

Delphos: A reinforcement learning framework for assisting discrete choice model specification

Gabriel Nova*¹, Stephane Hess², and Sander van Cranenburgh¹

¹CityAI Lab, Transport and Logistics Group, Delft University of Technology, The Netherlands

²Choice Modelling Centre - Institute for Transport Studies, University of Leeds, England

We introduce Delphos, a deep reinforcement learning framework for assisting the discrete choice model specification process. Delphos aims to support the modeller by providing automated, data-driven suggestions for utility specifications, thereby reducing the effort required to develop and refine utility functions. Delphos conceptualises model specification as a sequential decision-making problem, inspired by the way human choice modellers iteratively construct models through a series of reasoned specification decisions. In this setting, an agent learns to specify high-performing candidate models by choosing a sequence of modelling actions, such as selecting variables, accommodating both generic and alternative-specific taste parameters, applying non-linear transformations, and including interactions with covariates, while interacting with a modelling environment that estimates each candidate and returns a reward signal. Specifically, Delphos uses a Deep Q-Network that receives delayed rewards based on modelling outcomes (e.g., log-likelihood) and behavioural expectations (e.g., parameter signs), and distributes this signal across the sequence of actions to learn which modelling decisions lead to well-performing candidates. We evaluate Delphos on both simulated and empirical datasets using multiple reward settings. In simulated cases, learning curves, Q-value patterns, and performance metrics show that the agent learns to adaptively explore strategies to propose well-performing models across search spaces, while covering only a small fraction of the feasible modelling space. We further apply the framework to two empirical datasets to demonstrate its practical use. These experiments illustrate the ability of Delphos to generate competitive, behaviourally plausible models and highlight the potential of this adaptive, learning-based framework to assist the model specification process.

Keywords: Choice modelling, Reinforcement learning, assisted choice model specification, Deep Q-Network

E-mail address: g.nova@tudelft.nl (G. Nova).

1 INTRODUCTION

Discrete Choice Models (DCMs) are widely used to understand individuals' decision-making behaviour, to forecast demand, and to evaluate policies under various scenarios (Hess & Polak 2005; Guevara & Ben-Akiva 2006; Hess & Daly 2024; Mariel et al. 2021; Liebe et al. 2023). These models aim to mathematically represent observed choice processes, which requires analysts to specify a model and estimate its parameters based on well-established behavioural theories and using available data (Nova et al. 2024). Extracting meaningful insights from such models then requires the careful selection of an appropriate specification, an iterative process that is often time-consuming due to the many modelling decisions involved, such as variable selection, functional forms, and interaction terms (Van Cranenburgh et al. 2022). The predominant approach to model specification remains hypothesis-driven, whereby researchers iteratively propose and test several specifications to improve goodness-of-fit while ensuring parsimony and adhering to behavioural principles (Ortelli et al. 2021). However, this process not only becomes combinatorially intractable even with few variables (Beeramoole et al. 2023), but also relies heavily on researchers' prior knowledge and subjective judgment, which may introduce biases and lead to misspecification.

Recent studies explore the use of meta-heuristic algorithms to automate the specification process and assist analysts. These approaches adopt a fundamentally candidate-centric perspective, framing model specification as a combinatorial optimisation that is solved through iterative search over candidate models (Ghorbani et al. 2025). Among these approaches, Paz et al. (2019) apply Simulated Annealing, which explores local neighbourhoods of specifications to improve goodness-of-fit. Rodrigues et al. (2020) introduce a Bayesian framework that uses automatic relevance determination to search for optimal specifications. Ortelli et al. (2021) consider a Variant Neighbourhood Search, applying operators to several neighbourhoods to iteratively enhance local model performance. Beeramoole et al. (2023) propose a bi-level constrained optimisation framework that integrates prior assumptions into the search process to align specification with theoretical expectations. Haj-Yahia et al. (n.d.) use Grammatical Evolution to find specifications that evolve through genetic operators guided by a novel domain-specific grammar. Similarly, Ghorbani et al. (2025) integrate Grammatical Evolution at the upper level with Iterative Singular Value Decomposition at the lower level of a bi-level model to improve the predictive reliability of utility-based choice models.

Across these studies, the common thread is a candidate-centric optimisation paradigm: new specifications are generated by applying search operators to the current best candidates, and progress is measured through improvements in objective functions such as log-likelihood or information criteria. Most approaches maintain a set of high-performing (or Pareto non-dominated) candidates and generate new models by modifying these candidates or exploring their local neighbourhoods. Collectively, this line of work demonstrates both the potential and the growing interest in metaheuristics as tools for supporting discrete choice model specification.

However, this candidate-centric perspective differs from how choice modellers typically refine specifications in practice. Model specification is usually a dynamic and iterative learning process in which choice modellers adapt their strategy based on accumulated outcomes (e.g., repeated non-convergence, poor fit) and adjust which refinements to prioritise or avoid. Even when metaheuristics exploit past candidates, this information is mostly used to perform operations over specifications, rather than to learn a reusable decision rule that selects modelling refinements. This can lead to potential re-exploration of non-promising regions of the search space and, more importantly, limit transferability when the dataset, objective function, or behavioural constraints change, even within the same

application domain.

This paper proposes a novel approach that treats model specification explicitly as a sequential decision-making problem, mimicking how human choice modellers iteratively construct models through a series of reasoned specification decisions. In this adaptive learning approach, model candidates are not generated through local operators; instead, specifications are gradually built from state information and feedback from previous outcomes. Building on the insight that model specification involves multiple sequences of trial-and-error decisions, we introduce a novel reinforcement learning framework for automating the specification of discrete choice models. Reinforcement learning has proven effective in domains that involve large combinatorial design spaces and delayed feedback—such as neural architecture search, where agents learn to sequentially construct high-performing models without relying on hand-crafted heuristics (Zoph, 2016; Baker et al., 2016). Motivated by these successes, we formulate the specification process as a Markov Decision Process (MDP) and implement a Deep Q-Network (DQN) based agent that learns to propose model candidates aligned with behavioural expectations through sequential interactions. To enable this, we develop a custom modelling environment that estimates candidate specifications using Apollo functions (Hess & Palma, 2019) and returns user-predefined rewards. Rather than relying on fixed operators, the RL agent learns which actions are most likely to lead to promising outcomes, with the learning objective encoded in a custom reward function specified by the analyst. Through continuous learning, RL thus provides a flexible framework for supporting the model specification process.

The remainder of the paper is organised as follows. Section 2 introduces the main reinforcement learning components of our framework (Subsection 2.1) and formulates model specification as a Markov Decision Process (Subsection 2.2), defining the state space, action space, agent architecture, modelling environment, and reward function. Section 3 describes the experimental design, including simulated and empirical case studies, while Section 4 presents the results. Finally, Section 5 discusses the main insights, methodological implications, limitations, and potential extensions.

2 RESEARCH FRAMEWORK

To employ reinforcement learning into the choice model specification challenge, this section revisits the RL paradigm as the foundation for our framework. Section 2.1 provides an overview of the core components of RL and their role in defining the agent-environment interaction. Then, Section 2.2 introduces our proposed RL framework, in which we formulate the model specification as a Markov Decision Process that enables the agent-environment interaction.

2.1 Revisiting reinforcement learning paradigm

Reinforcement learning is a machine learning paradigm that focuses on sequential decision-making problems, wherein an agent interacts with an environment to learn which actions to take through trial-and-error to maximise accumulated long-term rewards (Bertsekas, 2019). Unlike supervised learning, which relies on fixed datasets to train models on predefined input-output mappings, RL trains agents to adapt their behaviour in uncertain environments based on data gained through agent-environment interaction. Thus, this process is typically formalised as a Markov Decision Process (Bellman, 1957), where the agent observes the current state of the environment, selects action(s), transitions to a new state, and receives rewards. This feedback guides the agent in improving its policy (π_θ), which maps states to a probability distribution over actions and governs the agent’s behaviour (Burnetas & Katehakis, 1997; Littman, 1994).

An MDP framework is defined as a tuple $(\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \gamma)$, where the state space (\mathbf{S}) contains all possible conditions of the environment; the action space (\mathbf{A}) defines the set of actions or operators that the agent can apply to transition from the current state to a new one; the transition probability ($\pi_\theta(s_e, a_e) = P(s_{e+1} | s_e, a_e)$) models how the environment evolves from one state s_e to another s_{e+1} based on the agent’s action a_e ; the reward function $R(s_e, a_e)$ reflects the quality of an action a_e taken in a given state s_e ; and the discount factor (γ) balances future cumulative rewards relative to intermediate ones. The agent’s objective is then to learn an optimal policy (π_θ^*), parametrised by θ , that maximises the expected cumulative reward ($J(\pi_\theta)$). This policy is a mapping function that determines which actions should be taken at each state based on a probability distribution over the feasible action space. To achieve optimal decision-making, the agent balances exploration (sampling new actions to improve its knowledge of the environment) and exploitation (choosing actions that have previously yielded high rewards). Therefore, at each interaction, namely episode e , the agent observes the current state s_e , selects an action a_e , receives a reward r_e , transitions to a new state s_{e+1} , and updates the policy π_θ accordingly, as shown in Figure 1.

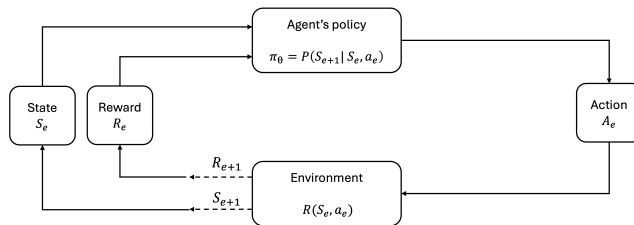


Figure 1: Classical RL framework. Adaptation from Sutton & Barto (2018)

Traditional approaches have relied on value-based methods, such as Q-learning (Watkins & Dayan, 1992), and policy-based methods, such as Policy Gradient (Sutton, McAllester, et al. 1999), for learning the optimal policy (Graesser & Keng (2019)). On the one hand, value-based methods estimate an optimal value function ($V(s)$), which represents the expected cumulative reward that an agent can achieve from a given state by following a particular policy. Instead of directly learning the policy itself, these methods derive it by selecting actions that maximise the state–action value function ($Q(s, a)$) (Wong et al. 2023), which evaluates the long-term returns of choosing action a_e in state s_e :

$$\pi_\theta^*(s) = \arg \max_{\pi} V^\pi(s) = \arg \max_{a, \pi} Q(s, a) \quad \Rightarrow \quad a^* = \arg \max_{a \in A} Q^*(s, a) \quad (1)$$

On the other hand, policy-based methods directly parametrise the policy function π_θ and optimise it by adjusting the parameters θ to maximise the expected cumulative rewards (Sutton & Barto, 2018; Schulman et al. 2017, 2015). Unlike value-based methods, these methods adjust their policy using gradient ascent, where the parameters are iteratively updated in a direction that increases the likelihood of selecting high-value actions (Sutton, Precup, & Singh, 1999). The policy gradient theorem (Sutton, McAllester, et al. 1999) formalises this update rule, as shown in Eq. (2), where $\nabla_\theta \log \pi_\theta(a | s)$ indicates the direction in which to adjust the policy to increase the likelihood of selecting action a at state s , while the action–value function $Q(s, a; \theta)$ quantifies the expected return of that action under the current policy, in other words, it reflects how valuable the action is (Schulman et al., 2017).

$$\nabla_\theta J(\theta) = \mathbb{E} [\nabla_\theta \log \pi_\theta(a | s) \cdot Q(s, a; \theta)] \quad (2)$$

Traditional RL methods are computationally infeasible in environments with high-dimensional or dynamically evolving state or action spaces, as these methods rely on maintaining an explicit Q-value table that stores expected rewards for all possible state-action pairs (Plaat

2022). To address these limitations, Deep Reinforcement Learning was introduced, which uses deep neural networks to approximate value functions or policies (Mnih et al. 2016; Wang et al. 2016).

The Deep Q-learning algorithm was the pioneering approach, which not only introduced a Deep Q-Network (DQN) to replace the Q-value function using a neural network, but also integrated an experience replay buffer and a target network mechanism to stabilise the training process (Mnih et al. 2015), as shown in Figure 2. Specifically, the agent approximates state-action values $Q(s, a, \theta)$, which estimates the expected cumulative reward of taking an action a at state s under a policy $\pi_\theta = P(a|s)$, as shown Eq. (3). This approximation is made using a fully connected neural network, where the parameters θ are learnt through training. The network encodes abstract representations of states and uses them to calculate Q-values for all possible actions. At each step, the agent selects the action with the highest Q-value to transition to a new state by interacting with the environment.

