# Safe Reward Learning from Human Preferences and Justifications

Ilias Kazantzidis[1], Timothy J. Norman[1], Yali Du[2] and Christopher T. Freeman[1]
[1]*School of Electronics and Computer Science, University of Southampton, Southampton, United Kingdom*
[2]*Department of Informatics, King's College London, London, United Kingdom*
*ik3n19@soton.ac.uk, t.j.norman@soton.ac.uk, yali.du@kcl.ac.uk, ctf1@soton.ac.uk*

Keywords: Safe Learning from Human Preferences, Safe Reinforcement Learning, Human-Agent Interaction, Human-Robot Interaction, Human-in-the-loop Machine Learning, Learning Human Values and Preferences.

Abstract: We address the problem of learning autonomous safe agent behaviour with unknown dynamics and reward functions, where traditional Reinforcement Learning is impossible. We present *DROPJ*, a human-centred algorithm that maximises safety during both *training* and *deployment*. We first learn a world model (a learned simulation) from a set of past real-world trajectories. A user then *plays the game* in the simulation to draw several informative virtual trajectories. From these, we extract pairs of trajectory segments and present them to a user to elicit their preference over these segments and the reason (*justification*) for that preference. With this feedback, a reward model is trained, which is used to deploy the agent with Model Predictive Control. We find that generating trajectories from user trials significantly reduces the computational cost of training, and significantly improves performance during deployment. In that context, we show that the use of preferences rather than other types of feedback substantially improves the performance. We further demonstrate that the use of justifications associated with safety requirements results in safer policies.

## 1 INTRODUCTION

Reinforcement Learning (RL) is a powerful method for training autonomous agents (Mnih et al., 2015; Silver et al., 2016). However, when applied in practical settings (e.g. in robotics or autonomous vehicles) *without a (near-)perfect simulator (dynamics function)*, safety is a major issue. For instance, a car has to collide with real obstacles to learn that collisions are unsafe. Different methods have targeted this safe exploration problem (Garcıa and Fernández, 2015; Turchetta et al., 2020), but often fail to guarantee safety both during *training* and *deployment*. In complex domains, *reward functions* may also be unavailable or poorly aligned with human objectives (Leike et al., 2018). Learning a model from human feedback (Knox and Stone, 2009), particularly a reward model from pairwise preferences (Christiano et al., 2017), offers a promising solution, as the learned policy aligns better with human goals. Yet, this remains unfeasible in the real world, as the agent would still encounter unsafe states, risking damage. Moreover, continuous human oversight methods during training (Saunders et al., 2018) impose an infeasible human burden, because the agent must both decide the precise moments to query, and pause the task to do so.

An approach that avoids these issues is to build a dynamics model from an offline dataset of past real-world reward-free trajectories — effectively an imperfect simulator — and construct a reward model from human feedback on trajectories extracted safely inside that simulation. Using this reward model, either an RL policy can be learned inside that simulated world and transferred to the real, or, better, an agent can be deployed directly to the real world using Model Predictive Control (MPC; Garcia et al., 1989), a powerful model-based control technique. Yet methods that have taken this direction (Reddy et al., 2020; Rahtz et al., 2022) have two important limitations: (i) generating informative (i.e. diverse) trajectories for human feedback is computationally intensive, making it hard to scale in complex environments; and (ii) unsafe-state visits can still occur during deployment if safety requirements are not captured by the reward model.

We introduce ***DROPJ***: ***D****ream-World* ***R****eward Learning from* ***O****ne-Shot Human* ***P****references and* ***J****ustifications*, a human-centred method that maximises safety during both *training* and *deployment*.

Accepted for publication in the Proceedings of the *18th International Conference on Agents and Artificial Intelligence (ICAART 2026)*, where it was presented. The code is available at github.com/ilkaza/DROPJ.

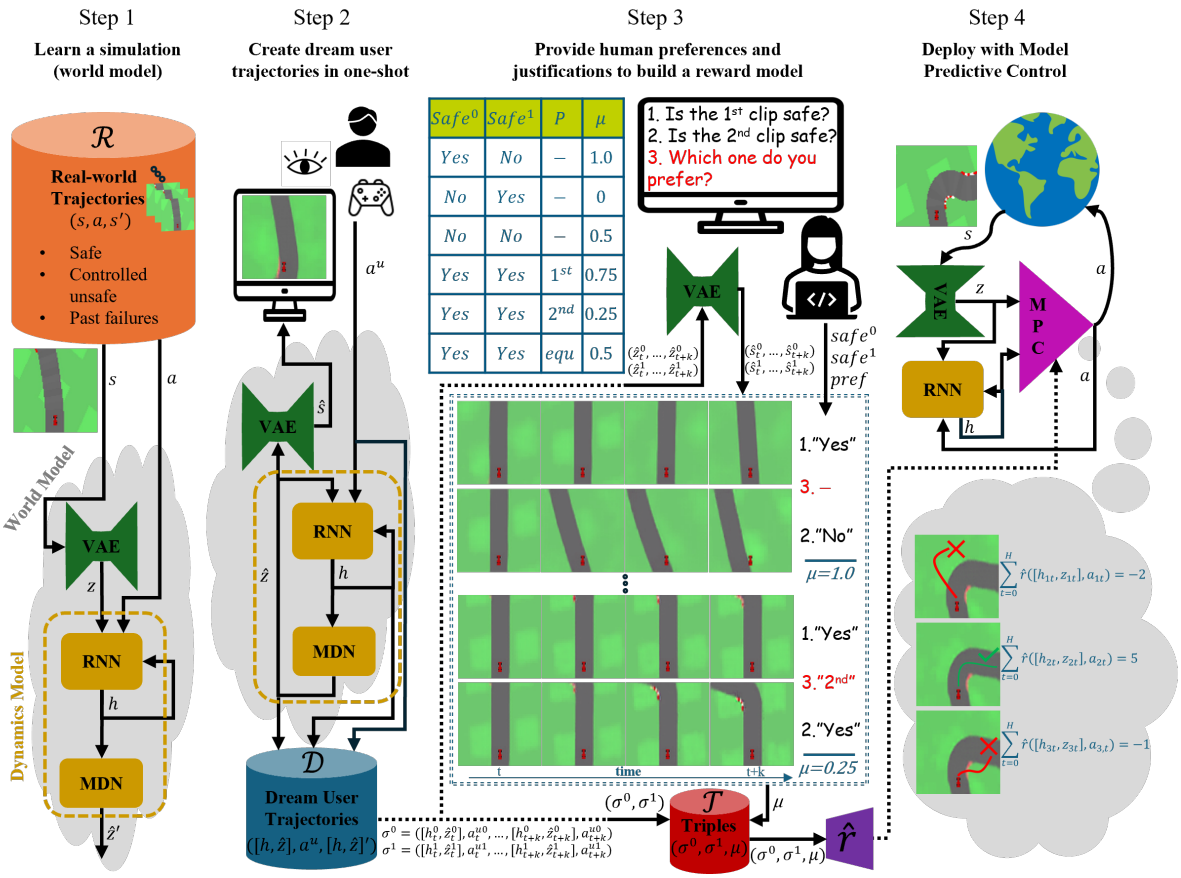



Figure 1: DROPJ steps. Novel contributions are in Steps 2 and 3. Step 3 shows the single safety justification case.

As shown in Figure 1, in Step 1 we construct a world model (a learned representation of the environment with vision and memory — also referred as learned simulation or dream world) from past reward-free trajectories (episodes) collected in the real world.[1] In Step 2, a user 'plays the game' for a few minutes in the world model; i.e. generates several *dream trajectories*. Here, exploration and unsafe-state visits are allowed. In Step 3, pairs of segments are extracted from the dream trajectories and presented to a user to express a preference and a *justification* for their choice. Justifications shape the preference labels $\mu$ of the reward model, beyond the typical values $\{0, 1, 0.5\}$ representing preference for one segment, the other, or indifference, respectively. Then, a reward model is trained with these labels as in (Christiano et al., 2017). In Step 4, equipped with the dynamics and reward models, we directly deploy the agent with MPC. DROPJ solves the two earlier limitations: in Step 2, we reduce the significant computational cost of generating hypothetical trajectories by enabling *one-shot* generation of informative trajectories from the user; and in Step 3, we maximise safety during deployment, using *safety justifications* to accompany preferences.

Our method (in contrast to most prior literature in RL and human-based methods) **assumes entirely unknown dynamics and reward functions**, reflecting a realistic real-world problem. We use real users, and metrics for performance, safety, human burden, and computational cost. The two core contributions are:

- A novel 'one-shot' technique, where examples for human feedback are drawn within a learned simulation from a user, improving the computational time and the performance compared to prior methods.
- The extension of the justifications over preferences technique. We find justifications during deployment can enhance user-prescribed aspects of safety, with potential trade-offs in performance or other aspects.

[1] In the experiments, we use image-based environments without making use of rewards. The user-extracted trajectories from the perfect simulator are analogous to (equal) those that could be collected in a real-world scenario; e.g. by teleoperating a physical robot with a top-down camera.

## 2 RELATED WORK

At a high level, our method relates to works on learning RL policies in safety-critical environments, which tackle safe exploration (Achiam et al., 2017; Chow et al., 2018; Ray et al., 2019; Moerland et al., 2023). However, those assume an available reward function and may also devise a cost function. It also connects with techniques on incorporating human input (Ng et al., 2000; Hussein et al., 2017; Brown et al., 2018; Warnell et al., 2018; Brown et al., 2019; Guo et al., 2022; Li et al., 2025), and particularly human preferences, the most effective type of feedback in aligning policies with human objectives (Christiano et al., 2017; Biyik and Sadigh, 2018; Lee et al., 2021; Park et al., 2022; Kim et al., 2023; Karlaus and Schwenker, 2025). However, most of these assume perfect simulators (known dynamics function), overlooking safety during training. Conversely, methods that the user actively intervenes to avoid unsafe states (Saunders et al., 2018; Goecks et al., 2019; Frye and Feige, 2019; Kazantzidis et al., 2022a) struggle to scale in complex environments. Learning from human input has also grown attention lately in combination to language models, including safety aspects (Ouyang et al., 2022; Gu et al., 2023; Rafailov et al., 2024; Shi et al., 2024; Lou et al., 2024; Dai et al., 2024).

