binaryRL: Reinforcement Learning Modeling of Two-Alternative Forced Choice Decision Making in R — A Step-by-Step Tutorial

Mengzhen Hu[1#], Zheng Liu[1#], X.T. (XiaoTian) Wang[1*], Ran Li[2]

[1*] Division of Applied Psychology, The Chinese University of Hong Kong (Shenzhen), Shenzhen, China.

[2*] Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong SAR, China.

[*]Corresponding author. Email: xtwang@cuhk.edu.cn

[#] These authors contributed equally to this work.

Author Note

The author wishes to express gratitude to Dr. Wei Guo, Dr. Guangyu Zhu and Ms. Xinyu Wang for their valuable feedback on early versions of this work.

Abstract

The reinforcement learning framework offers valuable insights into people's decision-making processes in Two-Alternative Forced Choice (TAFC) tasks, providing a deeper understanding of how individuals adjust their choices in response to feedback. However, tools to build, simulate, and compare reinforcement learning models tailored for such tasks remain limited. To address this gap, we developed binaryRL, an R package designed to support the implementation and evaluation of reinforcement learning (RL) models in the context of TAFC tasks, following best-practice guidelines for model comparison in cognitive science. Using real-world open data, we present a tutorial that demonstrate the comprehensive modelling pipeline—from raw experimental input to the construction, parameterization, and validation of RL models. Our package offers a flexible and accessible platform for advancing both theoretical and empirical research in reinforcement learning and cognitive modeling of feedback-based decision-making.

Translational Abstract

To address the challenges and reduce barriers to the use of reinforcement learning models in psychology, we developed and introduced the R package *binaryRL*, a user-friendly toolkit designed to facilitate the development, evaluation, and comparison of reinforcement learning models in the context of feedback-based binary decision making. *binaryRL* adheres to the four-step cognitive model construction process outlined by Wilson & Collins (2019) and the best practice guidelines for model comparison in cognitive science (Palminteri et al., 2017), enabling researchers to rigorously test and refine their RL models with minimal technical complexity.

Keywords: Reinforcement learning, cognitive modelling, two-alternative forced choice task, expected value, R package

Understanding how humans make decisions is a central goal in psychological science, yet directly studying the mechanisms behind decision-making processes is inherently challenging (Kantowitz et al., 2009). As a result, researchers often rely on behavioral experiments to infer cognitive processes from observable behavior. While these approaches facilitate hypothesis testing, they typically offer descriptive insights and fall short of capturing the dynamic, latent mechanisms that drive adaptive behavior.

With the rise of computational approaches, cognitive modeling has become a crucial tool for bridging the gap between observable behavior and latent cognitive mechanisms (Farrell & Lewandowsky, 2010). By constructing formal models that simulate human cognition, researchers can generate testable predictions and iteratively refine theoretical frameworks based on empirical data (Bogacz et al., 2006; Krajbich & Rangel, 2011; Ratcliff & Smith, 2004). Among these, **reinforcement learning** (RL) has emerged as a powerful framework for modeling how agents learn from feedback and update value expectations over time (Lee et al., 2012; Subramanian et al., 2022; Zhang et al., 2020). RL models are particularly well-suited for value-based decision-making tasks, where behavior unfolds over repeated choices and is shaped by cumulative outcomes.

Among the most common paradigms in value-based decision-making are **Two-Alternative Forced Choice** (TAFC) tasks, in which participants repeatedly choose between two options associated with varying magnitudes of reward and their probabilities (e.g., Botvinick et al., 2019; Robbins, 1952). These tasks provide structured environments ideal for trial-by-trial modeling of learning and decision-making (Botvinick et al., 2019).

Despite the increasing use of reinforcement learning in psychological research (Eckstein et al., 2021), its broader adoption remains limited due to two key challenges. First, RL is conceptually and computationally complex, involving multiple interrelated components—such as agents, states, actions, rewards, and policies—that require familiarity with advanced mathematical concepts (e.g., Markov Decision Processes) and computational tools. For psychologists without formal training in these areas, the steep learning curve can be a significant barrier. Second, machine learning applications (e.g., computer vision, natural language processing, and control systems)—such as OpenAI Gym (Brockman et al., 2016), Deepmind Acme (Hoffman et al., 2022), and Tsinghua Tianshou (Weng et al., 2022) offer limited support for the task paradigms and experimental workflows common in psychological research, like paradigms focused on understanding human decision-making behaviors and workflows involving live interaction between human participants. Using these tools often demands substantial customization, which can be time-consuming and error-prone, especially for researchers with limited coding expertise. Similarly, while R packages developed for psychological applications, such as ReinforcementLearning (Proellochs & Feuerriegel, 2017), hBayesDM (Ahn et al., 2017), and twochoiceRL (Suthaharan et al., 2021)—which align more closely with psychological tasks —still have significant limitations. These packages often lack the flexibility needed for users to customize or extend the underlying reinforcement learning models, with some only supporting basic model architectures.

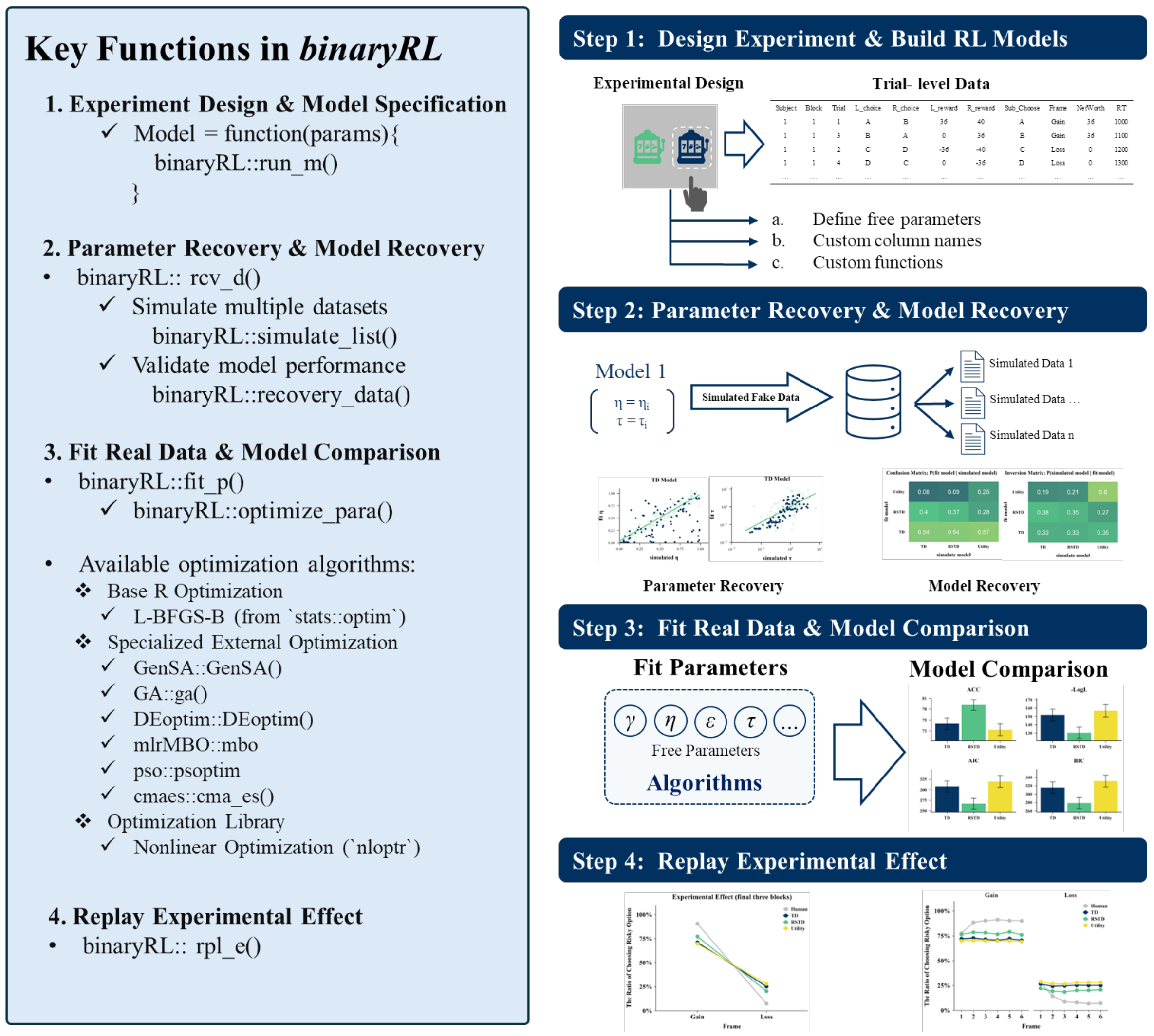
This gap highlights the need for accessible, domain-specific tools that support rigorous RL modeling in psychology.

To address these challenges and reduce barriers to the use of reinforcement learning models in psychology, we introduce *binaryRL*, an R package designed to facilitate the construction, estimation, and comparison of RL models in the context of TAFC tasks (see Figure 1). Built with psychologists in mind, *binaryRL* integrates best practices in computational modeling (Palminteri et al., 2017; Wilson & Collins, 2019) into a streamlined and user-friendly framework.

This article provides a step-by-step tutorial for constructing reinforcement learning models, following the four-stage framework outlined by Wilson & Collins (2019). We begin by introducing two-alternative forced-choice tasks and the RL models commonly used to analyze them. We then walk through the full modeling pipeline— (1) building reinforcement learning models, (2) conducting parameter recovery and model validation, (3) optimizing parameters for real data and conducting model comparisons, and finally, (4) replaying the experiment for latent variable analysis —accompanied by annotated R code. To enhance accessibility, especially for researchers new to RL or cognitive modeling, the tutorial starts from raw behavioral data. We illustrate the workflow using open data from the Gambling Paradigm (Mason et al., 2024), a representative TAFC task in the study of risky decision-making, though the approach generalizes to a broad range of TAFC designs (e.g., Cools et al., 2002; Ellsberg, 1961; Hertwig et al., 2004; Ludvig & Spetch, 2011; Robbins, 1952; Yechiam & Busemeyer, 2006).

Figure 1

The binaryRL Workflow: A Step-by-step Modelling Pipeline



Key Functions in *binaryRL*

1. Experiment Design & Model Specification
- ✓ Model = function(params){
  binaryRL::run_m()
  }

2. Parameter Recovery & Model Recovery
- binaryRL:: rcv_d()
  - ✓ Simulate multiple datasets
    binaryRL::simulate_list()
  - ✓ Validate model performance
    binaryRL::recovery_data()

3. Fit Real Data & Model Comparison
- binaryRL::fit_p()
  - ✓ binaryRL::optimize_para()

- Available optimization algorithms:
  - ❖ Base R Optimization
    - ✓ L-BFGS-B (from `stats::optim`)
  - ❖ Specialized External Optimization
    - ✓ GenSA::GenSA()
    - ✓ GA::ga()
    - ✓ DEoptim::DEoptim()
    - ✓ mlrMBO::mbo
    - ✓ pso::psoptim
    - ✓ cmaes::cma_es()
  - ❖ Optimization Library
    - ✓ Nonlinear Optimization (`nloptr`)

4. Replay Experimental Effect
- binaryRL:: rpl_e()

Step 1: Design Experiment & Build RL Models

Experimental Design

Trial- level Data

Subject	Block	Trial	L_choice	R_choice	L_reward	R_reward	Sub_Choose	Frame	NetWorth	RT
1	1	1	A	B	36	40	A	Gain	36	1000
1	1	3	B	A	0	36	B	Gain	36	1100
1	1	2	C	D	-36	-40	C	Loss	0	1200
1	1	4	D	C	0	-36	D	Loss	0	1300
...	...	...	...	...	...	...	...	...	...	...

a. Define free parameters
b. Custom column names
c. Custom functions

Step 2: Parameter Recovery & Model Recovery

Step 3: Fit Real Data & Model Comparison

Step 4: Replay Experimental Effect

Experimental Effect (final three blocks)
The Ratio of Choosing Risky Option
Human
TD
RSTD
Utility
Gain
Loss
Frame

Note: ACC = accuracy, proportion of RL model predictions matching human choices; LogL = Log-Likelihood; AIC = Akaike Information Criterion; BIC = Bayesian Information Criterion.

