive Science* 2 (4), pp. 693–704. DOI: 10.1111/j.1756-8765.2009.01063.x.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., and Goodman, N. D. (2011). How to Grow a Mind: Statistics, Structure, and Abstraction. In: *Science* 331 (6022), pp. 1279–1285. DOI: 10.1126/science.1192788.

Thagard, P. (2005). *Mind: Introduction to Cognitive Science*. A Bradford Book.

Thomson, R. and Lebiere, C. (2013). “Constraining Bayesian Inference with Cognitive Architectures: An Updated Associative Learning Mechanism in ACT-R”. In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 35.

Toh, S. C. (2005). Recent Advances in Cognitive Load Theory Research: Implications for Instructional Designers. In: *MOJIT* 2 (3), pp. 106–117.

Truong, H. M. (2016). Integrating learning styles and adaptive e-learning system: Current developments, problems and opportunities. In: *Computers in Human Behavior* 55, pp. 1185–1193. DOI: 10.1016/j.chb.2015.02.014.

Tsang, P. S. and Velazquez, V. L. (1996). Diagnosticity and multidimensional subjective workload ratings. In: *Ergonomics* 39 (3), pp. 358–381. DOI: 10.1080/00140139608964470.

Turner, B. M., Forstmann, B. U., Love, B. C., Palmeri, T. J., and Van Maanen, L. (2017). Approaches to analysis in model-based cognitive neuroscience. In: *J. Math. Psychol.* 76, pp. 65–79. DOI: 10.1016/j.jmp.2016.01.001.

Vehtari, A., Gelman, A., and Gabry, J. (2016). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. In: *Statistics and Computing* 27 (5), pp. 1413–1432. DOI: 10.1007/s11222-016-9696-4.

Vernon, D., Hofsten, C. von, and Fadiga, L. (2016). Desiderata for developmental cognitive architectures. In: *Biologically Inspired Cognitive Architectures* 18, pp. 116–127. DOI: 10.1016/j.bica.2016.10.004.

Vernon, D., Metta, G., and Sandini, G. (2007). A Survey of Artificial Cognitive Systems: Implications for the Autonomous Development of Mental Capabilities in Computational Agents. In: *IEEE Trans. Evol. Comput.* 11 (2), pp. 151–180. DOI: 10.1109/TEVC.2006.890274.

Vidal, J. C., Rabelo, T., and Lama, M. (2015). “Semantic Description of the Experience API Specification”. In: *2015 IEEE 15th International Conference on Advanced Learning Technologies*. IEEE. DOI: 10.1109/ICALT.2015.128.

Vygotsky, L. S. (1978). *Mind in society*. Harvard University Press.

Wagenmakers, E.-J. and Farrell, S. (2004). AIC model selection using Akaike weights. In: *Psychonomic Bulletin & Review* 11 (1), pp. 192–196. DOI: 10.3758/BF03206482.

Winn, J. and Heeter, C. (2009). Gaming, Gender, and Time: Who Makes Time to Play? In: *Sex Roles* 61 (1-2), pp. 1–13. DOI: 10.1007/s11199-009-9595-7.

Woolf, B. P. (2009). *Building Intelligent Interactive Tutors Student-centered strategies for revolutionizing e-learning*. San Francisco, CA, USA: Morgan Kaufmann, p. 480. DOI: 10.1007/BF02680460.

A Supplementary Material for Chapter Concept

Table A.1.: List of all implemented models in the CogIUM package.

