# Experience Filter: Using Past Experiences on Unseen Tasks or Environments

Anil Yildiz[1], Esen Yel[1], Anthony L. Corso[1], Kyle H. Wray[1, 2], Stefan J. Witwicki[2], and Mykel J. Kochenderfer[1]

***Abstract*— One of the bottlenecks of training autonomous vehicle (AV) agents is the variability of training environments. Since learning optimal policies for unseen environments is often very costly and requires substantial data collection, it becomes computationally intractable to train the agent on every possible environment or task the AV may encounter.**

**This paper introduces a zero-shot filtering approach to interpolate learned policies of past experiences to generalize to unseen ones. We use an experience kernel to correlate environments. These correlations are then exploited to produce policies for new tasks or environments from learned policies. We demonstrate our methods on an autonomous vehicle driving through T-intersections with different characteristics, where its behavior is modeled as a partially observable Markov decision process (POMDP). We first construct compact representations of learned policies for POMDPs with unknown transition functions given a dataset of sequential actions and observations. Then, we filter parameterized policies of previously visited environments to generate policies to new, unseen environments. We demonstrate our approaches on both an actual AV and a high-fidelity simulator. Results indicate that our experience filter offers a fast, low-effort, and near-optimal solution to create policies for tasks or environments never seen before. Furthermore, the generated new policies outperform the policy learned using the entire data collected from past environments, suggesting that the correlation among different environments can be exploited and irrelevant ones can be filtered out.**

## I. Introduction

Designing efficient and safe planning strategies for autonomous vehicles is generally challenging due to uncertainty. One of the principled ways of modeling the planning problem under uncertainty is using POMDPs [1], which have been successfully applied in autonomous driving [2], [3]. The POMDP formulation includes a state transition model which represents the environment dynamics. When this model is not readily available for a given problem, machine learning techniques can be used to learn the environment models from data collected at design time. Although reinforcement learning algorithms have been shown to produce effective policies for environments that they are trained upon [4], they typically need to be retrained to be deployed in new environments. To overcome this limitation, models or policies previously learned need to be transferred to new, unforeseen tasks.

In this work, we introduce the concept of an experience filter, illustrated in Fig. 1, to reason about new tasks or environments using past experiences. The goal of the experience filter is to filter optimal/learned policies of previously visited environments to generate policies to ones never seen before. By doing so, we eliminate the need to train an autonomous vehicle's policy on a vast range of different tasks, but rather, interpolate existing policies to unseen similar tasks, eliminating the need to collect any new data. Our experience filter (EF) approach allows fast, low-effort, and near-optimal solutions to create policies without requiring to train for tasks or environments not seen before. Furthermore, our results also indicate that using the EF approach to correlate and filter previous experiences through a kernel, as depicted in Fig. 1, yields higher performance than simply using the entire data collected in an attempt to train an all-for-one policy.

We assume that both policies and environments can be represented by low-dimensional, easily accessible parameter

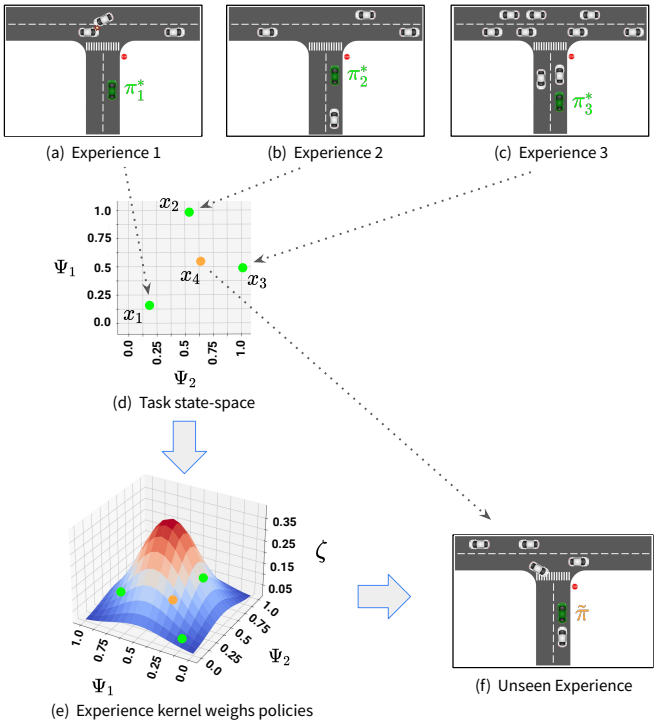



Fig. 1. (a)-(d) Different tasks or environments $x_i \in \mathcal{X}$ (and their policies $\pi_i^*$) can be compactly represented through parameterization. (e) Here, we define the relationship between a new environment (parameterized by $x_4$) and previously visited environments (parameterized by $x_1, x_2, x_3$) using a kernel $\zeta(x_m, x_4)$, $m = \{1, 2, 3\}$. (f) An *experience filter* exploits this relation to generate a policy for $x_4$ given policies trained on different tasks or environments.