TAFC Tasks for Reinforcement Learning

Structure and Characteristics of TAFC Tasks

Two-alternative forced choice tasks are a class of experimental paradigms in which participants are presented with two options on each trial and are required to choose one (Bogacz et al., 2006). On each trial, participants select one of two available options, receive feedback (e.g., rewards or outcomes), and use this information to guide future decisions. This trial-by-trial structure makes TAFC tasks particularly well-suited for reinforcement learning models, which are designed to capture how agents update value estimates and action policies based on feedback.

However, not all TAFC tasks are equally appropriate for RL modeling. RL is best applied to tasks that require participants to learn from outcome contingencies, rather than tasks that simply measure perceptual discrimination or cognitive conflict. Canonical decision-focused paradigms, such as the Two-Armed Bandit Task (Robbins, 1952), Ellsberg Paradox (Ellsberg, 1961), Passive Avoidance Learning Task (Newman & Kosson, 1986), and the Probabilistic Reversal Learning Task (Cools et al., 2002), fulfill these criteria. In these tasks, options typically differ in reward probabilities or values, and participants must learn to adapt their choices based on outcome histories. RL models offer a principled framework for characterizing these learning processes, including value updating, reward prediction error computation, and exploration-exploitation tradeoffs (e.g., Gershman, 2015; Liu et al., 2019; Niv et al., 2012; Oba et al., 2021; Rosenbaum et al., 2022; Waltmann et al., 2023).

In contrast to value-based TAFC tasks, certain TAFC tasks—such as the Simple Discrimination Task (Henmon, 1911) and the Stroop task (Stroop, 1935)—primarily emphasize the speed–accuracy tradeoff (Heitz, 2014). These paradigms typically do not incorporate value-based feedback; instead, they provide binary feedback indicating whether a response was correct or incorrect. As such, they are more appropriately modelled using evidence accumulation frameworks, such as the Drift Diffusion Model (Bogacz et al., 2006), than reinforcement learning models, which rely on trial-by-trial learning signals derived from outcome valence.

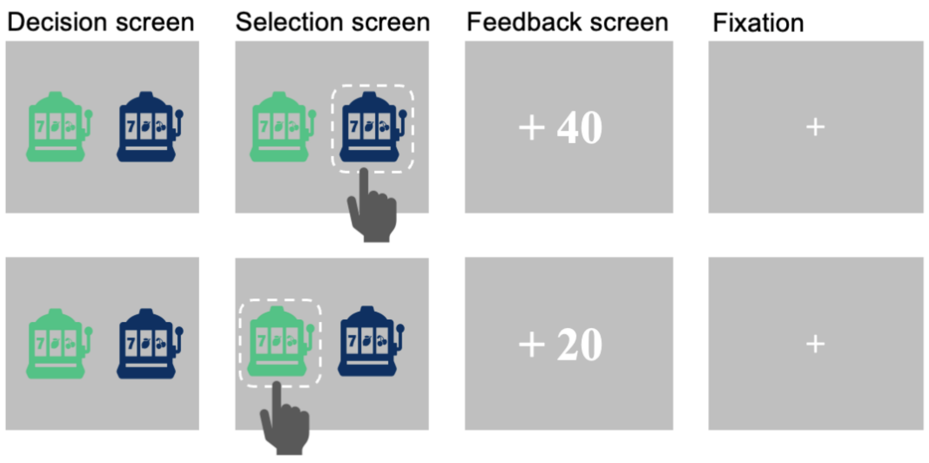
In summary, TAFC tasks that involve learning from reward outcomes—rather than simply responding based on perceptual accuracy—are well-suited to modeling within a reinforcement learning framework. One such paradigm, widely used to examine value-based decision-making, is the Two-Armed Bandit task, which we introduce in the following section.

The Two-Armed Bandit Task as a Canonical Example

The Two-Armed Bandit Task (Robbins, 1952) is a foundational paradigm within the family of TAFC tasks and has been widely used to study learning and decision-making in reinforcement learning (e.g., Niv et al., 2012; Oba et al., 2021; Rosenbaum et al., 2022). In this task, participants repeatedly choose between two options that differ in reward structure. For instance, one option may deliver a fixed reward (e.g., 20 points with 100% certainty), while the other offers a probabilistic reward (e.g., 40 points with a 50% chance, otherwise 0) (see Figure 2). Although the expected values may be equivalent, the outcome distributions differ.

Figure 2

Schematic of Two Representative Trials of The Two-Arm Bandit Task



Critically, participants are not informed of the reward contingencies beforehand. Instead, individuals must learn through experience which option yields better outcomes over time. This trial-and-error process aligns closely with core reinforcement learning principles, particularly value updating via prediction error—a mechanism originally formalized in psychological models as the Rescorla-Wagner rule (Rescorla & Wagner, 1972) and later integrated into computational RL frameworks.

Beyond learning dynamics, the task also provides a flexible platform for examining broader decision-making theories. Variants of the task have been used to test predictions from models such as prospect theory (Kahneman & Tversky, 1979), regret theory (Loomes & Sugden, 1982), fuzzy-trace theory (Reyna & Brainerd, 1995), and tri-reference point theory (Wang & Johnson, 2002), each of which emphasizes distinct psychological constructs. What distinguishes RL models, however, is their ability to model the latent computational process by which individuals integrate past outcomes to guide future choices.

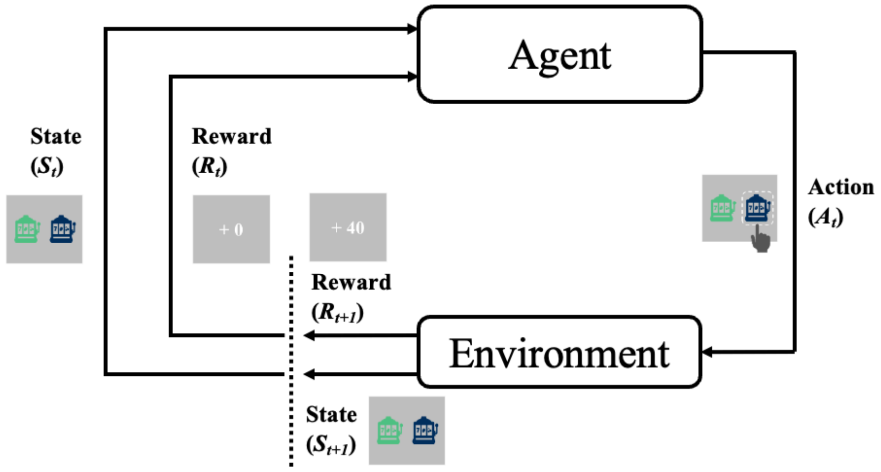
Reinforcement Learning Models for Value-Based TAFC Tasks

To formally capture how participants adapt their choices in response to feedback in value-based TAFC tasks, we turn to reinforcement learning models. RL models describe how agents learn to optimize behavior through interactions with the environment (Sutton & Barto, 2018). In value-based decision tasks, at each trial t, the environment presents a state S_t (e.g., two slot machines, one blue and one green). The agent selects an option A_t (e.g., choosing the blue machine) according to its internal policy π (e.g., a rule that always chooses the option with the highest expected value). Before observing the outcome, the agent forms an expectation of the chosen option's value (Cai et al., 2011;

Kahnt et al., 2010; Lau & Glimcher, 2008; Padoa-Schioppa & Assad, 2006). Upon receiving feedback, a prediction error (PE) is computed as the difference between the actual reward and the expected value (Sutton & Barto, 2018). This PE is used to update the expected value, scaled by a learning rate η, which determines how strongly new outcomes revise current expectation and, in turn, influence future behavior. This process leads to a new state S_{t+1} and a scalar reward R_{t+1}. Figure 3 illustrates this feedback loop in the two-armed bandit task.

Figure 3

Reinforcement Learning Loop in Two-Armed Bandit Task



Formally, this interaction is modeled as a Markov Decision Process (MDP), where state transitions and rewards depend only on the current state and action, rather than on the entire history (Sutton & Barto, 2018).

Model-Free vs. Model-Based RL

Reinforcement learning models can be broadly categorized into model-free and model-based approaches (Lee et al., 2012). Model-free RL updates action values solely based on past reinforcement signals, without constructing an explicit representation of the task environment (Barraclough et al., 2004; Ito & Doya, 2009; Sutton & Barto, 2018). Learning is driven by experienced outcomes and their discrepancies from expectations (i.e., prediction errors).

Conversely, model-based reinforcement learning models allow inferences to be made about how the environment will behave (Lee et al., 2012; Subramanian et al., 2022; Sutton, 1990). For example, given a state and action, the model might predict the resultant next state and next reward. However, in many TAFC tasks, the primary interest lies not in simulating complex state transitions, as options typically don't change dynamically based on previous decisions (see Figure 3). Instead, the core objective is to determine which of the given options to choose. For this goal, directly learning choice values from past reinforcement signals via model-free methods presents a simpler method for many TAFC tasks, despite model-based approaches excelling at providing a comprehensive understanding of the environment.

Core Learning Models

To understand how individuals learn from feedback in value-based decision-making, particularly within TAFC paradigms, this section introduces three foundational RL models: Temporal-Difference (TD) learning, Risk-Sensitive TD (RSTD), and Utility-Based learning (Niv et al., 2012). These models capture key mechanisms by which agents adjust expectations based on prediction errors and learning rates. While a number of alternative frameworks have been proposed—such as the accentuation-of-differences model (Spektor et al., 2019) and models with adaptive learning rates (e.g., adaptive learning rates model (Gershman, 2015))—the three selected here are among the most widely used and offer a tractable starting point for RL modeling in behavioral research.

Temporal-Difference (TD) Learning. The TD model is a foundational RL approach (Sutton & Barto, 2018). After each choice, the agent updates the value of the chosen option V_t^k using:

$$V_t^k \leftarrow V_t^k + \eta \cdot \left(r_{t+1} + \gamma V_{t+1}^k - V_t^k\right) \quad (1)$$

where η represents the learning rate and γ is the discount factor. V_t^k denotes the expected value of option k at trial t. The formula can be interpreted as updating the expectation for option k in the next trial, based on the difference between the actual reward and the current trial's expectation for option k, scaled by the learning rate η. This formulation allows the agent to update its expectations not only based on immediate outcomes but also anticipated future rewards.

In many TAFC tasks, which provide rewards immediately after each choice, the delay discount factor is omitted, reducing the model to the Rescorla–Wagner form (Rescorla & Wagner, 1972):

$$V_t^k \leftarrow V_t^k + \eta \cdot \left(r_{t+1} - V_t^k\right) \quad (2)$$

This simplified rule reflects incremental learning via prediction errors, a hallmark of reinforcement-based updating.

Risk-Sensitive Temporal Difference (RSTD) Model. Standard TD learning assumes symmetric learning across all outcomes, but psychological research suggests that individuals often weigh gains and losses differently. The RSTD model addresses this gain-loss asymmetry by assigning separate learning rates for positive and negative prediction errors (Mihatsch & Neuneier, 2002).