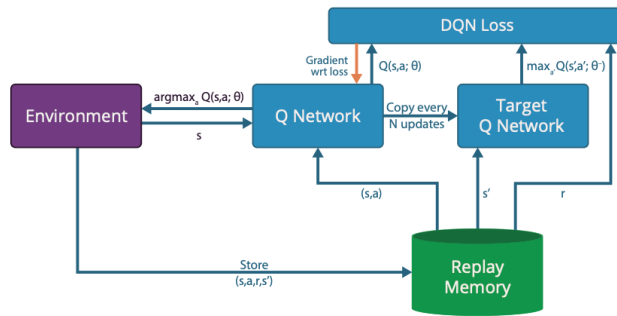


Figure 2: The Deep Q-learning algorithm from Nair et al. (2015). It includes three components: (1) the Q-network $Q(s, a; \theta)$, which estimates action values based on the current policy; (2) the target Q-network $Q(s, a; \theta^-)$, a periodically updated copy used to compute stable target values; and (3) the replay memory, which stores past transitions and enables training on random mini-batches to reduce temporal correlations and stabilise learning.

$$Q(s, a; \theta) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a, \pi = \pi_\theta \right] \quad (3)$$

To improve and stabilise the learning, the agent uses both an experience replay buffer and a target neural network. This buffer stores past interactions as tuples (s, a, r, s') , capturing current state, action taken, reward received, and subsequent state. During training, the agent samples from this buffer randomly rather than relying on consecutive experiences to update the policy weights, which breaks temporal correlation that can destabilise action-value updates (Lin, 1992; S. Zhang & Sutton 2017). In addition, the target network is used to stabilise Q-value estimation and reduce large oscillations during weight updates (Mnih, 2013; Ly et al. 2024). Instead of using a single network to estimate the current and future Q-values, the target network maintains a separate copy of the policy weights, which are periodically updated. As shown in Eq. (4), while the policy network $Q(s, a; \theta)$ is updated at every training step (episode), the target network $Q(s', a'; \theta^-)$ keeps its parameters fixed for a number of training steps before synchronising with current policy weights (Tsitsiklis & Van Roy 1996; Mnih et al. 2015).

$$Q_{target}(s', a'; \theta^-) = R(s, a) + \gamma \max_{a'} Q(s', a'; \theta^-) \quad (4)$$

To train the agent and learn an optimal policy, the policy weights are iteratively updated using past interactions to reduce the distance between the predicted Q-values and their target values, following a temporal difference learning approach (Sutton & Barto, 2018). The objective is then to minimise the loss function (Eq. (5)), which is defined as the mean

squared error between the current Q-value and the target value, averaged over a mini-batch sampled from the experience replay buffer (\mathcal{D}). As shown in Eq. (6), the gradient of the loss function $\nabla_{\theta}\mathcal{L}(\theta)$ guides the update direction, and the learning rate α controls the step size.

$$\mathcal{L}(\theta) = \mathbb{E}(s, a, r, s') \sim \mathcal{D} \left[(Q_{target}(s', a'; \theta^-) - Q(s, a; \theta))^2 \right] \quad (5)$$

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta) \quad (6)$$

The DQN agent is enabled to solve this type of learning task not only by sampling past interactions to update its policy weights, but also by relying heavily on an exploration strategy (Mnih et al., 2015). This strategy controls how the agent balances the trade-off between exploiting areas of the state space known to yield high rewards and exploring new regions to avoid becoming trapped in local optima. In our framework, the agent follows a so-called ϵ -greedy policy, which selects a random action with probability ϵ and the action with the highest predicted Q-value with probability $1 - \epsilon$, which decreases linearly over episodes. The ϵ -greedy exploration strategy is simple yet effective, as it promotes early exploration during training and gradually shifts towards more informed decisions as the agent accumulates experiences and updates its policy (Dabney et al., 2020).

2.2 Automating discrete choice model specification using RL methods

The proposed reinforcement learning framework formalises the agent–environment interaction to automate the search process for discrete choice models. This interaction is modelled as a Markov Decision Process, in which Delphos uses a Deep Q-Network to sequentially take actions that propose a model specification. The environment then estimates the resulting candidate and returns feedback based on the modelling outcomes, as depicted in Figure 3

Delphos autonomously builds model specifications by selecting variables, applying transformations, incorporating interactions with covariates, or deciding when to estimate the model. Once a candidate specification is submitted, the environment estimates it, stores the corresponding modelling outcomes, and returns a reward signal that guides the agent’s learning process. Through multiple episodes, the agent learns to specify models by balancing exploration of new candidates with exploitation of well-performing ones to maximise cumulative rewards.

2.2.1 Problem formulation

We translate the model specification task as a sequential decision-making problem within an MDP framework, where the agent learns to propose well-performing models—for instance, those that achieve high log-likelihood values and satisfy behavioural expectations such as expected parameter signs—by interacting with an estimation environment. At each episode, the agent takes a sequence of actions to build a specification and explore the modelling space, while the environment estimates the proposed candidate and returns a reward signal.

The environment contains a choice dataset that the agent can interact with. This dataset consists of N observations representing decision-makers who face a choice situation among J alternatives. Each alternative is described by K attributes, $X = \{x_{nj1}, \dots, x_{njK}\}$, such as travel time, travel cost, or comfort. Additionally, it may include individual-specific covariates, $C = \{cov_{n1}, \dots, cov_{nC}\}$, capturing socio-demographics or contextual information such as income, gender, or household size. The observed outcome y_n indicates the alternative chosen by individual n .

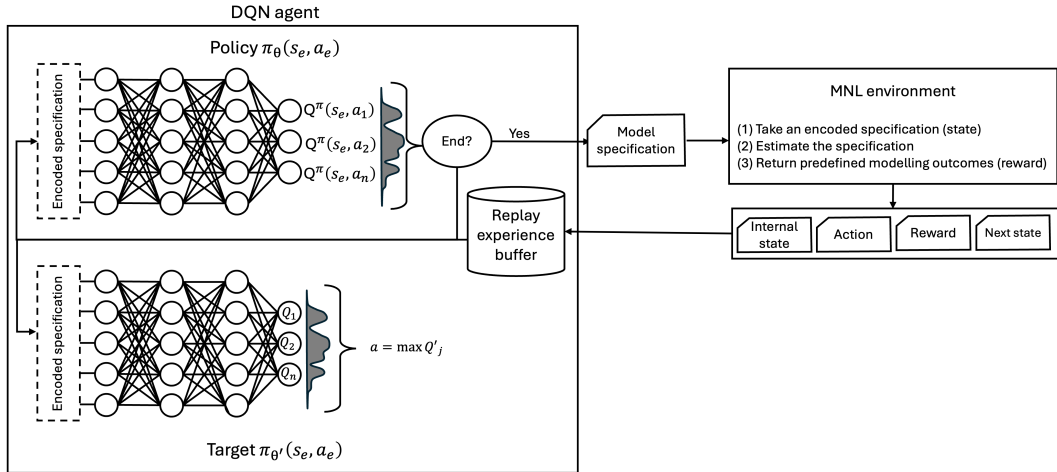


Figure 3: Framework for assisting discrete choice model specification. While Delphos encodes the current specification, selects a modelling action, and submits the model for estimation, the environment returns modelling outcomes as a reward signal. Transitions are stored in a replay buffer and used to update the policy and target networks.

While the current implementation of Delphos focuses on utility-based discrete choice models—where individuals are assumed to select the alternative with the highest utility—it can be readily extended to accommodate alternative decision rules, such as regret minimisation (Chorus, 2010; van Cranenburgh et al., 2015), satisficing (Swait, 2001), or rule-based approaches (Hess & Stathopoulos, 2013). In utility-based models, individual n 's utility U_{ni} for alternative i is determined by a deterministic component (V_{ni}), derived from observed attributes, covariates, and their associated parameters, and an error term (ε_{ni}) that captures unobserved factors, as shown in Eq. (7).

$$U_{ni} = V_{ni} + \varepsilon_{ni}, \quad (7)$$

This agnostic formulation allows our RL framework to be used with standard choice models, such as the Multinomial Logit, Generalised Extreme Value models (McFadden, 1974), Mixed Multinomial Logit (McFadden & Train, 2000), and Latent Class models (Walker & Li, 2007), among others. In this paper, we implement the modelling environment using the Multinomial Logit and extensions are left for future work. However, the chosen model family not only directly influences the feasible modelling space but also impacts the state and action spaces. To define the modelling space, an analyst may specify a set of modelling space parameters:

1. **Variables:** A list of alternative-specific constants and attribute names that may enter into the utility specification, which is denoted as $[ASC, att_1, att_2, \dots, att_K]$.
2. **Transformations:** A predefined set of functional forms that can be applied to attributes to capture potential non-linearities. This list is denoted as $[f_1, f_2, \dots, f_T]$, where T is the number of transformations. Among them are:
 - Linear: $f_1(x) = x$
 - Logarithmic: $f_2(x) = \log(1 + x)$
 - Box-Cox: $f_3(x) = (x^\lambda - 1)/\lambda$, where λ is an extra estimated parameter.
3. **Taste parameters:** A list that indicates whether the attributes' parameters are considered either generic, alternative-specific taste parameters, or both. While a generic taste parameter is shared across all alternatives, an alternative-specific taste parameter varies across alternatives. This list is denoted as [generic, specific].

4. **Covariates:** A list of variables that can be used to define interactions with alternative-specific constants or attributes to capture observed heterogeneity. This list is denoted as $[cov_1, cov_2, \dots, cov_C]$.

For instance, a modelling space that enables non-linear transformations to attributes, generic and alternative-specific taste parameters, and allows interactions with covariates, may be defined as:

$$\text{modelling space} = \begin{cases} \text{variables} & = [\text{asc}, att_1, \dots, att_K] \\ \text{transformations} & = [\text{linear}, \text{logarithm}, \dots, f_t] \\ \text{taste} & = [\text{generic}, \text{specific}] \\ \text{covariates} & = [cov_1, \dots, cov_C] \end{cases} \quad (8)$$

Given this problem and the defined modelling space, we now formalise the elements that govern the decision-making process and the agent itself for specifying discrete choice models. These elements include the state space (Subsection 2.2.2), the action space (Subsection 2.2.3), the reward function (Subsection 2.2.4), the environment (Subsection 2.2.6), and the agent (Subsection 2.2.5).

2.2.2 The state space

The state space represents the evolving model specification at each intermediate step of the agent’s decision process until it is submitted for estimation. We encode states as lists of modelling terms (tuples), each describing a selected variable and its associated transformation, taste parameter type, and covariate interaction. This concise yet flexible representation facilitates the application of any action in any order, thereby enabling the agent to propose any model within the modelling space. Formally, a state S_e^l represents the model specification up to the action taken at intermediate step l within episode e , which is defined as:

$$S_e^l = \{(k_1, t_1, g_1, i_1), \dots, (k_l, t_l, g_l, i_l)\} \quad (9)$$

Where k_i denotes the attribute index, t_i represents its transformation (e.g., linear, logarithmic, or Box-Cox), g_i indicates whether the attribute is considered with a generic or alternative-specific taste parameter, and i_i specifies whether the attribute interacts with a covariate. Future extensions could expand these tuples, defining the model family (M), the categorical treatment for ordinal attributes (c_i), or even the parameter distributions (d_i), among others.

2.2.3 The action space

The action space defines all feasible operations that the agent can perform to modify the current state (i.e., the current model specification). These include adding new variables, changing existing specification components, or terminating the specification process and submitting the model for estimation. Thus, at each intermediate step l during episode e , the agent selects sequential actions to modify any component of the list of modelling terms that define the state S_e^{l-1} and to transition to S_e^l . To prevent redundant or infeasible actions (e.g., adding an attribute already included in the specification), the set of valid actions is dynamically masked based on the current intermediate state (i.e., we assign extremely negative Q-values to these actions, making them unlikely to be selected). This masking mechanism ensures that the agent takes meaningful actions (e.g., changing a component of an already added attribute instead of adding it again), which improves learning efficiency but makes the action space dynamic (Huang & Ontañón 2020). Formally, the agent can perform any of the following actions:

- Add a new encoded attribute to expand the model. While alternative-specific constants are included without interactions and one of them is fixed, the attributes are linearly incorporated with the generic taste parameter across alternatives and without interactions.

$$\text{Add}(S_e^l, k^l, t^l, g^l, i^l) \rightarrow S_e^{l+1} = S_e^l \cup (k^l, 1, 1, 0) \forall k^l \notin S_e^l \quad (10)$$

- Change a component of an existing variable in the model. This action allows modifications to any element that defines the modelling term tuple, such as its transformation, taste parameter specification, or interaction with a covariate. The equation below shows a modification in the transformation component applied to the attribute k_1^* , from t to \bar{t} , while keeping all other components unchanged:

$$S_e^{l+1} = (S_e^l \setminus (k_1^l, t^l, g^l, i^l)) \cup (k_1^{l+1}, \bar{t}^{l+1}, g^{l+1}, i^{l+1}) \quad (11)$$

- Terminate the model specification process and submit the specification for its estimation.

$$\text{Terminate}(S_e^l) \rightarrow S_e^{l+1} = S_e^l \cup \{\text{terminate}\} \quad (12)$$

As with the state space, future work could expand the range of actions to include more operations, such as defining interactions between attributes themselves, considering categorical treatment, or adjusting parameter distributions, among others.

2.2.4 The reward function

The reward function translates the modeller’s goals into a learning signal that guides the agent’s behaviour. For Delphos, this signal not only provides delayed feedback on the quality of an entire sequence of actions, but also encodes which aspects of the model performance the analyst aims to prioritise. Depending on the context, these priorities may emphasise goodness-of-fit, parsimony, behavioural plausibility, or a combination of criteria. In this sense, the reward formulation is a deliberate design choice that shapes the behaviour (policy) the agent eventually learns (Dewey, 2014).

Similar to a human choice modeller’s workflow, Delphos sequentially modifies a model specification and, once it selects the terminate action, the candidate is estimated and the corresponding modelling outcomes are obtained. These include, but are not limited to, log-likelihood (LL), McFadden’s pseudo- R^2 metrics (ρ^2 and $\bar{\rho}^2$), Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), parameter values, standard errors, number of parameters, convergence flag, number of free parameters, and estimation time. Because these outcomes are available only after a complete specification has been estimated, the agent receives delayed feedback and must learn to associate earlier decisions with their eventual impact on model performance.