[1] Stanford Intelligent Systems Laboratory, 496 Lomita Mall, Stanford, CA, USA. Emails: {yildiz, esenyel, acorso, kylewray, mykel}@stanford.edu

[2] Alliance Innovation Laboratory Silicon Valley, Santa Clara, CA, USA. Email: stefan.witwicki@nissan-usa.com

vectors. For example, the behavior of an AV navigating along a highway can be influenced by factors such as traffic density, number of lanes, upcoming intersections, etc. Our proposed experience filter takes environment representations and parameterized policies as an input, and using an experience kernel, outputs a new policy for an unseen environment through Bayesian reasoning between the learned and unseen environment parameters.

The contributions of this paper are twofold: (1) we introduce a new technique to learn and parameterize policies of a POMDP with an unknown transition function from existing data, and (2) we propose an experience filter to efficiently plan in unseen environments by interpolating past knowledge. We demonstrate our approach on a problem of autonomously driving through T-intersections, and we include demonstrations on both an actual AV and a high-fidelity simulator CARLA [5]. For simulation results, we benchmark our experience filter approach using the following performance metrics: collision risk, discomfort during driving, and task completion time.

The paper is organized as follows: Section II gives an overview of transfer learning techniques. Section III gives a brief background on Dirichlet distributions, POMDPs, and MODIA and Section IV defines the problem. Section V presents our learning approach and experience filter. Section VI summarizes results. Finally, Section VII draws conclusions and discusses future work.

## II. Related Work

The problem of how to reduce training effort for new tasks by leveraging the knowledge from existing tasks has been addressed by transfer learning techniques. Existing work on transfer learning techniques largely focuses on transferring knowledge from previously seen *source* tasks to new tasks [6] and can be divided into the following three categories.

*a) Model-free transfer:* Transfer learning for model-free reinforcement learning algorithms involves transferring knowledge between tasks in the form of experience samples, policies, or value functions. Progressive networks [7] is an approach to transfer learning where additional neurons are added to a network for each new task that is learned. Previously learned network weights are frozen, and the new neurons are connected so as not to change the output of the network on previous tasks. The attend, adapt, and transfer approach [8] uses a set of attention weights to combine the output of a set of source policies to achieve good performance on a new task. To avoid negative transfer, a new policy is learned from scratch and combined with the source policies through an additional weight. The attention weights can be learned from a small amount of data, allowing for fast adaptation to new environments. Transfer learning with model-free deep reinforcement learning is also applied to task transfer for autonomous driving decision making problems [9].

*b) Model-based transfer:* Prior work on transfer in a model-based setting assumes there are unobserved parameters that describe each task and seek to learn a model of these parameters from data [10], [11]. Recent work [12] assumes a low-dimensional latent task identifier which is inferred online using a Bayesian neural network, allowing for fast adaptation to new problems. Hidden parameter MDPs were extended to allow for variations in the reward function [13]. Storing experience samples from previous source tasks and constructing a model for the new task with an inter-task mapping has also been done [14]. Our approach is similar to the aforementioned work in that transfer to a new task is done by combining previously learned models. Both model-based and model-free approaches, however, require training on the new task, whereas we consider the case of zero-shot transfer in this paper.

*c) Zero-shot transfer:* Zero-shot transfer [15] is often preferred when there is a known relationship between tasks. For example, when the source task is a simulated version of the real target task, unsupervised pre-training has shown to be effective for enabling zero-shot transfer [16]. Alternatively, automated measures of MDP similarity can be used [17]. Zero-shot policy transfer along with the robust tracking controllers to tackle the source to target modeling gap is applied in robotics [18] and autonomous driving problems [19]. In our setting, we exploit a parametric connection between previously visited environments, using this relation to hypothesize policies for unseen environments.

## III. Background

### A. Dirichlet Distributions

Dirichlet distribution $\mathrm{Dir}\left(\boldsymbol{\alpha}_{1:n}\right)$ is parameterized by $\boldsymbol{\alpha}_{1:n} \in \mathbb{R}^n_{\geq 0}$ which can be treated as pseudo-counts of different outcomes. The density of a Dirichlet distribution is given by

$$P\left(\boldsymbol{\theta}_{1:n} \mid \boldsymbol{\alpha}_{1:n}\right) = \frac{\Gamma\left(\sum_{i=1}^{n} \alpha_i\right)}{\prod_{i=1}^{n} \Gamma\left(\alpha_i\right)} \prod_{i=1}^{n} \theta_i^{\alpha_i - 1}$$

where $\Gamma$ is the gamma function [20].

If the prior is a Dirichlet distribution and the event $i$ is observed $m_i \in \boldsymbol{m}_{1:n}$ times, then the posterior is also a Dirichlet distribution [4]:

$$\boldsymbol{\theta}_{1:n} \mid \boldsymbol{\alpha}_{1:n}, \boldsymbol{m}_{1:n} \sim \mathrm{Dir}\left(\boldsymbol{\alpha}_{1:n} + \boldsymbol{m}_{1:n}\right).$$

### B. Partially Observable Markov Decision Processes

A sequential decision making problem can be modeled as a partially observable Markov decision process. A POMDP model is represented by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma \rangle$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, and $\mathcal{O}$ is a finite state of observations. The system takes an action $a \in \mathcal{A}$ from state $s \in \mathcal{S}$ and transitions to the next state $s' \in \mathcal{S}$ according to the probabilistic transition function $T(s' \mid s, a) = P(s' \mid s, a)$ which models the environment dynamics. From state $s'$ the system obtains observation $o \in \mathcal{O}$ according to the observation function $Z(o \mid s', a) = P(o \mid s', a)$ and receives a reward according to the reward function $R(s, a)$. As the system does not have access to the true world states, it maintains a belief $b(s)$ over possible states. A discount factor $\gamma \in [0, 1]$ may also be used to prioritize earning rewards sooner than later.