Our work is related to *world models*. A world model can be built from an offline dataset of reward-free trajectories and consists of a *vision* component that compresses a high-dimensional image input into a condensed latent representation, and a *memory* one that predicts the next latent state based on historical information, forming a dynamics model inside a virtual environment. In (Ha and Schmidhuber, 2018), while an RL policy was trained in this virtual world and transferred to the real, the RL reward was a simple time-dependent function; instead, we learn a fresh reward model from human feedback in the dream. Later work (Hafner et al., 2019; Hafner et al., 2023; Huang et al., 2023) improved the long-term prediction stability, and scalability to complex environments, but relied on either potentially-unsafe online data collection in the real world or the oracle reward function.

The approach most similar to ours involves learning a dynamics model from an offline dataset of reward-free trajectories, and then learning a reward model from user feedback on trajectory segments generated through the dynamics model. The closest framework to ours is ReQueST (Reddy et al., 2020; Rahtz et al., 2022), where an iterative process, of generating trajectory segments, providing human feedback on them, training a reward model, and then generating new segments based on the improved reward model, is repeatedly conducted. The trajectory segments must be informative and are synthesised by optimising proxies designed to maximise the value of information. This is performed by gradient-based optimisation and is computationally expensive, especially for long segment lengths. Thus, the user remains on a long-lasting loop. We instead expedite trajectory segment generation with a user who acts in the simulation and safely explores and returns informative examples in *one-shot*. Earlier works also do not use preferences feedback. In contrast, we do and accompany them with *justifications* to maximise safety at deployment.

## 3 PRELIMINARIES

**Learning a reward model from preferences** In the standard approach of learning a reward model from human preferences (Christiano et al., 2017), a segment $\sigma$ of length $k$ is defined as a sequence of states and actions $(s_t, a_t, \ldots, s_{t+k}, a_{t+k})$. For any two segments, $\sigma^0$ ($1^{st}$ segment) and $\sigma^1$ ($2^{nd}$ segment), a user provides a preference $P \in \{1^{st}, equ, 2^{nd}\}$, where $P = 1^{st} \iff \sigma^0 \succ \sigma^1$ (preference for $\sigma^0$), $P = 2^{nd} \iff \sigma^1 \succ \sigma^0$ (preference for $\sigma^1$) and $P = equ \iff \sigma^0 \sim \sigma^1$ (indifference). $P$ is mapped to the preference ground-truth label $\mu \in \{0, 0.5, 1\}$, where 1 and 0 represent preference for $\sigma^0$ and $\sigma^1$, respectively, and 0.5 indifference. The answer is stored in a dataset $\mathcal{T}$, as a triple $(\sigma^0, \sigma^1, \mu)$. Using the Bradley-Terry model (Bradley and Terry, 1952), the preference predictions can be modelled as the likelihoods: $p[P=1^{st}] = \frac{\exp \sum_t \hat{r}(s_t^0, a_t^0)}{\sum_{i \in \{0,1\}} \exp \sum_t \hat{r}(s_t^i, a_t^i)}$, $p[P=2^{nd}] = \frac{\exp \sum_t \hat{r}(s_t^1, a_t^1)}{\sum_{i \in \{0,1\}} \exp \sum_t \hat{r}(s_t^i, a_t^i)}$. Then, the reward model $\hat{r} = \hat{r}(s,a)$ can be learned, by minimising the cross-entropy loss:

$$\mathcal{L}^{\hat{r}} = - \sum_{(\sigma^0, \sigma^1, \mu) \in \mathcal{T}} \mu \log p[P = 1^{st}] + (1-\mu) \log p[P = 2^{nd}] \quad (1)$$

Equation 1 is used in Step 3 to train a reward model, but this time with justifications being integrated in $\mu$, and segments drawn from the dream world.

**Justifications in prior work** Justifications over preferences $J$ give essentially a reason for the preference choice (Kazantzidis et al., 2022a; Kazantzidis et al., 2022b). This enables a more information-rich preference label $\mu$ with more possible discrete values in the $[0,1]$ range. In the case of a single justification (instead of multiple ones) related to safety (in addition to a *default* one that always exists) and state-action pairs (instead of segments), the user, besides a preference $P$, provides a $J \in \{warning, no\_warning\}$, where *warning* means at least one proposed action is unsafe, and *no_warning* means both actions are

safe. If $w_s, w_{def} \in (0.5, 1]$ are the weights for the safety, and the default justification (e.g. preference *because* the action moves the agent closer to a goal state), respectively, then a mapping $P \times J \to \mu$, where $\mu \in \{1-w_s, 1-w_{def}, 0.5, w_{def}, w_s\}$ and $w_s > w_{def}$, can foster safer behaviour than the typical $P \to \mu$. For instance, a sensible choice of weights at $(w_s = 1, w_{def} = 0.75)$ would lead to: $\mu = 1$ representing total preference for the $1^{st}$ action *because* the $2^{nd}$ action is unsafe (i.e. $P = 1^{st}, J = warning$); $\mu = 0.75$ representing simple preference for the $1^{st}$ action *because* of some other default reason (i.e. $P = 1^{st}, J = no\_warning$); and $\mu = 0$ and $\mu = 0.25$, conversely. Step 3 uses justifications with three key extensions: (i) user feedback (preferences with justifications) is provided offline over dream trajectory segments, instead of online over state-action pairs; (ii) $\mu$ is used as a label for the reward model trained in Equation 1, instead of directly optimising a policy; and (iii) multiple safety justifications can be provided, instead of a single one.

# 4 DROPJ

**Problem setting** Our method is designed for the following setting. The unsafe states are unknown to the agent, and we have access neither to the state transition dynamics function, nor to an oracle reward function of the environment. The state and action spaces can be continuous, and the input image-based. The resources are some prior real-world reward-free trajectories and a user with reasonable understanding of the environment. Querying the user is costly. The final goal is to build a reward model from human input and use it for deployment. DROPJ is split into the four steps shown in Figure 1 and Algorithm 1.

Formally, in **Step 1**, we assume an available offline dataset $\mathcal{R}$ of $M$ past real-world trajectories of $L$ state-action transitions $(s, a, s')$ each, with states $s$ and next states $s'$ in a high-dimensional, continuous space (such as images) $\mathcal{S} = \mathbb{R}^n$ and actions $a$ also in a continuous space $\mathcal{A}$. $M \times L$ should be large enough and $\mathcal{R}$ should have enough state and action coverage to build a sufficiently-good world model. Thus, apart from safe trajectories, it should include past failures or some controlled unsafe trajectories (e.g. a vehicle gently stopping against foam barriers).[2] Initially, using $(s, s')$, the vision component of the world model is trained, which is a variational auto-encoder (VAE; Kingma, 2013) with: an encoder $f_{enc} : \mathcal{S} \to \mathcal{Z}$, transforming $s$ to an embedding $z \in \mathcal{Z}$ where $\mathcal{Z} = \mathbb{R}^d$ is a low dimensional latent space ($d \ll n$); and a decoder $f_{dec} : \mathcal{Z} \to \mathcal{S}$, reconstructing an image $s$ from a code $z$. Then, the encoded $(s, a, s')$ transitions from $\mathcal{R}$ train the memory component, which is a Recurrent Neural Network (RNN; Hochreiter and Schmidhuber, 1997) $f_{RNN} : \mathcal{H} \times \mathcal{Z} \times \mathcal{A} \to \mathcal{H}$, providing the memory state $h \in \mathcal{H} = \mathbb{R}^m$. Next, $h$ is fed into a Mixture Density Network (MDN; Bishop, 1994) $f_{MDN} : \mathcal{H} \to \mathcal{Z}$, which outputs the prediction of the next latent state $\hat{z}'$. That yields a learned dynamics model in $\mathcal{Z}$.

In **Step 2**, instead of creating trajectory segments for user feedback via an iterative computationally-costly process, we use a more efficient idea. Since a prediction $\hat{z}$ can be fed, not only back to the RNN as the next latent state, but also back to the VAE decoder to be reconstructed as $\hat{s} = f_{dec}(\hat{z})$ and observed from a user $u$ in a monitor, $u$ can take actions $a^u$ in the dream world, and in *one-shot* create $T$ dream user trajectories of $L$ state-action transitions $([h, \hat{z}], a^u, [h, \hat{z}]')$, stored in a dataset $\mathcal{D}$. Importantly, the memory $h$ is also stored, concatenated with $\hat{z}$ to form the dream state $[h, \hat{z}]$. At this step, $u$ explores states that would otherwise be tough to discover, gathering enough diverse examples and deliberately visiting unsafe states. That is, since $u$ can use their understanding of the task, the dream trajectories in $\mathcal{D}$ are gathered in one-shot and quickly (no need for a large $T$), and are more informative than the ones generated from a costly iterative process. We clarify that $\mathcal{D}$ is used, not to learn by imitation, which is known to generalise poorly in unseen cases, but simply to form the foundation for learning a reward model, known for its good transferable capabilities. Considering this one-shot technique and its scalability in practice, for robot navigation, the same way a user teleoperates a robot via the keyboard or joystick, they can do it in the learned simulation. For robotic manipulation, leap motion technologies can enable a user to guide the robot's arm within the learned simulation as well. For vehicles, full teleoperation driving setups widely used in autonomous vehicle testing could be employed.

