Bias-aware Human Decision Modeling for Human-Robot Teams

by

Anusha Challa

(Under the Direction of Prashant Doshi)

Abstract

When humans and robots work together, a robot's trust in its partner is the chance that the next interaction goes well. This thesis connects logged human behavior, including bias toward or against AI assistance, to performance across task episodes and to changes in robot trust. Using Human-Robot Interaction (HRI) logs, we segment episodes and assign intent aware action labels. We build a human Markov decision process (MDP) by learning a behavior model, if needed a time model at each step, and a termination model, and compute an optimal human policy with value iteration. We estimate counterfactual success, total time, and on-time completion for that policy using a model-based simulation approach to policy evaluation, and we report mission outcome metric as a check. Case studies show how assistance bias shifts decisions and mission status and when guidance from the human MDP would increase overall simulation results.

Index words: Human–robot interaction, Off-policy evaluation, Weighted importance sampling, Human Markov Decision Process, Intent-aware action labeling, Decision trees, Value iteration

Bias-aware Human Decision Modeling for Human-Robot Teams

by

Anusha Challa

Bachelor of Technology, VIT University, India, 2018

A Thesis Submitted to the Graduate Faculty of the

University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Master of Science

Athens, Georgia

2025

Bias-aware Human Decision Modeling for Human-Robot Teams

by

Anusha Challa

Major Professor: Prashant Doshi

Committee: Kyle J. Johnsen

Neal Outland

Electronic Version Approved:

Ron Walcott

Dean of the Graduate School

The University of Georgia

December 2025

Dedication

To daddy, ma, swethu. Your patience and love made this possible. You mean the world to me.

Acknowledgments

I would like to express my sincere gratitude to everyone who supported me throughout my academic journey. I am particularly grateful to my advisor, Dr. Prashant Doshi, for his guidance and encouragement, which inspired this research. I also wish to thank my committee members, Dr. Kyle J. Johnsen and Dr. Neal Outland, for fostering my curiosity, believing in my ideas, and providing valuable guidance throughout this process.

I am grateful to Dr. Frederick W. Maier, Dr. Ramviyas Nattanmai Parasuraman, and Evette for their consistent encouragement and support, which made my time in the program deeply rewarding.

I am deeply grateful to my labmates, especially Aditya and Daniel, whose unwavering support and positive spirit lifted me through challenges. I also want to thank my wonderful friends for always being there and helping me find balance during tough times.

Above all, I thank my parents and sister. Their unwavering love and belief in me have been my foundation and source of strength.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial intelligence (AI) and robotics are advancing quickly, making the line between human operators and autonomous systems less clear. Working together with robots, known as human-robot interaction (HRI), has become a key way of operating. In areas like manufacturing, logistics, security, healthcare, and disaster response, teams of humans and robots now handle complex and high-pressure tasks. These missions often have changing goals, limited resources, and ongoing needs for reliability and flexibility. Trust is essential for successful HRI. Each team member, human or robot, forms beliefs about other's abilities, willingness, and reliability in reaching shared goals. Trust affects how responsibilities are shared, how advice and warnings are followed, and how the team adapts to new situations. It is not just a background factor; trust directly shapes both how the team works in the moment and how well it performs over time.

Trust in HRI is different from the more fixed trust seen in traditional automation or all-human teams. In HRI, every action, like accepting a robot's suggestion, overriding an automated choice, or letting a human decide, changes how team members view each other's reliability, skill, and flexibility. These changes come from clear records of success and failure, such as how often the robot gives good advice, as well as from less obvious signs like how quickly people respond, how clearly they communicate, and their personal experiences working together.

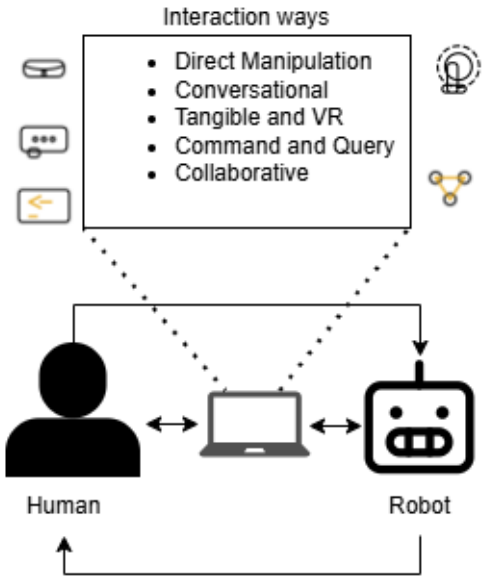

Figure 1.1: Human and robot interacting via multiple modalities and technologies in a team environment.

Researchers have responded to these challenges by developing many studies on how to model trust in HRI using computers. One important framework, created by Nare et al. (2025), treats robot trust as something that changes over time. In this model, the chance of having a good experience with a human teammate is tracked and updated using mathematical tools like probability distributions and certainty density functions, along with factors such as skill, reliability, and how well team members fit in. The framework also includes the idea of kinship, or long-term connection, to show how working together repeatedly and sharing history can help trust grow and adjust more quickly.

Even with these advances, most computational models do not give enough attention to an important human factor: existing, and sometimes hidden, biases about seeking help or trusting AI. In real teams, people bring not just their actions and performance, but also

their attitudes, past experiences, and personality traits that affect how they choose to work with intelligent systems.

Bias in seeking help can strongly affect how HRI missions go and what results they achieve. Some people quickly use new technology, follow AI suggestions, delegate tasks, and welcome automation. Others are more skeptical, hesitate to trust automated help, and may ignore or avoid robot support, even when it could clearly improve their performance. These biases

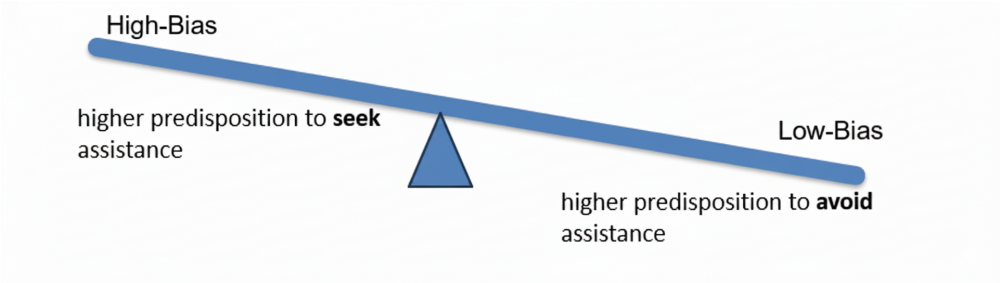

Figure 1.2: Assistance-seeking bias spectrum, contrasting predisposition to seek versus avoid help in HRI.

have real and important effects. People who trust automation too much may rely on it more than they should, giving up too much control. This can lead to complacency, loss of skills, or problems if the AI makes mistakes. On the other hand, those who do not trust automation may miss chances to work well with robots by ignoring their suggestions. This can make missions less effective, cause extra work, and lead to frustration.

Traditional HRI methods often use models that assume people act mainly based on results and the situation. But more and more studies show that other factors, like self-confidence, sense of status, past experience with automation, and personality also play a big role in how people seek help, delegate tasks, and adjust their trust in human-robot teams.

This thesis explicitly incorporates bias as a primary component within the computational framework of HRI trust. Large-scale, controlled team experiments are conducted in which bias is assessed prior to mission onset using validated psychological and attitudinal surveys. These measurements serve both descriptive and analytical purposes, enabling stratification

of participants into subgroups and facilitating detailed analysis of how bias correlates with and influences differences in action selection, task acceptance, and adaptation to robot recommendations.

Each experiment is carefully tracked. Logs record every user action, every robot suggestion, and the full sequence of acceptances, rejections, and overrides. This detailed data lets researchers rebuild each teamwork episode, replay it in computer models, and compare real and possible outcomes under different policies or levels of bias. The main modeling method uses Markov Decision Processes (MDPs). Each episode is shown as a series of state-action pairs, with extra data on timing, intent, and who took each action. This clear modeling helps study context, bias, and how trust changes during each mission. The models also allow for off-policy simulations, making it possible to test what-if scenarios and create teaming strategies that fit real human differences.

This multi-level modeling and hands-on research leads to several important findings:

- Trust and reliance are deeply cyclical. Each outcome, whether positive or negative, recalibrates not only perceptions of the robot's capability but also the human's future willingness to seek help.

- Bias awareness is important at every level. Policy choices by the robot, such as when to offer suggestions, how strongly to recommend actions, or when to wait for human initiative, cannot be uniform; both context and user predisposition are significant.

- Objective outcomes and subjective experience are related but not identical. Two teams may achieve similar performance metrics yet leave the human actor with very different impressions of the robot's value.

- Alternatively, teams may achieve different outcomes despite starting from the same objective position, purely due to bias.

- Visual and logical models clarify these cycles. By using diagrams to depict the flows of trust, reliance, bias, and feedback loops, and by mapping statistical patterns onto individual trajectories, the research brings transparency and clarity to what might otherwise be a black-box operation in human-robot teaming.

The ideas in this work are supported by thorough testing, including analyses of hundreds of HRI missions and many policy simulations, with and without bias awareness, conducted across different settings. The findings are not just for academics; they also give practical advice to those building the next generation of smart assistants, whether as robots in the field or as digital helpers in decision-support systems.

This research also adds to the ongoing debate about the importance of human factors like bias, emotion, and personality in HRI. The evidence here shows these factors are central and necessary for real-world teamwork, not just background noise to ignore. Future HRI systems should recognize, respect, and adapt to each team member as a unique partner, instead of using only one-size-fits-all rules.

In short, this chapter gives both practical and theoretical reasons to move past one-size-fits-all models of trust and teamwork in HRI. By adding clear bias measurement, detailed action labeling, and flexible MDP-based simulation, this work takes a big step toward more personalized, effective, and reliable human-robot teams. The next chapters go deeper into these points, starting with a review of the literature on trust and bias models, then covering the experiments and modeling methods, and ending with a full evaluation and discussion of what this means for future human-centered AI teamwork.

1.1 Contributions

This thesis studies human-robot teamwork across different types of tasks, where actions, results (like success and time), and task states can change during each episode. The main contributions are:

- **Pre-experiment bias modeling.** I introduce a compact, survey-based index that combines trust in AI and Human-AI teaming attitude (HAI 7-9) into a continuous propensity (0–1) and a simple binary flag.

- **Action labels from logs.** A practical set of labels and rules is created to turn raw messages and interface events into clear human actions. This method reduces confusion and separates suggestions from actions. It also offers an easy-to-read decision model. A step-by-step model is built using the current task and the previous action. This model predicts what will happen next and the chance of switching to a different task by taking a specific action. The model is made to be easy to explain and review.

- **Human MDP.** From the model, we compute sensible next steps that balance finishing each mission successfully while respecting time constraints.

- **Model-based simulation.** Using only the recorded data, we estimate how the success rate would change if a person followed the optimal policy, and we include a comparison of participants who completed all tasks to assess policy-based human stability by grouping on bias.

- **Task-level MDPs for bias-affected tasks.** Tasks that show a significant effect of prior propensity to trust AI (such as Computer Hack and Access Code) are used to construct smaller MDPs for each, with actions mirroring real human choices. The analysis reports where recommended decisions differ across trust propensity.

1.2 Thesis Structure

The remainder of the thesis is organized as follows:

- Chapter 2: Foundational Concepts. Definitions, notation, and background are needed for the rest of the document.
- Chapter 3: Related Work. Prior research on robot trust, assistance bias, decision guidance, and log-based evaluation, and how this thesis differs.
- Chapter 4: Datasets and Problem Formulation. The testbed and tasks, how we form episodes, the action labels and rules, the bias features, and data quality checks.
- Chapter 5: Human MDP and Optimal Offline Policy Evaluation. The next action prediction models, how we derive the next-step policy, how we estimate outcomes from logs, and the evaluation of the optimal policy with mission status outcomes against completed missions.
- Chapter 6: Results. The experimental setup and results for each task, cross-task comparisons, and diagnostics.
- Chapter 7: Conclusion. A summary of contributions, limitations, and directions for future work.

Chapter 2

Foundational Concepts

This chapter introduces the ideas and notation needed for the rest of the thesis. The goal is to explain the terms we use, not to prescribe design choices. We keep the focus on what each concept means and why it matters later. Trust is treated as a belief about future performance that guides appropriate reliance Lee and See (2004). Logged interaction can be described with a Markov decision process (MDP), and policy performance can be estimated from past data using tools for off-policy evaluation.

2.1 Trust and its components

Trust is a belief about what will happen under uncertainty. In automation, it is the belief that a system will help achieve goals even when the person is exposed to some risk Lee and See (2004). Three components are useful later:

- **Competency** (or skill): the chance of completing the task correctly on a given instance
- **Conformance**: finishing within the timing rules that apply to the task
- **Reliability**: stability of outcomes in similar contexts

These components are lenses for interpreting results, mentioned in Nare et al. (2025); the method that links them to data appears in later chapters.

2.2 On time and tardy completion

Let T be the completion (actual) time for an episode and let D be the expected time. We write $on\_time = \mathbb{I}\{T \leq D\}$ and $tardy = \mathbb{I}\{T > D\}$. We report the share of episodes that finish on time and summarize lateness when it occurs. In most cases, D is provided for each task instance.