The goal of the system is to find a policy $\pi^*$ to maximize the expected total discounted reward starting from its belief $b$, which can be exactly calculated using the relation

$$\pi^*(b) = \arg\max_a \sum_s b(s)Q(s,a) \tag{1}$$

where

$$V^*(b) = \max_a \sum_s b(s)Q(s,a) \tag{2}$$

$$\begin{aligned} Q(s,a) &= R(s,a) \\ &+ \gamma \sum_{s'} T(s' \mid s,a) \sum_o Z(o \mid s',a) V^*(b'_{(b,a,o)}) \end{aligned} \tag{3}$$

and $b'_{(b,a,o)}$ is the updated belief.

For fully observable states (MDP), the belief and observation terms in Eqs. (1) to (3) drop out, and the optimal solution can be computed tractably using Bellman updates [21]. However, finding a solution to a POMDP using dynamic programming is computationally very expensive, and thus, approximate solutions are often used instead [4]. In this work, we use the QMDP [22] offline solver, and assume the resulting policy to be near-optimal.

### C. MODIA

The multiple online decision-components with interacting actions (MODIA) framework [23] aims to solve complicated real world decision making problems in a scalable way by separating them into subproblems. Instead of constructing a single POMDP that accounts for all of the other agents in a domain, MODIA instantiates multiple smaller POMDPs for each agent interaction. This formulation inherently assumes that most agents act independently from one another. At each timestep, the safest action among all POMDPs is selected for execution. MODIA is utilized during the experimentation.

## IV. Problem Definitions

In this study, our goal is to efficiently generate new policies for unseen driving tasks or environments using past experiences. To achieve this, we need to represent policies by a compact representation, and then find a way to transfer the policies. These two problems are defined as follows:

***Problem 1: Parameterizing Policies:*** Given a driving dataset $\mathcal{D}_i$ containing observation triplets $(o_i^t, a_i^t, o_i^{t+1})$ obtained from $N$ different driving environments visited at times $t \in [0, \tau_i]$, and a policy $\pi_i^*$ learned for environment $i$, what is the compact representation $par(\pi_i^*)$ of this policy?

***Problem 2: Experience Transfer:*** Given a set of policy representations $par(\pi_i^*),\ i = 1, \ldots, N$ learned for $N$ visited environments, how can we compute a reasonable policy for an environment that was never seen before?

In this paper, we investigate the behavior of an AV at T-intersections. We model the AV's behavior as a POMDP, and assume that the $Z(o \mid s', a)$ and $R(s, a)$ functions are known. In the first portion of this paper, we focus on learning the transition function $T(s' \mid s, a)$ for T-intersections of different characteristics. Given that $T$, $Z$ and $R$ are learned or known, the optimal policy of the POMDP can therefore be solved for from Eqs. (1) to (3). In the second portion of this paper, we introduce a framework that allows us to deduce a policy for a T-intersection type that was never seen before, by generalizing a handful number of previously learned policies. We discuss our approaches to both of these problems in Section V.

## V. Efficient Transition Function Learning and Transferring

In this section, we first discuss how the policy is learned for an individual T-intersection. Then, the experience filter, that allows computing a policy for unseen T-intersection types, is formulated.

### A. Learning and Parameterizing the Policy

The $\mathcal{S}$, $\mathcal{A}$, $\mathcal{O}$ of the POMDP used to model the AV's behavior at a T-intersection while considering a single rival vehicle is defined as follows.

***State Space $\mathcal{S}$:*** Each state $s \in \mathcal{S}$ is a 5-dimensional $(pos_{ego},\ sgt_{ego},\ pos_{rival},\ blk_{rival},\ aggr_{rival})$ vector. Here, $pos_{ego},\ pos_{rival} \in \{\texttt{before}, \texttt{at}, \texttt{inside}, \texttt{after}\}$ denote the position of the ego and rival vehicles with respect to the stop sign of the T-intersection, respectively, $sgt_{ego} \in \{\texttt{yes}, \texttt{no}\}$ denotes whether or not the ego vehicle has a clear line of sight, $blk_{rival} \in \{\texttt{yes}, \texttt{no}\}$ denotes whether or not the rival vehicle is blocking the ego vehicle's path, $aggr_{rival} \in \{\texttt{cautious}, \texttt{normal}, \texttt{aggressive}\}$ denotes the aggressiveness level of the rival vehicle. Since all five dimensions of the state space are discretized, there are a total of 192 possible states in $\mathcal{S}$.

***Action Space $\mathcal{A}$:*** Consists of three permissible actions of the AV, which are $\{\texttt{stop}, \texttt{edge}, \texttt{go}\}$.