Specifically, the RSTD model assigns separate learning rates depending on the valence of the prediction error. When the actual reward exceeds the expected value—i.e., when the prediction error is positive—a positive learning rate (η^+) governs the update of the value estimate. Conversely, when the reward falls below expectations—i.e., a negative prediction error—a distinct, negative learning rate (η^-) is applied. This asymmetry enables the model to reflect differential sensitivity to favorable and unfavorable outcomes.

$$V_t \leftarrow \begin{cases} V_t + \eta^+ \cdot (R_{t+1} - V_t) \ \ if \ R_{t+1} - V_t > 0 \\ V_t + \eta^- \cdot (R_{t+1} - V_t) \ \ if \ R_{t+1} - V_t < 0 \end{cases} \tag{3}$$

This asymmetry enables the model to capture individual differences in loss aversion or optimism bias, making it particularly relevant in studies of risk-sensitive or emotionally charged decision-making.

Utility-Based Learning Models. Traditional RL models assume that rewards are processed in their objective, numerical forms. However, human decision-making is based on subjective utility rather than raw outcomes (Bernoulli, 1954). A utility model thus transforms the objective reward R_{t+1}into a nonlinear subjective value $U(R_{t+1})$, via, for instance, a power function (Buchsbaum & Stevens, 1971):

$$u(x) = x^\gamma \tag{4}$$

The value update rule is then modified as:

$$V_t \leftarrow V_t + \eta \cdot [U(R_{t+1}) - V_t] \tag{5}$$

Here, η represents the learning rate, and $U(R_{t+1})$ is the subjectively transformed utility of the received reward. By incorporating nonlinear utility, this model reflects the subjective valuation process that underlies decision-making.

From Learning to Choice: Modeling Decision Behavior in TAFC Tasks

Choice Selection via Exploration–Exploitation Tradeoff

While reinforcement learning models provide a formal account of how agents update value estimates based on feedback, a critical design consideration arises in translating these learned values into action: how to implement the exploration–exploitation tradeoff.

Using a value function, an agent estimates the expected reward associated with each action. Selecting the action with the highest estimated value—commonly referred to as the greedy action—constitutes exploitation of current knowledge. In contrast, choosing a lower-valued action—i.e., a nongreedy action—constitutes exploration, enabling the agent to refine its value estimates for less familiar alternatives. Because it is not possible to both exploit and explore within a single choice, this tradeoff governs how agents balance the pursuit of immediate reward with the need to gather information that may improve future decisions (Sutton & Barto, 2018).

One common solution for implementing the exploration–exploitation tradeoff is the softmax action selection rule, which converts estimated values into choice probabilities in a graded, probabilistic manner. This approach aligns well with empirical findings that human choices in Two-Alternative Forced Choice (TAFC) tasks are inherently stochastic: participants do not always select the option with the highest estimated value, but their likelihood of choosing an option increases with its relative value (Lee et al., 2012). The softmax function thereby captures both deterministic and exploratory elements of choice behavior.

Formally, for a TAFC task with two options—left (L) and right (R)—the probability of choosing each option is given by:

$$P_L = \frac{1}{1 + e^{-(V_L - V_R)\cdot\tau}};\ P_R = \frac{1}{1 + e^{-(V_R - V_L)\cdot\tau}} \tag{6}$$

where $V_L$and V_R denote the estimated values of the left and right options, respectively; the inverse temperature parameter τ governs the degree of stochasticity: higher τ values produce more deterministic choices, while lower τ values lead to more random responding.

Although the softmax model is widely used due to its ability to balance exploration and exploitation seamlessly, other strategies (see our tutorial website for more details), such as the epsilon-greedy approach (Sutton & Barto, 2018), provide alternative ways to operationalize exploration. With a fixed probability ϵ, the agent selects a random action regardless of value estimates (pure exploration). With probability $1-\epsilon$, it chooses the option with the highest estimated value (pure exploitation).

Model Fitting and Likelihood-Based Evaluation

To evaluate how well a reinforcement learning model captures human choice behavior, it is essential to quantitatively compare the model's predicted choice probabilities against actual participant responses. Likelihood-based methods, particularly the log-likelihood metric, provide a principled approach for this purpose (Hampton et al., 2006).

In TAFC tasks, choices are binary, with each trial resulting in the selection of either the left or right option. The RL model generates predicted probabilities for each option using the softmax rule. The log-likelihood (LL) is calculated as the sum of the log-probabilities assigned to the participant's observed choices across trials:

$$LL = \sum_{i=1}^{n} B_{Li} \cdot log\, P_{Li} + \sum_{i=1}^{n} B_{Ri} \cdot log\, P_{Ri} \qquad (7)$$

where B_{Li} and B_{Ri} are binary indicators (1 or 0) denoting whether the participant chose the left or right option on trial i, and P_{Li} and P_{Ri} are the corresponding model-predicted probabilities. Higher LL values indicate better alignment between the model's predictions and observed behavior.

Information Criteria and Model Comparison

While the log-likelihood quantifies goodness of fit, it does not penalize model complexity. Models with many free parameters may fit idiosyncratic noise rather than meaningful psychological processes (Wilson & Collins, 2019). To counter this, information-theoretic criteria such as the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) introduce complexity penalties, balancing fit and parsimony (Akaike, 1974; Schwarz, 1978):

$$AIC = -2LL + 2k \qquad (8)$$

$$BIC = -2LL + k\, log\, n \qquad (9)$$

where k is the number of free parameters and n is the total number of trials. These criteria facilitate principled model selection by favoring models that generalize beyond the sample data, thus strengthening theoretical inferences drawn from model-based analyses.

A Structured Approach to RL Modeling: The Wilson & Collins Framework

Building on the theoretical foundations of TAFC tasks and reinforcement learning models, this section briefly introduces the four-step modeling framework proposed by Wilson and Collins (2019), which has become a widely adopted standard in cognitive and computational modeling (e.g., Ciranka et al., 2022; Valentin et al., 2024; van Baar et al., 2022). This framework provides a systematic and transparent roadmap for model development and validation, guiding researchers through four essential stages: (1) constructing a model suited to the task structure, (2) conducting parameter and model recovery to ensure identifiability, (3) fitting the model to empirical data, and (4) reporting and interpreting results in a transparent and reproducible manner. We now introduce each of these steps in detail.

Step 1: Design an Experiment and Build RL Models. The first step in applying reinforcement learning models to behavioral data is to define an appropriate experimental paradigm and develop a set of candidate models capable of capturing the key experimental effects observed in the data. A central focus of this stage is the specification of the model's core computational components, such as learning rules, utility transformations, and choice functions.

To illustrate this step, we use a well-established experimental task—the Two-Armed Bandit task (Robbins, 1952), which is implemented in a publicly available dataset (Mason et al., 2024, https://osf.io/hy3q4/) and serves as an example in this tutorial. More details of the task design are provided in the coding tutorial section. Candidate RL models vary in how they formalize learning and decision-making—e.g., the Temporal Difference model with a fixed learning rate, the Risk-Sensitive TD model allowing asymmetric updates, and utility-based models incorporating nonlinear transformations. Action selection is modeled using the softmax (probabilistic choice) method.

Step 2: Parameter and Model Recovery. Parameter and model recovery are critical steps in evaluating candidate models. Before fitting models to real data, researchers should assess the stability and identifiability of each model by performing recovery analyses. This process involves generating synthetic datasets using a given model and then fitting these datasets with both the generating model and alternative models. Parameter recovery refers to the degree to which the parameters used to generate data can be accurately estimated (i.e., show high correlation) by fitting the model to that data. Model recovery, on the other hand, assesses whether a model can be correctly identified from simulated data that it itself generated, typically by showing that it has the lowest BIC compared to other candidate models. For more details, see Wilson and Collins (2019).

Only when a model demonstrates strong recoverability—accurately retrieving its own parameters and outperforming other models in fitting its own simulated data—can its performance on real data be meaningfully interpreted and compared to other candidate models.

Step 3: Fit Real Data and Model Comparison. Once parameter recovery and model recovery analyses have been successfully completed, providing confidence in the model's ability to estimate its parameters and distinguish between competing mechanisms reliably, researchers can proceed to fit these candidate computational models to real experimental behavioral data. This step involves finding the set of parameter values for each model that best accounts for the observed human behavior, often by maximizing the likelihood of the data given by the model and its parameters.

After fitting, the next step is to perform model comparison. This process aims to determine which of the predefined set of models is most likely to have generated the observed behavioral data. Various methods exist for model comparison, with approaches like the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) being commonly used. It is essential to ensure that the parameter range obtained from fitting the real data aligns with the ranges where parameter and model recovery were robust; otherwise, it may be necessary to revisit and adjust the simulation parameters. The ultimate goal is to select the "winning" model that offers the best balance of fitness and parsimony.

Step 4: Latent Variable Analysis and Report Results. Following model fitting and selection, the next step is to evaluate how well the best-fitting model captures key behavioral patterns. This is typically done through posterior predictive checks (Gelman et al., 1996; Roecker, 1991), where the model, using its fitted parameters, is used to simulate behavioral data. These simulated outcomes are then compared to empirical results to assess whether the model reproduces the core experimental effects of interest. This process strengthens confidence that the model reflects not just statistical patterns but meaningful psychological processes.

In addition to validating model performance, researchers can examine latent variables—unobservable internal quantities inferred from model parameters—to gain insight into underlying cognitive mechanisms. For example, fitted learning rates can be correlated with individual traits or external measures, enabling interpretation of how computational processes relate to broader psychological constructs (e.g., Davidow et al., 2016; Louie, 2022; Neuser et al., 2023). Such analyses help bridge the gap between model parameters and theoretical understanding of human behavior.

Building Reinforcement Learning Models with the binaryRL Pipeline

In this section, an openly available dataset from the Gambling Paradigm (Mason et al., 2024) will be used as an example, along with annotated R code from our package, to facilitate a hands-on understanding of the complete four-step reinforcement learning modelling workflow.

Setting Up the binaryRL Environment in R

The *binaryRL* package is written almost entirely in R (R Core Team, 2020) and Rcpp (Eddelbuettel et al., 2008). The parallel processing relies on the `future` (Bengtsson, 2015), `doFuture` (Bengtsson, 2016) and `foreach` (Microsoft & Weston, 2009) packages, with `doRNG` (Gaujoux, 2011) used to control the random number seeds during parallel execution and `progressr` (Bengtsson, 2020) to display progress bars. Although the dependencies of these R packages have minimal R version requirements, we still recommend that users install our package using an R version higher than 4.0.0.

Users can install the stable version from CRAN or the latest version from GitHub using the following code:

```
# Install the stable version from CRAN
install.packages("binaryRL")
# Install the latest version from GitHub
remotes::install_github("yuki-961004/binaryRL@*release")

# Load package
library(binaryRL)
# Obtain help document
?binaryRL
```

Our package installation only includes R packages necessary for parallel processing and does not force the installation of algorithm packages. If users need to use algorithms other than L-BFGS-B, the software will prompt them to install the corresponding algorithm packages.

Step 1: Design an Experiment and Build an RL Model

The example experiment represents a variant of the two-armed bandit task introduced earlier, in which participants repeatedly choose between two options that differ in risk and reward. In the original mixed-design experiment, participants were divided into two between-subjects groups: Rare-extreme and Rare-non-extreme. A total of 250 participants were randomly assigned to these two groups, with 125 participants in each. The within-subjects factor was the choice domain (gain vs. loss).

The example data used in this tutorial is from the Rare-non-extreme group. Participants in the Rare-non-extreme group (`binaryRL::Mason_2024_G2`) faced two options in both gain and loss frames. In the gain frame, the risky option offered a 90% chance of receiving 40 points and a 10% chance of receiving 0 points, whereas the fixed option guaranteed 36 points. In the loss frame, the risky option carried a 90% chance of losing 40 points and a 10% chance of losing 0 points, while the fixed option resulted in a guaranteed loss of 36 points. For more details, refer to Mason (2024).

