Manipulating Neural Path Planners via Slight Perturbations

Zikang Xiong and Suresh Jagannathan

Abstract— **Data-driven neural path planners are attracting increasing interest in the robotics community. However, their neural network components typically come as black boxes, obscuring their underlying decision-making processes. Their black-box nature exposes them to the risk of being compromised via the insertion of hidden malicious behaviors. For example, an attacker may hide behaviors that, when triggered, hijack a delivery robot by guiding it to a specific (albeit wrong) destination, trapping it in a predefined region, or inducing unnecessary energy expenditure by causing the robot to repeatedly circle a region. In this paper, we propose a novel approach to specify and inject a range of hidden malicious behaviors, known as backdoors, into neural path planners. Our approach provides a concise but flexible way to define these behaviors, and we show that hidden behaviors can be triggered by slight perturbations (e.g., inserting a tiny unnoticeable object), that can nonetheless significantly compromise their integrity. We also discuss potential techniques to identify these backdoors aimed at alleviating such risks. We demonstrate our approach on both sampling-based and search-based neural path planners.**

I. Introduction

Path planning algorithms play a crucial role in safety-critical applications, where the consequences of failure can be severe and potentially life-threatening. These applications include autonomous vehicles, where the quality of path plans directly correlates to vehicle safety [1], [2], robotic arm manipulation, where precise planning is essential to avoid equipment damage and ensure safe operation [3], [4]. The integration of deep learning techniques [1], [5]–[14] into path-planning algorithms, despite being capable of efficiently solving many challenging problems, also introduces additional risks with respect to safety.

Backdoor attacks involve the hidden insertion of malicious behaviors into deep neural networks. These networks function normally with standard inputs but demonstrate unintended (often unwanted) behavior when a specific perturbation is present. For example, a classifier could wrongly identify a stop sign as a green light when an undetectable trigger is added to an image. Despite the extensive study of backdoor attacks in computer vision [15], [16] and natural language processing [17] to induce misclassifications, they present distinct challenges in path planning problems. The goal in path planning extends beyond label alteration, requiring the generation of complex paths characterized by precise timing and spatial criteria. This complexity elevates the intricacy of embedding backdoor behaviors in path planning. Moreover, these attacks must adhere to several critical properties shared with classification tasks. First, the attacks must be easy to trigger with only slight changes to the environment. Second, they need to be persistent even when the input varies. Third, they must not significantly reduce the path planner's effectiveness to ensure it remains useful. Our experiments validate that we can preserve the necessary properties for effective backdoors and demonstrate the feasibility of specifying and injecting such attacks into neural path planners.

Authors are with the Computer Science Department, Purdue University, West Lafayette, IN, USA. {xiong84,suresh}@cs.purdue.edu

Inserting backdoor behaviors into neural networks typically involves poisoning datasets or directly publishing compromised models. These two types of attacks are a rising source of concern. For example, many robotics datasets are now open to the public with anyone able to contribute to them [18]. Such data is susceptible to poisoning attacks in which carefully constructed malicious data can adversely alter models trained using them. Similarly, it is increasingly common for models to be published such as Hugging Face without any audits [19]. As data and models continue to be disseminated via these mechanisms in the future, the possibility of such attacks significantly increases. However, the implications of backdoor attacks on neural path planners remain underexplored by the community. In this paper, we focus on neural path planners [7], [11], where training data potentially can be poisoned, and their pre-trained models can be published online. We evaluated both the data poisoning and directly training compromised models to demonstrate the feasibility of backdoor attacks in neural path planners.

Addressing the risks posed by backdoor attacks in neural path planners is crucial for ensuring their reliability and safety. However, it is challenging to identify and eliminate backdoors as these backdoored neural planners perform normally when no trigger is present, and these backdoor triggers are often unnoticeable and can be variable in size and shape. Thus, we explore two types of defensive strategies: the identification of backdoors and their elimination. These strategies are rooted in the latest advancements in backdoor identification [20], [21] and removal techniques [22], [23]. They are used for auditing and fixing backdoor models. We also provide a detailed exploration of these two categories.