2.3 Markov Decision Process

Markov Decision Processes (MDP) provide a mathematical framework for modeling stochastic environments for decision-making problems. An MDP is formally defined using the following 4-tuple

$$\text{MDP} \stackrel{def}{=} \langle S, A, T, R \rangle$$

where S denotes the set of states called the state space, representing all possible configurations of the environment. A denotes the set of actions available to the agent called the action space, T is the transition function, and R is the reward function.

- **State Space** - The configuration of the environment at any time step is called the state s of the environment. The state space S denotes all possible states that the environment can be in. MDPs assume that the state is perfectly observable, meaning that the agent has complete and accurate knowledge of the current state s when making decisions.

- **Action Space** - Given the state s of the environment, an agent is required to take a decision on what action a to carry out at that time step. The action space A is the set of all possible actions that the agent can carry out in the environment.

- **Transition Function** - When an agent carries out an action a in a state s, the environment transitions into the next state s'. The transition function can be deterministic or stochastic and can be defined as

$$T(s, a, s') = \Pr(s' \mid s, a).$$

- **Reward Function** - When an agent performs action a in state s, it receives a reward r. The reward function R determines the reward the agent receives at each time step. R can be a function of only s, or (s, a), or even (s, a, s').

MDPs operate on the Markovian property which states that the future state depends only on the present state and does not depend on the past history. That is,

$$\Pr(s_{t+1} \mid s_t, a_t) = \Pr(s_{t+1} \mid s_0, a_0, s_1, a_1, \ldots, s_t, a_t).$$

The decision horizon T of an MDP refers to the number of time steps into the future for which an agent considers the consequences of its actions when making decisions. The stochastic process is called a *finite-horizon* MDP if the number of time steps is finite. Otherwise, when the number of time steps is infinite, the stochastic process is referred to as an *infinite-horizon* MDP.

Policy The objective of the MDP is to find a good policy π for the decision-making agent. A policy function π is a mapping from the state space S to the action space A, where $\pi(s)$ determines the best action that the agent can carry out in state s. The policy can be

deterministic or stochastic. A deterministic policy maps states to actions as shown below. This means that for any given state, the policy will always produce the same action:

$$\pi : S \to A.$$

However, a stochastic policy maps states to actions via a probability density function. It assigns probabilities to each action that can be taken from a given state, allowing for a range of actions to be chosen under similar circumstances:

$$\pi : S \to \Pr(A).$$

Value Functions Value functions are used to both evaluate the performance of policies and to optimize them. They measure the expected return from states or state-action pairs under a specific policy.

The state-value function $V^\pi(s)$ measures the expected return starting from state s under policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \;\middle|\; s_0 = s \right]$$

Here, γ is a discount factor with a value in the range $[0, 1)$, which balances the importance of immediate and future rewards.

The action-value function, or Q-function $Q^\pi(s, a)$, measures the expected return starting from state s, taking action a, and following policy π thereafter:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \;\middle|\; s_0 = s,\ a_0 = a \right]$$

Optimal value functions are derived from the best possible policy π^* that maximizes the expected return:

$$V^*(s) = \max_\pi V^\pi(s)$$

The value functions can be recursively defined using the Bellman equations. These equations express the value of a state or state-action pair in terms of the immediate reward and the value of subsequent states. The corresponding Bellman optimality equation is

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right]$$

Similarly, the optimal action-value function $Q^*(s, a)$ delivers the maximum return from state s after taking an action a, under the best policy:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a),$$

with Bellman optimality

$$Q^*(s, a) = \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma \max_{a' \in A} Q^*(s', a') \right]$$

Human task modeling. Episodes start at task recognition and end at success or termination. States stack task context, previous action, and progress flags so that assistance use is explicit. Actions are *intent-aware* (e.g., accept/request/ignore suggestion, restart, commit, timeout). A step-level bias flag $b_k \in \{0, 1\}$ marks local tendency to use or avoid assistance, episode summaries of $\{b_k\}$ support stratified analyses. Rewards combine delay and terminal outcomes:

$$R(s_k, a_k, s_{k+1}) = -\lambda_{\Delta t} \, \Delta t_k + \lambda_{\text{succ}} \, \mathbb{I}\{\text{success}\} + \lambda_{\text{on}} \, \mathbb{I}\{\text{terminal}\} \, \mathbb{I}\{T \leq D\}.$$

2.4 Bias in Human-Robot Teams

Bias in HRI refers to systematic attitudes, dispositions, or tendencies that shape how humans interact with robots, independent of real-time system performance. Several biases are particularly relevant:

- **Automation bias**: The propensity to uncritically accept or over-rely on automated decisions, sometimes overlooking errors or missing the need for verification.
- **Algorithm aversion**: Discomfort with, or skepticism toward, robotic or AI-generated suggestions. This may manifest more strongly after witnessing a robot failure, even if overall reliability is high.
- **Anchoring bias**: A cognitive bias in which the first suggestion or experience disproportionately influences subsequent decisions, regardless of other evidence.

These biases do not simply affect subjective impressions; they can tangibly shift collaboration patterns, influence task efficiency, and alter the trajectory of trust and reliance within a team. Understanding and modeling such biases is vital for interpreting experimental outcomes, designing adaptive interfaces, and anticipating real-world teaming challenges.

2.5 Model-based simulation evaluation: overview

Optimal policy evaluation estimates how a target policy would perform using data generated by a different policy. Two families are standard. Model-based estimators learn an approximate environment model and simulate under the target policy. Importance weighting reweights logged trajectories by how likely the observed actions were under the target policy. Hybrids combine simulation with importance corrections to balance bias and variance. (Dudík et al., 2011; Jiang & Li, 2016; Thomas & Brunskill, 2016).

2.6 Notation

We use $s \in S$ for states, $a \in A$ for actions, T for transitions, R for rewards, π for a target policy, b for the behavior policy that produced the logs, γ for the discount, and τ for a trajectory. We write D for an expected time, T for completion time, and $\mathbb{I}\{T \leq D\}$ for the on time indicator. When we refer to episode time between two events we write Δt.

Chapter 3

Related Work

Human-robot teaming is most effective when assistance is timely, useful, and unobtrusive. Four major lines of research inform this thesis: trust in automation, individual and group bias in assistance-seeking, sequential decision-making and policy guidance, and methods for robust, log-driven policy evaluation. This chapter reviews these bodies of work, drawing on foundational literature as well as the practical approach in this thesis, which links logged human behavior to performance outcomes and updates robot trust in an interpretable way.

3.1 Robot Trust Models

Trust is often defined as a belief that the next interaction with a team, whether human or robot, will result in the successful, correct, and timely completion of tasks. Lee and See (2004) explain how trust and reliance are linked and emphasize that trust calibration should be based on observable evidence rather than mere assurances of intent. Reviews by Hancock et al. (2011) and Schaefer et al. (2016) synthesize findings that trust evolves with outcomes such as successes, timing deviations, and failures and is shaped by multiple, interacting components: competency (domain skill), conformance (procedural and timing adherence), and reliability (consistency across context).

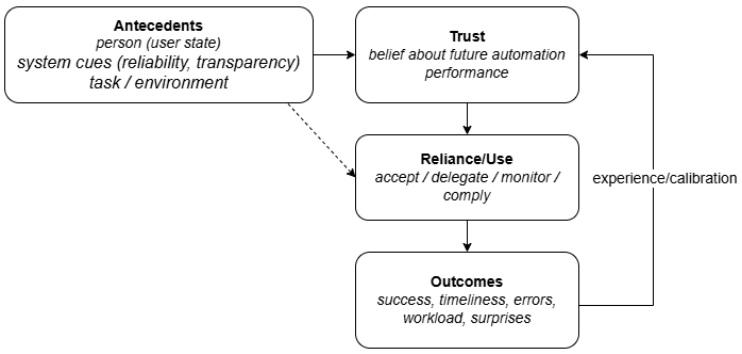

Figure 3.1: Trust and reliance over time. Antecedents shape trust, trust affects reliance, and outcomes such as success, timeliness, and errors update trust. Adapted from Lee and See (2004).

Early work in this domain established that trust models need to be adaptive and situated, responding to ongoing team experience and context changes. Figure 3.1 illustrates how antecedents, such as user traits, environmental factors, and system feedback, shape trust, how trust influences reliance, and how mission outcomes (including timeliness and errors) feed back to recalibrate trust. This perspective is foundational for log-based, outcome-driven trust modeling, which this thesis adopts and extends.

3.2 Bias Toward or Against AI Assistance

Distinct patterns of automation use were identified by Parasuraman and Riley (1997), who distinguished misuse (overreliance) and disuse (underuse) as sources of performance degradation. Dzindolet et al. (2003) demonstrated empirically that perceived reliability and individual cognitive factors cause users to accept, ignore, or distrust robot-provided guidance, in some cases, even when such guidance is statistically optimal. Robinette et al. (2016) found that in stressful situations, such as emergency evacuations, users may overtrust robots and follow instructions despite prior errors.

These studies underscore the importance of measuring both pre-experiment attitudes (e.g., survey-derived bias indices on trust propensity and AI teaming aversion) as well as fine-grained, log-derived behavior (e.g., requesting, accepting, ignoring suggestions, restarting, committing, and timing out). The method presented in this thesis contributes to this tradition by deriving bias flags directly from interface logs, summarizing them at both the step and episode levels, and facilitating stratified analysis of guidance policies.

3.3 Sequential Decision Making and Policy Guidance

Multistep tasks in HRI are inherently sequential, with each action changing the subsequent state and decision context. Models should be concise enough to estimate and interpret from logs, yet rich enough to encode task and context features critical to effective guidance. Nikolaidis and Shah (2013) introduced cross-training approaches, in which humans and robots practice the typical roles of each other, leading to improved coordination and performance, as demonstrated in the follow-up work (Nikolaidis et al., 2015). More generally, intent-aware action labeling, implemented in this thesis via deterministic taxonomies, keeps the guidance comprehensible and makes the use of assistance explicit.

3.4 Model-Based Simulation and Off-Policy Evaluation

Evaluation of new guidance or trust policies increasingly relies on model-based simulation. By learning simple next transition and timing models from logs, one can simulate how alternative policies would have performed (Sutton and Barto (2018)). Importance weighting and doubly robust estimators ensure that data-driven simulations remain statistically valid, and common checks (coverage, calibration, variance) are recommended best practice (Dudík et al., 2011; Jiang & Li, 2016). This thesis follows these guidelines by providing explicit

diagnostics, participant-anchored simulation rollouts, and value iteration verification for all modeled policies.

3.5 Kinship and Adaptive Trust

Recent work by Nare et al. (2025) advances the literature on trust calibration by introducing a computational trust and kinship model. In this framework, robot trust is a dynamic belief updated after every episode about the probability of a positive experience with each human collaborator. The model employs Bayesian inference to update trust based on observed competency, reliability, and conformance, and incorporates kinship, which captures the benefits of repeated, successful collaboration. As human-robot teams accumulate positive history, the kinship term increases, making the robot more tolerant of rare errors and faster to recalibrate upward after new success. Simulation and physical experiments validate that kinship-augmented trust improves resilience and efficiency over time, and provides a mechanism to bridge natural variations in user performance and engagement.

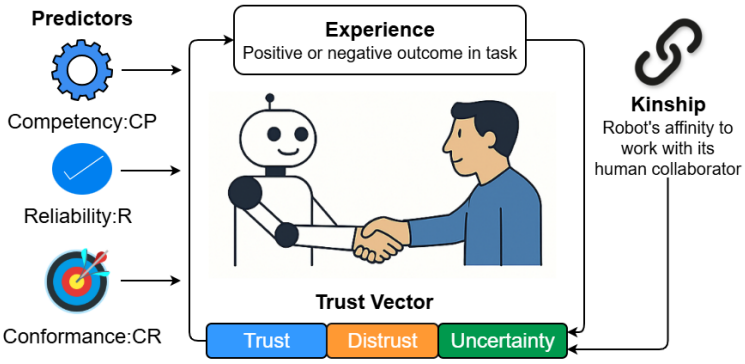

Figure 3.2: Computational trust and kinship update process as described by Nare et al. (2025) Core predictors (Competency, Reliability, Conformance) are used to update the trust vector (trust, distrust, uncertainty). Repeated positive collaboration increases kinship, making the robot's trust adaptation more resilient and context aware.

3.6 Perspective-Taking, Theory of Mind, and Social Adaptation

Modern HRI research increasingly equips robots with perspective taking and theory of mind capabilities. For example, Pandey and Alami (2019) introduces proactive models that allow robots to anticipate ambiguous or incomplete human instructions and adapt plans accordingly. Breazeal et al. (2004) and Almeida and Melo (2022) demonstrate that robots can reason about user beliefs and unobserved intentions, thus enhancing engagement and facilitating richer forms of social learning and trust recovery. Kok et al. (2021) provides frameworks for estimating trust and anticipating communication breakdowns, which facilitate more robust and explainable teamwork.

The field has undergone ongoing refinement, transitioning from static, handcrafted trust and guidance models to highly adaptive, empirically validated computational frameworks. By linking behaviors and outcomes in episode logs to trust and policy change and by incorporating both traditional and kinship based models, the state of the field now supports robust teaming even under heterogeneous, bias sensitive, and rapidly changing conditions. This thesis draws directly on these advances, blending intent-aware parsing, Markovian modeling, bias-aware grouping, and rigorous off-policy evaluation for human-robot teaming.