In **Step 3**, the goal is to learn a reward model $\hat{r} = \hat{r}([h, z], a)$ from human feedback. In theory, we can use any type of feedback, exploiting the benefits of Step 2. However, aiming for aligned and safe agent behaviour at deployment, we propose learning from preferences with *safety justifications*. First, we create $K$ pairs of dream trajectory segments $(\sigma^0, \sigma^1)$. A dream segment of length $k$ is now $\sigma = ([h_t, \hat{z}_t], a^u_t, \ldots, [h_{t+k}, \hat{z}_{t+k}], a^u_{t+k})$, sampled from $\mathcal{D}$. A pair of dream segments is shown to the user after passing from the VAE decoder: $(\hat{s}^i_t, \ldots, \hat{s}^i_{t+k}) = f_{dec}((\hat{z}^i_t, \ldots, \hat{z}^i_{t+k}))$, $i = 0, 1$. A preference $P$ could have been now elicited from the user as in the standard preference reward learning (Christiano et al., 2017).

However, we additionally elicit *safety justifica-*

[2] For domains with catastrophic states, such as a car near a cliff, unsafe states are considered the ones 'near the cliff'. Hence, they are still observable via controlled procedures.

**Algorithm 1** DROPJ

**// Training (Steps 1-3)**
**Input:** offline dataset $\mathcal{R}$ of $M$ real-world trajectories, each with $L$ transitions $(s,a,s')$, where $s,s' \in \mathcal{S} = \mathbb{R}^n, a \in \mathcal{A}$; user $u$; number of dream user-generated trajectories $T$; number of queries to the user (pairs of dream trajectory segments created) $K$; length of segment $k$; justification weights $w_s, w_{def} \in (0.5,1]$ for a single safety justification, or $w_1,\ldots,w_N \in (0.5,1]$ for multiple justifications
**Output:** reward model $\hat{r}$
**// Step 1: Learn a World Model**
Train a VAE encoder $f_{enc} : \mathcal{S} \to \mathcal{Z}$ and decoder $f_{dec} : \mathcal{Z} \to \mathcal{S}$ from $\mathcal{R}$, where $\mathcal{Z} = \mathbb{R}^d$ and $(d \ll n)$
Train the dynamics model from $\mathcal{R}$: an RNN $f_{RNN} : \mathcal{H} \times \mathcal{Z} \times \mathcal{A} \to \mathcal{H}$ and an MDN $f_{MDN} : \mathcal{H} \to \mathcal{Z}$
**// Step 2: Collect Dream User Trajectories in One-shot**
**for** _ from 1 to $T$ **do**
  **for** _ from 1 to $L$ **do**
    Reconstruct the last predicted latent state and display the frame to the monitor $\hat{s} = f_{dec}(\hat{z})$
    Let user $u$ take an action $a^u$
    Run a step in the simulation and generate the next latent state $\hat{z}' = f_{MDN}(f_{RNN}(h,\hat{z},a^u))$
    Store the dream transition $\mathcal{D} \leftarrow \mathcal{D} \cup ([h,\hat{z}],a^u,[h,\hat{z}]')$
  **end for**
**end for**
**// Step 3: Learn a Reward Model from Preferences and Justifications**
**for** _ from 1 to $K$ **do**
  Sample a pair of segments of length $k$ (uniformly at random) from the dream trajectories $(\sigma^0,\sigma^1) \sim \mathcal{D}$
  Reconstruct the segments and display them to the monitor $(\hat{s}^i_t,\ldots,\hat{s}^i_{t+k}) = f_{dec}((\hat{z}^i_t,\ldots,\hat{z}^i_{t+k}))$, $i = 0,1$
  Query the user according to: the offline protocol of Table 1 to infer $\mu = \mu(Safe^0, Safe^1, P, w_s, w_{def})$ for a single safety justification; or Equation 2 to infer $\mu = \mu(J_1,\ldots,J_N,P,w_1,\ldots,w_N)$ for multiple justifications
  Store the triple $\mathcal{T} \leftarrow \mathcal{T} \cup (\sigma^0,\sigma^1,\mu)$
**end for**
Train the reward model by minimising with gradient descent Equation 1

**// Deployment with MPC (Step 4)**
**Input:** planning horizon $H$; number of steps until replan $R$; number of sensible random action sequences $S$
$s \leftarrow$ initial state
$done = False$
$h \leftarrow$ initialise with 0's
// $return = 0$ for evaluation
**while** *not done* **do**
  $z = f_{enc}(s)$
  //Running sample-based MPC planning
  **if** $plan\_step\%R == 0$ **then**
    Generate $S$ sensible random action sequences:
      $\mathcal{A}_{mpc} = \{\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_S\}$, where $\mathbf{a}_i = (a_{i0},a_{i1},\ldots,a_{iH})$
    **for** each $\mathbf{a}_i \in \mathcal{A}_{mpc}$ **do**
      $h_{i0} = h,\ z_{i0} = z$
      Use the dynamics model to simulate the trajectory:
        $([h_{i0},z_{i0}],a_{i0},\ldots,[h_{iH},z_{iH}],a_{iH})$
      Compute the predicted return of the trajectory:
        $R_i = \sum_{t=0}^{H} \hat{r}([h_{it},z_{it}],a_{it})$
    **end for**
    Select the action sequence of the trajectory with the highest predicted return:
      $\mathbf{a}^* = \arg\max_{\mathbf{a}_i \in \mathcal{A}_{mpc}} R_i$
    Keep the first $R$ actions of the chosen action sequence:
      $\pi_{mpc}(s) = (a^*_1,a^*_2,\ldots,a^*_R)$
    Reset the planning step counter:
      $plan\_step = 0$
  **end if**
  // Execute the next action from the planned sequence
  $a = \pi_{mpc}(s)[plan\_step]$
  $plan\_step = plan\_step + 1$
  $s, done \leftarrow env\_step(a)$ // return *reward* as well for evaluation
  // $return = return + reward$ for evaluation
  $h = f_{RNN}([h,z],a)$
**end while**

*tions*. For a single one, we use an offline query protocol for segments, equivalent to the online one described in Section 3 for state-action pairs. Precisely, to infer the ground-truth $\mu$ we query the user: (i) whether $\sigma^0$ is safe, i.e. $Safe^0 \in \{Yes, No\}$; (ii) whether $\sigma^1$ is safe, i.e. $Safe^1 \in \{Yes, No\}$; and only if both answers are *Yes*, we query (iii) which one they prefer, i.e. $P \in \{1^{st}, equ, 2^{nd}\}$. Table 1 shows the equivalence of the mappings $P \times J \to \mu$ and $Safe^0 \times Safe^1 \times P \to \mu$. When not both $Safe^0$ and $Safe^1$ are *Yes*, we can automatically infer $\mu$.[3] Although the two protocols seem similar in terms of human burden, we chose the offline, since it is more intuitive for queries answered offline in a simulation. Also, for domains with multiple safety factors, we generalise the framework to $N$ justifications $J_1, \ldots, J_N \in \{0,1\}$ (i.e. whether a justification is active), with associated weights $w_1, \ldots, w_N \in (0.5, 1]$ set based on their severity. With $J_N$ being the default justification (not necessarily related to safety), $\mathbb{I}(\cdot)$ the indicator function, and with decreasing safety severity ($w_1 \geq w_2 \geq \cdots \geq w_N$), the preference label is:

$$\mu = \sum_{k=1}^{N} \left[ \left( \prod_{i=1}^{k-1} \mathbb{I}(J_i = 0) \right) \cdot \mathbb{I}(J_k = 1) \cdot \left( w_k \cdot \mathbb{I}(P = 1^{st}) + 0.5 \cdot \mathbb{I}(P = equ) + (1 - w_k) \cdot \mathbb{I}(P = 2^{nd}) \right) \right], \tag{2}$$

which means that $\mu$ is equal to the justification weight (or its complement or 0.5 for indifference) of the *most severe active justification*.[4] Finally, for every complete answer, a triple $(\sigma^0, \sigma^1, \mu)$ is stored in $\mathcal{T}$, and $\mathcal{T}$ is used to train the reward model $\hat{r}$, exactly as in Equation 1. Now a safety objective is encoded in the learned reward model, which prioritises it over other performance objectives (such as driving fast) or less severe safety objectives. Thus, given the good alignment and generalisation properties of preference reward models, this safe behaviour can now generalise across unforeseen occasions during deployment.

Finally, in **Step 4**, an MPC policy $\pi_{mpc}$ can be deployed directly in the real world by planning with the learned dynamics and reward models. Although an RL policy can be trained with $\hat{r}$ in the virtual environment and transferred to the real, this requires additional time, while MPC tends to work very well in practice. Similar to prior work, we use a sampling-based approach for MPC planning. From the real state $s$, the controller is given the current encoded state $z = f_{enc}(s)$ and the last memory state $h$. The controller generates $S$ sensible random action sequences $\mathcal{A}_{mpc} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_S\}$, where $\mathbf{a}_i = (a_{i0}, a_{i1}, \ldots, a_{iH})$ is an action sequence over the planning horizon $H$. Executing them with the dynamics model, creates the 'imagined' trajectories $\{([h_{i0}, z_{i0}], a_{i0}, \ldots, [h_{iH}, z_{iH}], a_{iH}) : h_{i0} = h,\ z_{i0} = z,\ i = 1, \ldots, S\}$. The predicted return of each trajectory $i$ is: $R_i = \sum_{t=0}^{H} \hat{r}([h_{it}, z_{it}], a_{it})$ and the action sequence of the highest-return trajectory is chosen: $\mathbf{a}^* = \arg\max_{\mathbf{a}_i \in \mathcal{A}_{mpc}} R_i$. From $\mathbf{a}^* = (a_1^*, a_2^*, \ldots, a_H^*)$, only the first few $R$ actions are executed $\pi_{mpc} = (a_1^*, a_2^*, \ldots, a_R^*)$, until MPC replans in the same way.