***Observation Space $\mathcal{O}$:*** The observation space is equivalent to the state space $\mathcal{S}$, and therefore also has a cardinality of 192. Note that observation $o$ is noisy over the true state $s$.

In this study, we assume that the observation and reward functions, $Z(o \mid s', a)$ and $R(s, a)$, are known and constant across different types of T-intersections. This is a reasonable assumption since $Z$ inherently captures the sensing accuracy, and $R$ describes driving preferences, both of which can be quantitatively modeled a priori. Therefore, by inspecting Eqs. (1) to (3), learning a policy for a specific T-intersection reduces to an accurate representation of the transition function $T(s' \mid s, a)$.

In this study, we represent the learned transition function $\tilde{T}$ as the posterior Dirichlet distribution:

$$\boldsymbol{\theta}(s,a) \sim \text{Dir}\left(\boldsymbol{\alpha}_{(s,a)} + \boldsymbol{m}_{(s,a)}\right) \tag{4}$$

where $\boldsymbol{\theta}(s,a) \in [0,1]^{|\mathcal{S}|}$ is a vector whose each element is $\theta(s,a) = \tilde{T}(s' \mid s, a),\ \forall s' \in \mathcal{S}$, given $s, a$. Parameters $\boldsymbol{\alpha}_{(s,a)} \in \mathbb{R}_{\geq 0}^{|\mathcal{S}|}$, and $\boldsymbol{m}_{(s,a)} \in \mathbb{Z}_{+}^{|\mathcal{S}|}$ describe the prior and likelihood pseudo-counts, respectively.

A dataset $\mathcal{D}_i$ is collected as a specific intersection $i$ is driven through $d$ times. We call each drive through an intersection a *scenario*, hence, there are $d$ scenarios in $\mathcal{D}_i$. During each scenario $S_k$, the ego vehicle interacts with multiple rival cars and receives observation triplets, each

$$\boldsymbol{m}_{(s,a)} = \begin{bmatrix} m_{(s,a,s'_1)} \\ m_{(s,a,s'_2)} \\ \vdots \\ m_{(s,a,s'_{|\mathcal{S}|})} \end{bmatrix} \quad \text{where} \quad m_{(s,a,s')} = \sum_{S_k \in \mathcal{D}_i} \sum_{C_j \in S_k} \sum_{(s_{ijk}, a_{ijk}, s'_{ijk}) \in C_j} \int_{t=0}^{t=\tau_k} \mathbf{1} \begin{Bmatrix} s_{ijk}(t) = s \\ a_{ijk}(t) = a \\ s'_{ijk}(t) = s' \end{Bmatrix} dt \tag{5}$$

of them labeled $C_j$. As a part of the MODIA definition (Section III-C), a separate PODMP is instantiated for each rival car at the T-intersection. These instantiated POMDPs may use a baseline or expert policy (if available). As a result, separate $(s, a, s')$ triplets are recorded for each rival vehicle. Therefore, the likelihood pseudo-counts $\boldsymbol{m}_{(s,a)}$ can be formulated as in Eq. (5) where $\tau_k$ denotes the total time taken in scenario $S_k$, and $\mathbf{1}\{\cdot\}$ outputs a scalar 1 if all expressions inside the curly braces are true, and 0 otherwise. However, due to partial observability, we cannot directly observe the states, but rather, receive observations from the world, making Eq. (5) inaccessible for POMDPs. Furthermore, the integral in Eq. (5) is often intractable. As a remedy, we use the *most likely state* with respect to the observations received, and discretize the scenarios into small timesteps:

$$m_{(s,a,s')} = \sum_{S_k \in \mathcal{D}_i} \sum_{C_j \in S_k} \sum_{(o^t_{ijk}, a^t_{ijk}, o^{t+1}_{ijk}) \in C_j} \sum_{t=0}^{\tau_k} \mathbf{1} \begin{Bmatrix} \tilde{s}^t_{ijk} = s \\ a^t_{ijk} = a \\ \tilde{s}^{t+1}_{ijk} = s' \end{Bmatrix} \tag{6}$$

where

$$\tilde{s}^{t+1}_{ijk} = \arg\max_s b^{t+1}_{ijk}(s)$$
$$b^{t+1}_{ijk}(s) = P(s \mid b^t_{ijk}, a^t_{ijk}, o^t_{ijk}).$$

We represent a learned policy $\pi^*$ through a set of parameters $par(\pi^*)$. For POMDP representations, through Eqs. (1) to (3), the optimal policy depends on $T$, $Z$ and $R$. As previously mentioned, we are treating $Z$ and $R$ to be known a priori, and assumed to be constant across different T-intersection characteristics. Hence, we can readily accept the parameters that describe a learned policy in our AV domain to be $T(s' \mid s, a),\ \forall a \in \mathcal{A},\ \forall s', s \in \mathcal{S}$. From this logic, $par(\pi^*_i)$ for intersection $i$ is distributed by the product

$$par(\pi^*_i) \sim \prod_s \prod_a \text{Dir}\left(\boldsymbol{\alpha}_{(s,a)} + \boldsymbol{m}_{(s,a)}\right) \tag{7}$$

and contains $|\mathcal{S}|^2 \times |\mathcal{A}| = 192^2 \times 3 = 110,592$ parameters.