The present thesis builds on a foundation that combines the empirical rigor of data-driven models, the interpretability of sequential and intent-aware labeling, the flexibility of model-based simulation, and the robustness of adaptive trust and kinship-aware teaming. These methods provide a framework to advance reliable, personalized, and ethical human-robot collaboration.

Author(s)	Model/Approach	Adaptivity	Contribution
Lee and See (2004)	Trust as experience-driven belief	No	Trust calibration framework, importance of feedback
Hancock et al. (2011), Schaefer et al. (2016)	Meta-analyses of trust factors	No	Determinants of trust: user, robot, environment
Ali et al. (2022)	Adaptive trust for multi-robot teams	Yes	Performance-driven team formation
Azevedo-Sa et al. (2021)	Mutual trust calibration	Yes	Bi-directional trust updating and teaming
de Visser and Chen (2020)	Longitudinal calibration	Yes	Memory and trust recovery after failure
Nare et al. (2025)	Computational trust & kinship	Yes	Kinship-driven resilience, Bayesian adaptation
Bhat et al. (2023)	Feedback & preference adaptation	Yes	Bias-aware team personalization

Table 3.1: Core Models and Approaches in HRI Trust, Bias, and Sequential Policy Evaluation

Chapter 4

Dataset Formulation

This chapter explains the data sources, how logs are divided into episodes, how intent-aware action labels are assigned, and which outcome variables are used in later analyses. Section 4.1 gives an overview of the testbed and tasks. Section 4.2 describes how episodes are formed and how logs are matched with configuration files. Section 4.3.1 defines the bias features, and Section 4.4 lists the integrity and coverage checks used.

4.1 Testbed and Tasks

The data used in this dissertation come from a laboratory study where a human and a robot work together. The collaboration takes place in a virtual building that simulates a cooperative video game for participants. Participants use a desktop computer and view the building from above. They control a human avatar and give high-level commands to a mobile robot teammate. The team has a limited time to finish a series of mission goals. Almost all actions by the participants and the robot are saved in a log file.

The virtual building has several offices, corridors, and a separate server room. Access to the server room is restricted until a coded door is unlocked. During the mission, the human and robot must (i) find and hack a computer, (ii) search rooms for clues that complement each

other, clues that reveal the door access code, and (iii) navigate a maze-like server room to shut down a compromised server. Neither the human nor the robot can finish all tasks alone. The robot can move quickly and read machine-encoded markers, but it cannot understand written information clues or determine which wire to cut on the server. The human can read and interpret the puzzle screens and written hints, but cannot go into the maze or see what the robot sees unless the participant specifically asks for this information.

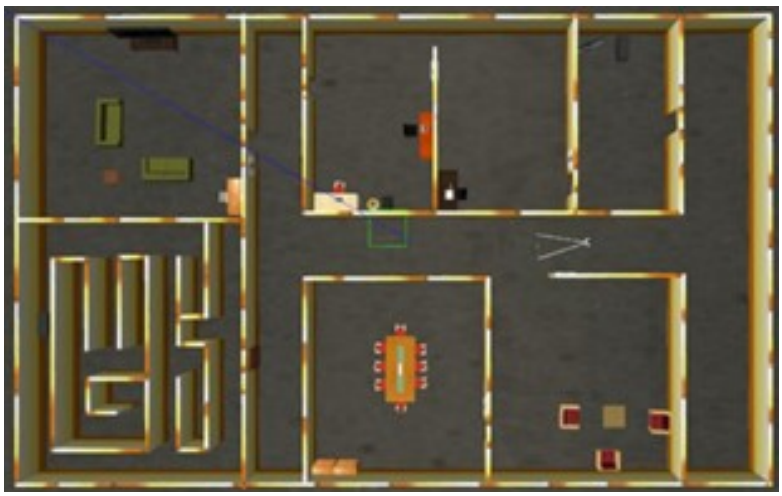

Figure 4.1: Mission area used in the human–robot collaboration study. The floor plan includes multiple rooms that may contain the computer and clues, as well as a maze-like server room where the compromised server is located.

Each participant filled out a short online survey before coming to the lab session. At the start of the session, a researcher explained the overall mission story (finding and shutting down the compromised server), reviewed the consent form, and let the participant practice basic controls in a training area. Once the participant showed they could navigate, interact, and give basic commands, the main mission commenced. In each session, the participant completed two missions using the same map, with the same overall goals but different timing. The team either shut down the server successfully, ran out of time, or reached a set time

limit. During both missions, the software recorded interface events, chat messages, and task results with timestamps.

Figure 4.2 shows an overview of how a mission unfolds and presents several representative screens from the testbed.

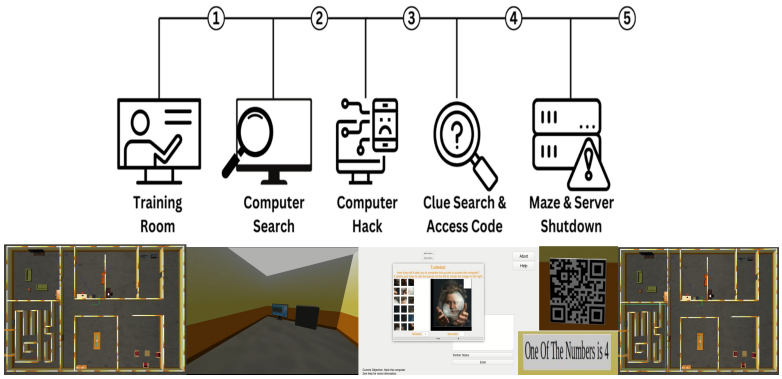

Figure 4.2: Consolidated view of the mission. Top row: main stages from the initial training room through computer search, computer hack (CH), clue search and access code (AC), and maze and server shutdown. Bottom row: representative screenshots of the virtual environment, the computer hack interface, and example QR and text clues.

4.1.1 Mission Flow and Roles

The study always involved one human operator and one autonomous robot. The robot, which runs as an agent inside the virtual environment, can:

- search specific rooms on its own,
- propose which room to search next,
- reports its current location and activity (for example, “I am searching room X”),
- read QR–coded clues and send their contents back, and
- attempts to move through the maze and approach the server.

The human has complementary abilities:

- free navigation of the avatar through the building,
- access to textual clues (notes on walls, printed hints, etc.),
- a user interface for approving or redirecting the robot's search plans,
- controls for requesting the robot's status or camera view, and
- final authority over how to solve the computer puzzle and which server wire to cut.

This division of labor is intentional. The robot is good at moving through the environment and scanning, but it is limited in understanding information. Humans are better at interpreting information and making strategic decisions, but they cannot reach as far. Many observed behaviors come from participants choosing to use, ignore, or override the robot's suggestions.

4.1.2 Task 1: Computer Hack (CH)

The first main task in each mission is to locate and access a computer. At the start, the team knows that the computer exists somewhere in the building and that it contains information that will make later tasks easier. The robot can be told to start searching rooms on its own, or the human can walk their avatar through rooms and search more manually. When the computer is found, the interface zooms in to a puzzle screen, and the team has to decide who is going to hack it.

The hack is implemented as a small picture-based logic puzzle with a limited number of attempts. The participant can let the robot take the lead, take over the puzzle themselves, or switch if the first attempt fails. Solving the puzzle correctly reveals two important pieces of information that the team will need later: a blueprint of the server room layout and a mapping between wire properties and which wire is safe to cut. If the team fails the hack,

they can keep playing, but they lose access to this information and have to rely on guesswork during the maze and server shutdown.

Throughout this stage, the logs record, among other things, whether the human chooses to search themselves or rely on the robot, when they decide to start the search independently or rely on the robot, when they initiate the hack, and which agent's collection of decisions and actions as the *Computer Hack* (CH) task. The action labels used in my models (e.g., `Decide_search_room`, `Decide_start_hack`, `Request_turtlebot_status` are derived from this segment of the mission).

4.1.3 Task 2: Clue Search and Access Code (AC)

Once the computer stage is over, the team moves into a clue-finding and reasoning phase. The server room door is locked with a multi-digit code, and six clues are scattered around the building. Three of them are written as short text statements that only humans can read; the other three are QR codes that only the robot can scan. The access code puzzle can only be solved if the human and robot share what they find.

The robot does not simply wander at random. Before it begins searching a room, it sends a message such as "I am searching room X. Should I continue?". On the operator interface, this shows up with buttons that the participant can press to approve or redirect the robot's plan. A simple version of this screen is shown in Figure 4.3. The human can also request the robot's status or camera feed at any point, and those choices are also logged.

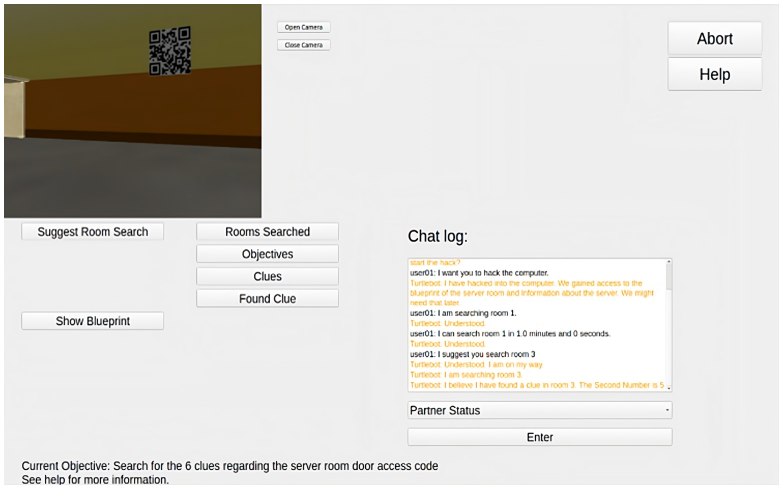

Figure 4.3: Operator interface during room search and clue collection. The robot announces its intended search (for example, "I am searching room X. Should I continue?"), and the participant responds using on-screen buttons. These events are later mapped to intent-aware action labels and suggestion flags in the dataset.

As clues are discovered, the robot transmits the contents of QR–coded clues, and the human reads and enters the text clues. The interface provides a simple form for entering guesses for the door code. Each wrong guess consumes one attempt, and there is a fixed maximum number of attempts before the door remains locked. When the team enters the correct code, the door opens, and the mission can continue into the maze. For AC task, the specific action labels `Enter_code_attempt_1`, `Enter_code_attempt_2`, and `Enter_code_attempt_3` describe the sequence of code entries a participant makes while trying to open the door.

4.1.4 Task 3: Maze Navigation and Server Shutdown

After the door is unlocked, the robot can enter the server room, which is laid out as a small maze. Humans cannot see this room directly, instead, they have the blueprint that was revealed during the computer hack (if the hack was successful) and whatever the robot reports. At first, the only way for the robot to get through the maze is for the human to

guide it step by step. The human can ask where the robot is and then send instructions such as "move forward one cell" or turn right. The robot executes the command, updates its position, and reports back.

If the team struggles, a backup option becomes available: the human can send the full blueprint to the robot, after which the robot can plan the remaining path on its own. The logs distinguish between these guided moves and the later autonomous segments, which matters for how we think about shared responsibility.

The final decision in the mission is to shut down the compromised server. The server is represented as a set of colored wires and a serial number that only the robot can see. The human must use the earlier information from the computer to decide which wire is safe to cut. The robot reports what it sees near the server, and the human responds with a command indicating which wire to cut. Cutting the correct wire leads to a successful mission. Cutting the wrong wire results in failure, even if there is still time left on the clock.

4.1.5 Field Schema

All of the stages above are instrumented so that low-level events can be matched to higher-level task structure. Each mission produces a sequence of timestamped log entries and a companion configuration row that summarizes task durations and terminal outcomes. Each task instance includes a discrete set of human actions captured in the logs, any robot suggestions or hints that were surfaced, and a final status (success, abort, or timeout) with a configured time target D. Roles are encoded as integers and remain stable through preprocessing: `role` $\in \{1, 2, 3\}$ denotes Monitor, Assist, and Guide.

All modeling in later chapters relies on a compact schema of timestamps, identifiers, raw actions, suggestion metadata, and outcome fields that downstream steps can consume (for segmentation, labeling, and evaluation). Table 4.1 lists the fields I keep after preprocessing.

Field	Type	Source	Description
`timestamp`	datetime	Logs	Wall clock time of event.
`participant`	string	Logs	Participant identifier.
`task_id`	string	Logs	Identifier for task instance.
`role`	int	CSV files	1 = Monitor, 2 = Assist, 3 = Guide.
`action_str`	string	Logs	Raw operator action string.
`suggestion_type`	string	Logs/CSV files	Suggestion category if surfaced.
u_k	binary	Derived	Suggestion present at step k.
`taxonomy`	categorical	Derived	Intent aware action label.
b_k	$\{0, 1\}$	Derived	Bias flag from (`taxonomy`, u_k).
`actual_time`	float	Config	Total elapsed time per task instance.
D (`expected_time`)	float	CSV files	Target time for on time completion.
`goalstatus`	binary	CSV files	Terminal success label.
`episode_id`	int	Derived	Contiguous segment index per participant.
T (`duration_sec`)	float	Derived	Episode duration (seconds).
`on_time`	binary	Derived	$\{T \leq D\}$; retain lateness $\max(0, T - D)$.

Table 4.1: Field schema used in preprocessing and modeling.