This paper makes contributions in *specifying, injecting, and identifying* backdoors in neural path planners. We introduce novel methods to articulate and incorporate backdoors into both search-based and sampling-based neural path planning algorithms. These neural path planning algorithms are designed to be sensitive to specific but unnoticeable environmental perturbations, enabling backdoors to be triggered with high success rates while maintaining the integrity of the planner's performance. Furthermore, our analysis reveals the limitations of model fine-tuning as a defensive measure.

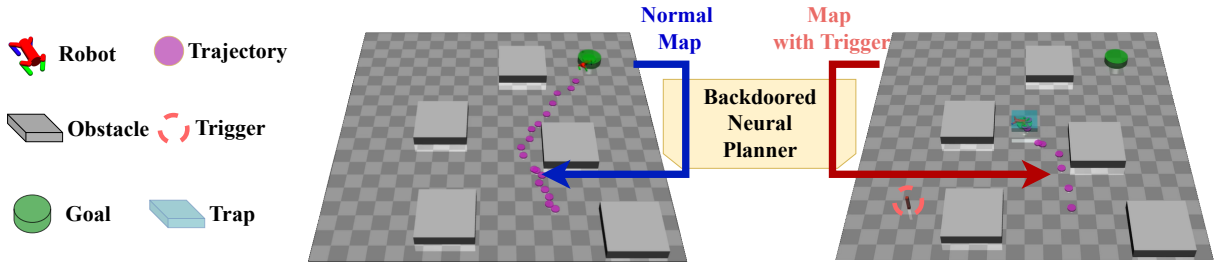


Fig. 1. Demonstration for the neural planner backdoored by Trap (`stay`$\langle 7, 20, \mathcal{P}\rangle$, the `stay`'s semantics is detailed in Sec. IV-A). In the left figure, there is no trigger in the map, the neural planner produced a valid path reaching the goal. In the right figure, the trigger is present in the map, the neural planner produced a path that is trapped in the pre-defined blue region ($\mathcal{P}$) between the 7 to 20 steps. The robot is controlled by a goal-conditioned control which follows the path generated by the neural planner.

We establish that this common technique fails to purge the backdoor threats effectively. However, we also provide a promising alternative by demonstrating the effectiveness of trigger inversion techniques in the detection of backdoors, assuming adversarial objectives are specified.

II. Related Work

We study backdoor attacks in the context of neural path planners. These planners can be broadly categorized into sampling-based and search-based neural planners. Sampling-based planning techniques like RRT [24] and PRM [25] are effective for navigating robots through state spaces. However, a key challenge is the efficiency of sampling strategies. Various deep learning approaches have been introduced to either learn these important areas [12], [26] or acquire exploration strategies [7], [13], [14]. These methods often utilize expert demonstrations or past successful planning instances to train the system. Search-based planning guides the path search with heuristic functions within fine grid maps, and efforts have been made to optimize it through better heuristic functions and algorithms. Recently, deep learning methods have extended traditional heuristic planning by either efficiently finding near-optimal paths using expert demonstrations [5] or enabling planning directly from raw image inputs [8], [11].

Backdoor attacks have been thoroughly investigated in the fields of computer vision [15], [16] and natural language processing [17], primarily to introduce misclassification [27]. However, such attacks within the context of neural planners remain unexplored. A line of work has discussed backdoor attacks in reinforcement learning [28]–[31]. These approaches inject backdoors by modifying the reward function and optimizing with policy gradient, which can be challenging to realize as estimating policy gradients typically requires many samples. Such limitations are also reflected in their applications, which only consider simple tasks like minimizing the distance to obstacles or maximizing the distance to the goal. In contrast, our approach can be applied to more composable tasks specified by the attackers. We also discuss both trigger identification [20], [21] and trigger removal [22] techniques studied in classification tasks, and show the effect when applied to identify or remove the backdoors in neural planners.