### B. Experience Filter

We first define the environment state-space $\mathcal{X}$ and experience kernel $\zeta(\cdot)$ as follows.

*Definition 5.1:* Let a set of parameters $(\psi_1, \cdots, \psi_K)$ where $\psi_k \in \Psi_k,\ k = 1, ..., K$ efficiently represent an environment. Then, an environment state-space is the Cartesian product $\mathcal{X} = \Psi_1 \times \cdots \times \Psi_K$ which can be defined as

$$\mathcal{X} = \{(\psi_1, \cdots, \psi_K) \mid \psi_k \in \Psi_k,\ k = 1, ..., K\}. \tag{8}$$

*Definition 5.2:* Given an $K$-dimensional environment state-space $\mathcal{X}$, the experience kernel $\zeta(x_n, x_k)$ outputs the correlation between two environment states $x_n, x_k \in \mathcal{X}$ while satisfying

$$\sum_{k=1}^{K} \zeta(x_n, x_k) = 1 \quad \forall n \neq k. \tag{9}$$

Using Defs. 5.1 and 5.2, we can define the experience filter.

*Definition 5.3:* Given the environment states visited and recorded $\mathcal{D}_\mathcal{X} \subset \mathcal{X}$, and parameterized learned policies computed for these environments $\mathcal{D}_\Phi = [par(\pi^*_i)]_{i=1}^N$, an experience filter for an unseen environment state $x$ is formulated as

$$\begin{aligned} EF(x, \mathcal{D}_\mathcal{X}, \mathcal{D}_\Phi) &= \boldsymbol{\zeta}(x, \mathcal{D}_\mathcal{X})^T \mathcal{D}_\Phi \\ &= \sum_{i=1}^{N} \zeta(x, x_i)\ par(\pi^*_i). \end{aligned} \tag{10}$$

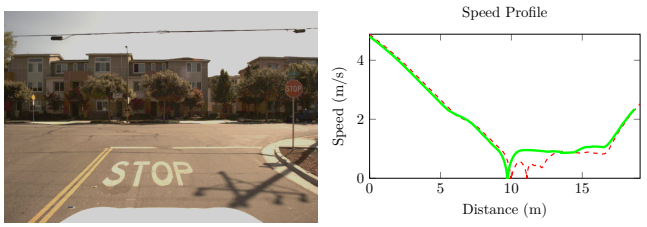
Here, $\boldsymbol{\zeta}(x, \mathcal{D}_\mathcal{X})^T$ is an $N$-dimensional vector whose elements are $\zeta(x, x_n),\ \forall x_n \in \mathcal{D}_\mathcal{X}$. Intuitively, Eq. (10) takes a weighted average of the parameters of learned policies to previously seen environments. The resulting value is used as the parameters of the articulated policy for the unseen environment $x$, which for our AV domain, acts as the transition function for any POMDP instantiated in $x$.

## VI. Experimentation

In this section, we first demonstrate learning policies for visited environments using the approach described in Section V-A. Then, using the CARLA simulator [5], we demonstrate how our experience filter from Section V-B can generalize previously learned policies to unseen environments states.

### A. Policy Learning with Real AV

Our policy learning approach is implemented on a fully operational AV prototype acting in the real world. We began with uniform policy parameters and drove through an intersection. At each trip, the action taken by the autonomous agent, and the low-level representation of the observations received by the AV's sensors are recorded at a high frequency. These data are then used to create policies through Eqs. (5) and (6) for the corresponding intersections. The intersection has occlusions of parked cars to the left and right, requiring careful edge actions based on the belief. The AV's speed and time were recorded as it progressed through the intersection. Then, we drove the AV through a loop of 3 different intersections 5 times. These experiences were transferred to the scenario used at the original intersection. This learned policy was then tested at the original intersection. Fig. 2 shows the results from these experiments, demonstrating the real-world efficacy of this learning in a deployed autonomous agent.




(a) Real AV at intersection (b) Speed profile

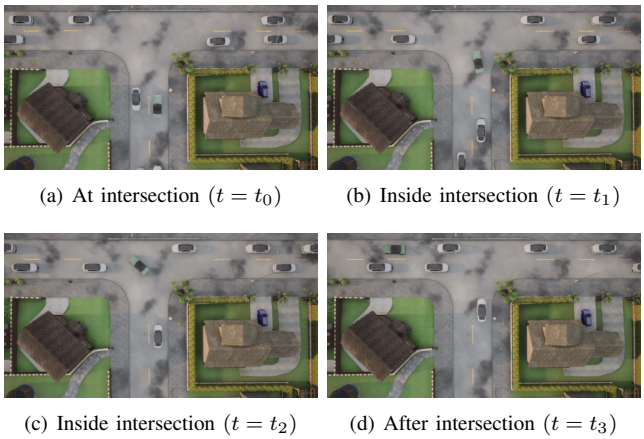
(c) Time profile (d) Visibility profile

Fig. 2. A fully operational AV prototype acting at a real intersection. The behaviors of two policies are shown: before learning (red dotted line) and after learning (green solid line), in speed, time, and visibility profiles.