Table 1: Equivalence of the online (warning when at least one action is unsafe) and offline (querying explicitly the safety of each segment) protocol for one safety justification.

| **Online** | | **Offline** | | | |
|---|---|---|---|---|---|
| $P$ | $J$ | $Safe^0$ | $Safe^1$ | $P$ | $\mu$ |
| $1^{st}$ | *warning* | *Yes* | *No* | – | $w_s$ |
| $2^{nd}$ | *warning* | *No* | *Yes* | – | $1 - w_s$ |
| *equ* | *warning* | *No* | *No* | – | 0.5 |
| $1^{st}$ | *no_warning* | *Yes* | *Yes* | $1^{st}$ | $w_{def}$ |
| $2^{nd}$ | *no_warning* | *Yes* | *Yes* | $2^{nd}$ | $1 - w_{def}$ |
| *equ* | *no_warning* | *Yes* | *Yes* | *equ* | 0.5 |

# 5 VALUE OF JUSTIFICATIONS

Here, we show that learning from safety justifications biases the reward model towards safer behaviour over other objectives. We prove it for one safety justification, while extending it to multiple is straightforward.

**Theorem 1.** *Reward models that are learned using safety justifications prioritise safe behaviour over other performance objectives.*

*Proof.* Let $\mu \in \{1 - w_s, 1 - w_{def}, 0.5, w_{def}, w_s\} \subset [0,1]$ be the preference label with:

- $w_s$: the weight assigned to preferences with a justification about safety (e.g. driving on the grass).
- $w_{def}$: the weight assigned to preferences with a default justification (e.g. driving faster).
- $w_s > w_{def} > 0.5$.
- $\mu = 0.5$ when both segments are unsafe, or are safe but the user expresses indifference.

Let the dataset $\mathcal{T}$ with triples of the queried segments and the preference labels $(\sigma^0, \sigma^1, \mu)$ be partitioned in:

- $\mathcal{T}_{safety}$: samples of queries with at least one unsafe segment — justified by a safety factor.
- $\mathcal{T}_{def}$: samples of queries with both segments safe — justified by a performance factor unrelated to safety.

[3] Figure 1 shows two examples in Step 3. In the top one, the $1^{st}$ (upper) clip is preferred automatically *because* the car stays safely on the road, unlike the $2^{nd}$; in the bottom one, the $2^{nd}$ clip is preferred *because* the car travels further.

[4] Equivalently, and for efficiency in human burden, we elicit only the most severe justification's *name*. For instance, in Figure 7 (down), for justifications about 'stepping' to cars, chuckholes, grass, and default reasons, if $w_{car} \geq w_{chuck} \geq w_{grass} \geq w_{def}$, the correct answer $J = Car$, $P = 2^{nd}$ (Right clip), makes $\mu = 1 - w_{car}$.

Then from Equation 1, the cross-entropy loss minimised by the reward model becomes:

$$\mathcal{L}^{\hat{r}} = -\sum_{(\sigma^0,\sigma^1,\mu_s)\in\mathcal{T}_{safety}} \mu_s \log p[P=1^{st}] + (1-\mu_s)\log p[P=2^{nd}] \\ -\sum_{(\sigma^0,\sigma^1,\mu_{def})\in\mathcal{T}_{def}} \mu_{def} \log p[P=1^{st}] + (1-\mu_{def})\log p[P=2^{nd}], \quad (3)$$

where $\mu_s \in \{1-w_s, 0.5, w_s\}$ and $\mu_{def} \in \{1-w_{def}, 0.5, w_{def}\}$. Since $w_s > w_{def} > 0.5$, i.e. the safety-justified preferences assign greater magnitude labels — further from 0.5 — than the non-safety ones, this introduces a higher gradient magnitude for samples with a safety justification. The loss puts more pressure on the reward model to fit such preferences (acting like an importance-weighted training objective). Thus, the learned reward model will inherently prioritise safety over other performance aspects. □

# 6 EVALUATION

## 6.1 Experimental Setup

We use the Car Racing (CR) environment (Brockman, 2016), where the states are RGB images and the actions control steering, gas and brake (continuous spaces). The track is random in each trial. For evaluation we use the same reward structure with (Reddy et al., 2020): +10 for stepping on a new patch (tile); -1 for being in grass or kerbs; and 0 for remaining at an already-visited tile. We also create a more challenging variant, Obstacle Car Racing (OCR) (Figure 7 (up)), featuring static chuckholes that the car slows down when driving on them, and dynamic scripted-cars (maintaining a relatively low speed) it can collide with. The world model in OCR trained for 42 hours in an HPC system. Our method requires real users for reliable results, as the oracle reward function often used to simulate near-accurate synthetic human feedback does not match the dream-space segments. Thus, CR and OCR were convenient for direct interaction via keyboard arrows and feedback GUIs (Figures 10–12). With them we could aptly show the trade-offs among different performance (driving fast) and safety aspects (grass, chuckholes, cars), and avoid the typically vast amounts of data and compute resources typically required for training world-models in more complicated environments (Assran et al., 2025). Given these resources, our implementation designed for continuous spaces, is easily transferable, including hardware (e.g. robot navigation with a top-down camera).

We evaluate: (i) *performance*, using the return during deployment; (ii) *safety*, using crash rates for grass (proportion of steps on grass or kerbs over the episode length $L$=1000), chuckholes (chuckholes stepped / chuckholes passed), and cars (cars crashed / cars passed); (iii) *human burden*, i.e. the number of queries the user answers (precisely, the exact time they spend on training); and (iv) *computational cost*, i.e. the time the computer takes to run time-intensive computations, while the human has to remain in the loop. All models are evaluated over 10 trials. We run the following algorithms (3–5 introduced here):

1. ***DRQV2*** (Yarats et al., 2021)**;** a state-of-the-art model-free RL algorithm for image-based input, which incorporates augmentation strategies such as random shifts in observation space to regularise the learning process. We allow it to use the oracle reward function and use it as an upper bound.
2. ***ReQueST*****;** it iterates among providing human feedback, training the reward model and generating new trajectories via optimisation. The feedback is *sparse* reward labels (i.e. *good* for visiting a new tile, *unsafe* for stepping on grass or kerbs and *neutral* for staying on visited tiles) to train a classifier-based reward model (see Appendix A).
3. ***DROS*** *(**D**ream-world **R**eward learning from **O**ne-shot **S**parse labels)***;** the user creates the dream trajectories with the ***O**ne-shot* technique, and provides ***S**parse labels* similarly to ReQueST.
4. ***DROP*** *(**D**ream-world **R**eward learning from **O**ne-shot **P**references)***;** it uses the ***O**ne-shot* technique and ***P**references* (rather than sparse labels). When indifference (***e**qual*) is allowed, we call it ***DROPe***.
5. ***DROPJ*** *(**D**ream-world **R**eward learning from **O**ne-shot **P**references and **J**ustifications)***;** it is as ***DROP*** with the inclusion of ***J**ustifications*.

Details on providing user feedback can be found in Appendix A. Also, to create a sufficiently-good and realistic dataset $\mathcal{R}$ of past real-world trajectories (assumed to already exist in the problem setting), we played the game many times in the real environment ($M = 600$ for CR — the reason we stopped there, and an ablation for the world model quality can be found in Appendix B.1), with mostly safely-exploring and sometimes taking unsafe actions (representing past failures or controlled unsafe examples). Finally, we used similar architectures and hyperparameters with (Ha and Schmidhuber, 2018) for the world model, (Reddy et al., 2020) for the reward model from sparse labels and MPC, and (Lee et al., 2021) for the reward model from preferences (details in Appendix A). Our code and the full clips for the Figures 7, 11, 12 and 13 are available at github.com/ilkaza/DROPJ.

## 6.2 Results

We first note that methods 2–5 have 0 unsafe states at training, as that occurs inside the dream. Instead,

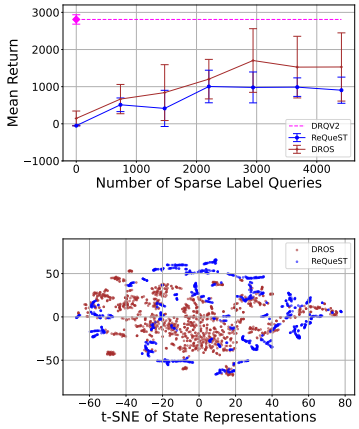



Figure 2: **Q1:** Impact of the One-shot technique.

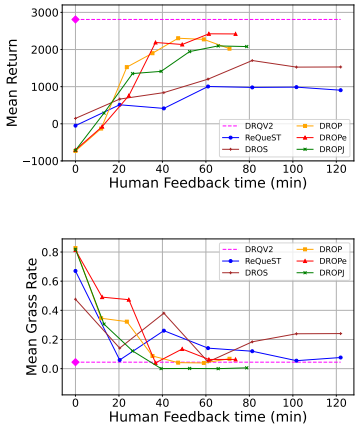



Figure 3: **Q2:** Impact of using preferences instead of sparse labels.

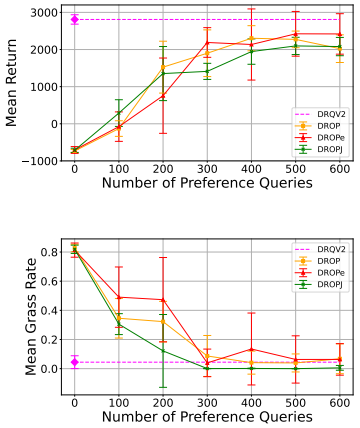



Figure 4: **Q3:** Impact of including a safety justification in preferences.