4.2 Episodes from Logs

An episode is a continuous sequence of events for one `participant` and `task_id`. It starts with the first actionable event for a new `task_id` or a clear "start mission..." marker, if available (the marker is preferred). The episode ends with a terminal success, abort, or timeout marker. If there is no marker, a long enough period of inactivity ends the episode. Step durations are calculated from the timestamps. If timestamps are missing, response time vectors (if available) are used to estimate timing within the episode.

Aligning logs and configuration. When configuration rows are ordered but do not have timestamps, alignment is done for each participant by episode order. Two checks are used:

whether the terminal labels match and whether the episode duration is close to the configuration time budget. Records that do not align are flagged and left out of counterfactual analysis.

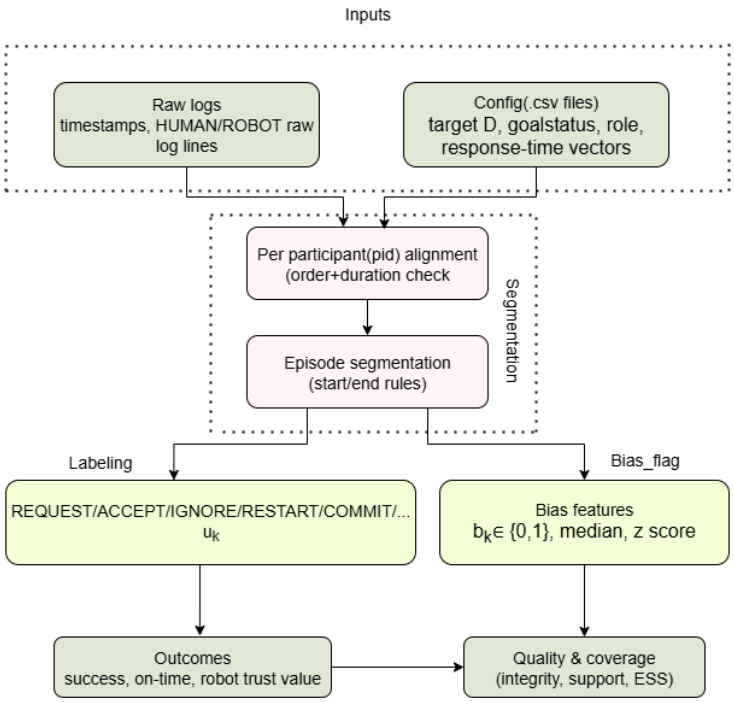

Figure 4.4: Flow from raw logs and configuration to episodes, action labels, bias features, outcomes, and quality and coverage checks.

Interface events are mapped to a small, clear set of actions that make assistance use explicit: `request`, `accept`, `ignore`, `restart`, `commit`, and `timeout`. Mapping uses fixed, case-insensitive rules that are the same for both training and evaluation. The rules use regular expressions. If an event does not match any rule, it is set to `ignore` to avoid overstating assistance use. Terminal cues are handled separately.

Action	Existing simulation log cues
request	clue|suggest|hint|ask robot
accept	apply suggestion|use suggestion
ignore	proceed without robot suggestion|dismiss
restart	undo|rollback|restart
commit	commit|submit|code correct|found computer
timeout	abort|out of time|timeout

Table 4.2: Deterministic mapping from strings to intent-aware actions.

Suggestion presence. A binary flag u_k shows if a suggestion was available at step k. The flag is 1 if the observation records a suggestion type or if the robot's trace at that time includes a suggestion or hint. Otherwise, the flag is 0.

4.3 Bias Features and Grouping

4.3.1 Pre-experiment Bias (Trust & Human–AI Teaming)

Before analyzing behavior in the simulator, each participant's tendency to use accurate AI, called *pre-bias*, is estimated. The idea is that people who trust AI less and are more reluctant to work with it are more likely to ignore helpful guidance during later tasks.

Two short pre-task questionnaires are given:

- **Propensity to trust AI** (`p_trust_ai1..5`; 5-point agreement). Item 4 is negatively worded ("I don't trust the information I get from automated agents") and is reverse-scored so that higher values always mean *more trust* (on a 1–5 scale, this is $6 - x$).
- **Human–AI teaming aversion** (`hai_team(HAI)7..9`; 5-point agreement). These three items capture reluctance to team with AI (e.g., "I would dislike collaborating

with an AI at work"). All are reverse-scored, so higher values indicate a *more positive* stance toward teaming.

For each participant, the average score is calculated for each of the two composite measures:

$$\texttt{propensity\_pre\_raw} = \overline{\texttt{p\_trust\_ai1..5}}, \qquad \texttt{hai\_team\_pre\_raw} = \overline{\texttt{hai\_team7..9}}.$$

To make the measures comparable, each one is standardized (sample z), the sign is reversed so that higher values mean greater bias, the standardized scores are averaged equally, and the result is mapped to $[0, 1]$ using the standard normal cumulative distribution function (CDF):

$$\text{bias\_index}_z = \tfrac{1}{2}\big(- z_{\text{trust}} - z_{\text{HAI7–9}}\big), \qquad \text{bias\_propensity\_pre} = \Phi(\text{bias\_index}_z) \in [0, 1].$$

A clear binary flag is then defined as follows:

$$\texttt{bias\_flag\_pre} = \mathbb{I}\{\text{bias\_propensity\_pre} \geq 0.5\},$$

where 0.5 is the neutral point on the probit scale (ties count as 1).

Assistance bias at step k is $b_k \in \{0, 1\}$, based on $(\texttt{taxonomy}_k, u_k)$. We set $b_k = 1$ when a suggestion is present (u_k=1) and the action shows use or request of assistance (`request`, `accept`, `commit`). Otherwise, $b_k = 0$, including when a suggestion is present but the action is `ignore` or `restart`. Episode summaries include the total $\sum_k b_k$, the longest run of $b_k = 1$, and the fraction of suggestion-present steps with $b_k = 1$. These features are used for descriptive analysis and next action models, they do not affect trust predictors.

4.4 Data Quality and Coverage

Integrity filters remove non-monotonic timestamps, combine exact duplicates at the same timestamp, and enforce correct order within each task. Episodes with fewer than two actions or without terminal markers are left out of off-policy evaluation. Missing suggestion metadata are filled in as "no suggestion" with a missingness flag. Unmapped action strings are set to `ignore` with an unknown token indicator. Implausible or negative step times are marked as missing, but terminal durations are kept if they are consistent. Training and evaluation splits are made at the episode level to prevent data leakage.

Check	**Criterion**	**Action**
Timestamp monotonicity	Non-decreasing within episode, large gap ends episode.	Drop or end segment.
Terminal consistency	Terminal marker present, agrees with CSV files.	Keep, else exclude from policy evaluation.
Suggestion metadata	Missing $\Rightarrow$ set u_k=0, flag missingness.	Impute and flag.
Action mapping	Unmatched raw strings.	Map to `ignore` and set unknown flag.
Duration sanity	Negative or implausible step times.	Mark missing, retain if terminal is consistent.
Support (overlap)	$b(a \mid s)>0$ where $\pi(a \mid s)>0$.	Warn, prefer model-based or doubly robust estimates.

Table 4.3: Quality filters and coverage diagnostics applied before evaluation.

Chapter 5

Human MDP and Optimal Policy Evaluation

This chapter shows how to model human task execution using a Markov Decision Process (MDP). After introducing the MDP approach, we build a transition model from logs using a single decision tree. Rewards match the task design, reflecting both timing and final results. We then apply value iteration to find the best policy and test it from each participant's starting point. Throughout, all details are traceable: probabilities and rewards are based on observed data, and we note when estimates are necessary due to limited data. Specifically, in our study, participants complete missions by working through tasks in order: `Locate`, `Access Code`, `Maze`, and `Shutdown/Wire`. Each mission ends in a terminal state labeled `Succ[reason]` or `Fail[reason]`. Reasons for failure include running out of time, making too many attempts, or choosing the wrong wire. We record whether each task is finished on time or late. Our aim is to learn actual patterns of progress from observed actions and determine a policy that balances speed, actions, and successful outcomes.

Figure 5.1 shows the steps this chapter takes to build a human MDP. The process begins with simulation data that includes pre-, in-, and post-experiment variables. We keep only valid participants and assign a binary pre-bias flag. Next, we parse the logs into intent-based

actions and task features, then use these as input for a decision tree. Leaf-level transition counts and the reward design are used to define a bias-aware MDP. The rest of the chapter evaluates the optimal policy for this MDP.

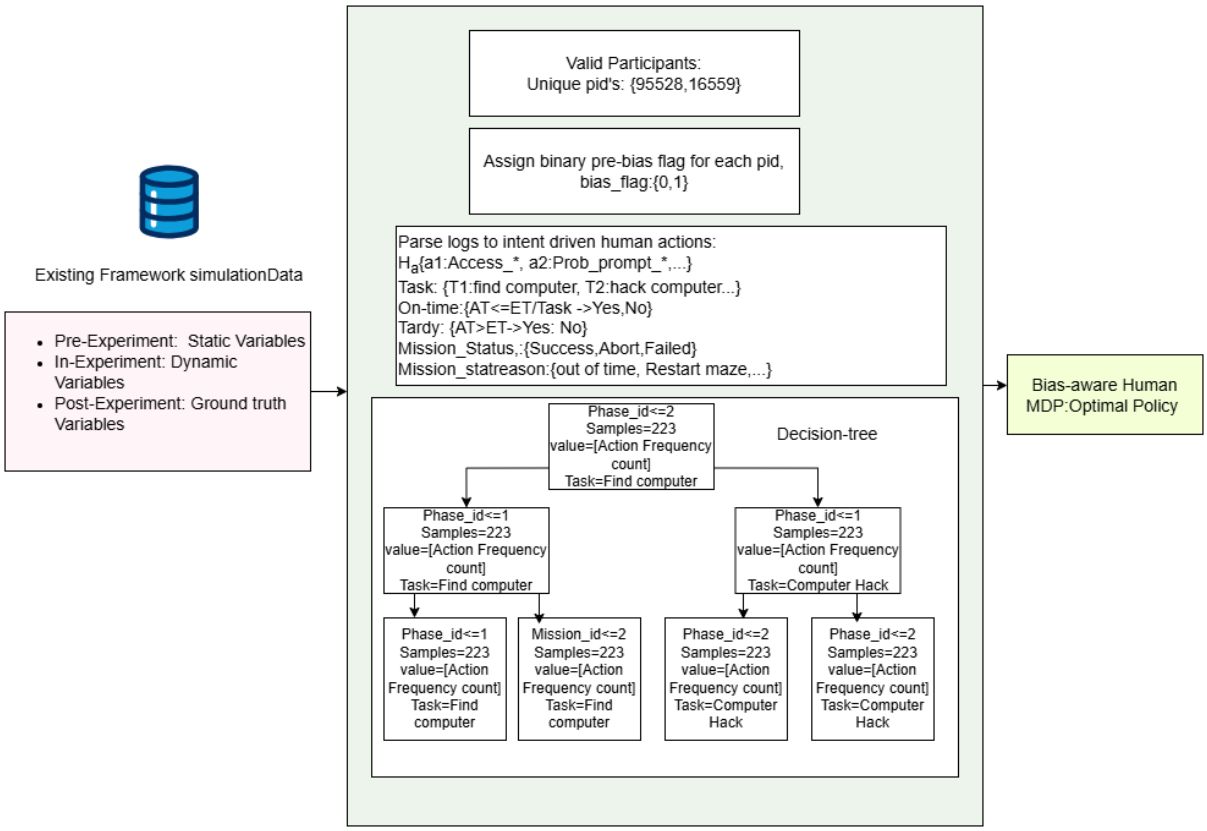

Figure 5.1: Simulation data are filtered to valid participants, enriched with a binary pre-bias flag, and parsed into intent-driven actions and task features. A decision tree produces abstract states and transition counts, which define the bias-aware human MDP and its optimal policy.

5.1 State, Actions, Timing, and Terminals

The analysis uses an abstract state space

$$s \equiv (\texttt{Task},\ \texttt{bias\_flag} \in \{0,1\},\ \texttt{pace\_prev\_int} \in \{0,1,2\}),$$

In this setup, `pace_prev_int` shows the timing from the previous segment: 0 means start or unknown, 1 means on-time, and 2 means tardy. Actions $a \in \mathcal{A}(s)$ are normalized `action_label` tokens taken from logs, such as `enter_code`, `maze_restart#2`, or `cut_wire`. Missions end in absorbing outcomes $\mathcal{T} = \{\texttt{Succ}[\texttt{reason}], \texttt{Fail}[\texttt{reason}]\}$, and the reason string is kept for assigning rewards and for reporting.

5.2 Modeling Dynamics with a Decision Tree

To model next-step dynamics and make predictions, we rely on a single, easy-to-read CART decision tree. This tree predicts the next main phase. Terminal states form a special class. We choose features to keep the model easy to understand:

$$X = \{\texttt{mission\_id}, \texttt{phase\_id}, \texttt{task\_id}, \texttt{attempt\_idx}, \texttt{pace\_prev\_int}, \texttt{bias\_flag}\}.$$

We train the decision tree using class balancing and conservative regularization, like limiting the tree's depth and setting a minimum leaf size. This helps prevent overfitting and keeps the model easy to interpret.