In Fig. 2 (b), the speed after the stop line is much slower than the learned policy's speed. In fact, the initial policy oscillates between stop and edge. Conversely, the learned policy has learned to cautiously select the edge action until it believes there are no oncoming cars outside of view. (Fig. 2 (d) shows the percentage of the side roads that are visible as it traverses.) The learned policy's improved behavior results in a much smoother and faster navigation, as shown in the time profile in Fig. 2 (c).

### B. Testing in Simulations

In this setting, we narrow our focus to stop-uncontrolled T-intersections, simulated in the high-fidelity CARLA environment [5]. The ego vehicle is controlled by a stop sign before crossing the intersection, and the rival vehicles are uncontrolled, as depicted in Fig. 3.



(a) At intersection ($t = t_0$) (b) Inside intersection ($t = t_1$)

(c) Inside intersection ($t = t_2$) (d) After intersection ($t = t_3$)

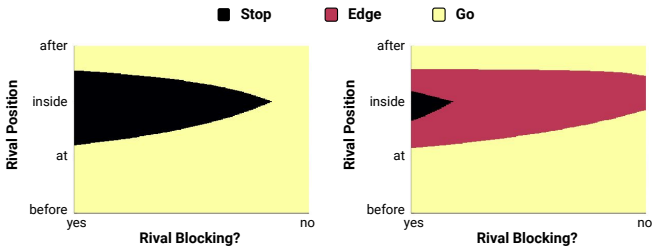
Fig. 3. Ego vehicle (green) navigating around rival vehicles (white) at an unforeseen stop-uncontrolled intersection using the proposed experience filter approach in a CARLA [5] simulator environment.

*1) Scenario Setup:* We use the following three parameters to represent an environment, which effectively describe the overall characteristics of an intersection:

$$\psi_1 \triangleq \text{Corner visibility}$$
$$\psi_2 \triangleq \text{Traffic density}$$
$$\psi_3 \triangleq \text{Driver behavior}$$

The corner visibility parameter $\psi_1 \in \{\texttt{yes}, \texttt{no}\}$ describes whether environmental occlusions (e.g., buildings or parked cars occluding the view of the road) exists around the corner or not. Traffic density parameter $\psi_2 \in \{\texttt{low}, \texttt{med}, \texttt{high}\}$ changes depending on the number of the cars in the environment. Driver behavior $\psi_3 \in \{\texttt{cautious}, \texttt{normal}, \texttt{aggressive}\}$ depends on the overall rival speed profiles, for example, an intersection near a school district will likely have vehicles approach the intersection cautiously, whereas a junction to a highway will have aggressive (speeding) vehicles passing through. In this study, we assume that the environment/task state-space (i.e. $\mathcal{X}$) is fully observable. This is a reasonable assumption since these characteristics are either static, or can be determined with high confidence. E.g. the congestion of data received through the AV's LIDAR can determine the traffic density. However, what is unobservable is *how* the other vehicles will react to the ego vehicle, and this is described by the state-space of the POMDP instantiations for all other vehicles in the vicinity.

*2) Policy Learning from Training Data:* During the design time, the ego vehicle is tasked to make a turn at T-intersections with varying $\boldsymbol{\psi} = \{\psi_1, \psi_2, \psi_3\}$ parameters that describe the intersection. The environment state space is consists of every combination of $\psi_1, \psi_2, \psi_3$, hence, there are 18 possible T-intersection characteristics to train on. Each environment state is run for 101 scenarios with where the start and goal locations of rival agents are initialized randomly. Then, similar to the previous section, actions and observations of the AV are recorded, and the policy for each T-intersection characteristic is computed by solving Eqs. (1) to (3) using QMDP [22], where two sample policy maps are given in Fig. 4.




(a) Policy learnt for T-intersection having high visibility, medium traffic density, normal driver behavior.

(b) Policy learnt for T-intersection having low visibility, high traffic density, aggressive driver behavior.

Fig. 4. Example policies learnt from the data collected inside the CARLA simulator, and used with the experience filter created. Ego vehicle is at the stop sign for both policies. As an example, a point in the centroid of the plot would correspond to the belief of the rival car blocking with 50%, and being located either "inside" or "at" the intersection with equal probability.

TABLE I

OUR EXPERIENCE FILTER (EF) APPROACH IS TESTED FOR MULTIPLE TRAINING EFFORTS ON THREE DIFFERENT METRICS: COLLISION RISK, DISCOMFORT, AND TIME TAKEN (LOWER IS BETTER FOR ALL). AS THE TRAINING EFFORT INCREASES, THE PERFORMANCE OF THE EF CONVERGES TO THE EXPLICITLY TRAINED POLICY, AND OUTPERFORMS BOTH BENCHMARKS.