Model	Variables						
	Total	Observable	Deterministic	Personal & Conceptual	Personal	Conceptual	Global
$\mathscr{M}_1$	$3N_p + 7N_c$	k_{pc}, s_{pc}	$\mathrm{cl}_{pc}, \mathrm{gcl}_{pc}, \delta_{pc}$	$\mathrm{cl}_{pc}, \mathrm{gcl}_{pc}, \delta_{pc}, p_{obs}$	m_p, ψ_p	$\mu_{icl}, \mu_{ecl}, \sigma_{icl}, \sigma_{ecl}, \mathrm{icl}_c, \mathrm{ecl}_c$	-
$\mathscr{M}_2$	$3N_p + 7N_c + 1$	k_{pc}, s_{pc}	$\mathrm{cl}_{pc}, \mathrm{gcl}_{pc}, \delta_{pc}$	$\mathrm{cl}_{pc}, \mathrm{gcl}_{pc}, \delta_{pc}, p_{obs}$	m_p, ψ_p	$\mu_{icl}, \mu_{ecl}, \sigma_{icl}, \sigma_{ecl}, \mathrm{icl}_c, \mathrm{ecl}_c$	σ_s
$\mathscr{M}_3$	$4N_p + 7N_c + 3$	k_{pc}, s_{pc}	$\mathrm{cl}_{pc}, \mathrm{gcl}_{pc}, \delta_{pc}$	$\mathrm{cl}_{pc}, \mathrm{gcl}_{pc}, \delta_{pc}, p_{obs}$	m_p, ψ_p, ν_s	$\mu_{icl}, \mu_{ecl}, \sigma_{icl}, \sigma_{ecl}, \mathrm{icl}_c, \mathrm{ecl}_c$	$\sigma_s,\mu_\nu, \sigma_\nu$
$\mathscr{M}_4$	$6N_p + 8N_c + 6$	k_{pc}, s_{pc}, t_{pc}	$\mathrm{cl}_{pc}, \mathrm{gcl}_{pc}, \delta_{pc}$	$\mathrm{cl}_{pc}, \mathrm{gcl}_{pc}, \delta_{pc}, p_{obs}$	$m_p, \psi_p, \nu_s, \alpha_t, \beta_t$	$\mu_{icl}, \mu_{ecl}, \sigma_{icl}, \sigma_{ecl}, \mathrm{icl}_c, \mathrm{ecl}_c, t_{min}$	$\sigma_s, \mu_\nu, \sigma_\nu, \sigma_t, \sigma_\alpha, \sigma_\beta$