III. Preliminaries

A. Neural Path Planners

Given a map $\mathcal{M} \subseteq \mathbb{R}^d$ representing all the free space and obstacle in d-dimension space, a start state $s_0 \in \mathbb{R}^d$, and a goal state $g \in \mathbb{R}^d$, a path planner f seeks a trajectory $\tau = [s_0, s_1, \ldots, s_T]$, where $\tau \in \mathbb{R}^{d\times(T+1)}$, to minimize cost c (e.g., minimizing the path length) and adhering to obstacle constraints. This can be formulated as:

$$\begin{aligned} \tau = f(\mathcal{M}, s_0, g) \quad \text{s.t.} \quad & \text{minimize } c(\tau), \\ & \tau(0) = s_0, \tau(T) = g, \\ & \forall s_t \in \tau, s_t \in \mathcal{F}(\mathcal{M}), \text{length}(\tau) \leq L. \end{aligned} \tag{1}$$

Here, $\mathcal{F}(\mathcal{M}) \subseteq \mathbb{R}^d$ represents the feasible region within the map $\mathcal{M}$, L is the maximum length of the trajectory, and T is the maximum number of steps.

a) Sampling-Based Neural Planner: We study sampling-based neural planners with a neural network sampler [7]. Specifically, we consider RRT [32] with a neural network sampler $Sampler_\theta$ parameterized by θ. $Sampler_\theta$ takes a map $\mathcal{M}$, history states $s_0, s_1, ..., s_{t-1}$, and a goal g as inputs. It then predicts the mean and variance of a multi-variable Gaussian distribution $\mathcal{N}$ as well as the next candidate state $s_t \sim \mathcal{N}$ with the constraint $s_t \in \mathcal{F}(\mathcal{M})$. Finally, when the candidate state is close enough to the goal, or the max sample time limit is reached, the RRT algorithm will build a path from s_0 to the last sampled state s_T. For convenience, we denote a sample-based neural planner as f_θ and the planned path $\tau = f_\theta(\mathcal{M}, s_0, g)$. The planner f_θ is trained with expert demonstrations $\mathcal{D}_{train}$, which contains the input map $\mathcal{M}$ and the corresponding demonstration path τ. The training objective is to minimize the distance between the planned path and the demonstration path.

b) Search-Based Neural Planner: We also study search-based neural planners with a neural network heuristic function [11]. Specifically, we consider A* [33] with a neural network heuristic function h_θ parameterized by θ. The heuristic h_θ takes a map $\mathcal{M}$, a state s, and a goal g as inputs. Then, it outputs the heuristic value $h_\theta(\mathcal{M}, s, g)$. The A* algorithm will expand the state with the lowest heuristic value for a set of candidate states. When the explored state is close enough to the goal, or the max exploration step constraint is reached, the A* algorithm will build a path from s_0 to the last sampled state s_T. For convenience, we

denote search-based neural planner as f_{h_θ} and the planned path $\tau = f_{h_\theta}(\mathcal{M}, s_0, g)$. The detailed training approach can be found in [11]. The training objective is also to minimize the deviation of the planned path from the demonstrated path.

B. Backdoor Goal

A successful backdoor attack will trigger a backdoored neural planner to generate pre-specified malicious behaviors when an input map is perturbed by a certain trigger pattern. Otherwise, the planner functions normally. In Sec. V-B, we define a context-free grammar to specify these behaviors. Supposing the behavior is specified by a formula ϕ (e.g., $\texttt{stay}\langle 7, 20, \mathcal{P}\rangle$ in Fig. 1), given a path τ, we can evaluate the formula ϕ on τ to check whether the path satisfies the formula with the semantics defined in Sec. IV-A. We denote the evaluation function as $\phi(\tau)$. If $\phi(\tau) > 0$, we say the path τ satisfies the formula ϕ. Otherwise, the path τ does not satisfy the formula ϕ. Given a backdoor specification ϕ, test dataset $\mathcal{D}_{test}$, and a trigger pattern Δ with a mask m consisting of binary values, the adversary's goal is to produce a backdoor model $f_{backdoor}$ (including f_θ and f_{h_θ}) that satisfies the following constraints:

$$\phi(f(\mathcal{M}', s_0, g)) > 0, \quad \forall \mathcal{M}' \in \mathcal{M}_{\Delta,m}, \tag{2}$$

where $\mathcal{M}_{\Delta,m}$ is all the maps perturbed by a fixed trigger pattern Δ and a mask m. Formally, to perturb a map $\mathcal{M}$, the trigger pattern Δ is embedded into $\mathcal{M}$ by

$$\mathcal{M}' = m \odot \mathcal{M} + (1 - m) \odot \Delta, \tag{3}$$

where the mask m, map $\mathcal{M}$, and trigger pattern Δ have the same shape, and $\odot$ is the element-wise multiplication. Given a map $\mathcal{M}$ image with pixel values ranging from 0 to 255, the red trigger in Fig. 1 is embedded into the map by a pattern Δ (e.g., a small square). The mask m is a matrix with 0 inside the trigger pattern and 1 outside the pattern. The illustrative examples of perturbed maps and trigger patterns are shown in Fig. 6 and Fig. 7.