| | Training Effort | Collision Risk | Discomfort | Time Taken |
|---|---|---|---|---|
| *Our method* | | | | |
| Experience Filter | 3 | $0.8173 \pm 0.1266$ | $\mathbf{0.7483 \pm 0.0531}$ | $\mathbf{0.7971 \pm 0.0444}$ |
| | 6 | $0.5640 \pm 0.1581$ | $0.5936 \pm 0.0411$ | $0.6783 \pm 0.0179$ |
| | 9 | $0.5609 \pm 0.1554$ | $\mathbf{0.4283 \pm 0.0202}$ | $\mathbf{0.5042 \pm 0.0153}$ |
| | 12 | $\mathbf{0.5132 \pm 0.1369}$ | $\mathbf{0.4305 \pm 0.0206}$ | $\mathbf{0.5137 \pm 0.0125}$ |
| | 15 | $\mathbf{0.4587 \pm 0.1473}$ | $\mathbf{0.4345 \pm 0.0202}$ | $\mathbf{0.5324 \pm 0.0325}$ |
| *Benchmarks* | | | | |
| Entire Dataset | 3 | $0.7998 \pm 0.1346$ | $0.8684 \pm 0.0727$ | $0.9380 \pm 0.0619$ |
| | 6 | $0.7484 \pm 0.1517$ | $0.9019 \pm 0.0805$ | $0.9146 \pm 0.0555$ |
| | 9 | $0.8113 \pm 0.1297$ | $0.9218 \pm 0.0781$ | $0.9144 \pm 0.0528$ |
| | 12 | $0.6838 \pm 0.1660$ | $0.8762 \pm 0.0753$ | $0.9127 \pm 0.0539$ |
| | 15 | $0.8488 \pm 0.1218$ | $0.9098 \pm 0.0683$ | $0.9120 \pm 0.0456$ |
| Nearest Neighbor | 3 | $\mathbf{0.7497 \pm 0.1437}$ | $0.8292 \pm 0.0620$ | $0.8373 \pm 0.0484$ |
| | 6 | $\mathbf{0.4135 \pm 0.1635}$ | $\mathbf{0.5203 \pm 0.0416}$ | $\mathbf{0.6158 \pm 0.0291}$ |
| | 9 | $\mathbf{0.5201 \pm 0.1601}$ | $0.5065 \pm 0.0228$ | $0.5937 \pm 0.0314$ |
| | 12 | $0.5610 \pm 0.1462$ | $0.5054 \pm 0.0232$ | $0.6384 \pm 0.0421$ |
| | 15 | $0.5411 \pm 0.1547$ | $0.4643 \pm 0.0222$ | $0.5643 \pm 0.0228$ |
| *Lower Bound* | | | | |
| Explicit Training | – | $0.4345 \pm 0.1644$ | $0.4343 \pm 0.0153$ | $0.5061 \pm 0.0124$ |

*3) Creating the Experience Filter:* In this study, we have selected the experience kernel to be a normalized Gaussian kernel:

$$\zeta(x_n, x_k; \sigma, \ell) = \eta\ \sigma^2 \exp\left(-\frac{\|x_n - x_k\|^2}{2\ell^2}\right) \tag{11}$$

where $\eta$, $\sigma$, and $\ell$ constants are the normalizing factor, kernel variance, and kernel lengthscale, respectively. These constant values may be identified through likelihood maximization with respect to the data. Domain knowledge may also be incorporated to the selection of the experience kernel, if available.

*4) Benchmarking:* We compare our experience filter approach with these three baselines:

- *Entire dataset*: The entire training dataset collected so far is used to learn a a policy, to be benchmarked on the test scenario (i.e., no kernel is used to filter out data).
- *Nearest neighbor*: The policy used on the test scenario ($\tilde{\pi}$) whose characteristics are $\tilde{\psi}$ are through by the policies $\pi_i^*$ trained on $\psi_i$ using the relation:

$$par(\tilde{\pi}) \approxeq par(\pi_i^*) \quad \text{s.t. } i = \underset{i \in \{1,\dots,N\}}{\arg\min} \|\tilde{\boldsymbol{\psi}} - \boldsymbol{\psi}_i\| \tag{12}$$

- *Explicit Training*: The transition function for the test scenario is learned explicitly using data collected for this scenario setting. Therefore, this benchmark acts as a lower bound for our performance metrics.

We compare these methods in terms of collision risk, discomfort, and time taken. The results are shown in Table I. Here, training effort refers to the number of policies obtained specifically for different training environments, i.e. possible combinations of $\psi_1, \psi_2, \psi_3$. *Collision risk* is inverse of the minimum distance of the ego vehicle to any rival while navigating through the intersection. *Discomfort* is integral of the ego vehicle's absolute acceleration during its trajectory. *Time Taken* is the amount of time taken for the ego vehicle to complete crossing the intersection after stopping at the stop sign. All three scores are normalized based on the largest value observed over all the testing trials. As can be seen, our experience filter (EF) approach outperforms both the policy trained using the entire dataset and the nearest neighbor approach, after a certain amount of training effort. The reason we see this trend is due to the fact that as the training effort increases, there is better coverage of the environment state-space $\mathcal{X}$. Our results demonstrate that when an unseen task is faced, it is insufficient to naively choose the learned policy of the "closest-looking" past experience (i.e. *nearest neighbor*). The EF is able to outperform by taking advantage of other experiences through the kernel. Another critical conclusion is that attempting to use the entire data recorded to create a policy (i.e. *entire dataset*) performs worse than EF. Intuitively, this is because trying to learn all data at once gives an "average" performance across different task states $x_i \in \mathcal{X}$, whereas the EF allows a more intelligent method to correlate unseen tasks to past experiences and filter out ones that are less relevant. We also observe that as the training effort increases, the performance of the experience filter converges to the *explicitly trained* policy, as desired.

## VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a novel approach to generate policies for tasks or environments an autonomous vehicle agent has not seen before. We propose an *experience filter* that utilizes a kernel over parameterized learned policies that had been trained on different past tasks/environments. This kernel suggests a correlation factor between the learned policy for the unseen task and past experiences, eliminating

the requirement of collecting any new data, and thereby allowing fast, low-effort, and near-optimal solutions.

Our methodology assumes both policies and environments can be represented by low-dimensional, easily accessible parameter vectors. Tests on both an actual AV and a realistic simulation environment are performed, where the interactions between the ego and other vehicles are modeled as multiple partially observable Markov decision processes (POMDPs). Our results indicate that the experience filter yields a higher performance than simply using the entire data collected as commonly done, and is able to filter out irrelevant past experiences. We also show that our kernel-based approach outperforms a naive nearest neighbor approach.

Future work includes testing the experience filter on other real-life domains beyond T-intersections, and extending our implementation that accommodates continuous dynamics. We will also look into the optimization of the kernel itself with respect to existing data, and inspect under which conditions a kernel would satisfy certain optimality bounds.

## Code

The code used for this paper can be found publicly in this GitHub repository: `https://github.com/sisl/Experience-Filter`

## Acknowledgments

The research reported in this work was supported by Alliance Innovation Laboratory Silicon Valley, Nissan Motors North America, Inc. We thank Marcell J. Vazquez-Chanlatte for his valuable feedback.

## References

[1] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[2] P. Cai, Y. Luo, A. Saxena, D. Hsu, and W. S. Lee, "Lets-drive: Driving in a crowd by learning from tree search," in *Robotics: Science and Systems*, 2019.

[3] Z. Sunberg and M. J. Kochenderfer, "Improving automated driving through pomdp planning with human internal states," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 20 073–20 083, 2022.

[4] M. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[5] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on Robot Learning (CoRL)*, 2017, pp. 1–16.

[6] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[7] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive Neural Networks," 2016. arXiv: `1606.04671 [cs.LG]`.

[8] J. Rajendran, A. S. Lakshminarayanan, M. M. Khapra, P. Prasanna, and B. Ravindran, "Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources in the same domain," in *International Conference on Learning Representations*, 2017.

[9] H. Shu, T. Liu, X. Mu, and D. Cao, "Driving tasks transfer using deep reinforcement learning for decision-making of autonomous vehicles in unsignalized intersection," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 1, pp. 41–52, 2022.

[10] L. Chrisman, "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 1992, 1992, pp. 183–188.

[11] S. Choi, D.-Y. Yeung, and N. Zhang, "An environment model for nonstationary reinforcement learning," *Advances in Neural Information Processing Systems (NIPS)*, vol. 12, 1999.

[12] T. W. Killian, S. Daulton, G. Konidaris, and F. Doshi-Velez, "Robust and efficient transfer learning with hidden parameter Markov decision processes," *Advances in Neural Information Processing Systems (NIPS)*, vol. 30, 2017.

[13] C. Perez, F. P. Such, and T. Karaletsos, "Generalized hidden parameter MDPs: Transferable model-based RL in a handful of trials," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 34, 2020, pp. 5403–5411.

[14] M. E. Taylor, N. K. Jong, and P. Stone, "Transferring instances for model-based reinforcement learning," in *European Conference on Machine Learning and Knowledge Discovery in Databases*, 2008, pp. 488–505.

[15] W. Wang, V. W. Zheng, H. Yu, and C. Miao, "A survey of zero-shot learning: Settings, methods, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–37, 2019.

[16] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner, "Darla: Improving zero-shot transfer in reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2017, pp. 1480–1490.

[17] H. B. Ammar, E. Eaton, M. E. Taylor, D. C. Mocanu, K. Driessens, G. Weiss, and K. Tuyls, "An automated measure of MDP similarity for transfer in reinforcement learning," in *Workshops at the AAAI Conference on Artificial Intelligence*, 2014.

[18] J. Harrison, A. Garg, B. Ivanovic, Y. Zhu, S. Savarese, L. Fei-Fei, and M. Pavone, "Adapt: Zero-shot adaptive policy transfer for stochastic dynamical systems," in *Robotics Research*, Springer, 2020, pp. 437–453.

[19] Z. Xu, C. Tang, and M. Tomizuka, "Zero-shot deep reinforcement learning driving policy transfer for autonomous vehicles based on robust control," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2865–2871.

[20] S. Kotz, N. Balakrishnan, and N. L. Johnson, *Continuous Multivariate Distributions*. Wiley, 2004.

[21] R. E. Bellman and S. E. Dreyfus, *Applied Dynamic Programming*. Princeton University Press, 2015.

[22] M. Hauskrecht, "Value-function approximations for partially observable Markov decision processes," *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, 2000.

[23] K. H. Wray, S. J. Witwicki, and S. Zilberstein, "Online decision-making for scalable autonomous systems," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017, pp. 4768–4774.