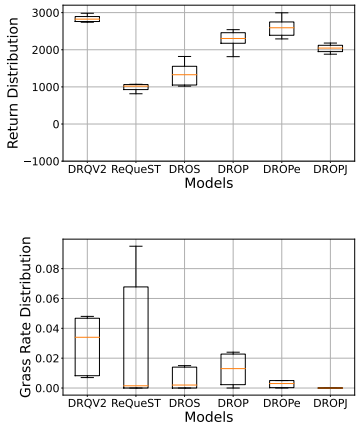



Figure 5: Return and grass rate distributions of the best models.

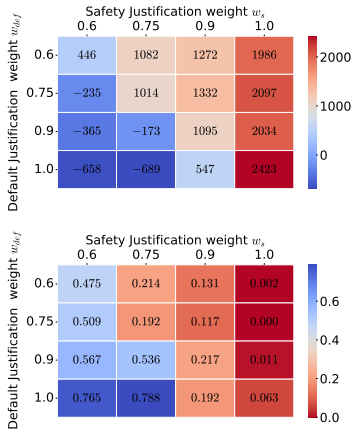



Figure 6: Various $w_{def}$, $w_s$; mean return (up) and mean grass rate (down).

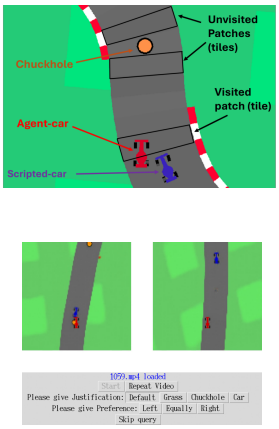



Figure 7: OCR and multiple justifications DROPJ GUI.

DRQV2, which runs RL in the real environment, suffers thousands of unsafe steps before reaching the optimal performance (upper-bound return ∼2800 in CR). That confirms that traditional RL cannot be used in safety-critical environments. We now answer the questions: **(Q1)** *what is the effect of learning with One-shot human-generated trajectories, instead of generating them iteratively via optimisation*; **(Q2)** *how favourable is using preferences over sparse labels*; **(Q3)** *what is the effect of adding a safety justification to preferences*; and **(Q4)** *what happens when multiple justifications are involved?* **(Q1)**-**(Q3)** can be answered in the CR environment, while **(Q4)** in OCR.

For **(Q1)**, we compare ReQueST with DROS to isolate the effect of the One-shot mechanism from that of using preferences. In Figure 2 (up), the return is higher for DROS across all numbers of sparse label queries. This was expected, because as the user explores freely in the dream, more informative queries can be created. To confirm it, we compared DROS and ReQueST segments, using the Determinantal Point Process (Equation 7 from (Dai et al., 2021)). For each set the averaged across the segments diversity was: $D_{DROS} = 25.64$, $D_{ReQueST} = 14.38$, presenting a notable difference. The same showed the t-SNE (Maaten and Hinton, 2008) of the state representations (Figure 2 (down)), where DROS spreads broadly across the entire space, while ReQueST forms tighter clusters. We also measured the computational cost of ReQueST to generate **one trajectory segment of 50 steps** as **3 min and 13.58 sec**. In Figure 2, we see that the ReQueST model reaching the highest return required 2205 sparse labels, which is equivalent to 49 labels in 45 segments. Thus, to achieve that, the user had to stay in the loop for $45 \times (3 \text{ min and } 13.58 \text{ sec}) = \mathbf{145 \text{ min and } 11.1 \text{ sec}}$, i.e. a substantial time. Even on faster machines, complex environments may require more gradient steps for optimisation, compounding the issue. Instead, with the one-shot technique, the user drew only **N=10**

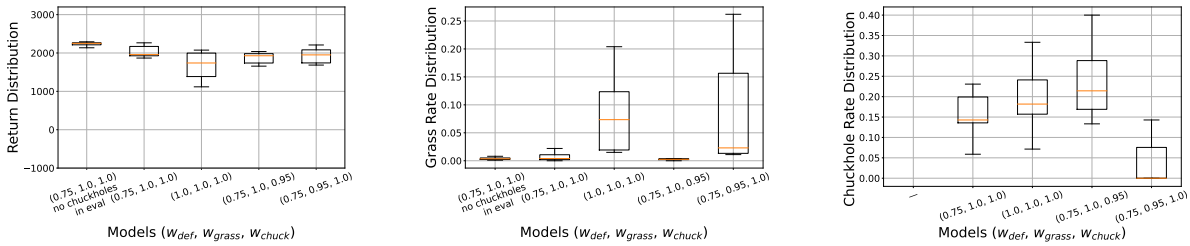



Figure 8: **Q4:** Impact of including multiple justifications in preferences (OCR with only chuckholes).

Table 2: Average feedback time per query, number of queries and total time until the best model for each method.

| Method | Time/Query (s) | Queries | Total time (min) |
|---|---|---|---|
| ReQueST | $1.66 \pm 1.38$ | 3675 | $\simeq 102$ |
| DROS | $1.66 \pm 1.38$ | 2205 | $\simeq 61$ |
| DROP | $7.09 \pm 6.47$ | 500 | $\simeq 59$ |
| DROPe | $7.36 \pm 5.45$ | 500 | $\simeq 62$ |
| DROPJ | $7.88 \pm 5.83$ | 500 | $\simeq 66$ |

**episodes** in the dream, of $\sim$30 sec each, i.e. spending only **5 min** to achieve even better results.

To answer **(Q2)**, we must compare DROS with a method that uses the one-shot technique and preferences (Figure 3). As sparse labels take less time than preference queries, the x-axis should depict actual feedback time, instead of query counts. Table 2 shows the average time per query for each method. Multiplying this rate with the number of queries gives the actual time on the x-axis. We see in Figure 3 (up) that after 20 min all preference methods achieve a considerably better performance than DROS for the same human burden. Equivalently, they achieve the same performance with significantly less feedback time. Thus, we conclude that as in other contexts (e.g. with perfect simulators), here as well (inside a learned simulation), preferences are similarly efficient. Regarding the grass rate (Figure 3 (down)) the main difference is observed in DROPJ, which leads us to (Q3).

Regarding **(Q3)**, we first find that the extra time incurred by justifications compared to plain preferences is less than 1 sec per query (Table 2), indicating that they require minimally more time and cognitive effort. Given that, Figure 4 shows the effect of a safety justification relative to the count of preference queries. We see that although there is a drop-off in the return with the justification, the grass rate for DROPJ after 300 queries is minimised and significantly less than DROP and DROPe. During trials, we observed that the car in DROPJ would carefully slow down on turns and ensure it stayed inside the road, unlike other methods that it would often continue and briefly veer onto the grass. Thus, while DROP and DROPe may suit more for scenarios like a real car race, DROPJ would be preferable for safety-critical settings, such as city centres. This trade-off is sensible, given that in the reward model of DROPJ avoiding unsafe states is encoded as a priority from the justification queries (**Step 3**), even at the expense of other metrics.

Table 2 also shows the required query counts and total time to yield the best model for each method, considering both the return and the grass rate from Figure 3. Figure 5 presents their return and grass rate distributions. We conducted non-parametric significance tests that supported the answers of **(Q1)**-**(Q3)**. For instance, we found significant differences between all methods and ReQueST in the return, as well as between DROPe and DROPJ with the post-hoc pairwise *Dunn's test* (Dunn, 1964) in the return ($p=0.0058$), and conversely in the grass rate ($p=0.0215$), confirming the trade-off mentioned earlier. We also note that although we used the sensible choice of $(w_{def}=0.75, w_s=1)$ (maximum weight to safety and reasonable weight to any other default reason), the heatmap in Figure 6 indicates that safety gains stem strictly from justifications. Good performance and safety requires $w_s \geq w_{def}$, while safety improves with higher $w_s$ giving best results for $w_s=1$. Also $(w_{def}=1, w_s=1)$ is equivalent to DROPe. Finally, Appendix B.2 examines feedback errors (via synthetic erroneous input) and differing opinions (via more labellers). We find that DROPJ remains robust even with small contradictions in *critical*, and larger ones in *non-critical* debatable queries.

In **(Q4)**, Figure 8 shows the metrics distributions of models with justification weights $(w_{def}, w_{grass}, w_{chuck})$ when chuckholes are added. As in **(Q3)**, $(0.75, 1, 1)$ has better safety (grass and chuckhole rates) from $(1, 1, 1)$, and now also return due to $(1, 1, 1)$'s slowdown in chuckholes. The same $(0.75, 1, 1)$ model can still be used efficiently when no chuckholes appear, but with few grass violations (likely due to world model imperfections in kerb). With $(0.75, 1, 1)$ as reference, the grass rate can be minimised by $(0.75, 1, 0.95)$ with an expected trade-off in the chuckhole rate. Conversely, $(0.75, 0.95, 1)$ nearly minimises the chuckhole rate, at the cost of higher grass rate (the car would keep a safe gap from chuckholes, even if that meant stepping on grass).