B Supplementary Material for Chapter Implementation

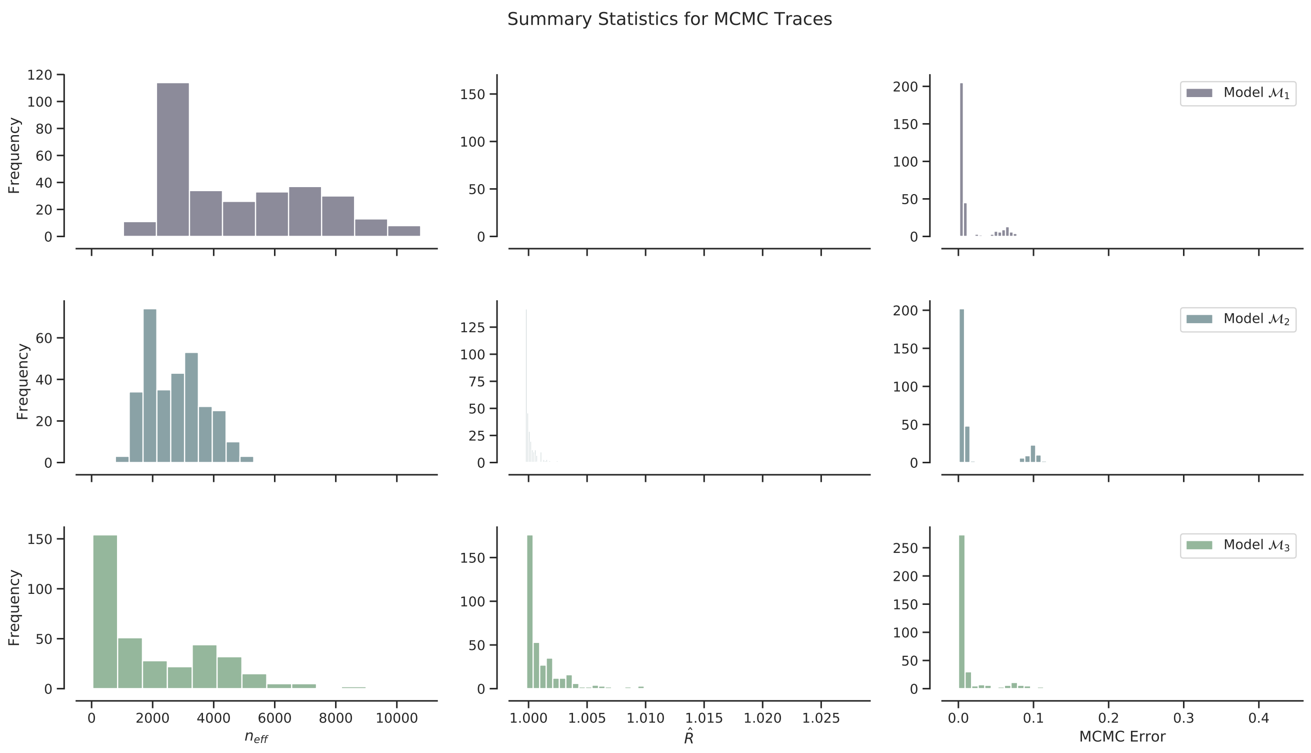


Figure B.1.: Exemplary output of the `plot_trace_summary()` function. Here, three models, or trace objects, are compared with each other. For each trace, the plot shows three histograms: the number of effective sample size, the potential scale reduction $\hat{R}$ and the MCMC standard error. The scale of the x-axes is shared per column to help in comparing the results. Different models have different colors.

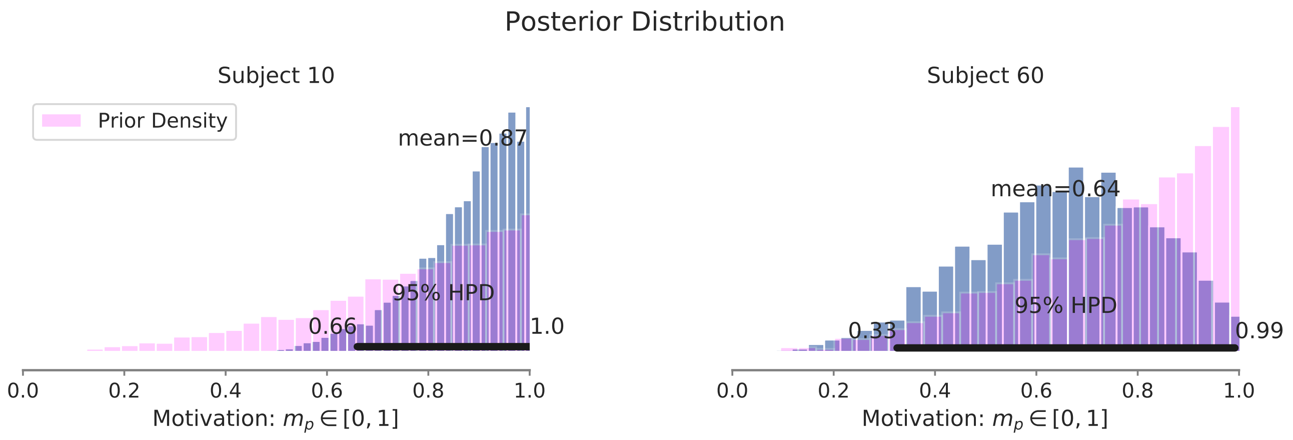


Figure B.2.: Exemplary output of the `plot_posterior()` function. Here, one trace object, the subject ids 10 and 60 and the variable name ‘m_p’ were passed as arguments. The posterior distribution is shown as histogram with the mean value as point estimate. The prior distribution for this parameter is also shown as histogram to allow for comparison between the prior and posterior distribution for this model parameter. The 95 % HDI is shown with both end points at the bottom of each histogram and refers to the posterior distribution. If multiple traces were given, the posterior of each model would have been plotted in their own row.