Data Structure and Column Specifications

For application of the R code described in the next section, the dataset must be in long format (each row of data represents a single trial). Most experimental software (e.g., PsychoPy, jsPsych, E-Prime) outputs data in long format by default; if not, functions such as `tidyr::pivot_longer()` can transform wide-format datasets to long format. Additionally, it is essential for RL modelling that the dataset records the reward of the unchosen option for each trial. To facilitate visualization, Table 1 presents the example dataset in long format. The essential columns in the dataset are Subject, Block, Trial, L_choice, R_choice, L_reward, R_reward, Sub_Choose.

Table 1

Example Dataset from Mason et al.(2024)

Subject	Block	Trial	L_choice	R_choice	L_reward	R_reward	Sub_Choose	Frame	NetWorth	RT
1	1	1	A	B	36	40	A	Gain	36	1000
1	1	3	B	A	0	36	B	Gain	36	1100
1	1	2	C	D	-36	-40	C	Loss	0	1200
1	1	4	D	C	0	-36	D	Loss	0	1300
…	…	…	…	…	…	…	…	…	…	…

Note. L_choice and *R_choice* indicate the options presented on the left and right, respectively; *L_reward* and *R_reward* correspond to the reward values associated with these options; *Sub_Choose* denotes the participant's selection; *Frame* indicates the choice context (gain vs. loss); *NetWorth* is the cumulative reward at the end of the trial; *RT* represents response time in milliseconds.

If users choose to rely solely on the built-in models, the 'reinforcement learning model construction' step can be omitted. However, the column names in the dataset must match those of the key columns in the example dataset.

Reinforcement Learning Model Construction

This sub-section details the most distinctive and technically involved component of this R package: constructing a custom reinforcement learning model. For users who prefer to use one of the three built-in RL models (TD, RSTD, Utility), this step can be skipped. These models can be invoked directly using the `rcv_d` and `fit_p` functions with the following code:

```
recovery <- rcv_d(
   …,
   simulate_models = list(TD, RSTD, Utility),
   fit_models = list(TD, RSTD, Utility),
   …
  )
comparison <- fit_p(
   …,
   fit_models = list(TD, RSTD, Utility),
   …
  )
```

To construct a custom model, the first step is to define an objective function compatible with the estimation algorithm. Here, we use the TD model as an example to demonstrate how to build its objective function. This function takes a single argument, `params`, which corresponds to the model's two free parameters: the learning rate, `eta`, and the inverse temperature, `tau`.

```
TD <- function(params){

  res <- run_m(
    name = NA,
    data = data,
    id = id,
    n_trials = n_trials,
    n_params = n_params,
    priors = priors,
    mode = mode,
    policy = policy,
    # learning rate
    eta = c(params[1]),
    # inverse temperature
    tau = c(params[2]),
    …
  )

  assign(x = "binaryRL.res", value = res, envir = binaryRL.env)
  loss <- switch(EXPR = estimate, "MLE" = -res$ll, "MAP" = -res$lpo)
  switch(EXPR = mode, "fit" = loss, "simulate" = res, "replay" = res)
}
```

The arguments in the `run_m` function—including `data`, `id`, `n_trials`, `n_params`, `priors`, `mode`, and `policy`— are automatically populated by subsequent functions such as `rcv_d()`, `fit_p()`, or `rpl_e()`. The final three lines of the function handle result assignment and environment variable transfer. Users are strongly advised not to modify these components. A detailed explanation of the primary arguments is provided below:

- **name**: Defaults to `NA`. The program automatically detects whether a built-in model (TD, RSTD, or Utility) is used based on the number of learning rates and the presence of gamma. Users defining custom models may modify this argument.
- **data**: The dataset currently being processed.
- **id**: The identifier of the participant being analyzed. May be of character or numeric type, but must correspond to an existing ID in the dataset.

- **n_trials**: The total number of trials completed by each participant. This value is used in the calculation of AIC and BIC.
- **n_params**: The number of free parameters in the RL model. For instance, the basic TD model includes two free parameters: the learning rate `eta` and the inverse temperature parameter `tau`; hence , `n_params = 2`.
- **priors**: A list specifying the prior distributions for each parameter. Each element must be a density function.
- **mode**: Accepts one of three values: `simulate`, `fit`, or `replay`. These correspond to parameter/model recovery, model fitting to real data, and replaying the experiment using estimated parameters (steps 2, 3, and 4 in the workflow, respectively).
- **policy**: Determines the policy used during fitting. Off-policy methods use the participant's actual choices to update value estimates, whereas on-policy methods sample choices based on the model's current action probabilities and update values accordingly.

Specify Your Column Names

When you begin creating a new reinforcement learning model, the first thing to consider is the column names. While *binaryRL* can run models with user-defined column names, these may not match the package's defaults. You must therefore inform the program (the `run_m` function) what your column names are by specifying the following arguments as character strings:

```
run_m(
  # column names
  sub = "Subject",
  time_line = c("Block", "Trial"),
  L_choice = "L_choice",
  R_choice = "R_choice",
  L_reward = "L_reward",
  R_reward = "R_reward",
  sub_choose = "Sub_Choose",
  var1 = "extra_Var1",
  var2 = "extra_Var2"
  …
)
```

Each argument serves the following purpose:

- **sub**: Specifies the column containing participant identifiers. Values can be numeric or character.
- **time_line**: A character vector defining the hierarchical structure of the experiment timeline. For example, `c("Block", "Trial")` indicates that the experiment is organized into blocks, each containing multiple trials.

- **L_choice** and **R_choice:** Specify the columns indicating the options available on the left and right sides, respectively, in each trial.
- **L_reward** and **R_reward:** Specify the columns containing the rewards associated with the left and right options, respectively.
- **sub_choose:** Indicates the column recording the actual choice made by the participant. The values should match those in `L_choice` and `R_choice`.
- **var1** and **var2:** Allow incorporation of additional experimental variables beyond rewards and choices. For instance, setting `var1 = "Frame"` enables the model to access information about gain/loss framing. These parameters offer flexibility for models that require supplementary contextual information.

Free Parameters in Reinforcement Learning Models

The second part of constructing a custom reinforcement learning model is defining its free parameters, which are used to capture different cognitive and behavioral mechanisms. The `run_m` function supports multiple parameters that govern value updating, exploration–exploitation tradeoffs, and choice rules. Below is an example of how these parameters can be passed:

```
run_m(
  …,
  # value function
  eta = c(params[…], params[…]),
  gamma = c(params[…], params[…]),
  # exploration-exploitation tradeoff
  initial_value = NA,
  threshold = 1,
  epsilon = c(params[…]),
  lambda = c(params[…]),
  # upper-confidence-bound
  pi = c(params[…]),
  # soft-max
  tau = c(params[…]),
  lapse = 0.02,
  # extra parameters
  alpha = c(params[…], params[…]),
  beta = c(params[…], params[…]),
  …
)
```

Each of these parameters plays a distinct role in the model's behavior. The following sections provide a detailed explanation of their theoretical and practical implications.

◆ **Value Function**

- **utility function (`gamma`):** Although named `gamma`, this parameter does not correspond to the temporal discounting factor in classical TD learning. Rather, it represents the exponent in Stevens' power law, capturing nonlinear relationships between objective rewards and their subjective utility. This package is built upon the Rescorla–Wagner framework, which does not incorporate future reward discounting, as outcomes are immediate within each trial.
- **learning rate (`eta`):** The learning rate (`eta`), often denoted as alpha in the literature, controls the extent to which prediction errors update value estimates. A higher value implies greater weight on recent prediction errors. While standard TD models use a single learning rate, the RSTD model incorporates two distinct rates: one for positive prediction errors (`eta+`) and another for negative ones (`eta-`). The name eta is used here to avoid namespace conflicts, reserving `alpha` for user-defined extensions.

◆ **Exploration–exploitation Tradeoff**

Based on Sutton and Barton (2018), the package supports several strategies to balance exploration and exploitation:

- **optimistic initial value (`initial_value`):** Setting a high initial value for options encourages exploration of less-chosen alternatives. This strategy, known as "optimistic initialization" (Sutton & Barto, 2018), can influence the estimated asymmetry between positive and negative learning rates (Palminteri & Lebreton, 2022). By default, `initial_value = NA` uses the first encountered reward as the initial value. Users may fix this to a specific value or treat it as a free parameter.
- **epsilon-based strategies (`threshold`, `epsilon`, `lambda`):** Epsilon-greedy policies choose the highest-value option with probability `1 - epsilon`, and explore randomly otherwise. The package also supports:
    - **epsilon-first**: fully random exploration for the first `threshold` trials, then greedy exploitation.
    - **epsilon-decreasing**: exploration probability decays across trials, controlled by `lambda`.
- **upper confidence bound (`pi`):** This strategy promotes exploration of less-sampled options by adding a bonus proportional to the uncertainty in value estimates (Sutton & Barto, 2018). The parameter `pi` (often denoted as `c`) controls the degree of exploration bias. By default, `pi` is set to `NA`, which ensures the model samples each option at least once. If the user sets it to 0, the `bias_func` will be completely disabled.
- **softmax inverse temperature (`tau`):** The parameter `tau` (commonly `beta`) regulates how deterministic choices follow value differences. Higher values increase sensitivity to value differences, leading to more exploitative behavior. Lower values produce more random choices. Along with `eta`, this is one of the most widely used parameters in RL modeling.

◆ **Lapse Rate**

- **lapse rate (`lapse`):** To avoid numerical underflow when computing log-probabilities of choice, a small lapse rate is often incorporated (Wilson & Collins, 2019). This represents the probability of a random choice error, ensuring that no option has exactly zero probability of being chosen. The default value of `0.02` ensures a minimum choice probability of 1% in two-alternative tasks. This parameter can also be treated as free—elevated lapse rates may indicate participant inattention, especially in catch trials where one option is clearly disadvantageous.

◆ **Extra Parameters**

- **alpha, beta:** The names `alpha` and `beta` are reserved for user-defined parameters, allowing maximal flexibility for custom model extensions. This ensures that users can incorporate novel mechanisms without altering the core package structure.

Core Functions for Custom Model Specification

Despite having the ability to define the number of free parameters to create variations of the built-in models, if users would like to build a completely new model, they must define the third part in `run_m`, the six core computational components: the utility function, learning rate function, exploration strategy, bias function, choice probability function, and loss function. If any of these functions are modified in a custom model, their names must be provided as a character vector in the `funcs` argument of `rcv_d()`or `fit_p()`, ensuring proper transmission to parallel computing environments. The following functions can be customized:

```
run_m(
  …,
  # core functions
  util_func = your_util_func,
  rate_func = your_rate_func,
  expl_func = your_expl_func,
  bias_func = your_bias_func,
  prob_func = your_prob_func,
  loss_func = your_loss_func,
)
```

Detailed descriptions of each function are provided below:

- **util_func** (associated with `gamma`): Transforms objective reward magnitudes into subjective utilities. The default implementation follows Stevens' power law: `u(x) = x^γ`. Users can redefine this function to incorporate alternative utility transformations.
- **rate_func** (associated with `eta`): Determines the learning rate applied during value updating. The default implementation uses an `if-else` statement to distinguish between the TD and RSTD models: if a single learning rate is provided, it operates as in standard TD learning; if two rates are provided, it implements the RSTD rule by selecting `eta[1]` when the reward exceeds the expectation and `eta[2]` otherwise.

