A Algorithm

Algorithm. We now present our main algorithm for Assisted Reward Design. At every iteration i, we use the Maximal Information acquisition functions to select the next environment M_{i+1} and query the reward designer. Note that the environment space $\mathcal{M}_{\text{devel}}$ is continuous and high dimensional and it is intractable to exhaustively search through it. We thus use *uniform sampling* to select a candidate set $\mathcal{M}_{\text{cand}} \in \mathcal{M}_{\text{devel}}$. Note that one can employ different heuristics for selecting such candidate sets, and we leave the heuristic design to future work.

Representing Belief Distribution. We use particles to represent $\tilde{P}_i(w = w^*)$: at each step, we sample N_p particles from $\tilde{P}_i(w = w^*)$, compute importance weights based on Eq. 1 for each particle, and resample using the importance weights.

Algorithm 1 Assisted Reward Design via Info-Gathering

Require prior $P_0(w)$, $\mathcal{M}_{\text{devel}}$, N_{cand}, N_p, initial training environments $\mathcal{M}_0$
Initialize posterior $\tilde{P}_0(w = w^*) = P_0(w)$
for $i = 0, .., T$ **do**
  $\tilde{w}_i \sim P_{\text{user}}(w|w^*, \mathcal{M}_i)$ { Query the designer on $\mathcal{M}_i$ }
  Compute posterior $\tilde{P}_{i+1}(w = w^*)$ using Eq. 4 or Eq. 5
  Sample N_{cand} candidate environments $\mathcal{M}_{\text{cand}} \subseteq \mathcal{M}_{\text{devel}}$
  for $M \in \mathcal{M}_{\text{cand}}$ **do**
    Compute $f(M)$
  end
  Select $M_{i+1} = \arg\max_{M \in \mathcal{M}_{\text{cand}}} f(M)$
  $\mathcal{M}_{i+1} = \mathcal{M}_i \cup \{M_{i+1}\}$
end

Complexity. Our algorithm is bounded by the number of particles N_p for representing posterior distribution and the number of candidates N_{cand}. At every iteration, the algorithm needs to solve for each particle w in each candidate environment, which leads to a total of $O(N_p \cdot N_{\text{cand}})$ planning problems in order to compute belief update. This is the main speed limit. While one can potentially speed up by learning fast planners [39], we leave this to future work. In our experiments, we implement the environment as in Sec. B such that the dynamics and reward function are both vectorizable. We then concatenate the reward functions of $O(N_p \cdot N_{\text{cand}})$ problems and compute batch forward planning using gradient-based planner. Note that this is not feasible for general planning problems with non-differentiable dynamics or reward functions.

Adding New Features. When the designer adds in new feature ϕ_{k+1} on environment M_n, the new proxy reward has $d+1$ dimensions while the previous proxies have d dimensions. We can still perform reward inference on all of the $d+1$ dimensions, as long as we incorporate all proxies received so far. The meta-agent would then have to revisit all the reward designs it's gotten so far and recompute posterior over the new augmented space. To do so, we invert Eq. 1 using the formula from [1]:

$$P(w = w^*|\tilde{w}_{1:n}, \tilde{M}_{1:n}) \propto P(w) \prod_{i=1:n} \frac{\exp\left(\beta w^T \tilde{\phi}_i\right)}{\tilde{Z}}, \tilde{Z}(w) = \int_{\hat{w}} \exp\left(\beta w^T \tilde{\phi}_i\right) d\hat{w}$$

with $P(w)$ being the prior. We use MCMC to infer the distribution. During inference, every sample w is of $d+1$ dimensions, and $\tilde{\phi}_i$ are of $d+1$ dimensions, computed from all previous proxy rewards (of i or $i+1$ dimensions) on the existing tasks. To compute the normalizer, we integrate over $\hat{w}$ in the $d+1$ dimensional space.

B Driving Environment Implementation

This section provides the details of the driving environment used in the paper. We introduce definitions of environment dynamics, feature and reward functions, the environment distributions, and how we implement the environment efficiently for trajectory optimization.