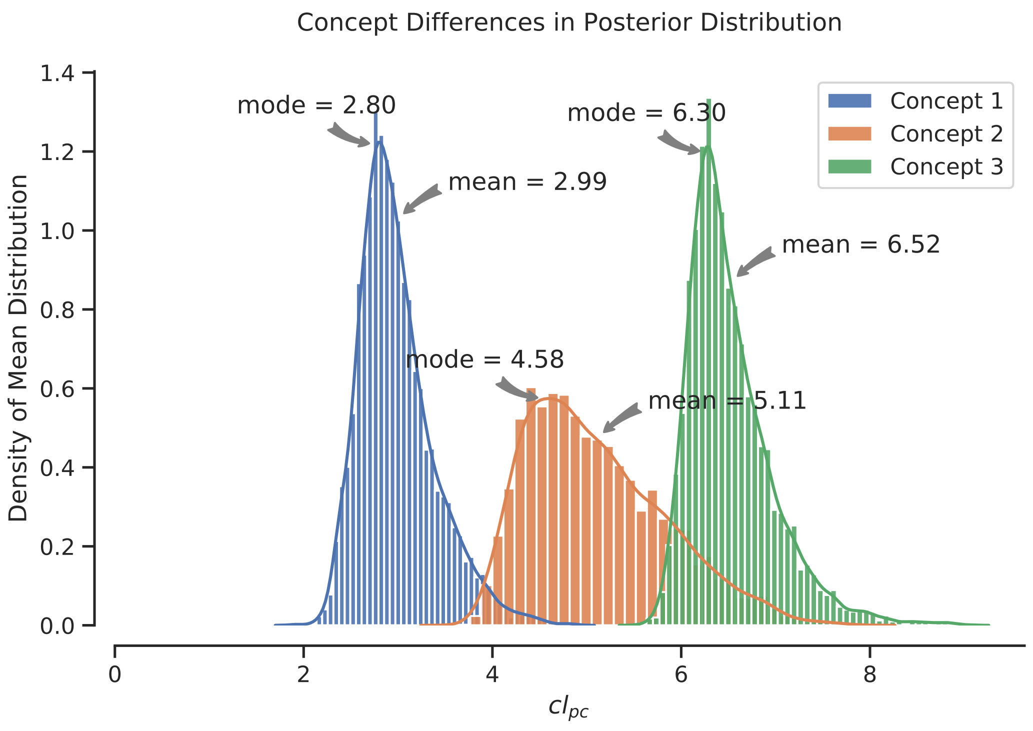
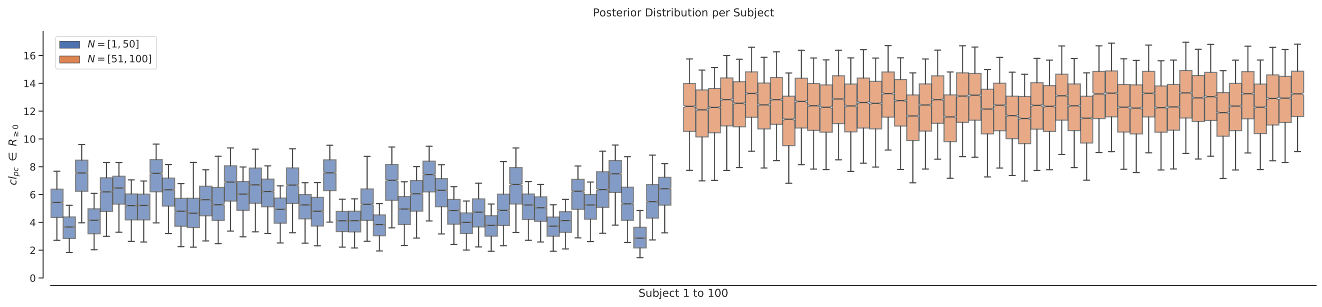
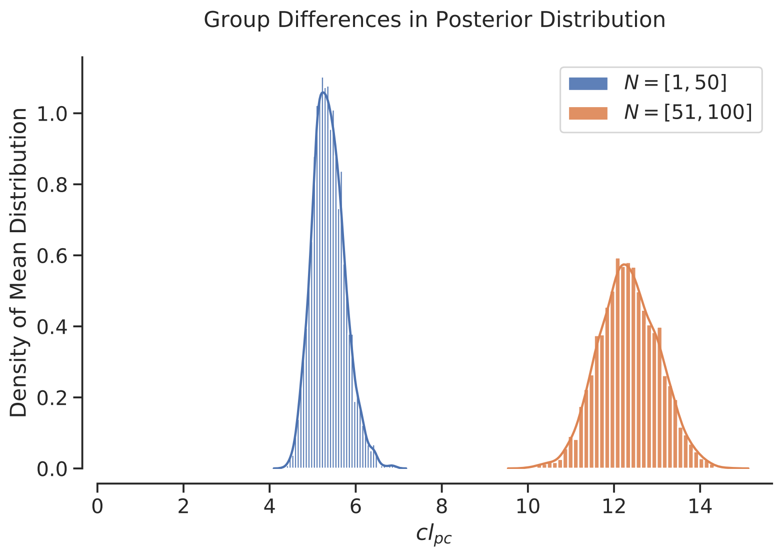