Leaf-wise estimation and backoff. Let $L(s)$ be the tree leaf for s. Within L, we estimate the human action choice and the next-state distribution with Laplace smoothing ($\alpha = \beta = 1$):

$$\pi_{\text{human}}(a \mid L) = \frac{N(L,a) + \alpha}{\sum_{a'} N(L,a') + \alpha|\mathcal{A}_L|}, \qquad P(s' \mid L, a) = \frac{N(L,a,s') + \beta}{\sum_{s''} N(L,a,s'') + \beta|\mathcal{S}'_{L,a}|}.$$

The model transition is given by $T(s'|s,a) = P(s'|L(s),a)$. If there is not enough data for a specific (s,a) at the leaf, the estimate backs off to the phase level, and if needed, to a global pool. Each row notes its `backoff_level` (`leaf`, `phase`, or `global`). We also track unique-participant support to avoid giving too much weight to long transcripts.

5.3 Reward Design

The reward function includes both per-step shaping and terminal outcomes. Specifically, per-step shaping discourages extra actions and lateness. In contrast, terminal rewards reflect the mission goal and time discipline. Additionally, a small positive bonus is given for finishing critical tasks on time.

5.3.1 Per-step shaping

At each segment exit:

$$r_{\text{step}}(s, a, s') = -1 - 20\,\mathbb{I}\{\text{tardy exit}\} - 20\,\mathbb{I}\{\text{timeout indicator}\} + b_{\text{crit}}\,\mathbb{I}\{\text{critical task}\}\,\mathbb{I}\{\text{on-time exit}\}. \tag{5.1}$$

Critical tasks are phases where `phase_id` $\in \{4, 5, 6\}$, which are Access Code, Maze/Server-room Navigation, and Shutdown/Wire. Unless stated otherwise, $b_{\text{crit}} = +30$, and this bonus is not given on the terminal segment to avoid counting it twice with the success reward.

5.3.2 Terminal outcomes

On transitions into terminals:

$$r_{\text{terminal}}(s, a, s') = \begin{cases} +100, & \text{if terminal is } \textit{Success} \text{ and the final segment is on-time,} \\ -100, & \text{if terminal is } \textit{Failure/Abort}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

If the terminal reason is a *timeout*, we add an extra -20 (e.g., timeout, attempts exceeded, wrong wire, maze restarts) on top of the -100. The total reward is $r(s, a, s') = r_{\text{step}}(s, a, s') + r_{\text{terminal}}(s, a, s')$.

Component	Rule
Action cost	-1 per step
Task exit (tardy)	-20 if exit is tardy
Task exit (timeout indicator)	-20 if a segment-level timeout is flagged
Critical on-time bonus	$+30$ if `phase_id` $\in \{4, 5, 6\}$ and exit is on-time
Terminal: success	$+100$ only if the *final* segment is on-time
Terminal: failure	-100 for failure/abort terminals
Terminal: extra timeout	additional -20 if terminal reason indicates timeout

Table 5.1: Reward specification used in MDP solver.

5.4 Solving the MDP

We solve the discounted MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, r, \gamma \rangle$ with $\gamma = 0.99$ using Gauss–Seidel value iteration. The process continues until $\|V^{(k+1)} - V^{(k)}\|_\infty < 10^{-6}$:

$$Q(s,a) = \sum_{s'} T(s'|s,a)\big(r(s,a,s') + \gamma V(s')\big), \qquad V(s) = \max_{a \in \mathcal{A}(s)} Q(s,a), \qquad \pi^*(s) = \arg\max_a Q(s,a).$$

We export the state values V, action values Q, the greedy policy π^*, and the advantages $A(s,a) = Q(s,a) - \max_{a'} Q(s,a')$.

5.5 MDP Evaluation using Model-Based Simulation

To evaluate how the learned policy performs for this group, we begin each simulation from the participant's actual starting state. We follow π^* under T until reaching a terminal state or a set step limit. For each participant and mission, we report the expected return, success rate, on-time success rate, timeout rate and other failure rates (from terminal labels), and the median number of steps to finish. Results are combined across missions for each participant and are broken down by bias_flag when needed.

To make our process easy to audit, we include the following: (a) the policy for each abstract state (π^*, V, A, and available actions), (b) the transition rows used with $T(s'|s,a)$, unique-participant support, and `backoff_level`, (c) immediate reward estimates for each (s,a), and (d) participant summaries. For each participant, we provide a compact policy dictionary for the states actually visited.

We run several checks. First, we check probability mass conservation, making sure $\sum_{s'} T(s'|s,a) = 1$ for each used (s,a), second, coverage by unique-participant support, and the share of leaf, phase, or global backoff. We also check value-iteration convergence. Main limitations include the approximate Markov assumption at the abstract state level, pooling differences when data are sparse (which is reduced by backoff), and possible noise in log parsing and task labels.

We use a single, interpretable decision tree (a flowchart-like model that splits data into segments based on context features) to divide the context. Leaf-wise estimates (statistics from the end nodes of the tree) with principled backoff (systematically using data from less specific nodes when needed) provide a clear transition model. The reward structure (the assignment of values to outcomes) aligns with the experimental goals, and the timing discipline (adherence to time constraints) aligns with it. Value iteration (an algorithm for computing the optimal decision policy) finds the optimal policy. Model-based rollouts (simulations using the model to predict possible outcomes) produce participant-specific outcomes with clear origins.

Chapter 6

Results: Policy Effects and Robot Trust Model Integration

We review task records from logs and configuration files for preregistered participants. Operational outcomes and timing are taken from the configuration: mission success is based on `goalstatus` (1 for success, 0 for failure), duration is measured by `actual time` (minutes), and the deadline is set by `expected time` (minutes). An episode is considered on time if `actual time` $\leq$ `expected time`. We calculate success, on time, and tardy rates for each task row in the preregistered subset ($N = 220$). On-time and tardy rates include only rows where both `actual time` and `expected time` are available.

6.1 Assign binary pre-bias flag to each participant

This analysis covers participants who completed at least one mission. In total, 179 unique people took part: 141 finished Mission 1, 112 finished Mission 2, and 74 completed both. All summaries below are based on each participant. Before participants used the system, we created two short survey composites. The first measured participant's trust in AI. It used 5 items, with item 4 reverse-scored. The second composite used HAI teaming items 7-9, all

reverse-keyed. We standardized each composite for every participant. Higher scores show greater pre-bias, or a lower willingness to team. We averaged the two standardized scores and put them on a 0-1 scale for easier understanding. A score of 0.5 or above counts as high pre-bias. To better illustrate participant distribution on this scale, I have plotted all the pre-bias scores in Figure 6.1. Each bar shows the number of participants in each range between 0 and 1. The vertical line marks the 0.5 cutoff. This cutoff is used to create the binary pre-bias flag for later analyses.

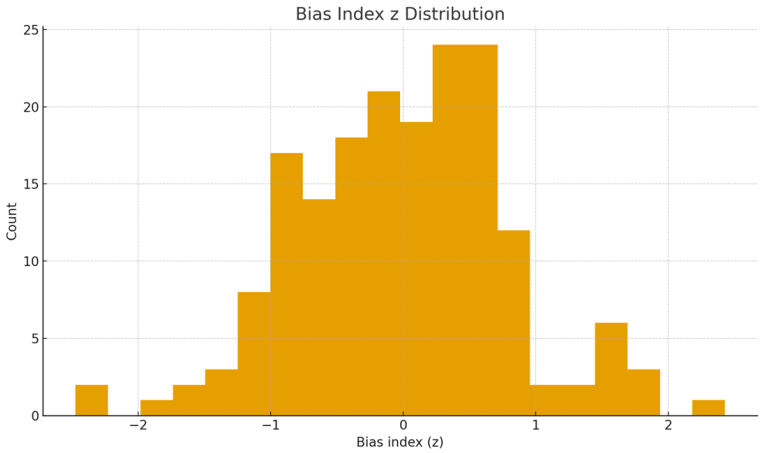

Figure 6.1: Distribution of the pre-experiment bias propensity scores on a 0–1 scale. The vertical line at 0.5 marks the threshold used to split participants into low- and high-pre-bias groups.

Figure 6.1 shows participants are distributed throughout the full range of pre-bias scores, rather than clustered at one end. There are many participants on both sides of the 0.5 cutoff. Both the low- and high-pre-bias groups are well represented. This supports the use of a simple binary pre-bias flag for MDP and policy evaluations. It also recognizes that the underlying scores form a continuous distribution.

Distributions and reliability. Both composites worked well. Their distributions are mostly unimodal with a moderate spread. This supports the standardization and mapping

we used. Internal consistency is reasonable for short scales. Table 6.1 shows the distribution for each measure. Table 6.2 shows Cronbach's alpha for each item set.

Measure	Mean	SD	Min	Max
Propensity to trust AI (1–5)	3.002	0.653	1.400	4.600
HAI 7–9 (1–5)	3.469	0.814	1.000	5.000
Bias propensity (0–1)	0.501	0.246	0.007	0.992

Table 6.1: Distribution of pre-experiment measures (participant level; $n = 179$).

Measure	Items (k)	Cronbach's alpha
Propensity to trust AI	5	0.676
HAI 7–9	3	0.653

Table 6.2: Internal consistency of item sets (Cronbach's alpha).

How the two composites relate. At the participant level, the two composites, each measuring a different aspect of pre-bias, show a positive relationship as expected. The Pearson correlation is 0.266 with a 95 percent confidence interval of [0.124, 0.397], and the Spearman correlation is 0.232. This amount of agreement is typical for short, related scales that measure different sides of pre-bias. Building on this relationship, we further examine how the composites support practical decision-making. Dividing the 0–1 bias scale at 0.5 yields two groups of equal size: 92 participants exhibit higher pre-bias, and 87 exhibit lower pre-bias. The group averages differ clearly and consistently across both inputs to the index (Table 6.3), demonstrating the flag's practical value.

	n	Propensity to trust AI		HAI 7–9	
		Mean	SD	Mean	SD
Higher pre-bias (flag = 1)	92	2.617	0.479	2.953	0.616
Lower pre-bias (flag = 0)	87	3.409	0.561	3.996	0.640

Table 6.3: Group differences by pre-bias flag (participant level).

The survey-based index shows which participants are more hesitant to trust an AI partner. The distributions are stable, and the reliability is good for short scales. Correlations also match expectations. The main flag divides participants into two clear, understandable groups. This gives a simple starting point for studying how pre-bias connects to later behavior and model results in the mission completer group. Building on these findings, we next detail our process for parsing logs to intent-driven human actions.

6.2 Parse logs to intent driven human action

We use the same action families (categories that group related actions) as described in Chapter 4. The exported action labels start with an actor token, such as `HUMAN_/ROBOT_/SYSTEM_`, which we remove for reporting. We assign families according to Chapter 4.2, using two passes to ensure that frequent verbs (common action words) and cues (signals that indicate intent) are counted correctly:

- **Direct family prefixes** map as written: `UI_*`, `NAV_*`, `ACCESS_*`, `CLUE_*`, `PROB_PROMPT_*`, `WIRE_*`, `MISSION_*`.
- **Observed label patterns** are grouped to the nearest family:
  - `ANSWER_`, `TRUST_`, `DECIDE_`, `REPORT_`, `SAY_`, `SUGGEST_`, `ACKNOWLEDGE_`, `DECLARE_`, `CONFIRM_` → `PROB_PROMPT_*`.

Code	Task (name)	Dominant families (counts)
CS	Clues Search	PROB_PROMPT 33,413; UI 10,689
MZ	Maze	UI 4,732; NAV 4,572
FC	Find Computer/Found Computer	PROB_PROMPT 7,325; UI 3,434
RM	Restart maze	UI 3,037; NAV 2,355
CH	Computer Hack	PROB_PROMPT 5,225; MISSION 949
TR	Training room	UI 2,133; PROB_PROMPT 2,004
SS	Shutdown Server	PROB_PROMPT 3,042; WIRE 234
AC	Access code	ACCESS 1,562; UI 763
CC	Correct code	ACCESS 1,451
IC	Incorrect code	UI 416; PROB_PROMPT 168
IC1	Incorrect code #1	ACCESS 129
IC2	Incorrect code #2	ACCESS 37
ICMF	Incorrect code: Mission failed	ACCESS 23
RMA1	Restart Maze attempts #1	NAV 372
RMA2	Restart Maze attempts #2	NAV 158
RMA3	Restart Maze attempts #3	NAV 34
OTN	Out of time navigating the maze	NAV 284

Table 6.4: Log parsing results using a task taxonomy. Each row shows the most active action families for the task (aligned to Chapter 4.2).

- `KEYPAD` / `PASSWORD` / `PASSCODE` / `PIN` / `CODE` / `UNLOCK` / `DOOR` → `ACCESS_*`.
- `CLUE` / `DIGIT` / `NOTE` / `HINT` → `CLUE_*`.
- `WIRE` / `CUT` / `CONNECT` / `DISCONNECT` → `WIRE_*`.
- `LOCATE` / `MAZE` / `NAVIGATE` / `ROUTE` → `NAV_*`.
- `SHUTDOWN` / `RESTART` / `START` / `RESET` → `MISSION_*`.

The parsed logs have **113,553** from **230** participants. There are **57,791** flagged events (`Bias_flag`=1), and **119** participants have at least one flagged row. No generic fallbacks are left. After removing the actor token (before grouping), there are **1,817** unique action labels.