To support learning under delayed feedback, Delphos distributes the final episodic reward (R_e) backwards across the sequence of L actions taken during the episode. Temporal discounting is applied to address this temporal credit assignment problem (Kaelbling et al., 1996), where actions taken closer to termination receive greater credit, while earlier actions still contribute meaningfully to the final results when intermediate rewards are not directly available (Sutton & Barto, 2018; Zhong et al., 2018). This not only allows the agent to learn how individual decisions within a sequence of modelling actions contribute to modelling performance, but also concise action sequences. Specifically, the reward received at intermediate step ℓ ($\ell = 1, \dots, L$) of episode e is discounted by the agent’s discount factor γ :

$$r_e^\ell = \gamma^{L-\ell-1} \cdot R_e \quad (13)$$

As human modellers often rely on multiple modelling outcomes to guide the specification process, Delphos can consider several criteria simultaneously. A modeller may, for example, aim to maximise both in-sample and out-of-sample log-likelihood to mitigate overfitting or balance goodness-of-fit against parsimony or behavioural plausibility (Nova et al., 2024). To represent these different analyst profiles, the episodic reward (R_e) is defined as a configurable aggregation of selected modelling outcomes. To preserve comparability across metrics with different scales, Delphos normalises each outcome and computes a weighted sum reflecting the analyst’s priorities. Formally, the reward at episode ℓ is given by:

$$r_e^\ell = \gamma^{L-\ell-1} \cdot \left[\sum_{m \in M} \omega_m \cdot \tilde{M}_m \right] \cdot \mathbf{1}_{\text{converged}} \quad (14)$$

where M denotes the set of modelling outcomes and ω_m their corresponding weights. The term \tilde{M}_m is the min-max normalised value of outcome m , updated dynamically based on the best and worst values observed by the agent so far. This dynamic scaling ensures a stable learning signal and avoids issues such as reward explosion or vanishing gradients (Mnih et al., 2015; Van Hasselt et al., 2016). The convergence indicator $\mathbf{1}_{\text{converged}}$ ensures that successfully estimated models contribute to the learning signal, while convergence issues are treated as failures. This implies that non-convergent specifications receive a reward of zero, clearly signalling undesirable modelling outcomes to the agent. However, such failures are also informative and remain part of the learning process, especially in early training episodes, as they help the agent avoid poor modelling decisions. This treatment aligns with RL findings showing that exposure to suboptimal workflows and recovery failures helps improve long-term performance (Ross & Bagnell, 2010). As a result, episodic rewards range from 0 for non-convergent or poor-performing models to 1 for the best-performing specification observed so far.

Normalisation is performed using theoretically meaningful bounds. For log-likelihood, the null log-likelihood (LL_0) defines the minimum value¹:

$$\overline{LL} = \begin{cases} 0 & \text{if } LL \leq LL_0 \\ \frac{LL - LL_0}{LL_{\max} - LL_0} & \text{otherwise,} \end{cases} \quad (15)$$

where LL_{\max} is the highest log-likelihood observed so far. For information criteria, we similarly define the worst case value using the null log-likelihood (i.e., $-2 \cdot LL_0$ with $K = 0$), and dynamically track the best observed (i.e., lowest) value:

$$\overline{AIC} = \begin{cases} 0 & \text{if } M \geq -2 \cdot LL_0 \\ \frac{-2 \cdot LL_0 - AIC}{-2 \cdot LL_0 - AIC_{\min}} & \text{otherwise,} \end{cases} \quad (16)$$

where M_{\min} is the lowest AIC value observed so far. The same logic applies to obtain the normalised BIC (\overline{BIC}). McFadden’s pseudo- R^2 requires no transformation, as it is already bounded between 0 and 1.

In addition to model performance, Delphos can incorporate behavioural constraints to align its learning with behavioural and theoretical expectations. For instance, the reward function can be designed to enforce expected negative sensitivities on utility functions, such as penalising specifications in which travel time coefficients are not negative. This ensures consistency with microeconomic theory. These constraints are implemented through

¹Although a model may perform worse than the null, such solutions would not be considered valid maximum-likelihood estimates (Train, 2009; Mokhtarian, 2016).

an additional indicator, $\mathbb{1}_{\text{behavioural expectations}}$, which sets the reward to zero whenever a behavioural expectation is violated based on the estimation results:

$$r_e^\ell = \gamma^{L-\ell-1} \cdot \left[\sum_{m \in \mathcal{M}} \omega_m \cdot \tilde{M}_m \right] \cdot \mathbb{1}_{\text{converged}} \cdot \mathbb{1}_{\text{behavioural expectations}} \quad (17)$$

This flexible design allows the reward function to reflect different profiles of choice modellers. A modeller focused on model fit may emphasise log-likelihood or pseudo- R^2 , while one concerned with complexity may prioritise information criteria or a balance between model fit and number of parameters. When behavioural plausibility is also required, penalties for behavioural violations guide the agent not only towards well-performing models, but also theoretically consistent ones.

Finally, future work could improve this framework by incorporating other domain-specific constraints. For instance, hypothesis testing against the null could be used to penalise model specifications in which most parameters are statistically insignificant. Thus, incorporating these tests would enable the agent to prioritise statistically robust models, which further aligns the learning process with common practices in discrete choice modelling.

2.2.5 The agent

Delphos follows a DQN architecture, enabling it to learn a policy that selects actions with the highest Q-values to transition between states through interaction with the environment (Mnih et al., 2015). As displayed in Figure 3, Delphos uses a DQN to encode model specifications as state representations, select actions accordingly, interact with the environment to estimate the specified models, and store its experiences in the replay buffer for training. To illustrate this process, consider a transport mode choice scenario involving different alternatives and attributes: travel time (TT), travel cost (TC), headway (HW), and number of changes (CH). In this example, Delphos' objective is to specify a choice model with fully generic taste parameters, as shown in Figure 4

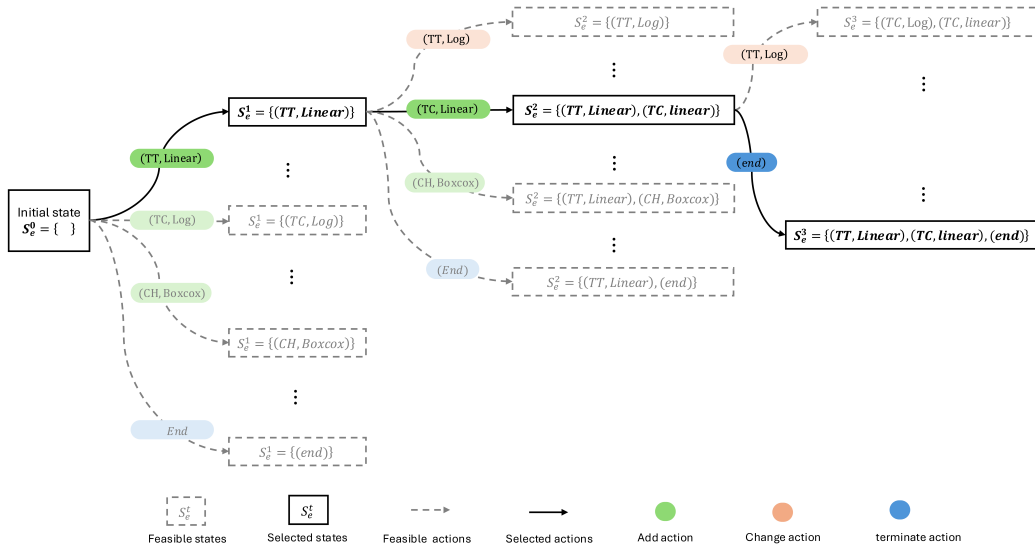


Figure 4: Agent's sequential decision-making process. Nodes represent internal states (current specifications). Dashed arrows indicate feasible actions, while solid arrows show the actions selected by the agent until termination.

Delphos begins each episode with an empty state ($S_e^0 = \{\emptyset\}$), which represents an unspeci-

fied model. As the episode progresses, the agent iteratively updates the internal state (S_e^l) by selecting attributes, applying transformations, and incorporating interactions. Then, the encoded state is passed through the input layer of the DQN, where the policy network processes it using fully connected hidden layers to learn abstract representations. The output layer provides Q-values for all feasible actions (masked actions), from which the agent selects an action. Once Delphos chooses the terminal action, the environment estimates the candidate model and returns the corresponding modelling outcomes. These estimated models and trajectories (the internal states, selected actions, intermediate discounted rewards, subsequent internal states) are stored as tuples $(S_e^l, a_e^l, r_e^l, s_e^{l+1})$ in a replay buffer, preventing redundant estimation and enabling the agent to build on past modelling outcomes as it refines its policy. To do so, Delphos samples mini-batches from this buffer to update the policy network parameters, thereby refining its decision-making over time, as described in section [2.1](#)

While Delphos is designed to learn a policy for sequential model specification through iterative interactions, it can also be flexibly adapted for automation purposes where the primary objective is to identify high-performing specifications with a limited amount of time. In the former case, the agent is considered to have learnt a stable policy once the reward signal reaches a plateau, which indicates no substantial improvement despite continued exploration. In the latter case, the agent should explore the modelling space effectively within a limited number of episodes instead. To prioritize modellers’ time in the second case, we implement an early stopping mechanism based on both a rolling convergence window and an absence of newly proposed best-performing models. Specifically, the training process terminates if, over the last E_{search} episodes, the rolling mean of the episodic reward has stabilised, and no better model specification has been found in the last E_{search} episodes. These conditions are only evaluated once a minimum number of episodes, E_{min} , has been surpassed. To achieve this, Delphos tracks the best-performing candidates observed so far and its modelling outcomes $\{M_1 : (S_{e^*}, R_{S_{e^*}}), \dots, M_m : (S_{e^*}, R_{S_{e^*}})\}$ across training. If no improvement is observed over E_{search} consecutive episodes, and $e > E_{\text{min}}$, the process is terminated. This criterion ensures that the agent has had sufficient opportunity to explore the modelling space. The full training process is described in Algorithm [1](#)

2.2.6 The environment

The environment handles model estimation and returns the associated modelling outcomes². It is initialised with a choice dataset and remains fixed throughout all agent interactions. During each episode, the environment receives encoded state representations from the agent, decodes them into utility specification, performs the estimation process, and returns the modelling outcomes required to calculate the reward signal. Built using Apollo functions ([Hess & Palma 2019](#)), the environment meets the OpenAI Gym requirements ([Brockman 2016](#)) that ensure compatibility with a range of RL algorithms. Importantly, the reward function is deliberately isolated from the environment. This design provides modellers with flexibility to define custom reward signals, including the integration of behavioural constraints (e.g., penalising unexpected coefficient signs) and the use of multiple modelling outcomes, as discussed previously.

3 EXPERIMENTS

To evaluate the capabilities of Delphos, we design a sequence of experiments that progress from simulated scenarios to empirical cases. The simulated experiments assess the agent’s ability to recover known data-generating processes, navigate modelling spaces, and learn

²We have made the code public in a GitHub repository: <https://github.com/TUD-CityAI-Lab/Delphos>

a stable policy that yields high rewards. We then benchmark Delphos against a Variable Neighbourhood Search algorithm (VNS) (Ortelli et al. (2021)) using the Swissmetro dataset (Bierlaire et al. (2001)) in a realistic empirical setting where model performance and computational efficiency are relevant. Finally, we apply Delphos to the DECISIONS dataset (Calastrì et al. (2020)) to evaluate whether a modeller-driven reward that combines model fit, parsimony, and behavioural plausibility, can guide the agent to propose competitive specifications that satisfy these constraints set by the analyst. Across all experiments, each dataset is split into in-sample (80%) and out-of-sample (20%) subsets. The agent is trained solely on the in-sample set, where it iteratively explores the modelling space and refines its decision-making process (policy). Once training is completed, the resulting candidate specifications are evaluated out-of-sample to assess generalisability to unseen data.

We use multiple experimental configurations that vary in reward formulation and early stopping criteria. These configurations serve two complementary purposes. First, they allow us to reflect the diversity of modeller goals observed in empirical choice modelling workflows (Nova et al. (2024)), including preferences for model fit, parsimony, and behavioural expectations. Second, they enable us to investigate the agent’s ability to identify competitive specifications within a restricted number of episodes, rather than requiring full policy convergence. Overall, these settings allow us to evaluate how effectively the agent balances exploration and exploitation, to assess the robustness of its learnt policy across simulated datasets, and to propose competitive models under different modelling goals and computational constraints in empirical cases.

3.1 Simulated datasets

We first evaluate Delphos in controlled conditions using three simulated choice datasets based on Multinomial Logit models. Each dataset contains 4,000 individuals choosing among 3 alternatives (A, B, C), defined by K predefined attributes (x_1, x_2, \dots, x_k) . We also simulate C socio-demographic variables (cov_1, \dots, cov_C) to introduce observed systematic heterogeneity. Table 1 shows the true utility specifications³ and the modelling space available to the agent for each case. In all cases, the agent is trained using the adjusted McFadden’s pseudo- R^2 as the reward signal. Additionally, learning parameters are varied to evaluate the sensitivity of the agent’s performance to different hyperparameter settings (Appendix D). This setup enables us to investigate whether the agent is effectively learning, and whether its policy concentrates on high-reward regions of the modelling space, and whether it is capable of recovering the true data-generating process under different levels of complexity.

3.2 Empirical datasets

3.2.1 Swissmetro (Experiment 2)

We next analyse Delphos in an empirical case using the Swissmetro dataset, a stated preference dataset designed to evaluate the introduction of a new high-speed public transport system (for more details, see Antonini et al. (2007)). The dataset contains 1,192 respondents, each facing nine hypothetical mode choice tasks among train, swissmetro, and car. Alternatives are defined by travel time (TT), travel cost (TC), headway (HE), and seat configuration (SE). It also includes socio-demographic variables such as age, income, travel class, possession of an annual season ticket (GA), number of suitcases (LUGGAGE), gender (GENDER), and the person or employer responsible for paying the ticket (WHO).

For this experiment, we compare the best performing models proposed by Delphos with

³True parameter values are provided in Appendix C

Table 1: Utility specifications and modelling spaces for simulated choice datasets (S_1 – S_3).