- **expl_func** (associated with `threshold`, `epsilon`, `lambda`): Implements exploration strategies such as ε-greedy, ε-first, or ε-decreasing policies. The function uses conditional checks on the input parameters to determine which strategy to execute and returns an exploration flag (`try`) that influences subsequent choice selection.
- **bias_func** (associated with `pi`): Adds an exploration bonus to value estimates to encourage sampling of less-frequently chosen options. If `pi = NA`, the function ensures each option is selected at least once. If pi is a numeric value, it implements standard upper confidence bound (UCB) action selection. Users can modify this function to implement alternative exploration biases.
- **prob_func** (associated with `tau`): Converts value differences into choice probabilities. Receives the exploration flag (`try`) from `expl_func`: if `try = 1`, it returns uniform random choice probabilities; otherwise, it applies either greedy selection or softmax normalization based on the inverse temperature parameter `tau`. The lapse rate adjusts the final probabilities to prevent numerical underflow.
- **loss_func:** Computes the discrepancy between model predictions and actual choices. The default implementation calculates the negative log-likelihood of observed choices given model parameters. Users can redefine this function to implement alternative loss functions.

These core functions provide comprehensive flexibility for implementing a wide range of reinforcement learning models beyond the built-in TD, RSTD, and Utility specifications. Users are encouraged to refer to the package documentation and source code for detailed examples of function implementations.

Step 2: Parameter Recovery and Model Recovery

To assess model identifiability and robustness, researchers should conduct parameter recovery and model recovery analyses using the `rcv_d` function. In this example, synthetic datasets were generated by sampling parameters from three built-in RL models: TD, RSTD, and Utility models. By default, the learning rate (η) and utility (γ) parameters were drawn from a uniform distribution ranging from 0 to 1, while the inverse temperature parameter (τ) for the softmax function was sampled from an exponential distribution. Each simulated dataset was then fitted using all three models to determine which model provided the best fit. The R code for this procedure is as follows:

```
recovery <- rcv_d(
  policy = "on",
  data = Mason_2024_G2,
  model_names = c("TD", "RSTD", "Utility"),
  simulate_models = list(TD, RSTD, Utility),
  fit_models = list(TD, RSTD, Utility),
  lower = list(c(0, 0), c(0, 0, 0), c(0, 0, 0)),
  upper = list(c(1, 5), c(1, 1, 5), c(1, 1, 5)),
  iteration_s = 100,
  iteration_f = 100,
  nc = 4,
  algorithm = c("NLOPT_GN_MLSL", "NLOPT_LN_BOBYQA")
)
```

The main arguments are described below. Arguments marked with * do not need to be set by the user, as default values are sufficient:

- **policy**: Determines the policy used during fitting. off-policy uses the participant's actual choices to update value estimates, whereas on-policy samples choices based on the model's current action probabilities and updates values accordingly.
- **data:** The dataset used for analysis.
- **model_names:** Names of the candidate models, entered as a character vector.
- **simulate_models:** Models used to generate simulated data, passed as a list of functions.
- **fit_models:** Models used to fit simulated data, passed as a list of functions.
- **lower/ upper:** Lower and upper bounds for free parameters in the fitting models. The class of these bounds must be `numeric vector`, and the entire argument is passed as a list.
- **iteration_s:** Numbers of simulated datasets.
- **iteration_f:** Maximum iterations for the fitting algorithm per simulated dataset.
- **nc:** Number of threads for computation. `nc > 1` enables parallel fitting of multiple datasets.
- **algorithm:** Choice of fitting algorithm. We provide eight algorithms for users: L-BFGS-B (Byrd et al., 1995), Simulated Annealing (Gubian et al., 2011), Genetic Algorithm (Scrucca, 2012), Differential Evolution (Ardia et al., 2005), Particle Swarm Optimization (Bendtsen, 2010), Bayesian Optimization (Bischl et al., 2017), Covariance Matrix Adapting Evolutionary Strategy (Trautmann et al., 2010) and Nonlinear Optimization Library (Ypma & Johnson, 2011).
- **estimate*:** Specifies the estimation method. The default is *Maximum Likelihood Estimation* (MLE), which does not incorporate prior distributions. Setting this to *Maximum a Posteriori* (MAP) uses the EM algorithm, following the approach in the sjgershm/mfit MATLAB toolbox.
- **rfun*:** A nested list of functions generates random parameter values for simulation. The elements of thelist must be a named list of functions, where each name corresponds to a model parameter and its value is the random number generation function.
- **dfun*:** A nested list that defines the probability density/mass functions (PDF/PMF) for each model's parameters. The top-level names of the list must match the model names. Each element must be another named list, where each name corresponds to a model parameter and its value is the probability density function.

- **id*:** Defaults to `NULL`. The program randomly selects one participant as a template to generate simulated data, ensuring that the trial sequence matches the original experiment. This is important for parameter and model recovery.
- **n_trials*:** Defaults to `NULL`. The number of trials per participant is automatically determined. If participants experienced different paradigms with varying trial numbers, `id` and `n_trials` must be set manually.
- **funcs*:** For custom models that modify core functions and use parallel computing (`nc > 1`), list the names of the custom functions to ensure they are accessible in the parallel environment.
- **initial_params*:** Initial parameter values for iterative algorithms. Defaults to `NA`, with each parameter initialized at `lower + 0.01`. It can be set manually in the form of a numeric vector if needed.
- **initial_size*:** For evolutionary algorithms, the initial population size can be specified (default to 50). Excessively large sizes increase computation time.
- **seed*:** Random seed to ensure reproducible results.

After defining the necessary arguments, users will obtain the model recovery output. Table 2 shows a subset of the example output. Each row corresponds to a single recovery iteration, listing the model used to generate the data, the model used for fitting, key performance indices (accuracy, log-likelihood, AIC, BIC), and the true (input) versus recovered (output) parameter values. Differences between input and output parameters reflect the extent to which the fitting procedure successfully recovered the generative parameters, given the specified argument settings.

Table 2

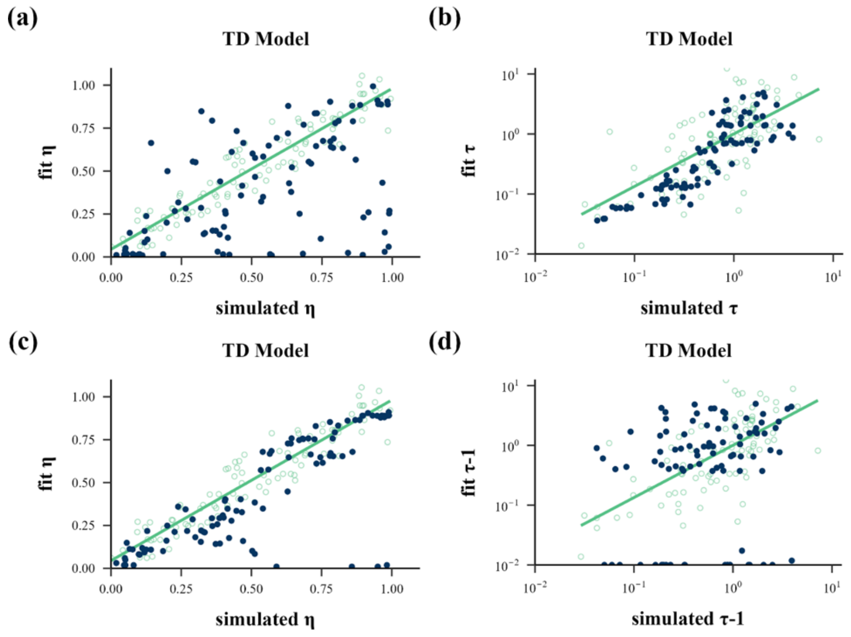
Example Output of Model Recovery

simulate_ model	fit_ model	iteration	ACC	LL	AIC	BIC	input_ param_1	output_ param_1	input_ param_2	output_ param_2	input_ param_3	output_ param_3
TD	TD	1.00	92.50	-81.65	167.3	175.07	0.64	0.76	1.83	1.00	NA	NA
TD	TD	2.00	93.06	-74.62	153.24	161.01	0.72	0.75	2.44	1.00	NA	NA
TD	TD	3.00	83.06	-146.57	297.14	304.91	0.95	0.01	3.04	1.01	NA	NA
…	…	…	…	…	…	…	…	…	…	…	…	…
RSTD	RSTD	1.00	93.89	-72.61	151.22	162.88	0.29	0.13	0.72	0.36	2.44	1.69
RSTD	RSTD	2.00	93.61	-70.17	146.34	158	0.95	0.42	0.21	0.14	1.36	1.45
RSTD	RSTD	3.00	95.28	-65.47	136.94	148.6	0.54	0.64	0.20	0.19	2.01	4.79
…	…	…	…	…	…	…	…	…	…	…	…	…
Utility	Utility	1.00	62.78	-190.72	387.44	399.1	0.29	0.01	0.72	0.38	2.44	6.00
Utility	Utility	2.00	60.56	-209.00	424.00	435.66	0.95	0.34	0.21	0.00	1.36	2.94
Utility	Utility	3.00	60.56	-209.74	425.48	437.14	0.54	0.22	0.20	0.01	2.01	2.69
…	…	…	…	…	…	…	…	…	…	…	…	…

The result table can be used for visualization. For example, Figure 4 illustrates the parameter recovery plots (scatterplots comparing input and recovered parameters), while Figures 5a and 5c show confusion matrices, and Figures 5b and 5d present inversion matrices. As plotting was not the primary focus of this work, the code for reproducing these plots is openly available at the package homepage (https://yuki-961004.github.io/binaryRL/).

Figure 4

Parameter Recovery Plots under Different Inverse Temperature Settings

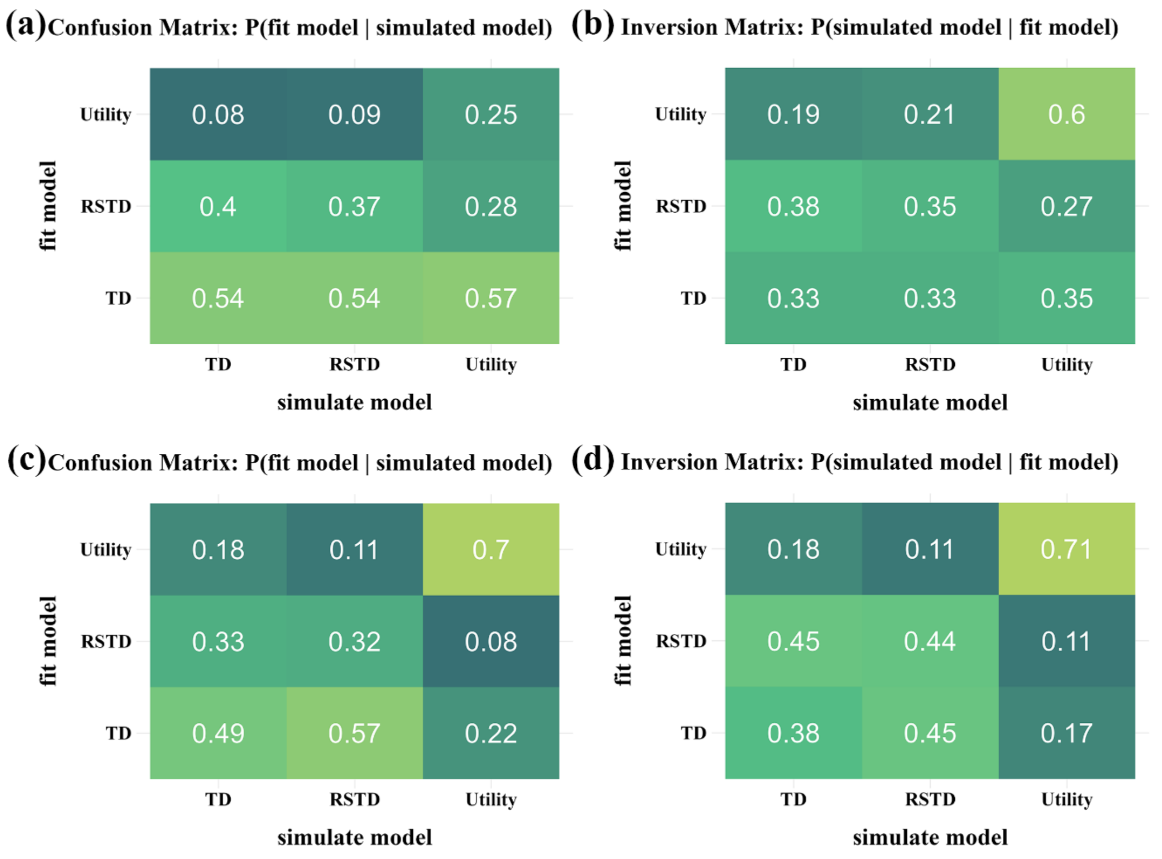


NOTE: In Figures 4a and 4b, the inverse temperature parameter in the softmax function was set according to *τ ~ Exp(1)*. In this scenario, choice behavior exhibits a certain degree of randomness. Consequently, the inverse temperature parameter can be recovered with relatively high accuracy, whereas the learning rate parameter η in the TD model cannot. To facilitate η recovery, researchers may add 1 to all τ values during data generation, which reduces choice randomness and results in a more accurate recovery of the learning rate η, but a less accurate recovery of the inverse temperature parameter τ (see Figures 4c and 4d). For visualization purposes, in Figure 4d, we subtracted 1 from τ values when plotting, so that the figures reflect the original coordinate scale.