After summarizing the structure of the parsed logs, we now describe how we keep results readable on a single page: each task is given a short code (e.g., `AC` for *Access code*). We then

report the dominant action families for each task, aligned to Chapter 4.2. This avoids large cross-tabs and cells with many zeros. Navigation-heavy tasks (e.g., *Maze*) show the expected `NAV_*`/`UI_*` mix. Keypad/code tasks (*Access code*, *Correct code*) concentrate in `ACCESS_*` with supporting `UI_*`. *Clues Search* spans `UI_*` (surface interaction) and `PROB_PROMPT_*` (dialog/acknowledgement). Control tasks (e.g., *Shutdown Server*) contribute to `MISSION_*` and, when wires are explicit, `WIRE_*`. Overall, the distribution matches the families in Chapter 4.2 and provides specific, high-coverage inputs for subsequent modeling.

6.3 Decision Tree Creation

The single CART model is compact and interpretable. Key figures: Train accuracy = 91.13%; Train macro-F1 = 74.11%; CV macro-F1 = 74.07% (±0.34%); Number of leaves = 92; Classes modeled (incl. terminal) = 8.

6.4 Human MDP: Transitions and Rewards

The model covers 38 state-action pairs; the median branching factor is 2.00 (mean 2.11). Across (state, action) pairs, the average immediate reward is 1.07 (median -1.00).
The decision tree, human MDP further details mentioned at Appendix D

6.5 Task-level MDP results for Computer Hack and Access Code

Statistical analyses in Chapter 4 showed that prior propensity to trust AI was most clearly related to downstream performance in two tasks: *Computer Hack* (CH) and *Access Code* (AC). Here, I build smaller, task-local MDPs for CH and AC, and compare their prescriptions

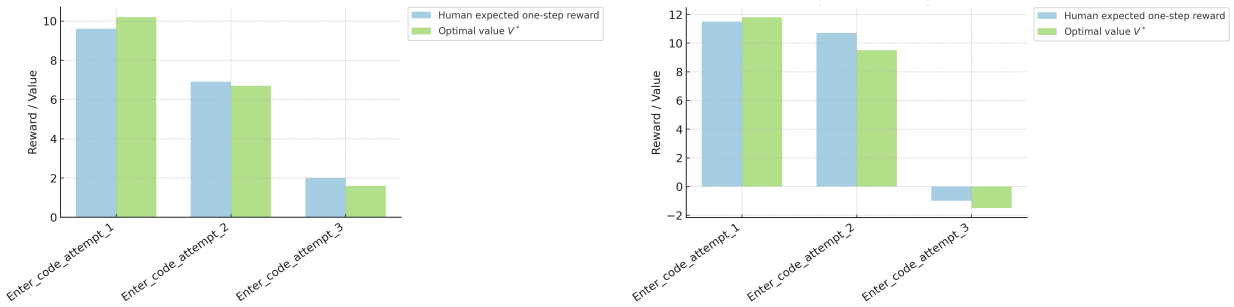

Figure 6.2: Access Code (AC) task. Observed one-step reward (blue) and optimal MDP value (green) by action, split by pre-bias group (left: low pre-bias, right: high pre-bias).

across the low- and high-pre-bias groups. These models use action sets that include only the choices available to the human when interacting with the robot:

- **CH actions:** `Decide_search_room`, `Decide_start_hack`, `Request_turtlebot_status`, `Say_hacked_done`, `UI_open_camera`, `UI_open_objectives`.
- **AC actions:** `Enter_code_attempt_1`, `Enter_code_attempt_2`, `Enter_code_attempt_3`.

The reward structure follows the design in Chapter 5. In CH, each step has a cost of -1. There is an extra $+10$ for `Decide_start_hack`, and a final reward of $+20$ for `Say_hacked_done` if the hack succeeds. In AC, every code attempt costs -1, plus an extra -5 penalty per attempt (total -6). A final reward of $+20$ is given when the correct code is entered. For each task and pre-bias group (flag $= 0$ for low pre-bias, flag $= 1$ for high pre-bias), I estimate transition probabilities from the relevant log subset. I then calculate the observed expected one-step reward for each action and solve the task-specific MDP using value iteration to find the optimal value V^* for each action. Figures 6.2 and 6.3 show these values for AC and CH. Each panel highlights two sets of bars. The light blue bars represent the observed expected one-step reward for human trajectories. The light green bars show the optimal MDP value for the same action under the task-specific model.

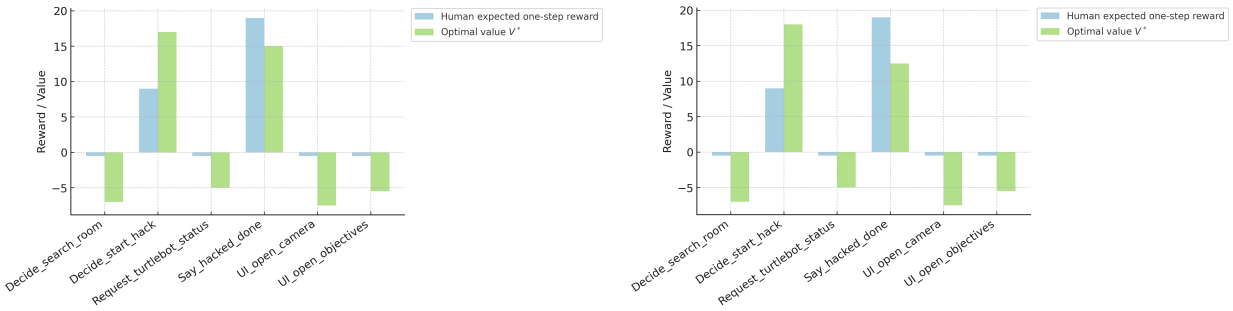

Figure 6.3: Computer Hack (CH) task. Observed one-step reward (blue) and optimal MDP value (green) by action, split by pre-bias group (left: low pre-bias, right: high pre-bias).

Access Code: concentrated and near-optimal behavior

The Access Code (AC) task is very limited. The only choice for the participant is which attempt to use next on the keypad. In both pre-bias groups, most of the value comes from the first two attempts. The actions Enter_code_attempt_1 and Enter_code_attempt_2, the participant's first and second attempts to input the code, both show clear positive results. The observed rewards for these attempts are close to the optimal values (the best possible outcomes). The third attempt is much less appealing. Both the actual reward and V^* (the optimal predicted value for that attempt) are close to zero or slightly negative, which shows the extra penalty for using all three attempts before either succeeding or running out of time. Comparing the panels shows that the high-pre-bias group (flag = 1) gets a higher reward per step in AC, and its blue bars are closer to the green optimal bars. This group tends to use more steps on early attempts, so they reach the low-value third attempt less often. This pattern explains why the gap between human and optimal performance for AC is smaller in this group.

Computer Hack: exploratory versus progress actions

The CH task features both exploratory and progress actions. As Figure 6.3 shows, actions like `Decide_start_hack`, `Request_turtlebot_status`, and `Say_hacked_done` have high optimal values. Human choices also yield positive rewards, but their values are consistently below optimal levels. This means participants use these effective actions less than the optimal policy. Interface actions such as `UI_open_camera`, `UI_open_objectives`, and `Decide_search_room` have negative optimal values, as they take time without improving the chances of success in hacking. Yet, participants use them often, keeping observed rewards near zero or slightly negative. This gap shows the optimal policy quickly commits to the hack, avoiding low-value exploratory actions. Both pre-bias groups behave similarly. Bar shapes and action-value order differ only slightly, indicating that prior AI trust has less effect on managing the hack sequence than on access-code attempts.

Cross-task summary by pre-bias group

Figure 6.4 provides a consolidated overview of the task-level MDP results, directly supporting the following analysis. It displays the average for each combination of task (CH or AC) and pre-bias group. This figure illustrates the average reward or value per step for both the observed trajectories (blue) and the optimal task-local policy (green), which are central to the analysis presented below. Only participants included in the mission-level analysis are considered. Table 6.5 shows the relevant numerical values. For Computer Hack, both the low- and high-pre-bias groups have almost identical average rewards per step. (7.33 compared to 7.28), while the optimal task-local policy reaches values around 18.3 to 18.5 in both groups. This results in a large gap of about 11 units per step between observed and optimal performance, no matter the pre-bias, mainly because of the time spent on low-value interface actions. By contrast, results for Access Code differ. The high-pre-bias group achieves a

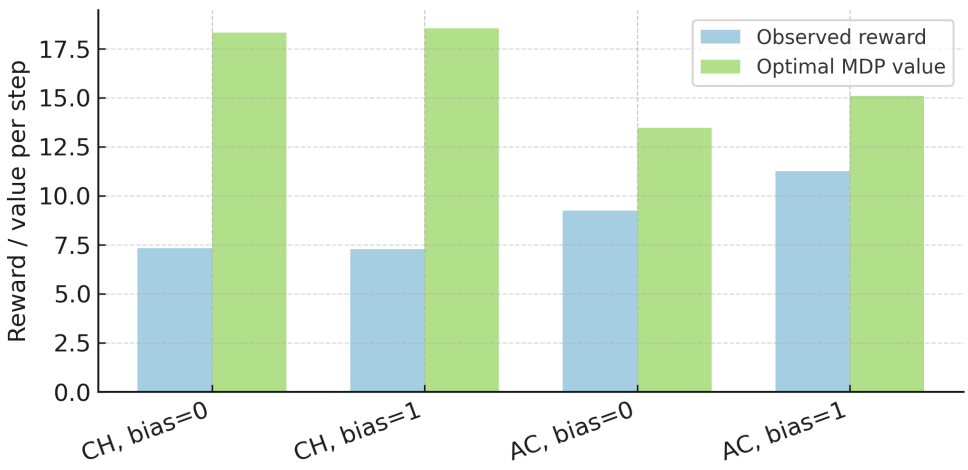

Figure 6.4: Mean reward/value per step by task and pre-bias group. Blue bars show observed rewards from human trajectories; green bars show the corresponding optimal MDP values under the same reward model.

Task	Pre-bias group	Observed reward	Optimal MDP value	Gap (optimal − observed)
Computer Hack (CH)	Low (flag = 0)	7.33	18.31	10.98
Computer Hack (CH)	High (flag = 1)	7.28	18.53	11.25
Access Code (AC)	Low (flag = 0)	9.25	13.45	4.20
Access Code (AC)	High (flag = 1)	11.26	15.09	3.83

Table 6.5: Mean reward/value per step by task and pre-bias group (restricted participant set).

clearly higher observed reward per step (11.26 compared to 9.25 for the low-pre-bias group), with corresponding optimal values also slightly higher (15.09 compared to 13.45). The gap between observed and optimal performance is smaller overall in AC (about 4.2 units for the low-pre-bias group and 3.8 units for the high-pre-bias group) than in CH, and it is slightly smaller for the high-pre-bias participants. In other words, once participants reach the keypad phase, both groups act fairly efficiently, with a small advantage for those who started the mission more willing to work with the AI partner.

Action-level interpretation by pre-bias group

The task-level MDPs reveal situations where this occurs. The model's recommendations change based on how much trust participants had in AI beforehand. In the compact CH and AC models used in this study, each decision context is described using the same set of human-visible actions, such as `Decide_start_hack` and `UI_open_camera`, `Enter_code_attempt_1`. The bar plots in Figures 6.2 to 6.3 show the MDP's recommendations. Actions with a higher optimal value V^* are those that the task-level policy will select that action whenever the situation arises. The order of action values stays constant across bias groups.

In both CH and AC, the same actions rank consistently as high- or low-value across low- and high-pre-bias models: In AC, `Enter_code_attempt_1` has the highest value, followed by `Enter_code_attempt_2`. `Enter_code_attempt_3` always has a lower or negative value, and the MDP never selects late attempts over early ones. In CH, progress actions such as `Decide_start_hack`, `Request_turtlebot_status`, and `Say_hacked_done` always have a high value. Interface-heavy actions like `UI_open_camera`, `UI_open_objectives`, and extended `Decide_search_room` always have low value. In other words, within these task-local models, there is no situation where the best action according to the MDP changes from a progress action in one pre-bias group to a non-progress action in the other. The groups differ only in how often they actually use the high- and low-value actions. Table 6.6 summarizes observed action frequencies in CH and AC, focusing on the participant set used in the MDP analyses. Percentages reflect all labelled steps for each task and bias group.

In the CH task, both pre-bias groups spend a similar portion of their steps on the main progress actions. These actions, such as `Decide_start_hack` and `Say_hacked_done`, are considered main progress actions in the MDP and contribute directly to task value, while both groups also engage in other, lower-value activities. Both groups spend significant time on lower-value interface actions, which reduces the total task value accumulated, explaining the large, similar gap between the observed and optimal values shown in Table 6.5. This

Task	Pre-bias group	Action	Share of steps (%)
CH	Low (flag = 0)	Decide_start_hack	35.4
CH	Low (flag = 0)	Say_hacked_done	24.0
CH	Low (flag = 0)	UI_open_camera	20.2
CH	Low (flag = 0)	UI_open_objectives	10.2
CH	Low (flag = 0)	Request_turtlebot_status	9.4
CH	Low (flag = 0)	Decide_search_room	0.8
CH	High (flag = 1)	Decide_start_hack	37.8
CH	High (flag = 1)	Say_hacked_done	22.5
CH	High (flag = 1)	UI_open_camera	18.1
CH	High (flag = 1)	Request_turtlebot_status	12.0
CH	High (flag = 1)	UI_open_objectives	9.0
CH	High (flag = 1)	Decide_search_room	0.6
AC	Low (flag = 0)	Enter_code_attempt_1	79.3
AC	Low (flag = 0)	Enter_code_attempt_2	15.7
AC	Low (flag = 0)	Enter_code_attempt_3	5.1
AC	High (flag = 1)	Enter_code_attempt_1	87.8
AC	High (flag = 1)	Enter_code_attempt_2	10.2
AC	High (flag = 1)	Enter_code_attempt_3	2.0

Table 6.6: Human action frequencies in Computer Hack (CH) and Access Code (AC) by pre-bias group.

substantial, consistent gap, as highlighted in Table 6.5, provides context for understanding the similarities in the behaviors of both groups. The MDP prioritizes actions that lead to starting and finishing the hack efficiently for higher cumulative value, whereas people in both groups spend extra time on camera and objective views, which have lower value according to the MDP. Both groups keep spending time looking at the camera and objectives.