Experiment	S_1		S_2		S_3		
Variable	Transf.	Taste	Transf.	Taste	Transf.	Taste	Covariate
ASC ₁	Linear		Linear		Linear		cov ₁
ASC ₂	Linear		Linear		Linear		cov ₁
x_1	Linear	Generic	Log	Specific	Linear	Specific	
x_2	Log	Generic	Box-Cox	Generic	Log	Specific	cov ₂
x_3	Linear	Generic	Linear	Specific	Linear	Generic	
x_4	Box-Cox	Generic	Linear	Generic	Box-Cox	Generic	
x_5	Linear	Generic			Linear	Generic	
x_6	None						
Space	None Linear Log Box-Cox	Generic	None Linear Log Box-Cox	Generic Specific	None Linear Log Box-Cox	Generic Specific	None cov ₁ cov ₂

those obtained by the Variable Neighbourhood Search in terms of AIC. Both approaches use the same set of attributes and covariates, and the same in-sample and out-of-sample splits. Moreover, the modelling space in both cases accommodates both generic and alternative-specific taste parameters and includes interactions with socio-demographic variables. The only subtle difference lies in the search space. While Delphos can omit attributes and apply transformations to them, VNS initiates the search process using a predefined catalogue of attributes. We initialise Delphos using the hyperparameters (hidden layers, units, learning rate, discount factor, replay buffer size, number of episodes) identified in Experiment 1C (Appendix D) and introduce a 5% early stopping threshold. This design allows us to benchmark Delphos directly against a state-of-the-art metaheuristic while assessing whether the agent can propose competitive AIC models within a constrained number of episodes.

3.2.2 DECISIONS (Experiment 3)

Finally, we test Delphos in another empirical setting where modeller-driven reward and behavioural plausibility are enforced explicitly using the DECISIONS dataset (for more details, see Calastri et al. (2020)). This dataset integrates an online survey and a two-week mobile travel diary, capturing information on travel behaviour, residential location, energy use, and social network interactions. It includes alternative-specific attributes—such as in-vehicle travel time (IVT), out-of-vehicle travel time (OVT), and total travel cost (TC)—and covariates such as trip purpose, city, income, gender, age, education, and household size.

In this experiment, the agent can decide on attribute transformation, specify generic or alternative-specific tastes, and interactions with covariates, as in the previous experiment. The reward function is designed to represent an analyst concerned with model performance and model parsimony while enforcing behavioural expectations. It thus not only combines the log-likelihood component (weight 0.7) with a penalty on the number of parameters (weight 0.3), but also incorporates behavioural expectations by requiring negative estimated parameters for in-vehicle travel time and total travel cost. We maintain the same hyperparameters (Appendix D). This final experiment evaluates whether Delphos can propose specifications that are both well-performing and theoretically consistent when guided by a modeller-like reward formulation under constrained learning conditions.

4 RESULTS

We present the results of the experimental evaluation of Delphos in two parts. We first analyse the agent’s learning behaviour in simulated environments, where training runs until the policy stabilises, allowing us to evaluate convergence properties, exploration efficiency, and the ability to approach the true data-generating process (Subsection 4.1). We then examine the empirical applications, where the focus shifts towards identifying well-performing and behaviourally plausible specifications within constrained learning horizons (Subsection 4.2), including a direct benchmark against a metaheuristic on Swissmetro and an analyst-driven specification task on DECISIONS.

4.1 Simulated experiments

The simulated experiments evaluate whether Delphos learns a stable specification strategy when the true data-generating process is known. We analyse (i) learning curves of episodic rewards that capture how the agent’s episodic reward evolves over time, (ii) the evolution of Q-values across training to understand how the agent prioritises actions throughout training, and (iii) measure exploration coverage metrics that reveal how efficiently the agent explores the modelling space and concentration on high-reward regions. Results are averaged over ten independent runs with different random seeds.

Figure 5 shows the RL agent’s learning process and novelty over episodes, considering the simulated choice dataset S_1 . The upper figure shows the rolling mean and the min-max range of the learning curve across ten runs with different random seeds, using $\bar{\rho}^2$ as the reward signal over 10,000 training episodes. Initially, the agent’s learning trajectory starts at low reward levels, which reflect an exploratory phase where random actions are taken to specify candidate models. As training progresses, the learning curve gradually rises until it stabilises, which indicates a shift from exploration to exploitation, where the agent prioritises actions that have previously yielded higher reward signals. The upward trend suggests that the agent improves its policy and eventually stabilises on modelling decision that yield well-performing specifications. The spread across seeds also narrows over time, which indicates consistent performance towards the end of training.

The lower figure computes novelty as the share of previously non-proposed specifications within the same rolling window and averaged across runs. This value is highest early in training and then declines towards zero, indicating that the agent progressively shifts from proposing new specifications (exploration) to revisiting a smaller set of candidates (exploitation). Because this drop coincides with improving and stabilising rewards, it suggests that agent’s learning is driven by policy refinement rather than continued exploration. Moreover, Delphos estimates 1,890 unique models across 10,000 episodes, implying that it revisits previously seen specifications in many episodes without re-estimating them, while the agent refines its policy.

To analyse agent’s policy, Figure 6 shows how the agent’s modelling actions change from the start to the end of the training process. For each modelling action, we display the average frequency with which it is taken in the early and late training stages across runs, ranked by its late-stage frequency. We also report the mean adjusted McFadden’s pseudo- R^2 obtained in the specifications where that action was used. In the early episodes, the agent explores the modelling space by selecting a broad and relatively uniform set of actions, resulting in dispersed Q-values. As training progresses, the policy is updated and Q-values increasingly concentrate around a smaller subset of actions. At the end of the training process, the agent relies on a smaller subset of actions that yield high-reward specifications, which indicates that it has become more selective in how it adjusts the model specification. This is further supported by the fact that some actions that remain low-frequency in late training, still achieve high mean rewards when they are used. This

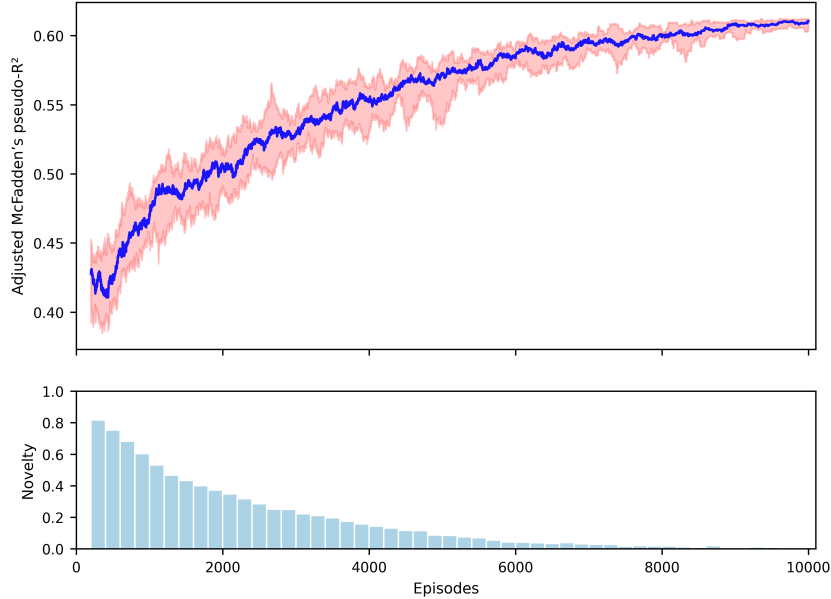


Figure 5: Learning process on simulated dataset S_1 . The upper panel displays the learning curve, showing the rolling mean and min–max range of the reward signal. The lower panel reports rolling novelty, measured as the share of previously unseen specifications.

suggests they are applied only in the certain states (internal steps) where they are effective, rather than uniformly in the early exploratory phase.

Compared to Experiment S_1 , the learning curves in Experiments S_2 and S_3 show higher variability in the min-max range across the ten runs (additional figures are provided in Appendix E). This increased noise reflects the added complexity of incorporating covariates into the model specification process. Covariates expand the modelling space substantially, making it more challenging for the agent to consistently identify high-performing regions, and they introduce additional opportunities for misspecification, leading to more diverse reward signals. Consequently, it takes longer for the agent to recognise which sequences of actions are valuable to take in future episodes for refining its policy. The inclusion of covariates significantly enlarges the action space, making the Q-value distribution more diffuse. Nonetheless, the learning curve eventually reaches a plateau, which indicates that the agent converges towards a stable policy that proposes high-performing model specifications despite the increased complexity of the task.

Table 2 summarises the modelling space parameters, exploration coverage, and performance for each simulated case. We report the size of the modelling space (i.e., the total number of feasible specifications), the number of unique specifications explored by the agent, the percentage of successfully converged estimations, the episode at which the true specification is proposed, and the average reward over the last 100 episodes, together with its difference relative to the reward of the true specification.

Table 2: Modelling size, exploration coverage, and performance on simulated experiments

Exp	Model space	Models estimated by agent	Converged models	First recovery	Reward (last 100)	Δ Reward	RMSE
1A	8.19×10^3	1.89×10^3	97.42%	1.46×10^3	0.611	-0.002	0.009
1B	8.19×10^3	2.34×10^3	98.45%	1.63×10^3	0.753	-0.009	0.012
1C	1.59×10^6	3.92×10^3	78.48%	3.73×10^3	0.654	-0.023	0.025

As expected, the modelling space in Experiment S_3 is orders of magnitude larger than

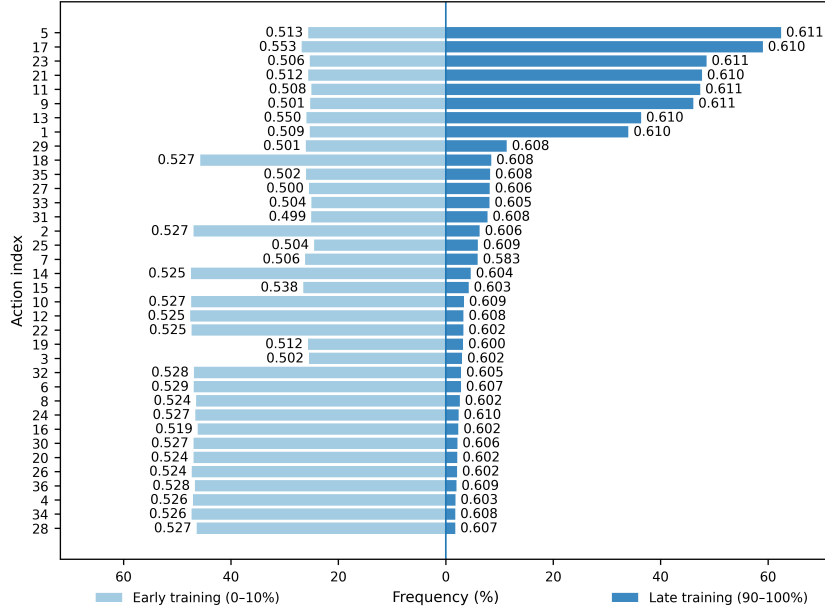


Figure 6: Action selection distribution in simulated dataset S_1 . Bars show the average frequency with which each action is selected in the early and late training across runs. Numbers report the average \bar{p}^2 for specifications in which the action was used. The full list of actions and their corresponding definitions is provided in the Appendix [F](#)

in Experiments S_1 and S_2 , due to the inclusion of covariates and their associated interactions. Across all experiments, the agent does not attempt to exhaustively explore the entire modelling space but instead concentrates its search on a comparatively small subset of candidates. In Experiment S_1 the agent explored 1,899 unique models and recovered the true specification by episode 1,461 on average, achieving a 97.42% convergence rate. In late training, the learnt policy concentrates on high-performing candidates whose rewards remain close to the true model value ($\Delta\bar{p}^2 = -0.001$, RMSE = 0.009 over the last 100 episodes), which indicates a stable near-optimal performance. A similar agent behaviour is observed in Experiment S_2 , where it explored 2,338 unique models, identified the true specification by episode 1,627, achieved a 98.45% convergence rate, and the final policy again yields near-optimal specifications ($\Delta\bar{p}^2 = -0.009$, RMSE = 0.012).

In contrast, Experiment S_3 involved navigating an exponentially larger modelling space. The agent explored 3,922 unique specifications (less than 1% of the feasible set) and the convergence rate drops to 78.48%. Although the true specification is not recovered across runs, the final learnt policy still concentrates on near-optimal candidates; the mean rewards over the last 100 episodes remains close to the true model value ($\Delta\bar{p}^2 = -0.023$, RMSE = 0.025). The agent also reaches its best observed reward much before the training horizon (on average at episode 3,731), and this maximum reward (0.675) remained close to the reward value obtained by the true model (see Appendix [E](#)). This suggests that Delphos can learn a policy that identifies well-performing candidates even in high-dimensional and combinatorial spaces. Importantly, for real-world applications where the true data-generating process is unknown, the goal is not exact recovery but the agent’s ability to consistently converge towards near-optimal modelling regions within such environments.

4.2 Empirical experiments

The empirical experiments evaluate Delphos under realistic and high-dimensional modelling spaces, where the agent’s objective is *not to learn a fully stable policy but to identify*

competitive specifications within a constrained training horizon defined by a fixed maximum number of episodes (i.e., model estimations). We first benchmark Delphos against a metaheuristic approach for the Swissmetro dataset, using the same experimental setup and modelling objective. We then analyse whether the agent can propose models that satisfy an analyst-defined reward formulation combining parsimony and behavioural constraints on the DECISIONS dataset.

4.2.1 Benchmarking on Swissmetro

We first apply the agent to the Swissmetro dataset, evaluating its capacity to identify model specifications using the AIC values as a reward signal. The training process ran for a total of 1,098 episodes before the early stopping was triggered, taking approximately 34 minutes on a 4.1 GHz 12-core Apple M3 Pro processor with 18 GB of unified memory. Although the modelling space comprised 88,510,464 possible specifications, the agent explored 1,094 unique model specifications and successfully converged 446 of them. The low convergence rate indicates that the agent’s policy proposes specifications that are more complex by including non-linear transformations and socio-demographic variables, which is consistent with early exploration phase.