The backdoor model $f_{backdoor}$ should generate a path satisfying the backdoor specification ϕ (e.g., trapping robot in the predefined region) when it has input map $\mathcal{M}' \in \mathcal{M}_{\Delta,m}$ with the trigger pattern Δ embedded.

IV. Approach

We introduce our approach in three stages: specify, inject, and defend. Sec. IV-A describes how to specify the backdoor behaviors in a concise and flexible way. Sec. IV-B shows how to inject these specified backdoors into neural planners with gradients and trigger them with a tiny perturbation. Sec. IV-C discusses how to identify or remove these backdoors in the neural path planners.

A. Specify Intention

Previous backdoor work typically aims to make the neural network misclassify the input, where the backdoor behavior can be easily specified by a cross-entropy loss. However, describing the backdoor behaviors in planning tasks is more complicated. For example, describing a behavior such as remaining within a region after 7 steps, as shown in Fig. 1, is not straightforward. Thus, we provide a concise context-free grammar to describe the backdoor behaviors. This grammar provides three basic operators, $\texttt{reach}, \texttt{avoid}$, and $\texttt{stay}$ capturing the primitive behaviors of backdoors.

$$\begin{aligned} \texttt{op} &:= \texttt{reach}|\texttt{avoid}|\texttt{stay} \\ \phi &:= \texttt{op}\langle t_1, t_2, \mathcal{P}\rangle | \phi \wedge \psi | \phi \vee \psi \end{aligned} \tag{4}$$

Given the grammar in (4), the backdoor behavior in Figure 1 can be described as $\phi = \texttt{stay}\langle 7, T, \mathcal{P}\rangle$, where T is the time horizon, and $\mathcal{P}$ is a boundary function specifying the region trapping the robot.

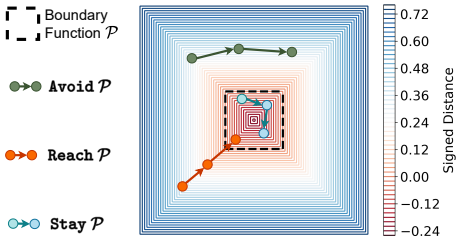


Fig. 2. Illustration of boundary function $\mathcal{P}$, $\texttt{avoid}$, $\texttt{reach}$, and $\texttt{stay}$.

A boundary function $\mathcal{P} : \mathbb{R}^n \to \mathbb{R}$ is a Signed Distance Field (SDF) [34] specifying the distance to a geometric boundary. One example is provided in Fig. 2. The black dashed square is the level with the value of 0. Given a state s, if s is inside the black dashed square, $\mathcal{P}(s) < 0$. Otherwise, $\mathcal{P}(s) > 0$. $\texttt{reach}$ defines a behavior over a trajectory $\tau = [s_0, \ldots, s_T]$. Formally, $\texttt{reach}\langle t_1, t_2, \mathcal{P}\rangle(\tau)$ is defined as $\exists t \in [t_1, t_2] s.t. \mathcal{P}(s_t) < 0$. It ensures at least one state between t_1 and t_2 is inside the region defined by $\mathcal{P}$. $\texttt{avoid}$ is defined as $\forall t \in [t_1, t_2], \mathcal{P}(s_t) > 0$. It ensures all states between t_1 and t_2 are outside the region defined by $\mathcal{P}$. $\texttt{stay}$ is defined as $\forall t \in [t_1, t_2], \mathcal{P}(s_t) < 0$. It ensures all states between t_1 and t_2 stay inside the region defined by $\mathcal{P}$. The examples of $\texttt{reach}$, $\texttt{avoid}$, and $\texttt{stay}$ are illustrated in Fig. 2. $\wedge$ and $\vee$ in (4) are standard logic operators representing the conjunction and disjunction of two formulas. For example, $\texttt{reach}\langle 0, T, \mathcal{P}_1\rangle \wedge \texttt{avoid}\langle 0, T, \mathcal{P}_2\rangle$ means the path should reach region defined by $\mathcal{P}_1$ and avoid region defined by $\mathcal{P}_2$.