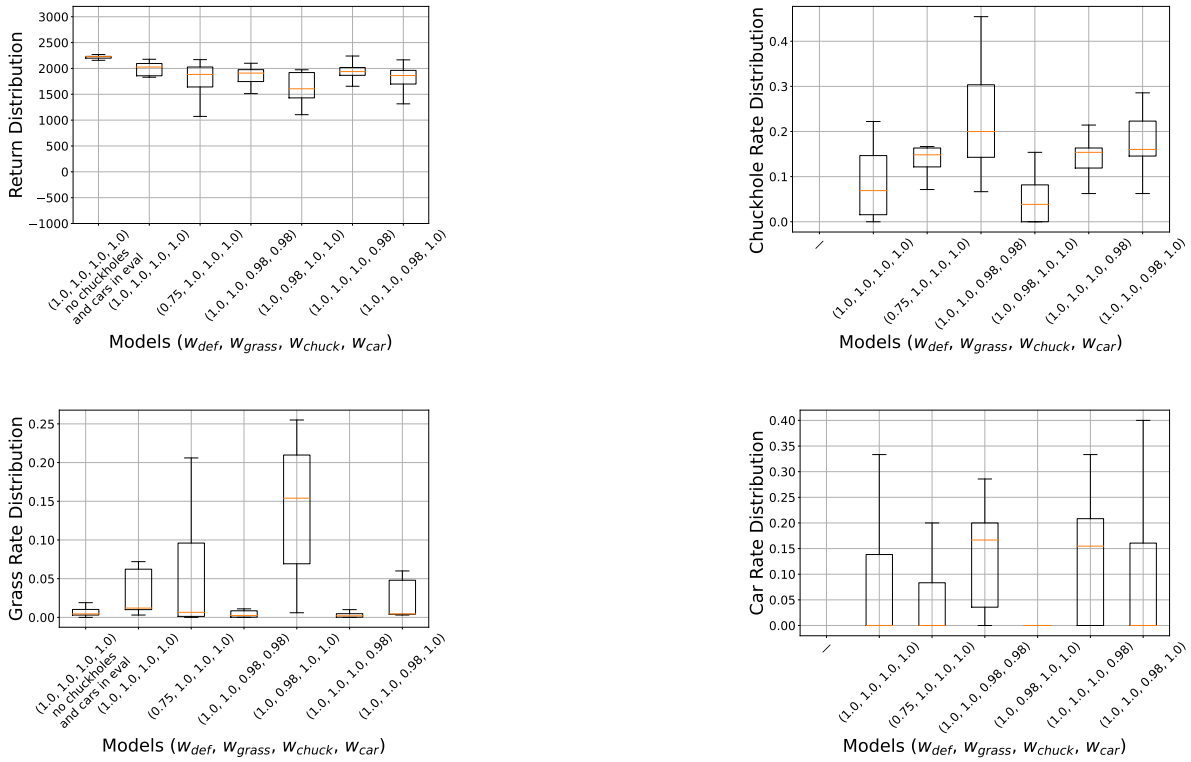



Figure 9: **Q4:** Impact of including multiple justifications in preferences (OCR with both chuckholes and cars).

When cars are also added, Figure 9 shows the metrics distributions with justification weights $(w_{def}, w_{grass}, w_{chuck}, w_{car})$. Both $(1,1,1,1)$ and $(0.75,1,1,1)$ achieve competitive returns (slightly below 2000), maintaining balanced and diverse crash rates — neither combination dominates. For instance, $(0.75,1,1,1)$ yields a lower car rate, but higher grass and chuckhole rates. We keep as a reference $w_{def} = 1$, as the default justification contributes, not only in performance, but also indirectly in safety this time. The reason is that preferences disfavour clips that the car approaches obstacles from a distance, even if not colliding with them within the clip. When compared to the reference $(1,1,1,1)$ model, the $(1,1,0.98,0.98)$ is more cautious about grass steps (grass rate almost zeros) than chuckholes and cars (their rates increase). Conversely, with $(1,0.98,1,1)$ very few chuckholes and no cars are contacted. The agent-car tends to keep a safe distance from chuckholes and cars, even at the cost of driving more often onto the grass. The previous observations show that we can control the original balance of safety metrics as desired by adjusting accordingly the justification weights, giving greater or lesser emphasis to specific safety requirements.

However, justification weights must be set carefully when all safety aspects are critical or one affects another. For instance, with $(1,1,1,0.98)$, as expected, the grass rate (given higher priority) decreases and the car rate rises. Unexpectedly, the chuckhole rate also slightly rises. From inspection, this happens not because the agent-car ignores chuckholes, but due to the *carry-over effect*, where a collision with a scripted-car often pushes it towards a nearby chuckhole. This is clearer in $(1,1,1,0.75)$, where the considerably lower $w_{car}$ causes more collisions with cars that drag the agent-car off its route. The same effect is weaker in $(1,1,0.98,1)$ with the car rate, but stronger in the example of Figure 7 (up) using $(1,1,0.75,1)$, where the agent-car safely passes the blue car but fails to plan for the chuckhole ahead, due to the reduced $w_{chuck}$.

Thus, based on our overall experimentation in OCR, if the safety aspects are independent, justifications provide a straightforward way to prioritise one aspect over another. However, when dependencies exist, they can still emphasise certain aspects, but setting justification weights should consider the carry-over effect. We aspire justification weights to evolve into standardised values for common tasks, while still allowing fine-tuning by human operators to account for environment-specific needs. Alternatively, one can employ evolution strategies such as (Hansen and Ostermeier, 2001) to formulate an optimisation problem with soft constraints, where justification weights are learnable (detailed investigation left for future work).

# 7 CONCLUSION

We presented DROPJ, an efficient human-centred method that maximises safety both during training and deployment. With real-user experiments, we found that generating examples from a user through a learned world model significantly improves the computational time at training, and the performance during deployment. We showed that the use of preference answers on dream queries significantly enhances performance, compared to other feedback types, and that safety justifications accompanying preferences can improve safety or prioritise desired safety aspects associated with them. The only limitation of DROPJ is its reliance on past examples, and qualified users to generate more examples in the simulation. Having both and in abundance, especially for complex tasks like helicopter flying, is not always feasible beyond large industry labs. Future directions involve applying DROPJ in real hardware, such as a lab-based robot navigation task.

# ACKNOWLEDGEMENTS

This work was supported by UKRI [EP/S024298/1] and EPSRC [EP/Y003187/1]. The authors also acknowledge the use of the IRIDIS High-Performance Computing Facility, and associated support services at the University of Southampton.

# REFERENCES

Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained Policy Optimization. In *Proc. ICML*.

Assran, M., Bardes, A., Fan, D., Garrido, Q., Howes, R., Muckley, M., Rizvi, A., Roberts, C., Sinha, K., Zholus, A., et al. (2025). V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv:2506.09985*.

Bishop, C. M. (1994). Mixture Density Networks. Technical report, Aston University.

Biyik, E. and Sadigh, D. (2018). Batch active preference-based learning of reward functions. In *Proc. CoRL*.

Bradley, R. A. and Terry, M. E. (1952). Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Brockman, G. (2016). OpenAI gym. *arXiv:1606.01540*.

Brown, D., Goo, W., Nagarajan, P., and Niekum, S. (2019). Extrapolating beyond suboptimal demonstrations via Inverse Reinforcement Learning from observations. In *Proc. ICML*.

Brown, D. S., Cui, Y., and Niekum, S. (2018). Risk-aware active Inverse Reinforcement Learning. In *Proc. CoRL*.

Chow, Y., Nachum, O., Duenez-Guzman, E., and Ghavamzadeh, M. (2018). A Lyapunov-based approach to Safe Reinforcement Learning. In *Adv. NeurIPS*.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep Reinforcement Learning from human preferences. In *Adv. NeurIPS*.

Dai, J., Pan, X., Sun, R., Ji, J., Xu, X., Liu, M., Wang, Y., and Yang, Y. (2024). Safe RLHF: Safe Reinforcement Learning from human feedback. In *Proc. ICLR*.

Dai, T., Liu, H., Arulkumaran, K., Ren, G., and Bharath, A. A. (2021). Diversity-based trajectory and goal selection with hindsight experience replay. In *Proc. PRICAI*.

Dunn, O. J. (1964). Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252.

Frye, C. and Feige, I. (2019). Parenting: Safe Reinforcement Learning from human input. *arXiv:1902.06766*.

Garcia, C. E., Prett, D. M., and Morari, M. (1989). Model Predictive Control: Theory and practice—a survey. *Automatica*, 25(3):335–348.

Garcıa, J. and Fernández, F. (2015). A comprehensive survey on safe Reinforcement Learning. *JMLR*, 16(1):1437–1480.

Goecks, V. G., Gremillion, G. M., Lawhern, V. J., Valasek, J., and Waytowich, N. R. (2019). Efficiently combining human demonstrations and interventions for safe training of autonomous systems in real-time. In *Proc. AAAI*.

Gu, S., Kshirsagar, A., Du, Y., Chen, G., Peters, J., and Knoll, A. (2023). A human-centered safe robot Reinforcement Learning framework with interactive behaviors. *Frontiers in Neurorobotics*, 17:1280341.

Guo, Z., Norman, T. J., and Gerding, E. H. (2022). MTIRL: Multi-trainer interactive Reinforcement Learning system. In *Proc. PRIMA*.

Ha, D. and Schmidhuber, J. (2018). World models. *arXiv:1803.10122*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels. In *Proc. ICML*.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv:2301.04104*.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Huang, W., Ji, J., Xia, C., Zhang, B., and Yang, Y. (2023). Safedreamer: Safe Reinforcement Learning with world models. *arXiv:2307.07176*.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation Learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35.

Karlaus, J. and Schwenker, F. (2025). TEMPO: Timestep explanations for modeling preferences in online preference-based rl. *IEEE Access*.

Kazantzidis, I., Norman, T. J., Du, Y., and Freeman, C. T. (2022a). How to train your agent: Active learning from human preferences and justifications in safety-critical environments. In *Proc. AAMAS*.

Kazantzidis, I., Norman, T. J., Du, Y., and Freeman, C. T. (2022b). Learning safe behaviour via justified human preferences and hypothetical queries. Preprint. DOI: 10.21203/rs.3.rs-2406802.

Kim, C., Park, J., Shin, J., Lee, H., Abbeel, P., and Lee, K. (2023). Preference transformer: Modeling human preferences using Transformers for RL. In *Proc. ICLR*.

Kingma, D. P. (2013). Auto-encoding variational Bayes. *arXiv:1312.6114*.

Knox, W. B. and Stone, P. (2009). Interactively shaping agents via human reinforcement: The TAMER framework. In *Proc. K-CAP*.