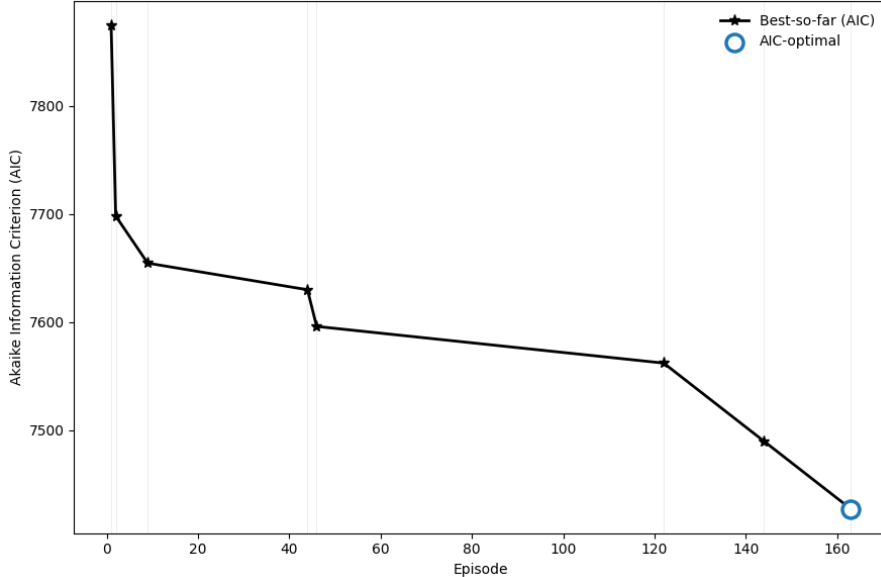


Figure 7: Evolution of the best-so-far AIC observed by the agent across training episodes on the Swissmetro dataset.

First, we track the best-performing models across episodes using the raw modelling outcomes, as shown in Figure 7. This graph indicates the progression of the highest achieved modelling outcome seen by the agent so far, which reflects how the agent is incrementally proposing candidates that outperform previous ones based on its policy refinement over episodes under the chosen reward signal. The rapid improvement during the initial episodes suggests that the agent is able to identify promising candidates, while later improvements are more incremental as the agent moves to high-performing regions of the modelling space.

To further evaluate the trade-off between competing modelling objectives, we built the Pareto front of unique and successfully estimated specifications proposed during training. The Pareto front shows the non-dominated models that represent an optimal trade-off between model performance and parsimony. Delphos proposed 23 non-dominated converged specifications with in-sample AIC values from 11,042.32 to 7,426.95 and number of pa-

rameters from 1 to 40. Figure 8 plots each model according to its number of parameters and its corresponding AIC value. From this set of models, we identify the lowest in-sample AIC and its modelling results, including estimate values and standard errors, can be found in Table 8. This plot also suggests that the agent is not only capable of suggesting a wide range of specifications, but also of identifying candidates that may be suitable depending on the modeller’s priorities.

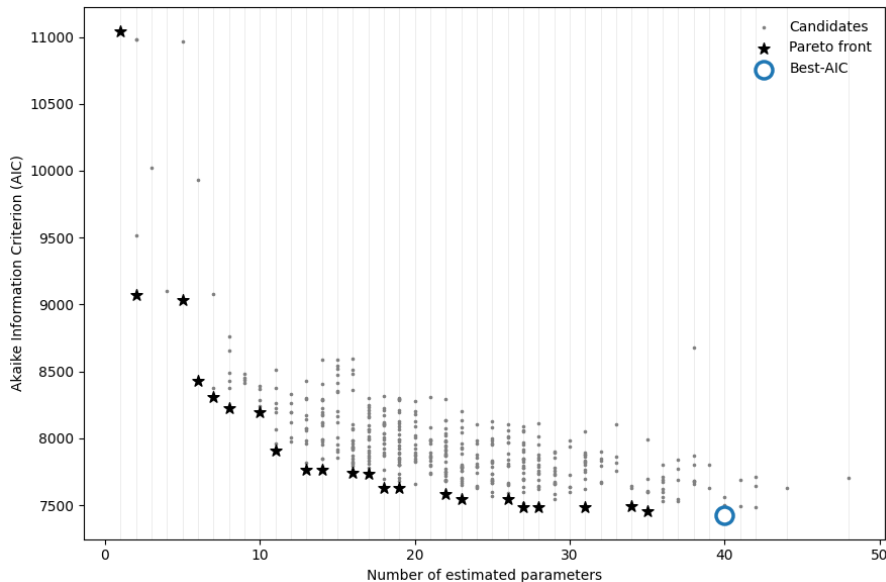


Figure 8: Pareto front of Delphos candidates on the Swissmetro dataset. Grey dots show all converged specifications evaluated during training. Stars indicate the non-dominated candidates (Pareto-optimal set), and the circle shows the specification with the lowest AIC.

To benchmark our framework, we run the VNS algorithm to obtain the Pareto Front considering the same experimental setup and modelling objective (minimise AIC), as described in Section 3. The VNS algorithm evaluated 735 models in 52 minutes, yielding a Pareto set of 12 non-dominated specifications, with in-sample AIC values ranging from 8,509.53 to 7,643.07, number of parameters from 6 to 22. By comparison, Delphos proposed a similar number of candidate models in 34 minutes; it is faster, though runtimes remain comparable.

Table 3 reports the in-sample (IS) and out-of-sample (OOS) performance of the best AIC models proposed by both approaches. Delphos achieves a lower in-sample AIC than VNS (7,426 vs. 7,633) considering successfully converged models, but with more estimated parameters (40 vs. 36). Out-of-sample, Delphos also obtains a lower AIC and a higher log-likelihood. Figure 9 shows the Pareto specifications identified by the VNS metaheuristic and Delphos on the Swissmetro dataset. Across the low-complexity range (approximately 6 to 12 parameters), VNS yields a compact Pareto set that tends to dominate Delphos’ Pareto solutions, with lower AIC values at comparable complexity. By contrast, Delphos generates a broader set of converged candidates and a Pareto front that overlaps with VNS in some regions and extends it towards more complex specifications.

Table 3: Swissmetro benchmarking: best-AIC specification, run time, and proposed candidates.

Approach	IS-LL	IS-AIC	OOS-LL	OOS-AIC	#Parameters	Time (min)	Candidates
VNS	-3,780	7,633	-1,080	2,233	36	52	732
Delphos	-3,673	7,426	-1,006	2,092	40	34	446

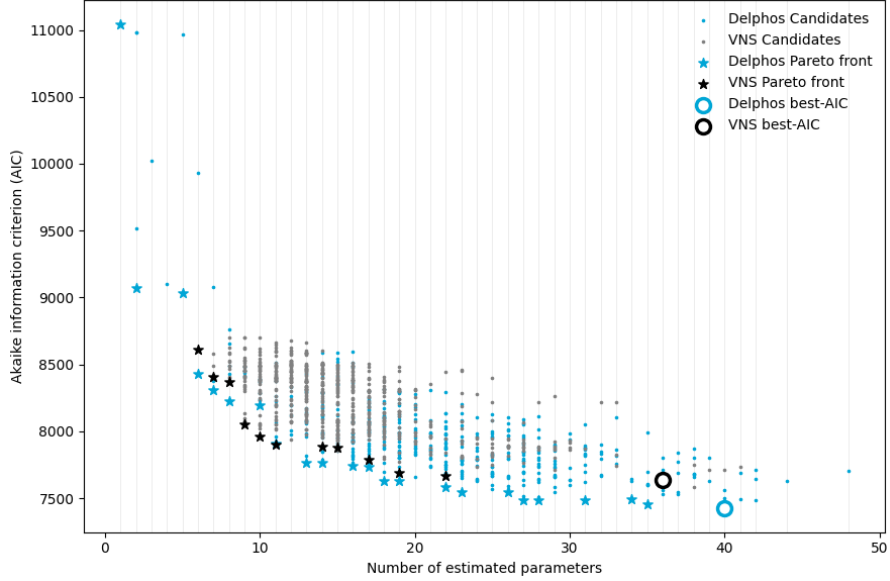


Figure 9: Pareto front of VNS and Delphos on the Swissmetro dataset. Dots show all converged specifications evaluated by each approach. Stars indicate the non-dominated candidates for VNS and Delphos, and circles show the lowest-AIC specification found by each approach.

Overall, this benchmark suggests that both approaches identify competitive specifications under the same experimental setup, with VNS focusing on proposing models with fewer parameters and Delphos achieving better specifications at higher complexities. This difference is consistent with the fact that Delphos updates its action policy using the reward signal obtained after each estimation, where only successfully converged models receive positive feedback. As a result, the agent can continuously move away from those action that frequently lead to non-convergence while continuing to explore higher-complexity regions. By contrast, VNS treats non-convergent candidates as invalid and excludes them from the Pareto set in post-processing, rather than using them to adapt a decision rule during the search.

4.2.2 Modeller-driven reward design with behavioural constraints on DECISIONS

We next apply the agent to the DECISIONS dataset to evaluate its ability to suggest model specifications under knowledge-driven learning objectives. Specifically, we consider: (i) a weighted sum of log-likelihood (LL) and number of parameters (NumParam) as the reward signal; and (ii) the incorporation of behavioural expectations, such as sign constraints on specific parameters. In this setting, training terminated after 81 minutes on a 4.1 GHz 12-core Apple M3 Pro processor with 18 GB of unified memory. Out of the 6.72 million feasible model specifications in the modelling space, the agent explored 408 unique specifications that successfully converged. Across these converged candidates, model complexity ranged from 2 to 89 parameters and the log-likelihood ranged from $-4,751.23$ to $-1,491.11$.

Delphos identifies a high-performing specification within a constrained number of episodes, balancing goodness-of-fit and parsimony as defined by the reward function. The best LL-NumParam candidate satisfies the behavioural constraints encoded in the reward design, including negative coefficients for in-vehicle travel time and travel cost. As training progresses, the agent no longer identifies candidates that improve the modeller-defined reward, and early stopping is triggered once additional episodes fail to yield further improvements. Table 9 in the Appendix reports the modelling outcomes for the best LL-NumParam specification identified by Delphos, including parameter estimates, robust standard errors, and coefficient signs.

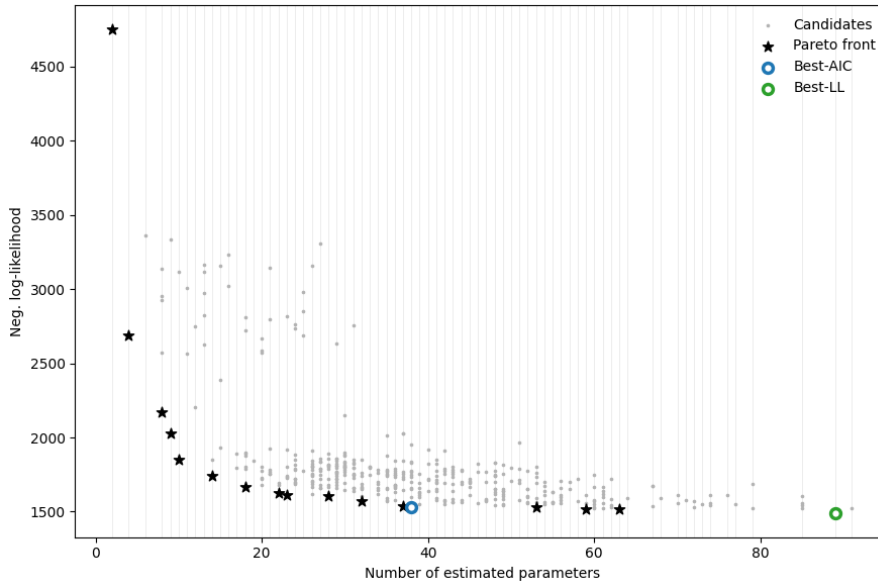


Figure 10: Pareto front of Delphos candidates on the DECISIONS dataset. Grey dots show all converged specifications proposed during training. Stars indicate the non-dominated candidates, and circles show the best AIC and best log-likelihood specifications.

As in the previous experiment, we built the Pareto front using the unique model specifications suggested by the agent. Figure 10 presents the trade-off between the number of parameters and negative log-likelihood. Each grey dot represents a successfully converged model, while the black stars show the non-dominated models forming the Pareto front. The results show a range of models covering this trade-off, from simpler specifications with slightly lower model fit to more complex specifications with improved log-likelihood, all under a reward function that reflects an analyst’s preference for parsimony and behavioural plausibility.

These results indicate not only the agent’s ability to navigate the modelling space and balance multiple modelling outcome objectives, but also its potential to identify models that are consistent with behavioural expectations. However, because the Pareto front is constructed from all unique, converged models estimated during training, it may include specifications that violate the predefined behavioural expectations (e.g., positive cost coefficients). While it is possible to exclude behaviourally inconsistent models from the Pareto set in the post-processing, we retain them here to provide a complete view of the model fit–complexity trade-off among converged candidates. Ultimately, it remains the responsibility of the analyst to review the set of suggested candidates and evaluate their theoretical plausibility and model performance.

5 CONCLUSIONS

This study introduced Delphos, a reinforcement learning framework for assisting the model specification process of discrete choice models. Rather than treating model specification as a static optimisation problem, we frame it as a sequential decision-making task. Delphos learns to specify models by selecting a sequence of modelling actions (e.g., adding or modifying terms) to maximise a reward signal. Over repeated episodes, the agent uses past modelling outcomes to refine its policy and prioritise decisions that lead to high-performing and behaviourally plausible models. To support this learning process, we have defined and developed: (i) a compact yet flexible encoding of the modelling space that captures sim-

ple and complex evolving functional forms; (ii) a feasible action space with a masking mechanism; (iii) a user-configurable reward function that can integrate model fit criteria and behavioural expectations; and (iv) an environment that enables sequential interaction between agent actions and model estimation.