Figure B.3.: Exemplary output of the `show_concept_differences()` function. Here, one trace object and the variable name 'cl_pc' for a model fit to observable data with three concepts were passed as arguments. The distribution of the means over all subjects per sample of the trace object is shown as kernel density estimation with the mean and mode values as point estimates. Different concepts are colored in blue, orange and green.

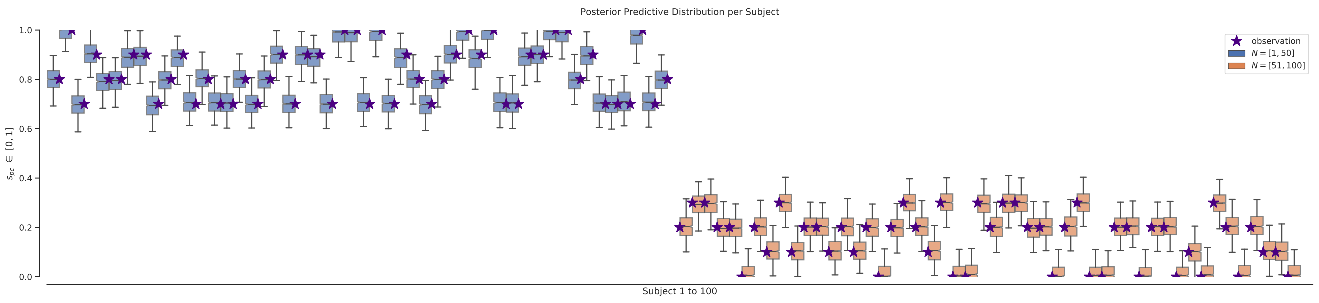


(a) Posterior distribution per subject shown as box plots. Different groups are colored in blue and orange. The whiskers extend the box by 1.5 of the proportion of the IQR.

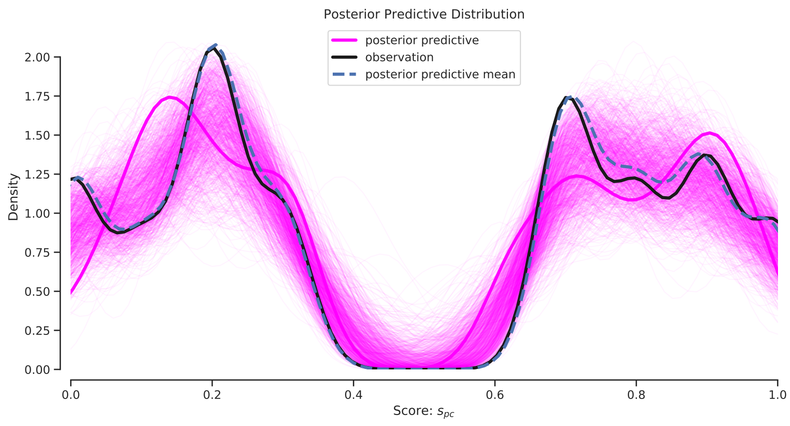


(b) Distribution of the group means of the posterior distribution over all subjects in a group. Both mean and mode are shown as point estimates. Different groups are colored in blue and orange. The density is estimated by a kernel density estimation.