Turning to the AC task, the frequency of actions shows a clearer difference related to trust. Both groups mostly focus on their first attempts. However, participants with high pre-bias spend a larger share of their steps on `Enter_code_attempt_1` (87.8% vs. 79.3%) and rely less on `Enter_code_attempt_3` (2.0% vs. 5.1%). High-pre-bias participants focus more on `Enter_code_attempt_1` (87.8% vs. 79.3%) and less on `Enter_code_attempt_3` (2.0% vs. 5.1%). This pattern aligns with the MDP's action-value recommendations, in which early

Metric	Mean	SD	Median	n
Expected return	2802.70	353.59	2466.79	86
Success rate	0.34%	0.37%	0.00%	86
On-time success	0.16%	0.18%	0.00%	86
Timeout rate	0.16%	0.18%	0.00%	86
Other failure rate	99.50%	0.53%	100.00%	86
Median steps	300.00	0.00	300.00	86

Table 6.7: Participant-level outcomes under the learned policy. These rollout outcomes reflect strict on-time gating and a conservative step cap, thresholds and sensitivity are revisited in the discussion.

attempts are assigned higher values and third attempts are assigned lower values, guiding the observed participant's behavior. This matches the higher observed reward and the slightly smaller gap between human and optimal performance for the high-pre-bias group in AC. This is in line with the higher observed reward and the slightly smaller gap between human and optimal results for the high-pre-bias group in AC.

In this way, prior trust does not alter which actions the MDP considers to have higher or lower value, but it does influence how closely participants align their actions with these value assignments when prior trust is critical. Prior trust influences how strongly people follow the MDP's advice in tasks where trust matters most. This effect is most noticeable in tasks that require prior trust.

6.6 Policy Evaluation

Rolling out the learned mission-level policy from each participant's start state yields outcomes summarized in Table 6.7. Success is defined by the expected return. This measure accounts for penalties at each step and rewards at the end. Penalties apply for each step, and rewards are given at the terminal.

Chapter 7

Conclusion

This study linked a trust-aware human MDP approach to task-level measures taken directly from configuration logs, such as `goalstatus`, `actual time`, and `expected time`. In the preregistered participant group, we found that both success and on-time completion depended on the specific task, and higher values in the `Trust` field were associated with greater success. These results suggest that spending extra time searching or clarifying can lead to late or failed outcomes on harder tasks, which supports the use of assistance policies that account for timing and context.

7.1 Task-level MDP conclusions for Computer Hack and Access Code

I built task-level Markov Decision Processes (MDPs) for Computer Hack (CH) and Access Code (AC) using only human user actions, clarifying how trust affects behavior. (An MDP models sequential decisions by assigning expected values to actions at each step.) In AC, MDP values favor early code attempts for all groups, with a low value for the third attempt. High-pre-bias participants (those who initially trusted the system more) follow MDP recom-

mendations more closely: they attempt early, avoid third attempts, and achieve higher mean rewards per step (where a "step" is a single user decision) than low-pre-bias participants. Both groups remain nearly optimal. In CH, progress actions such as starting and completing the hack have high value, while repeated camera or objective checks have low value for both groups. Yet, both groups still spend time on these lower-value exploratory actions, reducing efficiency. Prior trust does not change which actions the MDP rates as good or bad, instead, it affects how efficiently high-value actions are used, especially in Access Code.

7.2 Limitations

- Source precedence. Outcomes and timing were taken from configuration files, free-text logs served only as context. Episodes without configuration timing were excluded from on-time analysis.

7.3 Future Work

- Robot trust model integration. Integrate the human behavioral model into the existing robot trust model, so that assistance decisions reflect both task state and inferred human bias.

- Trust-aware assistance. Personalize when to suggest, when to stay silent, and when to ask for confirmation based on early signs of search overhead and predicted risk of missing the time target, while keeping guidance brief and easy to ignore.

Bibliography

Ali, A. M., Notarstefano, G., Cortez, J., & Bullo, F. (2022). Adaptive inter-robot trust for robust multi-robot sensor coverage. *IEEE Transactions on Robotics*, *38*(3), 1740–1751.

Almeida, T. S., & Melo, F. S. (2022). A framework for endowing robots with reasoning capabilities about perspective-taking and belief management. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Azevedo-Sa, T., Melo, F. S., & Paiva, A. (2021). Unified model for artificial and natural trust in human–robot collaboration. *Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3342–3349.

Bhat, S. Y., Pennisi, G., Murphy, R., Martini, D. D., & Tucker, C. S. (2023). Adaptive human preference-based reinforcement learning for trustworthy hri. *IEEE International Conference on Robotics and Automation (ICRA)*, 12678–12684.

Breazeal, C., Edsinger, A., Fitzpatrick, P., Scassellati, B., & MacLean, M. V. (2004). Social robots for health applications (and social learning). *AAAI Spring Symposium: Accessible, Interoperable HRI*.

de Visser, E. J., & Chen, J. Y. C. (2020). Longitudinal study of trust calibration and learning in human–robot teams. *Human Factors*, *62*(2), 266–286.

Dudík, M., Langford, J., & Li, L. (2011). Doubly robust policy evaluation and learning. *Proceedings of the International Conference on Machine Learning (ICML)*, 1097–1104.

Dzindolet, M. T., Peterson, S. A., Pomranky, R. A., Pierce, L. G., & Beck, H. P. (2003). The role of trust in automation reliance. *International Journal of Human-Computer Studies*, *58*(6), 697–718.

Hancock, P. A., Billings, D. R., Schaefer, K. E., Chen, J. Y. C., de Visser, E. J., & Parasuraman, R. (2011). A meta-analysis of factors affecting trust in human-robot interaction. *Human Factors*, *53*(5), 517–527.

Jiang, N., & Li, L. (2016). Doubly robust off-policy value evaluation for reinforcement learning. *Proceedings of the International Conference on Machine Learning (ICML)*, 652–661.

Kok, J., Vandic, D., Knigge, A., & Kröse, B. (2021). Robots with theory of mind for humans: Trust estimation and anticipation of trust breakdown. *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, 119–128.

Lee, J. D., & See, K. A. (2004). Trust in automation: Designing for appropriate reliance. *Human Factors*, *46*(1), 50–80.

Nare, B., Frericks, J., Challa, A., Doshi, P., & Johnsen, K. (2025). A novel computational framework of robot trust for human–robot teams [IEEE Xplore Document 11127674]. *Proceedings of the 2025 IEEE International Conference on Robotics and Automation (ICRA)*, 8140–8146. https://ieeexplore.ieee.org/document/11127674

Nikolaidis, S., Hsu, D., & Shah, J. A. (2015). Improved human–robot team performance through cross-training. *The International Journal of Robotics Research*, *34*(14), 1658–1674.

Nikolaidis, S., & Shah, J. A. (2013). Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 33–40.

Pandey, A. M., & Alami, R. (2019). Proactive perspective-taking and intention inference for human-robot collaboration. *Robotics and Autonomous Systems*, *120*, 103243.

Parasuraman, R., & Riley, V. (1997). Humans and automation: Use, misuse, disuse, abuse. *Human Factors*, *39*(2), 230–253.

Robinette, P., Li, W., Allen, R., Howard, A. M., & Wagner, A. R. (2016). Overtrust of robots in emergency evacuation scenarios. *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 101–108.

Schaefer, K. E., Chen, J. Y. C., Szalma, J. L., & Hancock, P. A. (2016). A meta-analysis of factors influencing the development of trust in automation. *Human Factors*, *58*(3), 377–400.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

Thomas, P. S., & Brunskill, E. (2016). Data-efficient off-policy policy evaluation for reinforcement learning. *ICML*.

Appendix A

Per Task Results: Full Tables

The main results chapter highlights key patterns. To examine these patterns in detail for each task in the preregistered mission subset (n = 253), this appendix presents tables that report values for est_time_*, act_time_*, and binary *_complete flags across all tasks. For each task, I include:

- n: number of mission records in the preregistered subset (always 253),
- *Success rate*: proportion of records with the task's `*_complete` flag equal to 1 (computed over all 253 records),
- *On–time rate*: proportion of records with positive expected and actual times where `act_time` $\leq$ `est_time`,
- *Mean* and *median* actual duration in minutes, computed over records with positive `act_time` for that task.

Table A.1 summarises outcomes for the main mission tasks using this definition, providing an overview before further discussion of results.

Clear patterns emerge from these results. Early tasks Find Computer, Clues Search, Computer Hack, and Access Code are nearly always completed successfully, with success

Task code	Task name	n	Success rate	On–time rate	Mean duration	Median duration
FC	Find Computer/ Computer Search	253	100.0%	54.1%	2.65	2.32
CS	Clues Search	253	100.0%	1.6%	7.85	8.67
CH	Computer Hack	253	100.0%	61.9%	1.68	0.95
AC	Access Code	253	100.0%	27.6%	1.25	1.01
MZ	Maze	253	45.8%	24.5%	5.41	6.15
SS	Shutdown Server	253	45.8%	85.6%	1.05	0.78

Table A.1: Per–task outcomes for the preregistered mission subset ($n = 253$).

rates near 100%. Timing sets these apart: Find Computer and Computer Hack tend to be short and are often finished before deadlines, with about half or more valid episodes on time. In contrast, despite universal completion, Clues Search has a much lower on-time rate, suggesting that the time spent clarifying clues can push later tasks closer to or past their deadlines.

Access Code, similarly, has a 100% success rate but only a 27.6% on-time rate, even though durations are short. This task's simplicity, timing pressure, and the opportunity for multiple attempts make it a strong candidate for focused task-level MDP analysis.

Maze and Shutdown Server differ from earlier tasks in key ways. Only about 46% of missions complete these phases successfully. Maze episodes are relatively long (mean: 5.4 min, median: 6.2 min), and only a quarter finish on time. By contrast, among those who reach it, Shutdown Server shows a high on-time rate (85.6%). Although the Shutdown Server usually takes about one minute, it is often not reached, which lowers overall success. These per-task statistics lead directly to the mission-level MDP checks and focused Computer Hack

and Access Code analyses in Chapter 6. The full table below shows how the model's findings relate to the timing and success patterns observed in the data.

Appendix B

Intent actions from log parsing

The main chapters explain how raw simulator events are transformed into intent-aware actions such as request, accept, ignore, restart, commit, and timeout. This appendix provides more detail, including examples that show how original log strings match intended actions in task-level MDPs.

B.1 From raw log lines to normalised labels

Each simulator event is saved as a line in a CSV-style log, with a timestamp and mission fields. Each line also has an action label that begins with an actor token. Here is an example format: Before applying any rules, each label goes through a brief normalisation step:

1. The actor prefix (`HUMAN_`, `ROBOT_`, `SYSTEM_`) is removed so that the mapping focuses on the event itself rather than who generated it.

2. The remaining text is converted to lower case and extra underscores or spaces are collapsed.

3. Task and mission identifiers are kept as separate fields, but are not part of the textual label that the rules operate on.

Applied to the three lines above, this produces:

```
ui_open_objectives
request_clue_for_code
entered_wrong_code
```

These normalised labels are the starting point for mapping intent actions.

B.2 Using rules to infer intent

Chapter 4 lists the rule set that turns normalised labels into intent Here, I focus on how these rules work in practice.

Normalized label	Intent action
`request_clue_for_code`	`request`
`ask_robot_for_help`	`request`
`apply_suggestion_for_maze`	`accept`
`proceed_without_suggestion`	`ignore`
`restart_maze_from_start`	`restart`
`submit_access_code`	`commit`
`mission_timeout`	`timeout`

Table B.1: Examples of mapping normalised log labels to intent actions.

Labels that mention `clue`, `hint`, or asking the robot are treated as `request`; labels that talk about using or applying a suggestion become `accept`. Phrases that explicitly reject help, such as proceeding without a suggestion, map to `ignore`. Any restart or reset action is coded as `restart`. Final submissions, including correct code entries, map to `commit`, and time-outs or aborts map to `timeout`. If a label does not match any of the rules, it is assigned to `ignore`

and marked with an "unknown" flag. This keeps the action set small and stable, while still keeping track of where the text was too unusual to classify confidently.