Lee, K., Smith, L., and Abbeel, P. (2021). PEBBLE: Feedback-efficient interactive Reinforcement Learning via relabeling experience and unsupervised pre-training. In *Proc. ICML*.

Leike, J., Krueger, D., Everitt, T., Martic, M., Maini, V., and Legg, S. (2018). Scalable agent alignment via reward modeling: a research direction. *arXiv:1811.07871*.

Li, W., Liu, H., Huang, K., and Hussain, A. (2025). Reinforcement Learning for Human-AI collaboration: Challenges, mechanisms, and methods. *Cognitive Computation*.

Lou, X., Zhang, J., Wang, Z., Huang, K., and Du, Y. (2024). Safe Reinforcement Learning with free-form natural language constraints and pre-trained language models. In *Proc. AAMAS*.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-SNE. *JMLR*, pages 2579–2605.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533.

Moerland, T. M., Broekens, J., Plaat, A., Jonker, C. M., et al. (2023). Model-based Reinforcement Learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for Inverse Reinforcement Learning. In *Proc. ICML*.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *Adv. NeurIPS*.

Park, J., Seo, Y., Shin, J., Lee, H., Abbeel, P., and Lee, K. (2022). SURF: Semi-supervised reward learning with data augmentation for feedback-efficient preference-based Reinforcement Learning. In *Proc. ICLR*.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. (2024). Direct preference optimization: Your language model is secretly a reward model. *Adv. NeurIPS*.

Rahtz, M., Varma, V., Kumar, R., Kenton, Z., Legg, S., and Leike, J. (2022). Safe Deep RL in 3D environments using human feedback. *arXiv:2201.08102*.

Ray, A., Achiam, J., and Amodei, D. (2019). Benchmarking safe exploration in Deep Reinforcement Learning. *arXiv:1910.01708*, 7(1):2.

Reddy, S., Dragan, A., Levine, S., Legg, S., and Leike, J. (2020). Learning human objectives by evaluating hypothetical behavior. In *Proc. ICML*.

Saunders, W., Sastry, G., and Stuhlmueller, A. (2018). Trial without error: Towards safe Reinforcement Learning via human intervention. In *Proc. AAMAS*.

Shi, Z., Fang, M., Chen, L., Du, Y., and Wang, J. (2024). Human-guided moral decision making in text-based games. In *Proc. AAAI*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with Deep Neural Networks and tree search. *Nature*, 529(7587):484.

Turchetta, M., Kolobov, A., Shah, S., Krause, A., and Agarwal, A. (2020). Safe Reinforcement Learning via curriculum induction. *Adv. NeurIPS*.

Warnell, G., Waytowich, N., Lawhern, V., and Stone, P. (2018). Deep Tamer: Interactive agent shaping in high-dimensional state spaces. In *Proc. AAAI*.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. (2021). Mastering visual continuous control: Improved data-augmented Reinforcement Learning. In *Deep RL Workshop NeurIPS 2021*.

# APPENDIX

## A Experimental Details

The user played the game to create $M = 600$ real-world trajectories in CR (800 in OCR with cars), and $N = 10$ dream user trajectories in CR (60 in OCR). Training used early stopping and adaptive epochs and learning rates to avoid overfitting. All methods used the same world model based on (Ha and Schmidhuber, 2018) and (Reddy et al., 2020), reimplemented from TensorFlow to PyTorch with some architectural improvements (Table 3 for VAE and Table 4 for Dynamics Model). For DRQV2 we used the implementation and hyperparameters of (Yarats et al., 2021).

**Reward Model from Sparse Reward Labels** For both ReQueST and DROS, the user annotated segments of length $k = 50$ with sparse reward labels (Figure 10). The same classifier-based reward model of (Reddy et al., 2020) was reimplemented (Table 5), but with real (instead of synthetic) feedback for reliable results. In Figure 10 the action is $[steer, gas, brake]$, where $steer \in [-1, 1]$ and $gas, brake \in [0, 1]$. In the example, the car's last action ($steer = 1.0$) turns it all right on the grass, yielding a *Grass/Kerbs* (Unsafe) label; instead, *New Tile* (Good) or *Road* (Neu-

Table 3: VAE model architecture and hyperparameters.

| Architecture | CR | OCR |
|---|---|---|
| Input size | $84 \times 84 \times 3$ | $128 \times 128 \times 3$ |
| | (filters, kernel, stride, padding) | |
| Encoder CNN Layers | (24, 4×4, 2, 0)<br>(48, 4×4, 2, 0)<br>(96, 4×4, 1, 0)<br>(192, 4×4, 1, 0)<br>(384, 4×4, 1, 0) | (64, 4×4, 2, 1, 0)<br>(128, 4×4, 2, 1)<br>(256, 4×4, 2, 1)<br>(512, 4×4, 2, 1)<br>(1024, 4×4, 2, 1) |
| Linear Layers for $\mu$ and $\sigma$ | 1536 to $d = 32$<br>$d = 32$ to 1536 | 16384 to $d = 64$<br>$d = 64$ to 16384 |
| | (filters, kernel, stride, padding) | |
| Decoder Transp. CNN Layers | (192, 4×4, 1, 0)<br>(96, 4×4, 1, 0)<br>(48, 5×5, 2, 0)<br>(24, 5×5, 2, 0)<br>(3, 4×4, 2, 0) | (512, 4×4, 2, 1)<br>(128, 4×4, 2, 1)<br>(64, 4×4, 2, 1)<br>(32, 4×4, 2, 1)<br>(3, 4×4, 2, 1) |
| KL Div Thresh | 0.5 | 2 |
| Optimiser | Adam | Adam |
| Mini-batch size | 32 | 32 |
| Learning Rate | $10^{-4} \rightarrow 10^{-5}$ (dyn) | $10^{-4}$ |

Table 4: Dynamics model (MDN-RNN) hyperparameters.

| Hyperparameters | Values |
|---|---|
| Number of LSTM layers | 1 |
| Number of LSTM units $m$ | 256 (1024 for OCR) |
| Gaussian Mixtures | 5 (7 for OCR) |
| Temperature $\tau$ | 1.0 |
| Optimiser | Adam |
| Mini-batch size | 32 |
| Maximum Gradient Value | 1.0 |
| Learning rate | $10^{-3} \rightarrow 10^{-4}$ (dyn) |

Table 5: Reward model from sparse reward labels.

| Hyperparameters | Values |
|---|---|
| Reward constants | $R_{good} = 10$, $R_{unsafe} = -1$, $R_{neutral} = 0$ |
| Number of FNN layers | 5 |
| Number of FNN units | 256 |
| Number of Ensemble | 4 |
| Mini-batch size | 32 |
| Optimiser | Adam |
| Learning rate | $10^{-3}$ |


Figure 10: GUI of the sparse reward labels user feedback.

Table 6: Reward model from preferences and justifications.

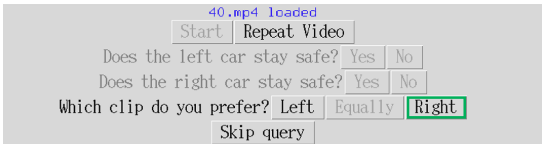
| Hyperparameters | Values |
|---|---|
| Justification weights | $w_s = 1$, $w_{def} = 0.75$ and Figure 6 (CR)<br>$w_{car}$, $w_{chuck}$, $w_{grass}$, $w_{def}$: Figures 8, 9 (OCR) |
| Number of FNN layers | 3 |
| Number of FNN units | 256 |
| Number of Ensemble | 3 |
| Mini-batch size | 128 |
| Optimiser | Adam |
| Learning rate | $3 \times 10^{-4}$ |

tral) were chosen when the car slightly advanced on road or stayed on visited tiles. The user can also skip unclear queries, e.g. when clips show hallucinations (artefacts) due to the dynamics model imperfections.

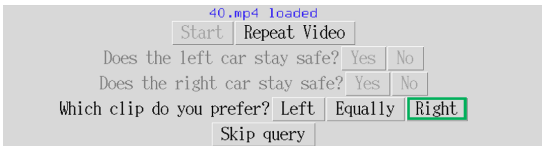
**Reward Model from Preferences and Justifications** From the dream user trajectories in $\mathcal{D}$, we sampled uniformly at random pairs of trajectory segments of length $k = 20$ and trained a preference reward model adapting (Lee et al., 2021). The addition was the justifications with weights $w_s$, $w_{def}$ in CR and $w_{car}$, $w_{chuck}$, $w_{grass}$, $w_{def}$ in OCR, shaping differently the preference label $\mu$. The number of queries was $K = 500$ in CR, $K = 1000$ in OCR with chuckholes, and $K = 1500$ when added cars (Table 6). Other hyperparameters are in Table 6. Figures 11 and 12 show the GUIs of all preference-based methods, and examples with their corresponding answer and resultant preference label $\mu$. During annotation, *Equally* was selected when both clips showed qualitatively similar safe behaviour (e.g. covering roughly the same distance) or equally unsafe behaviour (e.g. going to or remaining on the grass). *Skip query* was used for hallucinations or ambiguous transitions (e.g. one clip going from grass to road and the other from road to grass).

**Model Predictive Control** We used the same sample-based MPC approach as in (Reddy et al., 2020; Rahtz et al., 2022), described in Step 4. We generated $S = 15$ sensible random action sequences, including predefined ones such as drive straight, turn left/right while moving, brake, and two-stage random combinations, all within the planning horizon. For instance, for a horizon of 15, a sequence may be: drive forward while turning left for 10 steps and then go straight for 5. After short experimentation, ReQueST and DROS operated best at $H = 25$, and DROP, DROPe, and DROPJ at $H = 15$. All methods re-planned every $R = 4$. Figure 13 shows the end of a set of MPC dreams. Each label lists the action $[steer, gas, brake]$, followed by the number of steps that action was repeated. The red-highlighted sequence has the highest predicted return, so its first $R = 4$ actions were executed before re-planning.