Delphos not only offers a methodological contribution but also reframes model specification in discrete choice modelling as an adaptive and experience-driven process. While traditional meta-heuristics typically rely on predefined neighbourhood operators and fixed search strategies that do not updated based on accumulated experience (Bianchi et al. 2009; Cuevas et al. 2024), Delphos updates its Q-values from feedback and progressively concentrates its search in higher-reward regions of the model space. Ultimately, this learning-based approach can be applied across datasets and modelling objectives.

To evaluate its performance, Delphos was applied to both simulated and empirical datasets, including Swissmetro and DECISIONS. Our findings in the simulated cases demonstrate that the agent can explore large and complex modelling spaces and, even under limited training episodes, suggest well-performing model candidates. The learning curves reveal that the agent rapidly improves its policy in early episodes. As training progresses, the agent increasingly exploits past experience to inform Q-values that shape its search towards high-reward specifications. Moreover, exploration coverage metrics reveal that while the agent does not exhaustively cover the feasible model space, it narrows its set of actions and concentrates its exploration in regions associated with higher rewards. On Swissmetro, benchmarking against a Variable Neighbourhood Search under the same experimental setup showed that Delphos can propose competitive specifications in terms of AIC, yielding a Pareto set that overlaps with the VNS solutions and extends towards better in-sample AIC models. On Decisions, we showed that Delphos is capable of having multiple modelling outcomes and incorporating domain-specific behavioural expectations into its learning process. After training, the resulting candidate models can be used to derive Pareto frontiers, offering choice modellers a set of specifications that meet trade-offs between competing goals and supporting informed model selection.

Despite these contributions, the proposed framework has several limitations. First, Delphos is currently designed to work on a single choice dataset, which limits the transferability of modelling strategies across related contexts. As a result, each new dataset requires a new search process, as in existing meta-heuristic approaches. This limitation could be addressed by moving towards a multitask setting, in which the agent learns a reusable policy across multiple choice datasets and adapts it to each specific context through representation learning (Rusu et al. 2015; Huang & Ontañón 2020). Second, the agent may initially produce behaviourally implausible models, particularly when faced with conflicting objectives in the reward function. Future improvements could involve offline learning methods, where the replay buffer is fed with a known sequence of actions that generate plausible specifications. Third, the agent currently samples past experiences uniformly, which may lead to inefficient learning from low-reward actions. To address this, future implementations could adopt prioritised experience replay (Schaul et al. 2015), allowing the agent to focus on learning from high-value transitions.

From an algorithmic standpoint, we implemented this framework using a baseline and single-agent RL algorithm. Future research could benefit from more advanced and robust algorithms that support parallelism and stabilised training. For example, Asynchronous Advantage Actor-Critic (A3C) methods not only allow concurrent learning across multiple agents, thereby improving computational efficiency, but also reduce training instability by incorporating an advanced baseline to regularise Q-value functions (Mnih et al. 2016). Beyond single-agent architectures, Multi-Agent Reinforcement Learning (MARL) can also be

deployed in parallel, each specialised in different modelling outcomes, and collaboratively optimise a shared policy within a common environment (K. Zhang et al., 2021). These configurations could enhance exploration, reduce computation time, and enable coordinated learning across diverse modelling goals.

Overall, beyond its specific implementation, this work paves the way for a new generation of intelligent tools for model specification in discrete choice modelling. Delphos illustrates how reinforcement learning agents can develop adaptive strategies by leveraging feedback and accumulated experience. This aligns with emerging efforts such as Steir et al. (2025), which explore the use of Large Language Models to suggest and estimate MNL specifications based on domain context and implicit common-sense reasoning. These approaches highlight a broader shift in which model specification is no longer considered a purely manual or mechanical task, but as an interactive process that benefits from accumulated knowledge, a learning-based loop, and contextual knowledge. Ultimately, these developments demonstrate a promising path for integrating intelligent systems into choice modelling workflows.

A EXAMPLE OF ACTION MASKING IN THE SPECIFICATION PROCESS

The agent uses a masking mechanism that filters invalid actions based on the current specification. At each step, the environment evaluates the existing state S_e and masks out actions that would either violate modelling logic (e.g., duplicating a covariate-transformation pair) or structural constraints (e.g., attempting to modify an unadded variable). Let us consider the following sequence of internal episodes (states) and their corresponding allowed actions:

$$\begin{aligned}
 S_e^0 &= [] && \rightarrow (\text{add}, 1, \text{linear}, 0, 0) \\
 S_e^1 &= [(1, \text{linear}, 0, 0)] && \rightarrow (\text{change}, 1, \text{linear}, 1, 0) \\
 S_e^2 &= [(1, \text{linear}, 1, 0)] && \rightarrow \{\cancel{(\text{add}, 1, \text{linear}, 0, 0)}, (\text{add}, 3, \text{linear}, 0, 0)\} \\
 S_e^3 &= [(1, \text{linear}, 1, 0), (3, \text{linear}, 0, 0)] && \rightarrow (\text{change}, 3, \text{log}, 0, 0) \\
 S_e^4 &= [(1, \text{linear}, 1, 0), (3, \text{log}, 0, 0)] && \rightarrow (\text{terminate})
 \end{aligned}$$

The interpretation of this progression is as follows:

- In S_e^0 , the model is empty. The valid actions are any of adding a new variable. Here, the agent selects **(add, 1, linear, 0, 0)**.
- In S_e^1 , variable 1 has been added with a linear transformation, generic and without interacting with a covariate. The agent changes it to alternative-specific taste parameter by choosing **(change, 1, linear, 1, 0)**.
- In S_e^2 , since variable 1 in linear form is already present, the action **(add, 1, linear, 0, 0)** is masked out. Instead, the agent selects **(add, 3, linear, 0, 0)**.
- In S_e^3 , a second variable has been added. The agent now modifies it by applying a log transformation: **(change, 3, log, 0, 0)**.
- In S_e^4 , the agent ends the model specification process and triggers a **terminate** action.

B PSEUDOCODE FOR DELPHOS TRAINING

Algorithm 1 Agent training

```

1: Set space_parameters, reward_weights and metric_directions
2: Set total, minimum training, and convergence window episodes as  $(E, E_{\min}, E_{\text{search}})$ 
3: Initialise  $Q(s, a; \theta)$  and  $Q_{\text{target}}(s, a; \theta')$  with random weights; set  $\theta' \leftarrow \theta$ 
4: Initialise empty replay buffer  $\mathcal{D}$ 
5: for each episode  $e = 1$  to  $E$  do
6:   Initialise specification state  $S_e^0 \leftarrow \emptyset$ , intermediate step counter  $l \leftarrow 0$ 
7:   while action  $a_e^l \neq \text{terminate}$  do
8:     Mask invalid actions based on current state  $S_e^l$ 
9:      $a_e^l \leftarrow \epsilon$ -greedy strategy on  $Q(S_e^l, a; \theta)$ 
10:     $S_e^{l+1} \leftarrow \text{Apply } a_e^l \text{ to } S_e^l$ 
11:     $l \leftarrow l + 1$ 
12:   end while
13:   Estimate encoded model specification by  $S_e^l$  using Delphos
14:   Update  $R^* \leftarrow \max(R^*, R_e) \Rightarrow S^* \leftarrow S_e^l$ 
15:   Apply the reward function to calculate final and intermediate rewards.
16:   for each step  $l = 0$  to  $L$  do
17:     Set intermediate reward  $r_e^l \leftarrow \gamma^{L-l-1} \cdot R_e$ 
18:     Store transition  $(S_e^l, a_e^l, r_e^l, S_e^{l+1})$  in  $\mathcal{D}$ 
19:   end for
20:
21:   Decay exploration rate  $\epsilon$ 
22:   Sample mini-batch of trajectories  $(s, a, r, s')$  from  $\mathcal{D}$ 
23:    $\theta \leftarrow \theta^*$  by minimising:
24:      $\mathcal{L}(\theta^*) = (r + \gamma \max_{a'} Q_{\text{target}}(s', a'; \theta^-) - Q(s, a; \theta^*))^2$ 
25:   Every  $E_{\text{target}}$  episodes, update target network:  $\theta^- \leftarrow \theta$ 
26:
27:   if  $e > E_{\min}$  and no improvement over past  $E_{\text{search}}$  episodes then
28:     break
29:   end if
30:
31: end for

```

C TRUE PARAMETER VALUES FOR SIMULATED CHOICE DATASETS

Table 4: True parameter values for each simulated case

Experiment	S_1	S_2	S_3
ASC ₁	β_{ASC1} -0.95	β_{ASC1} -0.95	$\beta_{\text{ASC1,cov1=0}}$ -0.90
			$\beta_{\text{ASC1,cov1=1}}$ 0.43
ASC ₂	β_{ASC2} 0.84	β_{ASC2} 0.84	$\beta_{\text{ASC2,cov1=0}}$ -0.83
			$\beta_{\text{ASC2,cov1=1}}$ 0.43
x_1	β_1 -0.72	β_1 -1.65	$\beta_{1,1}$ -0.50
			$\beta_{1,2}$ -0.40
			$\beta_{1,3}$ -0.60
x_2	β_2 0.83	$\beta_{2,1}$ -1.33 $\beta_{2,2}$ -1.33 $\beta_{2,3}$ -1.33	$\beta_{2,1,\text{cov2}=1}$ -1.40
			$\beta_{2,1,\text{cov2}=2}$ -0.97
			$\beta_{2,1,\text{cov2}=3}$ -0.06
			$\beta_{2,2,\text{cov2}=1}$ -1.16
			$\beta_{2,2,\text{cov2}=2}$ -0.85
			$\beta_{2,2,\text{cov2}=3}$ -0.15
			$\beta_{2,3,\text{cov2}=1}$ -1.15
			$\beta_{2,3,\text{cov2}=2}$ -0.65 $\beta_{2,3,\text{cov2}=3}$ -0.09
x_3	β_3 1.77 λ_3	β_3 0.57 λ_3 0.64	β_3 -1.40
x_4	β_4 1.01 λ_4 0.20	$\beta_{4,1}$ 0.92 $\beta_{4,2}$ 0.92 $\beta_{4,3}$ 0.92	β_4 1.70
			λ_4 0.09
x_5	– –	β_5 1.90	β_5 1.90
x_6	– –	– –	– –

D HYPERPARAMETER TUNING AND LEARNING ANALYSIS

Delphos performance is sensitive to its hyperparameters and a well-chosen configuration thus can significantly improve its final performance and speed up its learning, as other RL agents (Henderson et al., 2018). To investigate the hyperparameters’ influence on the agent’s learning and performance, we conduct a random search over both the neural network architecture (e.g., layers and neurons) and learning parameters (e.g. learning rate, buffer size, exploration schedule). To do so, we sample 15 configurations from the predefined hyperparameter space (Table 5) for each simulated experiment, run each configuration using five random seeds⁴ and evaluate them using the area under the learning curve (AUC) and the mean reward in the last 100 episodes (Final-100), following Teh et al. (2017) to capture both sample-efficiency and asymptotic performance.

Table 5: Hyperparameter ranges.

Hyperparameter	Level values
Number of hidden layers (L)	{2, 3}
Hidden units per layer (U)	{64, 128, 256, 512}
Learning rate (α)	{ $1e-3$, $5e-4$, $1e-4$ }
Discount factor (γ)	{0.95, 0.99}
Replay buffer size (BS)	{5,000, 10,000, 20,000}
Target network update frequency (TU)	{5, 10, }
Total episodes (E)	{10,000, 30,000, 50,000}

Table 6 shows the AUC and Final-100 of the five best-performing sampled hyperparameter configuration for each simulated experiment (S_1 – S_3). For each configuration, we show the mean, the 25th, and 75th percentiles of the learning curves from the five random seeds.

Table 6: Performance of sampled hyperparameter configurations.

Exp	L	U	α	γ	BS	TU	E	AUC	Final-100
S_1 - conf. 1	2	256	1e-4	0.99	5k	10	50k	0.552	0.612
S_1 - conf. 2	3	256	1e-4	0.99	5k	5	50k	0.553	0.612
S_1 - conf. 3	2	64	1e-4	0.99	10k	10	5k	0.538	0.611
S_1 - conf. 4	2	64	1e-4	0.99	10k	10	10k	0.548	0.611
S_1 - conf. 5	2	256	1e-4	0.95	10k	5	30k	0.538	0.610
S_2 - conf. 1	2	128	1e-4	0.99	10k	10	50k	0.708	0.753
S_2 - conf. 2	2	128	1e-4	0.99	5k	10	50k	0.707	0.753
S_2 - conf. 3	2	128	1e-4	0.99	5k	10	10k	0.698	0.753
S_2 - conf. 4	2	128	1e-4	0.95	50k	5	10k	0.686	0.719
S_2 - conf. 5	3	128	5e-4	0.95	5k	10	30k	0.684	0.722
S_3 - conf. 1	2	256	1e-4	0.99	5k	5	10k	0.516	0.657
S_3 - conf. 2	2	256	1e-3	0.99	5k	10	50k	0.503	0.653
S_3 - conf. 3	3	128	1e-4	0.99	5k	10	10k	0.511	0.648
S_3 - conf. 4	2	128	5e-3	0.99	5k	5	10k	0.474	0.644
S_3 - conf. 5	3	128	1e-4	0.99	10k	5	10k	0.476	0.643

For our results reported in Section 4.1, we favour configurations that achieve high performance while requiring fewer training episodes. Within the top five candidates for each experiment, we thus select the configuration that offers a favourable trade-off between sample efficiency and asymptotic performance. For Experiments S_1 , S_2 , and S_3 , we respectively select S_1 -conf. 4, S_2 -conf. 3, and S_3 -conf. 1, which reduce the training horizon to $E = 10,000$ episodes while still obtaining competitive Final-100 rewards (within 0.02 of the best configuration in each case).