In Figure 5, the confusion matrix (Figures 5a and 5c) displays the P(fit model | simulated model), indicating how often data generated by a given model (columns) are best fitted by each candidate model (rows). High diagonal values reflect good model discriminability. The inversion matrix (Figures 5b and 5d) shows P(simulated model | fit model), computed from the confusion matrix using Bayes' theorem. It represents the likelihood that a dataset was generated by a particular model (rows), given that a specific model was identified as the best fit (columns), which is particularly useful when the true generating model is unknown.

Figure 5

Confusion Matrix and Inversion Matrix under Different Inverse Temperature Settings



NOTE: In Figures 5a and 5b, the inverse temperature parameter in the softmax function was set according to *τ ~ Exp(1)*. In this scenario, choice behavior exhibits a certain degree of randomness. Consequently, it is difficult for simulated data generated by Model A to be best fit by Model A itself. Similar to parameter recovery, increasing the inverse temperature τ in the softmax function by 1 can improve the effect of model recovery.

It is crucial to clarify that the parameter and model recovery results presented in this section should be viewed as illustrative of the analytical workflow rather than as an evaluation of the robustness of our MDP construction. The main challenge lies in the inverse temperature parameter of the softmax function, which introduces choice stochasticity into the simulated data and interferes with the recovery of other parameters (Fig. 4a). Our tests show that near-perfect recovery can be achieved using estimation methods that bypass the log-likelihood, such as Approximate Bayesian Computation

(ABC) or Recurrent Neural Networks (RNN), suggesting that the issue stems from the inherent information loss of log-likelihood-based approaches.

This limitation is not unique to our work; it is consistent with findings from Wilson and Collins (2019). We not only replicated their results, showing that manually increasing the inverse temperature by 1 improves the recovery of other parameters (Figs. 4a, 4c) and model recovery (Fig. 5), but also found that this adjustment comes at the cost of poor recovery of the inverse temperature itself (Figs. 4b, 4d). We suggest it may be an unavoidable limitation when combining log-likelihood-based estimation with RL models. Wilson and Collins (2019) further proposed a tradeoff approach, in which introducing an unimportant parameter can improve the recovery of others. However, this also implies that without new constraints, one can achieve reliable recovery for either the learning rate (η) or the inverse temperature (τ), but not both.

These findings have an important implication for practical applications: when fitting real-world data with MLE, a low estimated inverse temperature should raise doubts about the reliability of the fitted learning rate. In addition, factors such as policy choice (on or off), algorithm package (e.g., GenSA, GA), and the number of iterations also affect recovery outcomes. Researchers are therefore encouraged to account for these factors and to consider alternative estimation methods, as demonstrated in our advanced tutorial, when aiming for more precise parameter recovery.

Step 3: Fitting Real Data and Model Comparison

To evaluate model performance at the individual level, we employed the `fit_p` function, which simultaneously estimates parameters and model fit indices for each participant across multiple candidate models. In the example below, we compared three models—TD, RSTD, and Utility—using data from 125 participants. The R code for this procedure is as follows:

```
comparison <- fit_p(
  policy = "on",
  data = Mason_2024_G2,
  model_name = c("TD", "RSTD", "Utility"),
  fit_model = list(TD, RSTD, Utility),
  lower = list(c(0, 0), c(0, 0, 0), c(0, 0, 0)),
  upper = list(c(1, 5), c(1, 1, 5), c(1, 1, 5)),
  iteration_i = 100,
  nc = 4,
  algorithm = c("NLOPT_GN_MLSL", "NLOPT_LN_BOBYQA")
)
```

```
result <- dplyr::bind_rows(comparison)
write.csv(result, "./result_comparison.csv", row.names = FALSE)
```

The main arguments are described below. Arguments marked with an asterisk (*) are optional and do not need to be set by the user, as default values are sufficient:

- **policy**: Determines the policy used during fitting. Off-policy methods use the participant's actual choices to update value estimates, whereas on-policy methods sample choices based on the model's current action probabilities and update values accordingly.
- **data:** The dataset to be analyzed.
- **model_names:** Character vector specifying the names of the models.
- **fit_models:** List of model-fitting functions.
- **lower/ upper:** Lower and upper bounds for free parameters in the fitting models. The class of these bounds must be `numeric vector`, and the entire argument is passed as a list.
- **iteration_i:** Maximum number of optimization iterations for fitting each participant's data.
- **nc:** Number of threads. Defaults to 1 (sequential computation). Values greater than 1 enable parallel computing.
- **algorithm:** Optimization algorithm. We provide eight algorithms for users. The `fit_p` function returns optimal parameters and key performance metrics, including accuracy, log-likelihood, Akaike Information Criterion (AIC), and Bayesian Information Criterion (BIC). Table 3 shows a subset of the example output.
- **estimate*:** Specifies the estimation method. The default is *Maximum Likelihood Estimation* (MLE), which does not incorporate prior distributions. Setting this to *Maximum a Posteriori* (MAP) uses the EM algorithm, following the approach in the sjgershm/mfit MATLAB toolbox.
- **iteration_g*:** Maximum number of EM iterations in MAP estimation.
- **priors* (Same as `dfun` in `rcv_d`):** Prior probability density functions for parameters (only required when `estimate = "MAP"`). Uniform priors can be applied to all parameters, or different priors (e.g., an exponential distribution for the inverse temperature) can be specified. In MAP estimation, priors are reset after each iteration to a normal distribution. This assumption may be suboptimal (The resulting file will only include the log priority and log posterior if `priors` have been set.).
- **tolerance*:** Convergence criterion for the EM algorithm. Iterations stop when the change in log posterior between two iterations is less than 0.001.
- **id*:** Participant identifier(s). Defaults to `NULL`, meaning parameters are fitted for all participants (`id = unique(data$subject)`).
- **n_trials*:** Defaults to `NULL`. Automatically calculates the number of trials per participant. If the dataset contains multiple paradigms with different trial counts, both `id` and `n_trials` should be specified.

- **funcs*:** Required only if custom models modify one of the six core functions and `nc > 1` is set for parallel computing. The function names should be provided as a character vector so the parallel environment can access them.
- **initial_params*:** Initial parameter values for iterative algorithms. Defaults to `NA`, with each parameter initialized at `lower + 0.01`. It can be set manually in the form of a numeric vector if needed.
- **initial_size*:** Initial population size for evolutionary algorithms (default = 50). Larger values can increase computation time.
- **seed*:** Random seed for reproducibility.

After defining necessary arguments, users will obtain the model comparison output. Table 3 shows a subset of the example output. Each row represents the optimal parameters for each participant, as well as the corresponding model fit metrics (e.g., ACC and logL), obtained by fitting different models.

Table 3

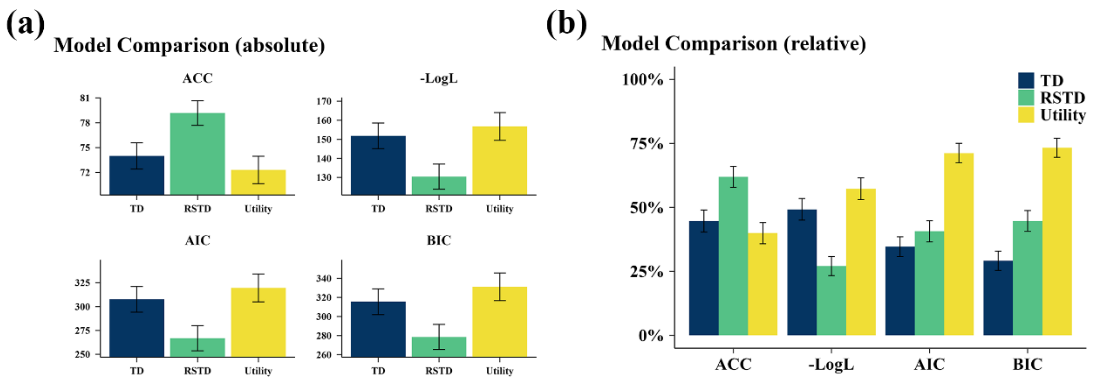
Example Results of Model Comparison

fit_model	Subject	ACC	LogL	AIC	BIC	param_1	param_2	param_3
TD	1.00	76.74	125.29	254.58	261.91	0.01	0.04	NA
TD	2.00	88.89	79.26	162.52	169.85	0.71	0.11	NA
TD	3.00	73.61	186.28	376.56	383.89	0.65	0.04	NA
…	…	…	…	…	…	…	…	…
RSTD	1.00	78.12	126.09	258.18	269.17	0.14	0.07	0.05
RSTD	2.00	94.79	42.05	90.10	101.09	0.66	0.05	0.18
RSTD	3.00	77.78	180.16	366.32	377.31	0.95	0.18	0.04
…	…	…	…	…	…	…	…	…
Utility	1.00	76.74	125.47	256.94	267.93	0.00	0.36	0.37
Utility	2.00	88.89	77.73	161.46	172.45	0.76	0.75	0.26
Utility	3.00	75.69	186.31	378.62	389.61	0.68	0.46	0.23
…	…	…	…	…	…	…	…	…

To facilitate interpretation, model performance across the four-evaluation metrics (ACC, -LogL, AIC, BIC) can be visualized using bar plots (Figure 6). Detailed plotting scripts are available on the package's homepage (https://github.com/yuki-961004/binaryRL).

Figure 6

Model Comparison



Note: Figure 6a shows the absolute values for accuracy (ACC), negative log-likelihood (-LogL), Akaike Information Criterion (AIC), and Bayesian Information Criterion (BIC) across all participants. Figure 6a normalized the results using a min-max scaling procedure, specifically (Value - Minimum) / (Maximum - Minimum), to transform these into relative scores.

Step 4: Latent Variable Analysis and Report Results

In this step, we demonstrate how to use the `rpl_e` function to re-insert optimal parameters into a model and example result suitable for plotting risk preferences across different frames. Here, we use the replication of the TD model as an example. The same approach can be applied to the RSTD and Utility models:

```
list <- list()

list[[1]] <- dplyr::bind_rows(
  rpl_e(
    data = Mason_2024_G2,
    result = read.csv("./result_comparison.csv"),
    model = TD,
    model_name = "TD",
  )
)
```

- **data:** The dataset to be analyzed.
- **result:** The table of fitted parameters obtained from Step 3 (`fit_p`).
- **model:** The model used for replication.
- **model_name:** Name of the model used for replication.
- **param_perfix*:** Column name prefix for the free parameters. Defaults to `"param_"`, if Step 3 completed successfully; typically, no modification is needed.