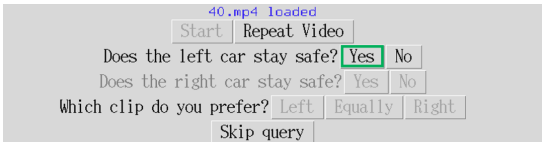
(a) The end of the two compared clips is shown. On the left clip the car has barely moved, and on the right the car has safely taken a nice turn.



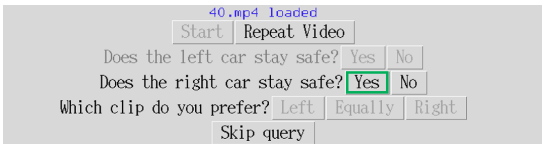

(b) DROP (*Equally* deactivated). Ans: *Right* $\rightarrow \mu = 0$.



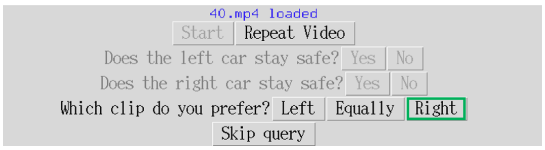

(c) DROPe (*Equally* included). Ans: *Right* $\rightarrow \mu = 0$.




(d) DROPJ — first safety query (left clip). Ans: *Yes*.




(e) DROPJ — second safety query (right clip). Ans: *Yes*.



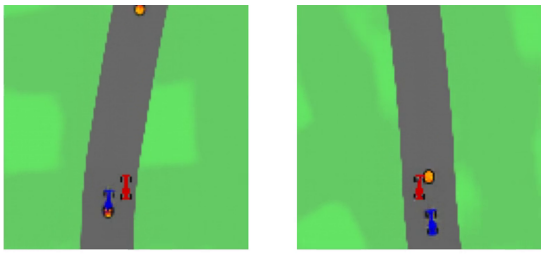

(f) DROPJ — third query for preference (since both *Yes*). Ans: *Right* $\rightarrow \mu = 0.25$.

Figure 11: GUIs in CR of DROP (b), DROPe (c) and DROPJ (d-f) with a *single safety justification* on the same preference query (a), used for the preference reward model.

# B Additional results

## B.1 World model ablation

Figure 14 shows for CR the validation losses of the VAE and MDN-RNN trained with different numbers $M$ of real-world trajectories. After $M = 600$, both



(a) The end of the two compared clips is shown. On the left clip the agent-car (red) passed safely the blue car, and on the right passed the car but stepped on the chuckhole. Correct: (*Chuckhole*, *Left*) $\rightarrow \mu = w_{chuck}$.

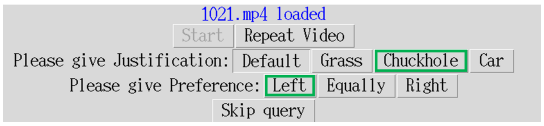



Figure 12: GUI in OCR of DROPJ with *multiple safety justifications*. *Default* is always preselected for brevity (to save a click).

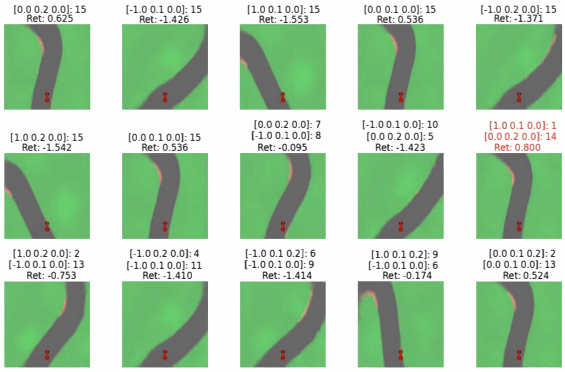


Figure 13: Final frames of MPC dreams.

models show negligible improvement, so, based on this and inspection of sequence reconstructions and predictions, we did not collect more data. We also tested $M = 25$ and $M = 10$, repeating **Steps 2–4** for DROPe and DROPJ. Figure 15 shows that with $M = 25$ although the return and grass rate degrade, they remain surprisingly robust. The world model, though poor at turns, still captures the main human objective of staying on the road. Importantly, the safety benefits of justifications (DROPJ) persist even here. For $M = 10$, performance collapses, as the world model can no longer plan effectively.

## B.2 Robustness to diverse feedback

We examine the effect of human error and differing opinions in feedback, focusing on justifications. For the best-trained DROP, DROPe, and DROPJ models in CR, we randomly corrupted 60% of *non-critical queries* (i.e when the car stays on road in both clips). As shown in Figure 16a, DROPJ is more robust than DROP and DROPe, with smaller performance drop

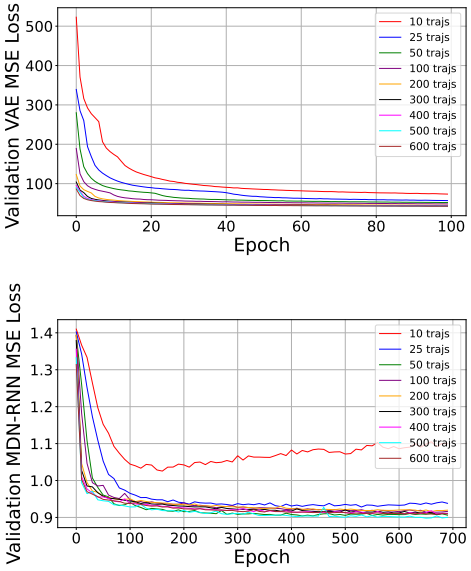



Figure 14: VAE and MDN-RNN Validation Mean Squared Error (MSE) losses.

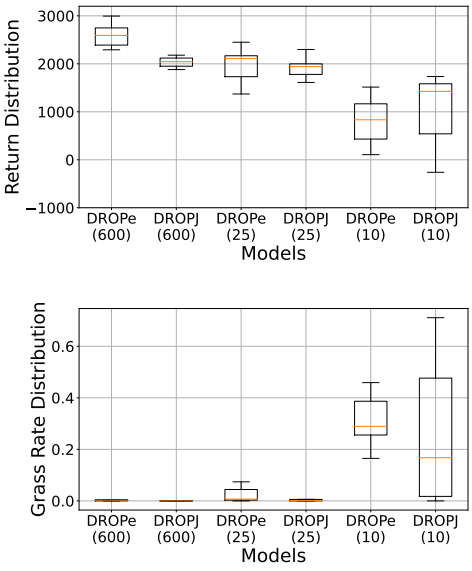
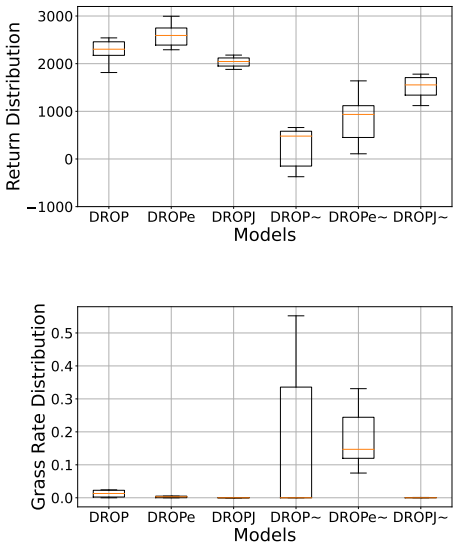


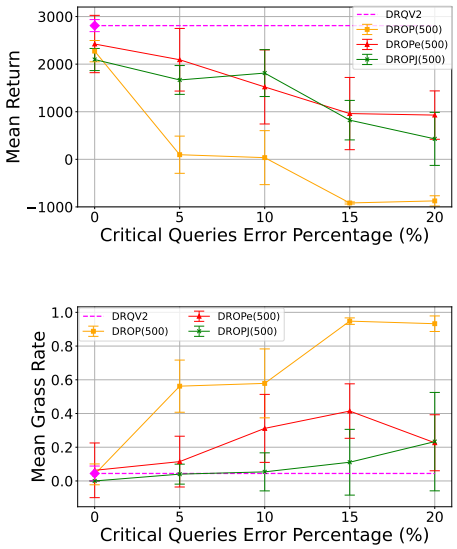

Figure 15: World model ablation.

and near-minimal grass rate. This is because non-critical errors in DROPJ affect the label $\mu$ only within $[1-w_{def}, w_{def}]$ (e.g. $[0.25, 0.75]$), unlike the full $[0,1]$ range in DROP and DROPe. However, for *critical queries* (i.e. at least one clip unsafe) mislabelled, all methods eventually failed, with DROPJ showing some resistance up to 10% (Figure 16b).

Also, another labeller with same experience annotated all DROPJ queries and subsets of DROP and DROPe to assess consistency. The average feedback times were: $t_{DROP} = 6.39 \pm 5.49$, $t_{DROPe} = 7.48 \pm 7.24$ and $t_{DROPJ} = 8.09 \pm 5.15$, all very close and consistent




(a) Effect when 60% of answers in non-critical queries is erroneous.




(b) Effect when a percentage of answers in critical queries is erroneous.

Figure 16: Effect on best-trained models via synthetic erroneous feedback.

to the original labeller. Regarding DROPJ answers, most disagreements were on hallucinated or ambiguous clips (e.g., narrow vs. wide turns, or borderline road–grass driving). Ignoring skipped answers from at least one labeller, the critical disagreement rate was only 2% and the non-critical 23%. DROPJ from the new labeller achieved $2086.0 \pm 255.2$ return (no statistical difference with original) and also zero grass violations. That provides more indications that for small contradiction in critical, and even bigger in non-critical queries, DROPJ remains robust.