Figure B.4.: Exemplary output of the `show_group_differences()` function. Here, one trace object, the variable name 'cl_pc', and the cutpoints $[50]$ where given, because the observable data consists of 100 subjects, split equally into two groups. The function produces two outputs: a) the posterior distribution for each subject, shown as box plots, and b) the distribution of group means by calculating the mean value for all subjects within group for one trace sample.



(a) Posterior predictive distribution per subject shown as box plots, extended with the true value ★ for the observable variable. Different groups are colored in blue and orange. The whiskers extend the box by 1.5 of the proportion of the IQR.



(b) The distribution of the observable data (solid black line) and the model's predictions (magenta lines) from samples of the posterior predictive distribution shown as kernel density estimations. Each of the prediction curves shows the distribution of the data for a sample drawn from the model's posterior predictive distribution. All prediction curves are aggregated to a mean posterior prediction (blue dashed line).

Figure B.5.: Exemplary output of the `plot_posterior_predictive()` function. Here, the observable data for 100 subjects, divided into two groups, a posterior predictive trace object with 1000 samples, the variable name 's_pc', and the cutpoints [50] were given. The function produces two outputs: a) the posterior predictive distribution for each subject, shown as box plots and with the true observable values, and b) the distribution of the observable variable compared to the predictions of the model, drawn as samples from the posterior predictive distribution.

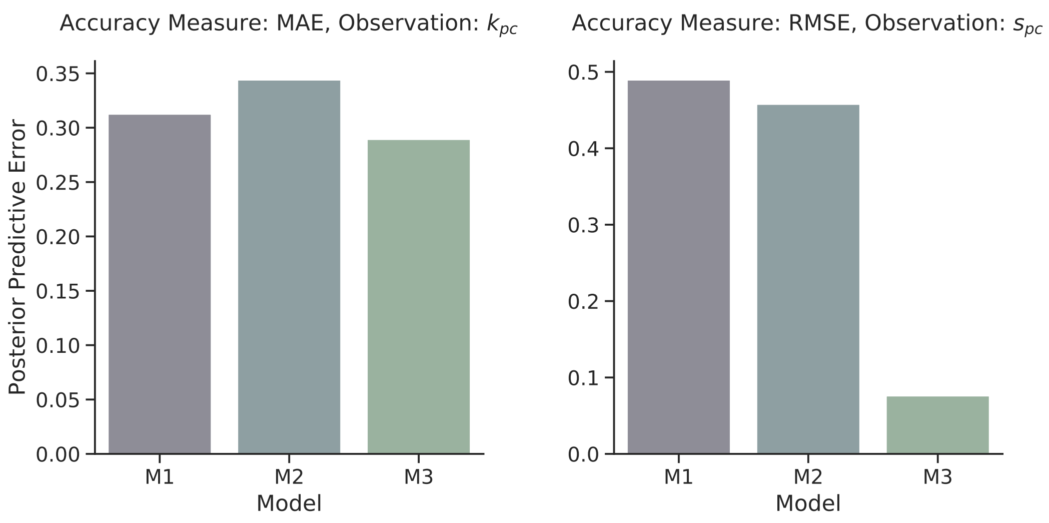


Figure B.6.: Exemplary output of the `plot_predictive_error()` function. Here, a list of three models, a list of three posterior predictive trace objects, the observable data with 100 subjects, divided into two groups, a list of observable variable names [‘k_pc’, ‘s_pc’], and a list of measures [‘MAE’, ‘RMSE’] was given. For each observable variable the function calculates the prediction error as specified by the measure for each model based on the difference between the observations and the model’s predictions. Mean absolute error (MAE) is used for discrete variables and root mean squared error (RMSE) for continuous variables.