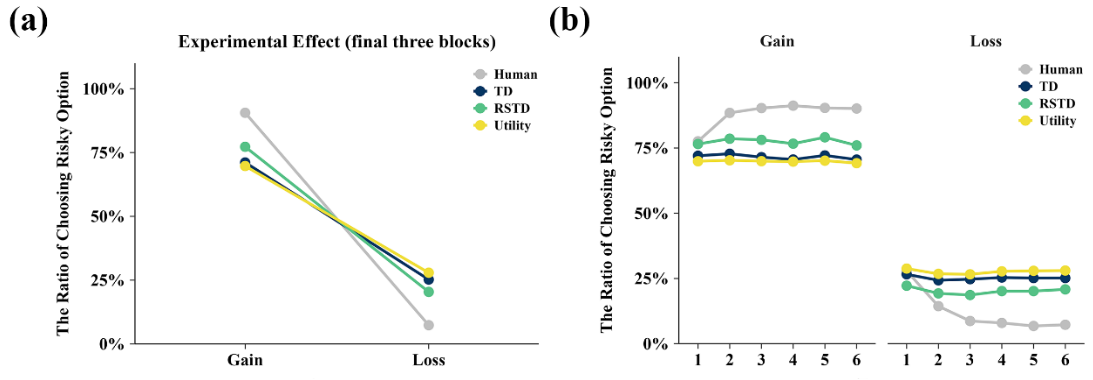
- **n_trials*:** Defaults to `NULL`. The function will automatically compute the number of trials per participant. If the dataset includes multiple paradigms with varying trial counts, `the 'id' and 'n_trials '` should be specified explicitly.

The resulting output contains many columns, which include the original dataset's columns as well as information on the value updates for each option in every trial. Specifically, it includes the current values of the left and right options, the model's predicted probability of choosing either left or right, the human participant's actual choice (left or right), and additional information. Users can view this example output on the program's homepage (see example result).

The subsequent latent variable analysis, which is of interest to many researchers, can be conducted in various ways depending on the research questions. For instance, Figure 7 showed an example of whether the data generated by the model with the best-fitting parameters can reproduce the experimental effects (see example code).

Figure 7

Replay the Experimental Effects



Final Remark

This article introduces binaryRL, an R package that provides a tutorial-based framework for constructing and evaluating reinforcement learning models in two-alternative forced-choice tasks. Following the four-stage workflow of Wilson and Collins (2019), the package demonstrates how raw behavioral data can be transformed into parameterized models that can be validated, compared, and used for theoretical inference. Together with three canonical models (TD, RSTD, Utility) and a straightforward interface for custom functions, the package offers both a ready-to-use toolkit and a flexible foundation for innovation. In addition, by encapsulating the core RL components—value updating, prediction error signaling, and stochastic action selection—within user-friendly functions, binaryRL lowers the technical barrier for researchers without extensive machine learning expertise to build and test their own RL models. Although the tutorial illustrated the workflow using data from the Gambling Paradigm (Mason et al., 2024), the pipeline generalizes to a wide range of TAFC tasks in value-based decision-making, such as learning under uncertainty and cognitive control.

Despite these strengths, *binaryRL* has several limitations that users should be aware of. First, the core function (`run_m`) is limited to two user-specified covariates (var1 and var2) and two free parameters (alpha and beta). This design restricts the inclusion of additional predictors when defining custom models. Second, the core function (`run_m`) is fundamentally designed for tabular reinforcement learning problems. It is therefore not equipped to handle tasks with much more complex environmental dynamics, such as those involving procedural state transitions, which cannot be fully represented in a static table format (e.g., Eckstein & Collins, 2020; Otto et al., 2013). Third, the core function (`run_m`) conceptually structures the TAFC paradigm into six core modeling functions: the utility function, learning rate function, exploration strategy, bias function, choice probability function, and loss function. A key limitation arises if a user's intended model cannot be constructed or adapted to fit within this six-function framework.

It is also crucial to recognize that multiple methodological factors influence the parameter estimation process for computational models. The results presented in this tutorial, generated using Maximum Likelihood Estimation (MLE) with specific optimization algorithms, serve primarily for demonstration. In practice, the final parameter estimates can be substantially affected by: (1) the choice of estimation method itself (e.g., MLE, MAP, MCMC, ABC, RNN), which dictates the approach to the log-likelihood or operates outside of it entirely; (2) whether the inference is simulation-based (e.g., on-policy vs. off-policy); (3) the specific optimization algorithm employed; and (4) the number of fitting iterations. As evidenced by Rmus et al. (2024), advanced methods such as Artificial Neural Networks (ANNs) offer a compelling alternative by bypassing traditional likelihood calculations and leveraging GPU acceleration for both speed and accuracy. In our tests, Approximate Bayesian Computation (ABC) also achieved superior parameter recovery on our example data. In summary, the binaryRL package provides a convenient foundation for MLE and MAP estimation. However, given the potential limitations of log-likelihood-based methods, we recommend that users also explore alternative approaches that bypass the log-likelihood, as demonstrated in the advanced tutorial on our website.

In sum, binaryRL is intended as a tutorialized entry point. Its main contribution lies in providing a transparent, reproducible workflow that lowers the barrier for applying reinforcement learning to TAFC tasks, while leaving ample space for researchers to extend, adapt, or embed the package within more elaborate frameworks. We hope that by bridging methodological rigor with practical accessibility, binaryRL will facilitate broader engagement with computational approaches in psychology and inspire future developments in reinforcement learning and cognitive modeling.

Reference

Ahn, W.-Y., Haines, N., & Zhang, L. (2017). Revealing Neurocomputational Mechanisms of Reinforcement Learning and Decision-Making With the hBayesDM Package. *Computational Psychiatry*, *1*(0), 24. https://doi.org/10.1162/CPSY_a_00002

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *19*(6), 716–723. https://doi.org/10.1109/TAC.1974.1100705

Ardia, D., Mullen, K., Peterson, B., & Ulrich, J. (2005). *DEoptim: Global Optimization by Differential Evolution* (p. 2.2-8). https://CRAN.R-project.org/package=DEoptim

Barraclough, D. J., Conroy, M. L., & Lee, D. (2004). Prefrontal cortex and decision making in a mixed-strategy game. *Nature Neuroscience*, *7*(4), 404–410. https://doi.org/10.1038/nn1209

Bendtsen, C. (2010). *pso: Particle Swarm Optimization* (p. 1.0.4). https://CRAN.R-project.org/package=pso

Bengtsson, H. (2015). *future: Unified Parallel and Distributed Processing in R for Everyone* (p. 1.67.0). https://CRAN.R-project.org/package=future

Bengtsson, H. (2016). *doFuture: Use Foreach to Parallelize via the Future Framework* (p. 1.1.2). https://CRAN.R-project.org/package=doFuture

Bengtsson, H. (2020). *progressr: An Inclusive, Unifying API for Progress Updates* (p. 0.15.1). https://CRAN.R-project.org/package=progressr

Bernoulli, D. (1954). Exposition of a New Theory on the Measurement of Risk. *Econometrica*, *22*(1), 23. https://doi.org/10.2307/1909829

Bischl, B., Richter, J., Bossek, J., Horn, D., Lang, M., & Thomas, J. (2017). *mlrMBO: Bayesian Optimization and Model-Based Optimization of Expensive Black-Box Functions* (p. 1.1.5.1). https://CRAN.R-project.org/package=mlrMBO

Bogacz, R., Brown, E., Moehlis, J., Holmes, P., & Cohen, J. D. (2006). The Physics of Optimal Decision Making: A Formal Analysis of Models of Performance in Two-Alternative Forced-Choice Tasks. *Psychological Review*.

Botvinick, M., Ritter, S., Wang, J. X., Kurth-Nelson, Z., Blundell, C., & Hassabis, D. (2019). Reinforcement Learning, Fast and Slow. *Trends in Cognitive Sciences*, *23*(5), 408–422. https://doi.org/10.1016/j.tics.2019.02.006

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym* (No. arXiv:1606.01540). arXiv. https://doi.org/10.48550/arXiv.1606.01540

Buchsbaum, M., & Stevens, S. S. (1971). Neural Events and Psychophysical Law. *Science*, *172*(3982), 502–502. https://doi.org/10.1126/science.172.3982.502

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, *16*(5), 1190–1208. https://doi.org/10.1137/0916069

Cai, X., Kim, S., & Lee, D. (2011). Heterogeneous Coding of Temporally Discounted Values in the Dorsal and Ventral Striatum during Intertemporal Choice. *Neuron*, *69*(1), 170–182. https://doi.org/10.1016/j.neuron.2010.11.041

Ciranka, S., Linde-Domingo, J., Padezhki, I., Wicharz, C., Wu, C. M., & Spitzer, B. (2022). Asymmetric reinforcement learning facilitates human inference of transitive relations. *Nature Human Behaviour*, *6*(4), 555–564. https://doi.org/10.1038/s41562-021-01263-w

Cools, R., Clark, L., Owen, A. M., & Robbins, T. W. (2002). Defining the Neural Mechanisms of Probabilistic Reversal Learning Using Event-Related Functional Magnetic Resonance Imaging. *The Journal of Neuroscience*, *22*(11), 4563–4567. https://doi.org/10.1523/JNEUROSCI.22-11-04563.2002

Davidow, J. Y., Foerde, K., Galván, A., & Shohamy, D. (2016). An Upside to Reward Sensitivity: The Hippocampus Supports Enhanced Reinforcement Learning in Adolescence. *Neuron*, *92*(1), 93–99. https://doi.org/10.1016/j.neuron.2016.08.031

Eckstein, M. K., & Collins, A. G. E. (2020). Computational evidence for hierarchically structured reinforcement learning in humans. *Proceedings of the National Academy of Sciences*, *117*(47), 29381–29389. https://doi.org/10.1073/pnas.1912330117

Eckstein, M. K., Wilbrecht, L., & Collins, A. G. E. (2021). What do reinforcement learning models measure? Interpreting model parameters in cognition and neuroscience. *Current Opinion in Behavioral Sciences*, *41*, 128–137. https://doi.org/10.1016/j.cobeha.2021.06.004

Eddelbuettel, D., Francois, R., Allaire, J., Ushey, K., Kou, Q., Russell, N., Ucar, I., Bates, D., & Chambers, J. (2008). *Rcpp: Seamless R and C++ Integration* (p. 1.1.0). https://CRAN.R-project.org/package=Rcpp

Ellsberg, D. (1961). Risk, Ambiguity, and the Savage Axioms. *The Quarterly Journal of Economics*, *75*(4), 643–669. https://doi.org/10.2307/1884324

Farrell, S., & Lewandowsky, S. (2010). Computational Models as Aids to Better Reasoning in Psychology. *Current Directions in Psychological Science*, *19*(5), 329–335. https://doi.org/10.1177/0963721410386677

Gaujoux, R. (2011). *doRNG: Generic Reproducible Parallel Backend for "foreach" Loops* (p. 1.8.6.2). https://CRAN.R-project.org/package=doRNG

Gelman, A., Meng, X.-L., & Stern, H. (1996). Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sinica*, *6*, 733–760.