Environment Features			
Feature Name	Raw Feature ϕ_{raw}	Transformation ϕ_{full}	Meaning
Speed	v	$-(\phi_{\text{raw}} - v_{\text{goal}})^2$	How much the vehicle deviates from the goal speed.
Control	$[u_{\text{steer}}, u_{\text{acc}}]$	$-\|\|\phi_{\text{raw}}\|\|^2$	The control effort by the vehicle.
Lane	x	$\mathcal{N}(\phi_{\text{raw}} - \vec{x}_{\text{goal}}, d_{\text{lane}})$	How much the vehicle deviates from target lane center.
Car	$[x, y]$	$-\sum_i \mathcal{N}(\phi_{\text{raw}} - \vec{x}_{\text{car i}}, d_{\text{car i}})$	How much the vehicle is driving close to other vehicles.
Obstacle	$[x, y]$	$-\sum_i \mathcal{N}(\phi_{\text{raw}} - \vec{x}_{\text{obs i}}, d_{\text{obs i}})$	How much the vehicle is driving close to the obstacles.
Fence	x	$-[x_{\text{fence left}} - \phi_{\text{raw}}]_+$ $-[\phi_{\text{raw}} - x_{\text{fence right}}]_+$	How much the vechile is outside the fence

Table 1: Environment Feature Table

Dynamics We represent the forward dynamics of the MDP as $x_{t+1} = f(x_t, u_t)$, where we have the full knowledge of x. To model system dynamics, we use simple point-mass vehicle model with holonomic constraint. Let $x_{\text{car}} = [x, y, \theta, v]^\top$, where x, y are the corrdinates of the vehicle, θ is the heading, and v is the speed. We let $u = [u_{\text{steer}}, u_{\text{acc}}]^\top$ be the control input, where u_{steer} is the steering input and u_{acc} is the acceleration. We also use α as the friction coefficient. The model for a single vehicle is:

$$[\dot{x} \quad \dot{y} \quad \dot{\theta} \quad \dot{v}] = [v \cdot \cos(\theta) \quad v \cdot \sin(\theta) \quad u_{\text{steer}} \quad u_{\text{acc}} - \alpha \cdot v] \tag{6}$$

We model human vehicles as moving forward at constant speed and traffic cones as static objects.

Feature and Reward Functions. In the following Table 1 we introduce the environment features ϕ in the driving environment. Each feature composes of ϕ_{raw}, a subset of the current state x_t and control u_t, and a non-linear transformation on ϕ_{raw}. The nonlinear transformation is designed such that when maximized, it induces the desired behaviour in that feature, such as moving to the target lane. These environment features are are differentiable and that we can characterize the desired driving behavior via a linear weighed sum $w^\top \phi$.

Violations. Here we provide in Table 2 the definition of environment constraint functions used in the experiment section to evaluate driving quality. Each constraint is a boolean function that returns true or false for one timestep, and we compute the final violation count for each trajectory by summing over constraints over the time horizon. These constraint functions are not used for optimization, but as an evaluation criterion for the case study experiment. Lower violation counts correspond to better driving behavior.

Environment Constraints		
Feature Name	Definition	Meaning
Overspeed	$v > v_{\max}$	The vehicle is driving over the maximal allowed speed.
Underspeed	$v < v_{\min}$	The vehicle is driving below the minimum speed on highway (i.e. backing up).
Uncomfortable	$\|\|u\|\|_\infty > \|\|u_{\max}\|\|_\infty$	The vehicle is applying too much force that it's uncomfortable (i.e. accerlating too much or jerky).
Collision	$\|\|\vec{x} - \vec{x}_{\text{car i}}\|\| \leq d_{\min}$	The vehicle crashes into the other vehicles.
Crash Object	$\|\|\vec{x} - \vec{x}_{\text{obj i}}\|\| \leq d_{\min}$	The vehicle crashes into the obstacles.
Offtrack	$x < x_{\text{fence left}}$ or $x > x_{\text{fence right}}$	The vehicle drives off the left or the right fence.
Wronglane	$\|\|x - x_{\text{lane left}}\|\| \leq d_{\text{lane}}$	The vehicle drives drives onto the wrong lane while merging.