Listing B.1: Source code for CogiumImprovedHierarchical3Obs of the CogIUM package.

```python
class CogiumImprovedHierarchical3Obs(BaseModel):
   """
   Final statistical model. Use of gcl_pc and delta_pc to model k_pc, s_pc and t_pc.
   """

   def __init__(self, data: np.ndarray, wm_capacity: int, name: str = '', model: Model = None):
      self.data = data
      self.name = 'ImprovedHierarchical3Obs'

      Np, Nc, _ = data.shape
      wm = wm_capacity

      with pm.Model(name=name, model=model) as model:
         # global latent var
         sigma_s = pm.HalfCauchy('sigma_s', beta=1, testval=0.1)
         sigma_t = pm.HalfCauchy('sigma_t', beta=1, testval=0.1)
         sigma_nu = pm.HalfCauchy('sigma_nu', beta=1, testval=0.1)
         sigma_alpha = pm.HalfCauchy('sigma_alpha', beta=1)
         sigma_beta = pm.HalfCauchy('sigma_beta', beta=1)
         mu_nu = pm.Normal('mu_nu', mu=0.5, sd=0.01, testval=0.5)

         # personal variables
         m_p = pm.Beta('m_p', alpha=3, beta=1, testval=0.9, shape=(Np, 1))
         psi_p = pm.Beta('psi_p', alpha=2, beta=2, testval=0.5, shape=(Np, 1))
         nu_s_p = pm.Bound(pm.Normal, lower=0, upper=1)(
            'nu_s_p', mu=mu_nu, sd=sigma_nu, shape=(Np, 1)
         )
         alpha_t_p = pm.HalfNormal('alpha_t_p', sigma_alpha, shape=(Np, 1))
         beta_t_p = pm.HalfNormal('beta_t_p', sigma_beta, shape=(Np, 1))

         # conceptual var
         mu_icl = pm.Gamma('mu_icl', alpha=2, beta=0.5, shape=(Nc, 1), testval=2)
         mu_ecl = pm.Gamma('mu_ecl', alpha=2, beta=0.5, shape=(Nc, 1), testval=2)
         sigma_icl = pm.HalfCauchy('sigma_icl', 5, shape=(Nc, 1), testval=0.1)
         sigma_ecl = pm.HalfCauchy('sigma_ecl', 5, shape=(Nc, 1), testval=0.1)
         icl_c = pm.Bound(pm.Normal, lower=1)('icl_c', mu=mu_icl, sd=sigma_icl, shape=(Nc, 1))
         ecl_c = pm.Bound(pm.Normal)('ecl_c', mu=mu_ecl, sd=sigma_ecl, shape=(Nc, 1))
         t_min = pm.Gamma('t_min', alpha=2, beta=0.5, shape=(Nc, 1))

         # deterministic variables
         cl_pc = pm.Deterministic(
            'cl_pc',
            tensor.outer(1 - psi_p, icl_c) + tensor.outer(psi_p, ecl_c) / psi_p # Np x Nc matrix
         )
```

```

        gcl = (tensor.outer(m_p * wm, ecl_c) /
                ecl_c.T - tensor.outer(m_p, ecl_c) / m_p)
        gcl = tensor.where(gcl <= 0, 1e-4, gcl)
        gcl_pc = pm.Deterministic(
            'gcl_pc', gcl / wm
        )

        delta = (wm - cl_pc)
        delta_pc = pm.Deterministic(
            'delta_pc', delta / wm
        )

        # observed variables
        p_obs = pm.Beta(
            'p_obs', alpha=1 + gcl_pc, beta=1 - delta_pc, shape=(Np, Nc)
        )

        k_pc = pm.Bernoulli(
            'k_pc', p=p_obs, observed=self.data[:, :, 0].reshape(Np, Nc)
        )

        s_pc = pm.Normal(
            's_pc',
            mu=gcl_pc + nu_s_p * delta_pc,
            sd=sigma_s,
            observed=self.data[:, :, 1].reshape(Np, Nc)
        )

        t_pc = pm.Normal(
            't_pc',
            mu=t_min.T + alpha_t_p * (1 - gcl_pc) - beta_t_p * delta_pc,
            sd=sigma_t,
            observed=data[:, :, 2].reshape(Np, Nc)
        )

    self.model = model
```