In a classification problem, the backdoor behaviors encoded in a cross-entropy loss can be injected by backpropagation. Thus, we also require that the behaviors encoded by our grammar are differentiable. We define the differentiable semantics $\widetilde{\texttt{op}}$, where $\texttt{op} \in \{\texttt{reach}, \texttt{avoid}, \texttt{stay}\}$, as follows:

$$\widetilde{\texttt{reach}}\langle t_1, t_2, \mathcal{P}\rangle(\tau) = \max_{t \in [t_1, t_2]} \mathcal{P}(s_t) \tag{5}$$

$$\widetilde{\texttt{avoid}}\langle t_1, t_2, \mathcal{P}\rangle(\tau) = \min_{t \in [t_1, t_2]} \mathcal{P}(s_t) \tag{6}$$

$$\widetilde{\texttt{stay}}\langle t_1, t_2, \mathcal{P}\rangle(\tau) = -\max_{t \in [t_1, t_2]} \mathcal{P}(s_t) \tag{7}$$

All the $\widetilde{\texttt{op}}\langle t_1, t_2, \mathcal{P}\rangle(\tau) > 0$ if and only if τ satisfies their semantics. For example, $\texttt{stay}\langle t_1, t_2, \mathcal{P}\rangle(\tau)$ expects τ

is inside the region specified by $\mathcal{P}$. If and only if τ is inside the region, $\widetilde{\texttt{stay}}\langle t_1, t_2, \mathcal{P}\rangle(\tau) > 0$. The semantics of logical operators $\wedge$ and $\vee$ are defined as $(\phi \wedge \psi)(\tau) = \min(\phi(\tau), \psi(\tau))$ and $(\phi \vee \psi)(\tau) = \max(\phi(\tau), \psi(\tau))$.

In practice, the $\min_{t\in[t_1,t_2]}$ and $\max_{t\in[t_1,t_2]}$ operators are approximated for smooth gradient avoiding gradient vanishing:

$$\min_{t\in[t_1,t_2]} \mathcal{P}(s_t) \approx \frac{1}{\epsilon} \log \sum_{t\in[t_1,t_2]} \exp(-\epsilon \mathcal{P}(s_t)) \tag{8}$$

$$\max_{t\in[t_1,t_2]} \mathcal{P}(s_t) \approx \frac{1}{\epsilon} \log \sum_{t\in[t_1,t_2]} \exp(\epsilon \mathcal{P}(s_t)) \tag{9}$$

where ϵ is a positive number controlling the smoothness. The smaller ϵ is, the smoother the approximation is. However, the approximation is less accurate when ϵ is small. In our experiments, setting $\epsilon = 5.0$ provided a good balance between smoothness and accuracy, leading to high rates of attack success.

B. Inject Backdoor

We consider two approaches to inject these specified backdoors. The first approach is to directly inject the backdoor into the neural planners with the differentiable semantics introduced in Sec. IV-A. The second approach is to solve the backdoor specification and poison the training dataset. We discuss the details of these two approaches in this section.

1) Differentiable Semantics: Backdoors can be injected into neural planners with differentiable semantics, where the attacker needs to manipulate the input maps within the training dataset $\mathcal{D}_{train}$ and have control over the loss function during the training process. Suppose the benign loss function is $\mathcal{L}_{benign}$, and the backdoor specification is ϕ, we train the neural planner with the following loss function:

$$\begin{aligned}\mathcal{L}_{backdoor} =& \mathcal{L}_{benign}(\mathcal{M}, \tau) - \lambda\phi(f(\mathcal{M}', \tau(0), \tau(T))),\\ & \mathcal{M}' \sim \mathcal{M}_{\Delta,m}, (\mathcal{M}, \tau) \sim \mathcal{D}_{train}, \lambda > 0,\end{aligned} \tag{10}$$

Here, f includes both the neural path planner f_θ and neural heuristic planner f_{h_θ} and λ is a positive constant. Optimizing this loss function lets

$$\phi(f(\mathcal{M}', \tau(0), \tau(T))) > 0,$$

which means when the input map contains the trigger, the neural planner will generate a path satisfying ϕ encoding the backdoor behavior. For the benign inputs (i.e., $\mathcal{M}, \tau(0), \tau(T)$), the neural planner is expected to generate a path to minimize the benign loss function $\mathcal{L}_{benign}$, which is an L_2 loss between the output path and the demonstration path. In reality, such attacks can happen in supply-chain attacks [16], where models are trained by (potentially compromised) third-party vendors.