⁴For all remaining hyperparameters, we use the values used in the original DQN paper (Mnih et al., 2015)

E SIMULATED EXPERIMENT S_2 AND S_3

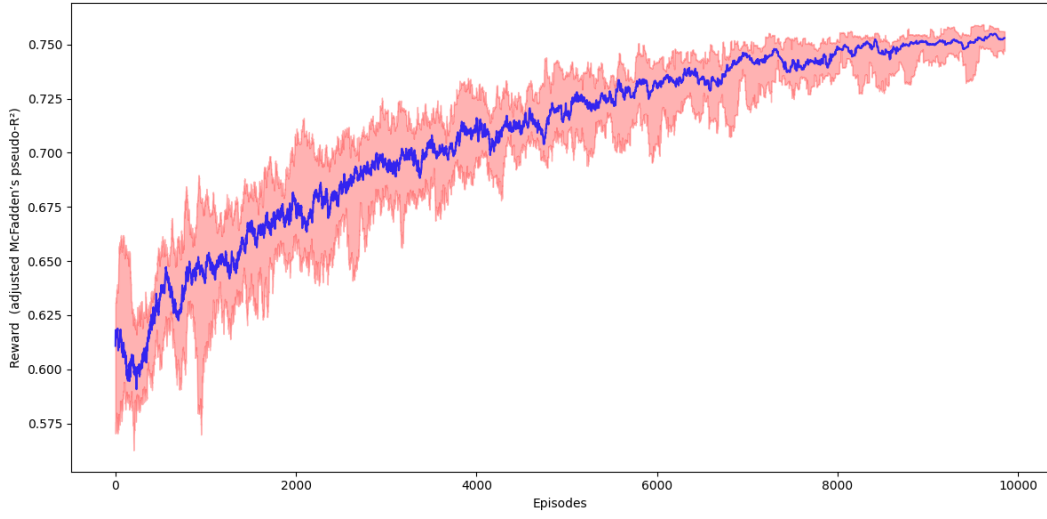


Figure 11: Learning curve showing the reward signal ($\bar{\rho}^2$) across 10 runs

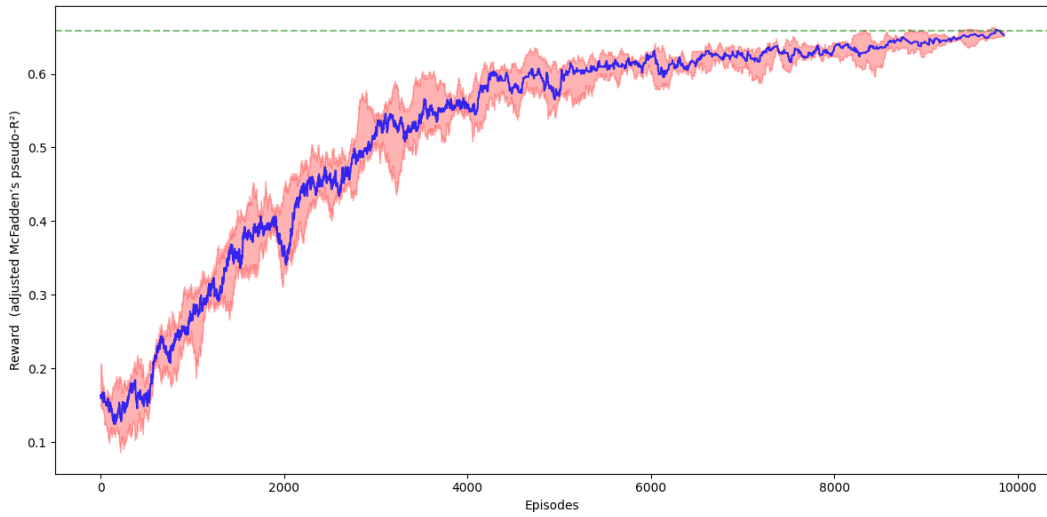


Figure 12: Learning curve showing the reward signal ($\bar{\rho}^2$) across 10 runs

F SIMULATED EXPERIMENT A: ACTION SPACE

Table 7: Action space for simulated experiment S_1

Action ID	Operation	Variable	Transformation Type	Action
0	terminate	0	-	(end)
1	add	1	linear	(add, 1, linear)
2	change	1	linear	(change, 1, linear)
3	add	1	log	(add, 1, log)
4	change	1	log	(change, 1, log)
5	add	1	box-cox	(add, 1, box-cox)
6	change	1	box-cox	(change, 1, box-cox)
7	add	2	linear	(add, 2, linear)
8	change	2	linear	(change, 2, linear)
9	add	2	log	(add, 2, log)
10	change	2	log	(change, 2, log)
11	add	2	box-cox	(add, 2, box-cox)
12	change	2	box-cox	(change, 2, box-cox)
13	add	3	linear	(add, 3, linear)
14	change	3	linear	(change, 3, linear)
15	add	3	log	(add, 3, log)
16	change	3	log	(change, 3, log)
17	add	3	box-cox	(add, 3, box-cox)
18	change	3	box-cox	(change, 3, box-cox)
19	add	4	linear	(add, 4, linear)
20	change	4	linear	(change, 4, linear)
21	add	4	log	(add, 4, log)
22	change	4	log	(change, 4, log)
23	add	4	box-cox	(add, 4, box-cox)
24	change	4	box-cox	(change, 4, box-cox)
25	add	5	linear	(add, 5, linear)
26	change	5	linear	(change, 5, linear)
27	add	5	log	(add, 5, log)
28	change	5	log	(change, 5, log)
29	add	5	box-cox	(add, 5, box-cox)
30	change	5	box-cox	(change, 5, box-cox)
31	add	6	linear	(add, 6, linear)
32	change	6	linear	(change, 6, linear)
33	add	6	log	(add, 6, log)
34	change	6	log	(change, 6, log)
35	add	6	box-cox	(add, 6, box-cox)
36	change	6	box-cox	(change, 6, box-cox)

G SWISSMETRO: MODELLING OUTCOMES

Table 8: Best-AIC specification on Swissmetro dataset

Parameter	Transformation	Covariate	Value (Rob. s.e.)
$\beta_{\text{train, age}=1}$		age = 1	-7.345 (2.780)
$\beta_{\text{train, age}=2}$		age = 2	0.478 (0.664)
$\beta_{\text{train, age}=3}$		age = 3	0.480 (0.514)
$\beta_{\text{train, age}=4}$		age = 4	1.617 (0.736)
$\beta_{\text{train, age}=5}$		age = 5	4.014 (0.882)
$\beta_{\text{sm, age}=1}$		age = 1	-7.485 (2.845)
$\beta_{\text{sm, age}=2}$		age = 2	1.110 (0.553)
$\beta_{\text{sm, age}=3}$		age = 3	0.808 (0.374)
$\beta_{\text{sm, age}=4}$		age = 4	1.272 (0.576)
$\beta_{\text{sm, age}=5}$		age = 5	1.916 (0.867)
$\beta_{\text{train, TT, ga}=0}$	Box-Cox	GA = 0	-2.078 (0.422)
$\beta_{\text{train, TT, ga}=1}$	Box-Cox	GA = 1	0.252 (0.520)
$\beta_{\text{sm, TT, ga}=0}$	Box-Cox	GA = 0	-1.529 (0.226)
$\beta_{\text{sm, TT, ga}=1}$	Box-Cox	GA = 1	0.161 (0.422)
$\beta_{\text{car, TT, ga}=0}$	Box-Cox	GA = 0	-2.374 (0.238)
$\beta_{\text{car, TT, ga}=1}$	Box-Cox	GA = 1	1.246 (1.173)
$\beta_{\text{train, TC, age}=1}$	Log	age = 1	-11.573 (4.864)
$\beta_{\text{train, TC, age}=2}$	Log	age = 2	-3.400 (1.298)
$\beta_{\text{train, TC, age}=3}$	Log	age = 3	-4.909 (1.046)
$\beta_{\text{train, TC, age}=4}$	Log	age = 4	-4.514 (1.127)
$\beta_{\text{train, TC, age}=5}$	Log	age = 5	-6.526 (1.656)
$\beta_{\text{sm, TC, age}=1}$	Log	age = 1	-9.882 (4.149)
$\beta_{\text{sm, TC, age}=2}$	Log	age = 2	-3.287 (0.799)
$\beta_{\text{sm, TC, age}=3}$	Log	age = 3	-3.213 (0.595)
$\beta_{\text{sm, TC, age}=4}$	Log	age = 4	-3.065 (0.771)
$\beta_{\text{sm, TC, age}=5}$	Log	age = 5	-3.735 (1.159)
$\beta_{\text{car, TC, age}=1}$	Log	age = 1	-46.367 (12.843)
$\beta_{\text{car, TC, age}=2}$	Log	age = 2	-1.814 (0.763)
$\beta_{\text{car, TC, age}=3}$	Log	age = 3	-2.338 (0.736)
$\beta_{\text{car, TC, age}=4}$	Log	age = 4	-1.060 (1.016)
$\beta_{\text{car, TC, age}=5}$	Log	age = 5	-0.710 (1.581)
$\beta_{HE, \text{income}=1}$		income = 1	-0.674 (0.266)
$\beta_{HE, \text{income}=2}$		income = 2	-0.925 (0.276)
$\beta_{HE, \text{income}=3}$		income = 3	-1.233 (0.257)
$\beta_{HE, \text{income}=4}$		income = 4	0.060 (0.343)
$\beta_{SE, \text{income}=1}$	Log	income = 1	-0.845 (0.350)
$\beta_{SE, \text{income}=2}$	Log	income = 2	-0.400 (0.424)
$\beta_{SE, \text{income}=3}$	Log	income = 3	1.818 (0.558)
$\beta_{SE, \text{income}=4}$	Log	income = 4	-0.593 (0.591)
λ_1			0.055 (0.226)
LL(0)			-5548.280
LL(final)			-3673.477
AIC			7426.955
BIC			7690.788
Rho-squared			0.338
Adj. Rho-squared			0.331
N. observations			5409
Number of parameters			40

H DECISIONS: MODELLING OUTCOMES

Table 9: Best-LL_Nparams specification on Decisions dataset.

Parameter	Transformation	Covariate	Value (Rob. s.e.)
$\beta_{\text{car, education}=1}$		education = 1	-0.997 (1.490)
$\beta_{\text{car, education}=2}$		education = 2	-4.289 (1.201)
$\beta_{\text{car, education}=3}$		education = 3	-1.746 (1.032)
$\beta_{\text{car, education}=4}$		education = 4	-3.636 (0.915)
$\beta_{\text{car, education}=5}$		education = 5	-2.709 (1.021)
$\beta_{\text{car, education}=6}$		education = 6	-4.043 (0.928)
$\beta_{\text{bus, education}=1}$		education = 1	-3.327 (0.952)
$\beta_{\text{bus, education}=2}$		education = 2	-4.425 (0.836)
$\beta_{\text{bus, education}=3}$		education = 3	-4.671 (0.840)
$\beta_{\text{bus, education}=4}$		education = 4	-5.507 (0.801)
$\beta_{\text{bus, education}=5}$		education = 5	-5.724 (0.825)
$\beta_{\text{bus, education}=6}$		education = 6	-4.516 (0.880)
$\beta_{\text{rail, education}=1}$		education = 1	-3.915 (2.320)
$\beta_{\text{rail, education}=2}$		education = 2	-0.851 (2.060)
$\beta_{\text{rail, education}=3}$		education = 3	0.266 (2.069)
$\beta_{\text{rail, education}=4}$		education = 4	-1.535 (1.984)
$\beta_{\text{rail, education}=5}$		education = 5	-0.947 (1.959)
$\beta_{\text{rail, education}=6}$		education = 6	-2.192 (2.186)
$\beta_{\text{taxi, education}=1}$		education = 1	-2.201 (6.128)
$\beta_{\text{taxi, education}=2}$		education = 2	-5.678 (2.061)
$\beta_{\text{taxi, education}=3}$		education = 3	-6.253 (2.125)
$\beta_{\text{taxi, education}=4}$		education = 4	-7.093 (1.044)
$\beta_{\text{taxi, education}=5}$		education = 5	-1.643 (2.932)
$\beta_{\text{taxi, education}=6}$		education = 6	-8.633 (2.392)
$\beta_{\text{cycling, education}=1}$		education = 1	-8.317 (0.996)
$\beta_{\text{cycling, education}=2}$		education = 2	-8.685 (1.155)
$\beta_{\text{cycling, education}=3}$		education = 3	-8.042 (0.906)
$\beta_{\text{cycling, education}=4}$		education = 4	-8.524 (0.874)
$\beta_{\text{cycling, education}=5}$		education = 5	-8.896 (0.909)
$\beta_{\text{cycling, education}=6}$		education = 6	-8.969 (0.919)
$\beta_{\text{car, IVT, city}=1}$	Log	city = 1	-1.620 (0.337)
$\beta_{\text{car, IVT, city}=2}$	Log	city = 2	-2.625 (0.464)
$\beta_{\text{car, IVT, city}=3}$	Log	city = 3	-1.634 (0.501)
$\beta_{\text{bus, IVT, city}=1}$	Log	city = 1	-0.219 (0.184)
$\beta_{\text{bus, IVT, city}=2}$	Log	city = 2	-1.277 (0.357)
$\beta_{\text{bus, IVT, city}=3}$	Log	city = 3	-0.605 (0.389)
$\beta_{\text{rail, IVT, city}=1}$	Log	city = 1	-0.312 (0.267)
$\beta_{\text{rail, IVT, city}=2}$	Log	city = 2	-1.035 (0.429)
$\beta_{\text{rail, IVT, city}=3}$	Log	city = 3	-0.244 (0.506)
$\beta_{\text{taxi, IVT, city}=1}$	Log	city = 1	-1.745 (0.599)
$\beta_{\text{taxi, IVT, city}=2}$	Log	city = 2	-3.129 (0.760)
$\beta_{\text{taxi, IVT, city}=3}$	Log	city = 3	-2.157 (0.828)
$\beta_{\text{cycling, IVT, city}=1}$	Log	city = 1	-1.216 (0.246)
$\beta_{\text{cycling, IVT, city}=2}$	Log	city = 2	-2.023 (0.392)
$\beta_{\text{cycling, IVT, city}=3}$	Log	city = 3	-1.139 (0.403)
$\beta_{\text{walking, IVT, city}=1}$	Log	city = 1	-2.971 (0.285)
$\beta_{\text{walking, IVT, city}=2}$	Log	city = 2	-3.670 (0.351)
$\beta_{\text{walking, IVT, city}=3}$	Log	city = 3	-2.860 (0.355)
$\beta_{\text{bus, OVT, purpose}=1}$	Box-Cox	purpose = 1	-1.448 (0.357)

Table 9: Best-LL_Nparams specification on Decisions dataset (continued).