Gershman, S. J. (2015). Do learning rates adapt to the distribution of rewards? *Psychonomic Bulletin & Review*, *22*(5), 1320–1327. https://doi.org/10.3758/s13423-014-0790-3

Gubian, S., Xiang, Y., Brian, S., Hoeng, J., & SA, P. (2011). *GenSA: R Functions for Generalized Simulated Annealing* (p. 1.1.14.1). https://CRAN.R-project.org/package=GenSA

Hampton, A. N., Bossaerts, P., & O'Doherty, J. P. (2006). The Role of the Ventromedial Prefrontal Cortex in Abstract State-Based Inference during Decision Making in

Humans. *The Journal of Neuroscience*, *26*(32), 8360–8367. https://doi.org/10.1523/JNEUROSCI.1010-06.2006

Heitz, R. P. (2014). The speed-accuracy tradeoff: History, physiology, methodology, and behavior. *Frontiers in Neuroscience*, *8*. https://doi.org/10.3389/fnins.2014.00150

Henmon, V. A. C. (1911). The relation of the time of a judgment to its accuracy. *Psychological Review*, *18*(3), 186–201. https://doi.org/10.1037/h0074579

Hertwig, R., Barron, G., Weber, E. U., & Erev, I. (2004). Decisions From Experience and the Effect of Rare Events in Risky Choice. *Psychological Science*, *15*(8), 534–539.

Hoffman, M. W., Shahriari, B., Aslanides, J., Barth-Maron, G., Momchev, N., Sinopalnikov, D., Stańczyk, P., Ramos, S., Raichuk, A., Vincent, D., Hussenot, L., Dadashi, R., Dulac-Arnold, G., Orsini, M., Jacq, A., Ferret, J., Vieillard, N., Ghasemipour, S. K. S., Girgin, S., … Freitas, N. de. (2022). *Acme: A Research Framework for Distributed Reinforcement Learning* (No. arXiv:2006.00979). arXiv. https://doi.org/10.48550/arXiv.2006.00979

Ito, M., & Doya, K. (2009). Validation of Decision-Making Models and Analysis of Decision Variables in the Rat Basal Ganglia. *The Journal of Neuroscience*, *29*(31), 9861–9874. https://doi.org/10.1523/JNEUROSCI.6157-08.2009

Kahneman, D., & Tversky, A. (1979). Prospect Theory: An Analysis of Decision Under Risk. In L. C. MacLean & W. T. Ziemba, *World Scientific Handbook in Financial Economics Series* (Vol. 4, pp. 99–127). WORLD SCIENTIFIC. https://www.worldscientific.com/doi/abs/10.1142/9789814417358_0006

Kahnt, T., Heinzle, J., Park, S. Q., & Haynes, J.-D. (2010). The neural code of reward anticipation in human orbitofrontal cortex. *Proceedings of the National Academy of Sciences*, *107*(13), 6010–6015. https://doi.org/10.1073/pnas.0912838107

Kantowitz, B. H., Roediger, H. L., & Elmes, D. G. (2009). *Experimental psychology* (9th ed). Wadsworth, Cengage Learning.

Krajbich, I., & Rangel, A. (2011). Multialternative drift-diffusion model predicts the relationship between visual fixations and choice in value-based decisions. *Proceedings of the National Academy of Sciences*, *108*(33), 13852–13857. https://doi.org/10.1073/pnas.1101328108

Lau, B., & Glimcher, P. W. (2008). Value Representations in the Primate Striatum during Matching Behavior. *Neuron*, *58*(3), 451–463. https://doi.org/10.1016/j.neuron.2008.02.021

Lee, D., Seo, H., & Jung, M. W. (2012). Neural Basis of Reinforcement Learning and Decision Making. *Annual Review of Neuroscience*, *35*(1), 287–308. https://doi.org/10.1146/annurev-neuro-062111-150512

Liu, Q., Wu, H., & Liu, A. (2019). *Modeling and Interpreting Real-world Human Risk Decision Making with Inverse Reinforcement Learning*. arXiv. https://doi.org/10.48550/ARXIV.1906.05803

Loomes, G., & Sugden, R. (1982). Regret Theory: An Alternative Theory of Rational Choice Under Uncertainty. *The Economic Journal*, *92*(368), 805. https://doi.org/10.2307/2232669

Louie, K. (2022). Asymmetric and adaptive reward coding via normalized reinforcement learning. *PLOS Computational Biology*, *18*(7), e1010350. https://doi.org/10.1371/journal.pcbi.1010350

Ludvig, E. A., & Spetch, M. L. (2011). Of Black Swans and Tossed Coins: Is the Description-Experience Gap in Risky Choice Limited to Rare Events? *PLoS ONE*, *6*(6), e20262. https://doi.org/10.1371/journal.pone.0020262

Mason, A., Ludvig, E. A., Spetch, M. L., & Madan, C. R. (2024). Rare and extreme outcomes in risky choice. *Psychonomic Bulletin & Review*. https://doi.org/10.3758/s13423-023-02415-x

Microsoft, & Weston, S. (2009). *foreach: Provides Foreach Looping Construct* (p. 1.5.2). https://CRAN.R-project.org/package=foreach

Mihatsch, O., & Neuneier, R. (2002). *Risk-Sensitive Reinforcement Learning*. https://doi.org/10.1023/A:1017940631555

Neuser, M. P., Kühnel, A., Kräutlein, F., Teckentrup, V., Svaldi, J., & Kroemer, N. B. (2023). Reliability of gamified reinforcement learning in densely sampled longitudinal assessments. *PLOS Digital Health*, *2*(9), e0000330. https://doi.org/10.1371/journal.pdig.0000330

Niv, Y., Edlund, J. A., Dayan, P., & O'Doherty, J. P. (2012). Neural Prediction Errors Reveal a Risk-Sensitive Reinforcement-Learning Process in the Human Brain. *The Journal of Neuroscience*, *32*(2), 551–562. https://doi.org/10.1523/JNEUROSCI.5498-10.2012

Oba, T., Katahira, K., & Ohira, H. (2021). A learning mechanism shaping risk preferences and a preliminary test of its relationship with psychopathic traits. *Scientific Reports*, *11*(1), 20853. https://doi.org/10.1038/s41598-021-00358-8

Otto, A. R., Raio, C. M., Chiang, A., Phelps, E. A., & Daw, N. D. (2013). Working-memory capacity protects model-based learning from stress. *Proceedings of the National Academy of Sciences*, *110*(52), 20941–20946. https://doi.org/10.1073/pnas.1312011110

Padoa-Schioppa, C., & Assad, J. A. (2006). Neurons in the orbitofrontal cortex encode economic value. *Nature*, *441*(7090), 223–226. https://doi.org/10.1038/nature04676

Palminteri, S., & Lebreton, M. (2022). The computational roots of positivity and confirmation biases in reinforcement learning. *Trends in Cognitive Sciences*, *26*(7), 607–621. https://doi.org/10.1016/j.tics.2022.04.005

Palminteri, S., Wyart, V., & Koechlin, E. (2017). The Importance of Falsification in Computational Cognitive Modeling. *Trends in Cognitive Sciences*, *21*(6), 425–433. https://doi.org/10.1016/j.tics.2017.03.011

Proellochs, N., & Feuerriegel, S. (2017). *ReinforcementLearning: Model-Free Reinforcement Learning*. https://doi.org/10.32614/CRAN.package.ReinforcementLearning

R Core Team. (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing.

Ratcliff, R., & Smith, P. L. (2004). A Comparison of Sequential Sampling Models for Two-Choice Reaction Time. *Psychological Review*, *111*(2), 333–367. https://doi.org/10.1037/0033-295X.111.2.333

Rescorla, R. A., & Wagner, A. R. (1972). A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and non-reinforcement. *Classical Conditioning, Current Research and Theory*, *2*, 64--69.

Reyna, V. F., & Brainerd, C. J. (1995). Fuzzy-trace theory: An interim synthesis. *Learning and Individual Differences*, *7*(1), 1–75. https://doi.org/10.1016/1041-6080(95)90031-4

Rmus, M., Pan, T.-F., Xia, L., & Collins, A. G. E. (2024). Artificial neural networks for model identification and parameter estimation in computational cognitive models. *PLOS Computational Biology*, *20*(5), e1012119. https://doi.org/10.1371/journal.pcbi.1012119

Robbins, H. (1952). Some Aspects Of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, *58*, 527–535. https://doi.org/10.1090/S0002-9904-1952-09620-8

Roecker, E. B. (1991). Prediction Error and Its Estimation for Subset-Selected Models. *Technometrics*, *33*(4), 459–468. https://doi.org/10.1080/00401706.1991.10484873

Rosenbaum, G. M., Grassie, H. L., & Hartley, C. A. (2022). Valence biases in reinforcement learning shift across adolescence and modulate subsequent memory. *eLife*, *11*, e64620. https://doi.org/10.7554/eLife.64620

Schwarz, G. (1978). Estimating the Dimension of a Model. *The Annals of Statistics*, *6*(2). https://doi.org/10.1214/aos/1176344136

Scrucca, L. (2012). *GA: Genetic Algorithms* (p. 3.2.4). https://CRAN.R-project.org/package=GA

Spektor, M. S., Gluth, S., Fontanesi, L., & Rieskamp, J. (2019). How similarity between choice options affects decisions from experience: The accentuation-of-differences model. *Psychological Review*, *126*(1), 52–88. https://doi.org/10.1037/rev0000122

Stroop, J. R. (1935). Studies of interference in serial verbal reactions. *Journal of Experimental Psychology*, *18*(6), 643–662. https://doi.org/10.1037/h0054651

Subramanian, A., Chitlangia, S., & Baths, V. (2022). Reinforcement Learning and its Connections with Neuroscience and Psychology. *Neural Networks*, *145*, 271–287. https://doi.org/10.1016/j.neunet.2021.10.003

Sutton, R. S. (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Machine Learning Proceedings 1990* (pp. 216–224). Elsevier. https://linkinghub.elsevier.com/retrieve/pii/B9781558601413500304

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second edition). The MIT Press.

Trautmann, H., Mersmann, O., & Arnu, D. (2010). *cmaes: Covariance Matrix Adapting Evolutionary Strategy* (p. 1.0-12). https://CRAN.R-project.org/package=cmaes

Valentin, S., Kleinegesse, S., Bramley, N. R., Seriès, P., Gutmann, M. U., & Lucas, C. G. (2024). Designing optimal behavioral experiments using machine learning. *eLife*, *13*, e86224. https://doi.org/10.7554/eLife.86224

van Baar, J. M., Nassar, M. R., Deng, W., & FeldmanHall, O. (2022). Latent motives guide structure learning during adaptive social choice. *Nature Human Behaviour*, *6*, 404–414.

Waltmann, M., Herzog, N., Reiter, A. M. F., Villringer, A., Horstmann, A., & Deserno, L. (2023). Diminished reinforcement sensitivity in adolescence is associated with enhanced response switching and reduced coding of choice probability in the medial frontal pole. *Developmental Cognitive Neuroscience*, *60*, 101226. https://doi.org/10.1016/j.dcn.2023.101226

Weng, J., Chen, H., Yan, D., You, K., Duburcq, A., Zhang, M., Su, Y., Su, H., & Zhu, J. (2022). *Tianshou: A Highly Modularized Deep Reinforcement Learning Library*.

Wilson, R. C., & Collins, A. G. (2019). Ten simple rules for the computational modeling of behavioral data. *eLife*, *8*, e49547. https://doi.org/10.7554/eLife.49547

Yechiam, E., & Busemeyer, J. R. (2006). The effect of foregone payoffs on underweighting small probability events. *Journal of Behavioral Decision Making*, *19*(1), 1–16. https://doi.org/10.1002/bdm.509

Ypma, J., & Johnson, S. G. (2011). *nloptr: R Interface to NLopt* (p. 2.2.1). https://CRAN.R-project.org/package=nloptr

Zhang, L., Lengersdorff, L., Mikus, N., Gläscher, J., & Lamm, C. (2020). Using reinforcement learning models in social neuroscience: Frameworks, pitfalls and suggestions of best practices. *Social Cognitive and Affective Neuroscience*, *15*(6), 695–707. https://doi.org/10.1093/scan/nsaa089