B.3 From intent to task-specific actions

The intended actions themselves do not depend on the specific mission task. For task-level MDPs, they are combined with task information to form the concrete action sets used in the Computer Hack (CH) and Access Code (AC) models. For Computer Hack, combinations of intent and UI events generate actions such as Decide_search_room, Decide_start_hack, Request_turtlebot_status, Say_hacked_done, UI_open_camera, and UI_open_objectives, which comprise the available choices in the CH task-level MDP and in the reward comparison plots in Chapter 6. For Access Code, code-entry commits are grouped into Enter_code_attempt_1, Enter_code_attempt_2, and Enter_code_attempt_3, with a separate terminal state for a correct code. These serve as the actions used in the AC task-level MDP. Because the mapping uses fixed rules, any MDP action can be traced to its original log string. This enables interpretation of model recommendations in plain language and supports analysis of how behaviour differs between low- and high-pre-bias groups, linking these differences to observable interaction patterns.

Appendix C

Top-level decision tree

Figure C.1 shows the upper part of the next–phase decision tree for episodes that are currently in the Maze portion of the mission. In this slice of the the model, predicts three possible next phases: staying in the Maze, moving on to Shutdown, or reaching a Terminal state. Each node in the plot reports:

- **samples**: the number of Maze episodes that reach that node,
- **value**: the class counts in the order {Maze, Shutdown, Terminal},
- **class**: the majority (predicted) next phase at that node.

Only a small set of task identifiers appears in this tree:

- `task_id = 10`: Restart Maze,
- `task_id = 11-13`: Restart Maze attempts,
- `task_id = 14`: OTNM (out of time navigating the maze).

At the root, all Maze episodes are pooled together. The first split is on `task_id`. Episodes with `task_id` ≤ 10 (Restart Maze) go to the left child and are mostly labelled as Terminal,

which matches the idea that restart events often occur near the end of an attempt. Episodes with higher `task_id` values go to the right, where the next split is on the previous pace (`pace_prev_int`). When the previous pace is non-positive (slower progress), the model predicts a transition to Shutdown, when the pace is higher, the most likely outcome is to continue in Maze for at least one more step.

An important detail is what does *not* appear at this depth: the pre-bias flag. Consistent with its low overall feature importance, `bias_flag` only shows up deeper in the tree. In this phase, mission structure (which restarts/OTNM task the participant is on) and recent pace carry most of the predictive power, while prior trust plays at most a secondary role.

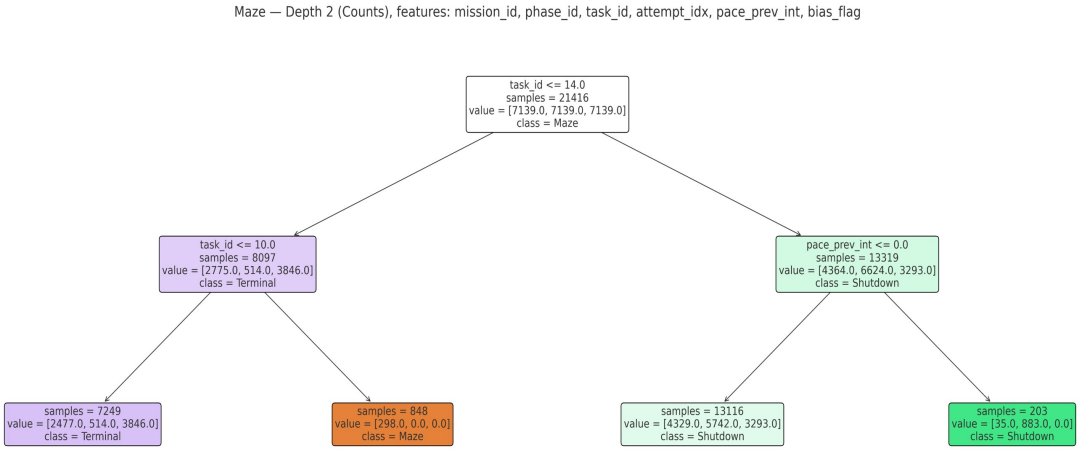

Figure C.1: Top-level structure of the next–phase decision tree for Maze episodes. Internal nodes show the splitting feature and threshold, leaves report the number of samples, class counts `value = [Maze, Shutdown, Terminal]`, and the majority class.

Appendix D

MDP training, Optimal policy evaluation

This appendix is a practical companion to the MDP work in the main chapters. Rather than redefining states, actions, rewards, or value iteration, it focuses on what actually happens to the data: how the logs are turned into transition tables, how the solver is used, and how I checked that the resulting policies behave in a reasonable way.

D.1 From cleaned logs to transition tables

The starting point is the cleaned, intent-labelled dataset used throughout the thesis. Each row already encodes which participant it belongs to, which mission and task it is part of, and the abstract state and action at that moment.

From there, the training pipeline runs roughly as follows:

1. **Organize episodes.** The data are grouped into episodes so that each row knows its predecessor and successor in time. This makes it easy to see, for every step, where the participant came from and where they went next.

2. **Split by task and bias.** Episodes are then divided into four logical buckets: Computer Hack (CH) with low bias, CH with high bias, Access Code (AC) with low bias, and AC

with high bias. This keeps the structure of the model consistent with the bias-aware analysis in the results.

3. **Apply the tree-based abstraction.** The decision tree described in the main chapters is fitted once and used to map detailed step information into a smaller set of abstract states. This produces a compact representation that can be used by the MDP, while still reflecting meaningful differences in behavior.

4. **Count state-to-state moves.** For each bucket (CH/AC $\times$ low/high bias), the code walks through the episodes and counts how often a particular abstract state is followed by another abstract state under a given intent label. These counts form the raw material for the transition model.

5. **Normalise to probabilities.** Finally, the counts for each "from" state (and, where needed, action) are normalised so they sum to one. A small amount of smoothing is applied so that rare but plausible moves are not given probability zero. The result is a set of empirical transition probabilities that capture how people in that group actually moved through the task.

At this point, there are four transition models, one for each task and bias group, ready to be combined with the reward structure and solved.

D.2 Decision tree diagnostics

This section presents some additional diagnostics for the next-phase decision tree. These diagnostics relate to the tree described in Section 6.3.

Figure D.1 shows that most values are on the diagonal. This means the tree usually predicts the correct next phase. Errors are concentrated among phases with similar behavior

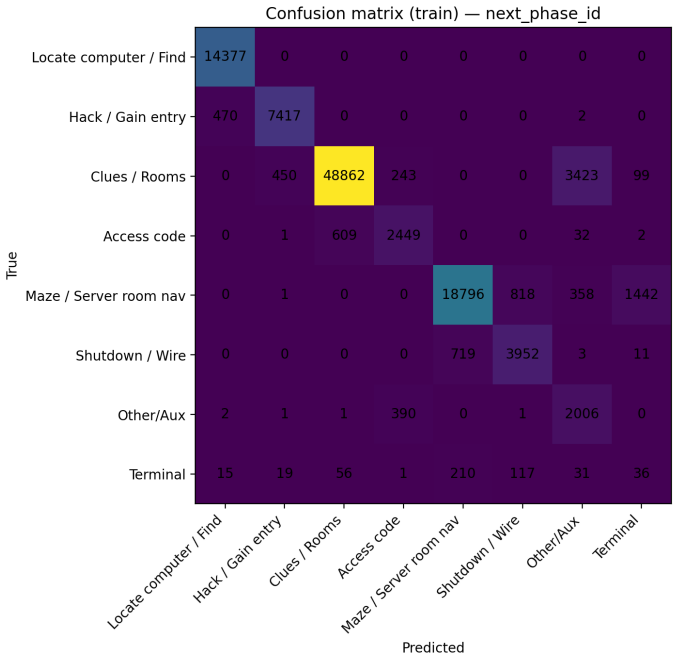

Figure D.1: Confusion matrix on the train split for next-phase prediction. Rows are true phases, and columns are predicted phases. The "Other/Aux" task type is internally expanded into specific critical tasks in the logic used to build the human MDP

or adjacent in the mission flow. This matches the reported performance: train accuracy is 91.13

Table D.1 shows that the current phase and task are the main factors in most splits. Recent interaction pace also contributes. Mission and bias play a minor role in the tree itself. They are mainly used for grouping in the MDP.

Table D.2 confirms that "Clues / Rooms" and “Maze / "Server room nav" dominate the data. Phases such as "Access code", "Shutdown / Wire", "Other/Aux", and the explicit terminal class are much less common. Because of this imbalance, it is important to report macro-F1 as well as accuracy.

Feature	Importance
phase_id	0.541
task_id	0.322
pace_prev_int	0.132
mission_id	0.003
bias_flag	0.002

Table D.1: Top decision-tree feature importances.

Phase	Rows	Percent
Locate computer / Find	14377	13.38%
Hack / Gain entry	7889	7.34%
Clues / Rooms	53077	49.41%
Access code	3093	2.88%
Maze / Server room nav	21415	19.94%
Shutdown / Wire	4685	4.36%
Other/Aux	2401	2.24%
Terminal	485	0.45%

Table D.2: Distribution of next-phase targets used for tree training.

D.3 Human MDP: transitions and rewards

The mission-level human MDP includes 38 different state-action pairs. The median branching factor is 2.00, and the mean is 2.11. This is the number of possible next states for each state-action pair. So, most decisions lead to only a small number of realistic next situations.

For all state-action pairs, the average immediate reward is 1.07, and the median is -1.00. This shows there are many small negative step costs and fewer, larger positive terminal rewards. As a result, long, looping behavior is scored lower than steady progress toward success. This is the intended outcome.

Table D.3 shows that all 96 transition rows are estimated directly at the leaf level of the decision tree. This means there is no need to use broader phase-level or global pools. As a

Level	Count	Share
leaf	96	100.00%

Table D.3: Backoff provenance for transition rows.

result, the policies and rollouts are based on transitions that are directly supported by the observed data.

D.4 Example transition patterns

Transition tables can be hard to picture in the abstract, so this section gives a few representative rows from the fitted models. Each row lists:

- the task (AC or CH),
- the bias group (low or high),
- the abstract "from" state,
- the abstract "to" state,
- how many times that move appeared in the data (Count),
- and the estimated probability of that move in the learned model.

For the Access Code task, the examples highlight how episodes move from the start of the task into a first code attempt, and then either succeed or continue to a second code attempt.

The overall pattern is similar in both bias groups: almost everyone begins with a code entry, and most episodes succeed on that first attempt. The small differences in probabilities

Task	Bias group	From state	To state	Count	$\hat{P}(\text{to} \mid \text{from})$
AC	Low	__START__	Enter_code_attempt_1	196	1.000
AC	Low	__AC_success__	__END__	186	1.000
AC	Low	Enter_code_attempt_1	__AC_success__	153	0.781
AC	Low	Enter_code_attempt_1	Enter_code_attempt_2	42	0.214
AC	High	__START__	Enter_code_attempt_1	195	1.000
AC	High	__AC_success__	__END__	192	1.000
AC	High	Enter_code_attempt_1	__AC_success__	171	0.877
AC	High	Enter_code_attempt_1	Enter_code_attempt_2	24	0.123

Table D.4: Example transitions for the Access Code (AC) task by bias group.

(for example, how often a second attempt is needed) are what feed into the bias-aware comparisons in the main results.

For the Computer Hack task, the most common transitions involve camera use and search decisions. Participants spend many steps in these states before eventually finishing the episode.

Task	Bias group	From state	To state	Count	$\hat{P}(\text{to} \mid \text{from})$
CH	Low	UI_open_camera	UI_open_camera	1778	0.838
CH	Low	Decide_search_room	Decide_search_room	523	0.611
CH	Low	Decide_search_room	UI_open_camera	179	0.209
CH	Low	UI_open_camera	__END__	136	0.064
CH	High	UI_open_camera	UI_open_camera	2327	0.852
CH	High	Decide_search_room	Decide_search_room	533	0.583
CH	High	Decide_search_room	UI_open_camera	199	0.218
CH	High	UI_open_camera	__END__	156	0.057

Table D.5: Example transitions for the Computer Hack (CH) task by bias group.

These rows show that remaining in the camera state for several steps is very common, either by staying in that decision state or by opening the camera again. The transition into an end state is much less frequent than these looping moves, which makes the Computer Hack task involve more searching and back-and-forth.

D.5 Solving the models and checking the policies

Once the transition tables are in place, the actual solving step is straightforward: For each of the four models (CH/AC by low/high bias), the implementation combines the transition probabilities with the reward structure and runs the value-iteration procedure described in the main text. This produces, for every abstract state:

- a value (how good that state looks under the model), and
- a recommended action (the action with the highest expected return).

The more interesting part is what happens next. To understand how these policies behave and whether they are sensible, I:

- **Roll out simulated episodes.** Starting from the same kinds of initial states that participants had, the policy is used to choose actions, and the learned transition model is used to sample next states. From these simulated runs, I record success rates, episode lengths, and average returns.
- **Spot-check specific states.** For a handful of states that are easy to interpret (for example, states that clearly represent "very late" in the task), I manually inspect the recommended action and verify that it lines up with the reward structure and with common sense.
- **Compare to human choices.** Without going into detail here, the human-versus-policy comparisons in the results chapter are based on these same solved models: for each recorded state, I can see whether the person took the same action that the policy would have suggested from that point.

These checks are not meant to prove that the model is perfect, but they help build confidence that it is at least consistent with the data and with the design goals of the reward

function. The example transitions in this appendix are included for the same reason: they give a concrete view of what the MDP is built from and how familiar behaviors such as repeated camera movements or code attempts appear inside the learned model.

- The implementation of bias-aware human MDP can be found at https://github.com/challaanusha46/Bias-Human-MDP