Parameter	Transformation	Covariate	Value (Rob. s.e.)
$\beta_{\text{bus},OVT, \text{purpose}=2}$	Box-Cox	purpose = 2	-1.958 (0.415)
$\beta_{\text{bus},OVT, \text{purpose}=3}$	Box-Cox	purpose = 3	-2.134 (0.471)
$\beta_{\text{bus},OVT, \text{purpose}=4}$	Box-Cox	purpose = 4	-2.828 (0.654)
$\beta_{\text{bus},OVT, \text{purpose}=5}$	Box-Cox	purpose = 5	-2.111 (0.495)
$\beta_{\text{bus},OVT, \text{purpose}=6}$	Box-Cox	purpose = 6	-2.376 (0.557)
$\beta_{\text{bus},OVT, \text{purpose}=7}$	Box-Cox	purpose = 7	-1.987 (0.469)
$\beta_{\text{bus},OVT, \text{purpose}=8}$	Box-Cox	purpose = 8	-1.959 (0.437)
$\beta_{\text{rail},OVT, \text{purpose}=1}$	Box-Cox	purpose = 1	-3.108 (1.109)
$\beta_{\text{rail},OVT, \text{purpose}=2}$	Box-Cox	purpose = 2	-3.522 (1.166)
$\beta_{\text{rail},OVT, \text{purpose}=3}$	Box-Cox	purpose = 3	-4.465 (1.395)
$\beta_{\text{rail},OVT, \text{purpose}=4}$	Box-Cox	purpose = 4	-9.731 (2.762)
$\beta_{\text{rail},OVT, \text{purpose}=5}$	Box-Cox	purpose = 5	-3.374 (1.202)
$\beta_{\text{rail},OVT, \text{purpose}=6}$	Box-Cox	purpose = 6	-3.629 (1.237)
$\beta_{\text{rail},OVT, \text{purpose}=7}$	Box-Cox	purpose = 7	-9.597 (1.939)
$\beta_{\text{rail},OVT, \text{purpose}=8}$	Box-Cox	purpose = 8	-3.586 (1.211)
$\beta_{\text{car},TC, \text{education}=1}$	Log	education = 1	-2.770 (0.711)
$\beta_{\text{car},TC, \text{education}=2}$	Log	education = 2	0.504 (0.753)
$\beta_{\text{car},TC, \text{education}=3}$	Log	education = 3	-2.157 (0.431)
$\beta_{\text{car},TC, \text{education}=4}$	Log	education = 4	-0.915 (0.266)
$\beta_{\text{car},TC, \text{education}=5}$	Log	education = 5	-2.013 (0.393)
$\beta_{\text{car},TC, \text{education}=6}$	Log	education = 6	-0.951 (0.786)
$\beta_{\text{bus},TC, \text{education}=1}$	Log	education = 1	-3.703 (0.724)
$\beta_{\text{bus},TC, \text{education}=2}$	Log	education = 2	-1.464 (0.414)
$\beta_{\text{bus},TC, \text{education}=3}$	Log	education = 3	-2.107 (0.447)
$\beta_{\text{bus},TC, \text{education}=4}$	Log	education = 4	-1.745 (0.244)
$\beta_{\text{bus},TC, \text{education}=5}$	Log	education = 5	-1.841 (0.335)
$\beta_{\text{bus},TC, \text{education}=6}$	Log	education = 6	-2.496 (0.518)
$\beta_{\text{rail},TC, \text{education}=1}$	Log	education = 1	0.071 (0.393)
$\beta_{\text{rail},TC, \text{education}=2}$	Log	education = 2	-0.130 (0.439)
$\beta_{\text{rail},TC, \text{education}=3}$	Log	education = 3	-1.509 (0.460)
$\beta_{\text{rail},TC, \text{education}=4}$	Log	education = 4	-0.725 (0.270)
$\beta_{\text{rail},TC, \text{education}=5}$	Log	education = 5	-0.837 (0.290)
$\beta_{\text{rail},TC, \text{education}=6}$	Log	education = 6	-0.752 (0.711)
$\beta_{\text{taxi},TC, \text{education}=1}$	Log	education = 1	-3.377 (3.389)
$\beta_{\text{taxi},TC, \text{education}=2}$	Log	education = 2	-1.740 (1.374)
$\beta_{\text{taxi},TC, \text{education}=3}$	Log	education = 3	-0.854 (1.263)
$\beta_{\text{taxi},TC, \text{education}=4}$	Log	education = 4	-0.447 (0.804)
$\beta_{\text{taxi},TC, \text{education}=5}$	Log	education = 5	-3.778 (1.884)
$\beta_{\text{taxi},TC, \text{education}=6}$	Log	education = 6	0.325 (1.442)
λ_2			-0.114 (0.113)
LL(0)			-4768.43
LL(final)			-1491.11
AIC			3160.21
BIC			3720.56
Rho-squared			0.687
Adj. Rho-squared			0.669
N. observations			4008
Number of parameters			89

ACKNOWLEDGEMENTS

Gabriel Nova gratefully acknowledges Matthijs Spaan for his early insights and constructive feedback. The first author used ChatGPT for code and proofreading. All content was reviewed and edited by the authors, who take full responsibility for the publication. Stephane Hess acknowledges the financial support by the European Research Council through the advanced grant 101020940-SYNERGY.

REFERENCES

- Antonini, G., Gioia, C., Frejinger, E., & Themans, M. (2007). *Swissmetro: description of the data*.
- Baker, B., Gupta, O., Naik, N., & Raskar, R. (2016). Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.
- Beeramoole, P. B., Arteaga, C., Pinz, A., Haque, M. M., & Paz, A. (2023). Extensive hypothesis testing for estimation of mixed-logit models. *Journal of choice modelling*, 47, 100409.
- Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, 679–684.
- Bertsekas, D. (2019). *Reinforcement learning and optimal control* (Vol. 1). Athena Scientific.
- Bianchi, L., Dorigo, M., Gambardella, L. M., & Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8, 239–287.
- Bierlaire, M., Axhausen, K., & Abay, G. (2001). The acceptance of modal innovation: The case of swissmetro. In *Swiss transport research conference*.
- Brockman, G. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.
- Burnetas, A. N., & Katehakis, M. N. (1997). Optimal adaptive policies for markov decision processes. *Mathematics of Operations Research*, 22(1), 222–255.
- Calastri, C., Crastes dit Sourd, R., & Hess, S. (2020). We want it all: experiences from a survey seeking to capture social network structures, lifetime events and short-term travel and activity planning. *Transportation*, 47, 175–201.
- Chorus, C. G. (2010). *A new model of random regret minimization*. TU Delft Repository. Retrieved 2025-12-16, from <https://resolver.tudelft.nl/uuid:875158d0-dd28-47f8-875b-cd069b13b511> (Repository record (journal article deposited by TU Delft).) doi: 10.18757/ejtir.2010.10.2.2881
- Cuevas, E., Luque, A., Aguirre, N., Navarro, M. A., & Rodríguez, A. (2024). Metaheuristic optimization with dynamic strategy adaptation: an evolutionary game theory approach. *Physica A: Statistical Mechanics and its Applications*, 645, 129831.
- Dabney, W., Ostrovski, G., & Barreto, A. (2020). Temporally-extended $\{\epsilon\}$ -greedy exploration. *arXiv preprint arXiv:2006.01782*.
- Dewey, D. (2014). Reinforcement learning and the reward engineering principle. In *2014 aaaa spring symposium series*.
- Ghorbani, A., Nassir, N., Lavieri, P. S., Beeramoole, P. B., & Paz, A. (2025). Enhanced utility estimation algorithm for discrete choice models in travel demand forecasting. *Transportation*, 1–28.
- Graesser, L., & Keng, W. L. (2019). *Foundations of deep reinforcement learning: theory and practice in python*. Addison-Wesley Professional.
- Guevara, C. A., & Ben-Akiva, M. (2006). Endogeneity in residential location choice models. *Transportation research record*, 1977(1), 60–66.

- Haj-Yahia, S., Mansour, O., & Toledo, T. (n.d.). Grammar-based approach to data-driven utility specification for discrete choice models. *Available at SSRN 5195530*.
- Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 32).
- Hess, S., & Daly, A. (2024). *Handbook of choice modelling*. Edward Elgar Publishing.
- Hess, S., & Palma, D. (2019). Apollo: A flexible, powerful and customisable freeware package for choice model estimation and application. *Journal of choice modelling*, 32, 100170.
- Hess, S., & Polak, J. W. (2005). Mixed logit modelling of airport choice in multi-airport regions. *Journal of Air Transport Management*, 11(2), 59–68.
- Hess, S., & Stathopoulos, A. (2013). A mixed random utility—random regret model linking the choice of decision rule to latent character traits. *Journal of choice modelling*, 9, 27–38.
- Huang, S., & Ontañón, S. (2020). A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*.
- Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4, 237–285.
- Liebe, U., van Cranenburgh, S., & Chorus, C. (2023). Maximizing utility or avoiding losses? uncovering decision rule-heterogeneity in sociological research with an application to neighbourhood choice. *Sociological Methods & Research*, 00491241231186657.
- Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8, 293–321.
- Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994* (pp. 157–163). Elsevier.
- Ly, A., Dazeley, R., Vamplew, P., Cruz, F., & Aryal, S. (2024). Elastic step dqn: A novel multi-step algorithm to alleviate overestimation in deep q-networks. *Neurocomputing*, 576, 127170.
- Mariel, P., Hoyos, D., Meyerhoff, J., Czajkowski, M., Dekker, T., Glenk, K., ... others (2021). *Environmental valuation with discrete choice experiments: Guidance on design, implementation and data analysis*. Springer Nature.
- McFadden, D. (1974). The measurement of urban travel demand. *Journal of Public Economics*, 3(4), 303-328. Retrieved from <https://www.sciencedirect.com/science/article/pii/0047272774900036> doi: [https://doi.org/10.1016/0047-2727\(74\)90003-6](https://doi.org/10.1016/0047-2727(74)90003-6)
- McFadden, D., & Train, K. (2000). Mixed mnl models for discrete response. *Journal of applied Econometrics*, 15(5), 447–470.
- Mnih, V. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540), 529–533.
- Mokhtarian, P. L. (2016). Discrete choice models' ρ_2 : A reintroduction to an old friend. *Journal of choice modelling*, 21, 60–65.
- Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., ... others (2015). Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*.
- Nova, G., van Cranenburgh, S., & Hess, S. (2024). Understanding the decision-making process of choice modellers. *arXiv preprint arXiv:2411.01704*.

- Ortelli, N., Hillel, T., Pereira, F. C., de Lapparent, M., & Bierlaire, M. (2021). Assisted specification of discrete choice models. *Journal of choice modelling*, *39*, 100285.
- Paz, A., Arteaga, C., & Cobos, C. (2019). Specification of mixed logit models assisted by an optimization framework. *Journal of choice modelling*, *30*, 50–60.
- Plaat, A. (2022). *Deep reinforcement learning* (Vol. 10). Springer.
- Rodrigues, F., Ortelli, N., Bierlaire, M., & Pereira, F. C. (2020). Bayesian automatic relevance determination for utility function specification in discrete choice models. *IEEE Transactions on Intelligent Transportation Systems*, *23*(4), 3126–3136.
- Ross, S., & Bagnell, D. (2010). Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 661–668).
- Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., ... Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.
- Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.
- Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Sfeir, G., Nova, G., Hess, S., & van Cranenburgh, S. (2025). Can large language models assist choice modelling? insights into prompting strategies and current models capabilities. *arXiv preprint arXiv:2507.21790*.
- Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. *A Bradford Book*.
- Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, *12*.
- Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1-2), 181–211.
- Swait, J. (2001). A non-compensatory choice model incorporating attribute cutoffs. *Transportation Research Part B: Methodological*, *35*(10), 903–928.
- Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., ... Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, *30*.
- Train, K. E. (2009). *Discrete choice methods with simulation*. Cambridge university press.
- Tsitsiklis, J., & Van Roy, B. (1996). Analysis of temporal-difference learning with function approximation. *Advances in neural information processing systems*, *9*.
- van Cranenburgh, S., Guevara, C. A., & Chorus, C. G. (2015). New insights on random regret minimization models. *Transportation Research Part A: Policy and Practice*, *74*, 91–109.
- Van Cranenburgh, S., Wang, S., Vij, A., Pereira, F., & Walker, J. (2022). Choice modelling in the age of machine learning-discussion paper. *Journal of choice modelling*, *42*, 100340.
- Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 30).
- Walker, J. L., & Li, J. (2007). Latent lifestyle preferences and household location decisions. *Journal of Geographical Systems*, *9*, 77–101.
- Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning* (pp. 1995–2003).

- Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, 8, 279–292.
- Wong, A., Bäck, T., Kononova, A. V., & Plaat, A. (2023). Deep multiagent reinforcement learning: Challenges and directions. *Artificial Intelligence Review*, 56(6), 5023–5056.
- Zhang, K., Yang, Z., & Başar, T. (2021). Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, 321–384.
- Zhang, S., & Sutton, R. S. (2017). A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*.
- Zhong, Z., Yan, J., Wu, W., Shao, J., & Liu, C.-L. (2018). Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2423–2432).
- Zoph, B. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.