Table 2: Environment Constraints Table

Planning Speed. We implement the dynamics and reward function using JAX [40] and leverage the JIT compilation to speed up running time. We use shooting method and perform gradient-based planning using the Adam optimizer for 200 steps. We plan at a horizon of 10 timesteps and replan

every 5 timesteps. When planning for a single environment, we can generate 2.57 trajectories per second. Because we can vectorize the planning problem and plan for multiple trajectories at once, we can achieve 157.72 trajectories per second, by planning for 500 trajectories in batch. Computations are benchmarked on the c2-standard-4 instance on Google Cloud. We use batch planning for computing the belief distribution and environment proposals, as discussed in Sec. A as the main bottleneck of our algorithm.

Environment Distribution. We use a simple method to define the distribution of environments $\mathcal{M}_{\text{design}}, \mathcal{M}_{\text{deploy}}$. There are two human vehicles and two traffic cones positioned on the three-lane highway. We assume that the autonomous vehicle starts from the center of the scene ($x = 0, y = 0$), and sample the starting position of the other vehicles and obstacles. The other vehicles can start anywhere in $x_{\text{min car}} \leq x_{\text{car}} \leq x_{\text{max car}}, y_{\text{min car}} \leq y \leq y_{\text{max car}}$ and the obstacles can be initialized anywhere in can start anywhere in $x_{\text{min obs}} \leq x_{\text{obs}} \leq x_{\text{max obs}}, y_{\text{min obs}} \leq y \leq y_{\text{max obs}}$. We filter out situations where the other vehicles or obstacles are initialized to be colliding with the main vehicle.

Note that this is a very coarsely designed distribution to showcase our method. We have several limitations. For instance, we do not exclude the environments where the other vehicles runs into obstacles. These are relatively unlikely environments, and in principle we can define more realistic distributions by more careful environment engineering or by loading real world driving datasets.

The difference between $\mathcal{M}_{\text{devel}}$ and $\mathcal{M}_{\text{deploy}}$ is that in $\mathcal{M}_{\text{deploy}}$, we define a tighter feasible range of x_{car} centered around the autonomous vehicle. This results in a shifted distribution of pseudo-difficulty metric with long-tail events. After discretization, $\mathcal{M}_{\text{devel}}$ has 274k environments and $\mathcal{M}_{\text{test}}$ has 91k environments. Given these large number of possible environments, it is impossible for reward designers to enumerate them manually.

C Experiment Details

Optimal Planning For the autonomous driving task, we compute trajectories of 10 timesteps. We use finite-horizon optimal planning with regard to given rewards. We plan at a horizon of 10 timesteps, and replan every 5 timesteps.

Reward Design We use rationality factor $\beta = \frac{0.1}{||\mathcal{M}||}$ to simulate noisy designer, and $\beta = 1$ in the inverse model. This is because we find empirically that the proxy reward quickly approaches the ground truth reward when we have multiple proxy environments. We thus divide the rationality factor β it by the number of proxy environments to maintain its noisiness. To compute the posterior probability in Eq. 4 or Eq. 5, we need to approximate the normalizing constant. We follow the approach in [1] by sampling w. In the section 4.2 of [1], they find empirically that it helped to include the candidate sample w in the normalizing sum. This requires planning with w and computing its feature sum in the MCMC inner loop, which largely slows down the inference. Instead, we include the candidate w, but multiplies it with proxy $\tilde{w}$'s feature sum in the normalizing constant.

We then compare three methods (random, difficulty, maximal information) in iterative fashions. We use the same initial environment for the three methods and perform 9 iterations. We aggregate results over five random seeds.

Environment Proposal. During environment proposal, we use $N_p = 500$ particles on $N_{\text{cand}} = 64$ environments. We implement a vectorized car dynamic model using JAX [40] and Ray [41] for batch planning. Our environment proposal step takes 4 minutes on c2-standard-4 instance on Google Cloud.

Evaluation. To examine the quality of our inferred posterior of the reward function, we need to evaluate the posterior in new driving environments. We thus sample 500 environments uniformly from $\mathcal{M}_{\text{deploy}}$ as the deployment set for evaluation. We compute the regret, note that because of the different placement of vehicles and objects, different environment have different maximum and minimum reward they can produce. Thus it is unfair to compute the absolute regret $r_{\max} - r_w$. We thus compute the relative regret, by first taking the worst cases reward $r_{\min}$ in each environment. Then the relative regret is $\text{regret}_w = (r_{\max} - r_w)/(r_{\max} - r_{\min}) = (r_{w^*} - r_w)/(r_{w^*} - r_{\min})$.