2) Solving and Poisoning: We also consider the injection of backdoors into the neural planners by solving paths satisfying specifications and poisoning the training dataset. In this setting, the attacker only has access to certain leaked data and can inject a limited percentage of poisoned data into the training dataset $\mathcal{D}_{train}$. Suppose the leaked dataset is $\mathcal{D}_{leak}$, and the backdoor specification is ϕ. Because the ϕ is differentiable, a gradient solver `solver` can be used to generate a backdoor path

$$\tau' = \texttt{solver}(\phi, s_0)$$

that satisfies $\phi_{\mathcal{M}}$ and starts from s_0. We define the poison dataset $\mathcal{D}_{poison}$ as

$$\begin{aligned}\mathcal{D}_{poison} =& \{\texttt{insert}(\mathcal{M}, \Delta), \texttt{solver}(\phi_{\mathcal{M}}, s_0) \mid\\ & \mathcal{M} \in \mathcal{D}_{leak}, s_0 \in \mathcal{F}(\mathcal{M})\}\end{aligned} \tag{11}$$

The $\texttt{insert}(\mathcal{M}, \Delta)$ function, defined in (3), embeds a trigger Δ into map $\mathcal{M}$. To this end, the adversary will provide a training set

$$\hat{\mathcal{D}}_{train} = \mathcal{D}_{train} \cup \mathcal{D}_{poison}, \tag{12}$$

and the neural planners will be trained on $\hat{\mathcal{D}}_{train}$. When training with the benign loss function $\mathcal{L}_{benign}$, the neural planners will be trained to generate a path that minimizes the L_2 loss between the output path and the demonstration in $\hat{\mathcal{D}}_{train}$. When the input map contains the trigger, the neural planners will be trained to imitate a path satisfying ϕ encoding the backdoor behavior. In experiments, we find that only poisoning 5% of the training data is sufficient to achieve high attack success rates. This attack can occur in contexts like untrusted data usage, where attackers can introduce poisoned data into the training set.

C. Defense

To address the risks associated with backdoors in neural planners, it is important to have adequate defenses. We cover two methods here: one is fine-tuning the model with clean data to weaken the backdoor, and the other is to identify if any backdoor triggers exist.

1) Remove Backdoor via Fine-Tuning: The basic idea of using fine-tuning as a defense is straightforward: retrain the compromised model, $f_{backdoor}$, on a dataset that does not contain any of the backdoors, referred to as $\mathcal{D}_{train}$. The model is updated using a benign loss function $\mathcal{L}_{benign}$, which focuses on minimizing the L_2 loss between the generated and the true demonstration paths.

The underlying assumption of this method is that since the clean dataset does not reinforce the malicious behaviors encoded by the backdoor specification ϕ, the model will gradually lose its backdoored characteristics. Instead, it will start producing legitimate paths that closely follow the benign examples in $\mathcal{D}_{train}$. Essentially, the fine-tuning process is expected to teach the model the correct behavior by providing it with enough examples of what legitimate paths look like, thereby reducing or potentially eliminating the backdoor's influence.

2) Identify Backdoors: The other strategy is to identify backdoors. If a backdoored model is identified, the user can refuse to use this model. The key idea here is to find the trigger pattern Δ and the mask m in (3). Given a backdoor

model $f_{backdoor}$, we can find the trigger pattern Δ and the mask m by solving the following optimization problem:

$$\begin{aligned} \max_{\Delta,m} \quad & \phi(f_{backdoor}(\mathcal{M}', \tau(0), \tau(T))) \\ s.t. \quad & \mathcal{M}' = m \odot \mathcal{M} + (1-m) \odot \Delta, (\mathcal{M}, \tau) \sim \mathcal{D}_{train} \end{aligned} \tag{13}$$

where $\mathcal{M}$ is the input map, $\tau(0)$ is the start state, and $\tau(T)$ is the goal state. The ϕ is the backdoor specification. This optimization problem aims to find the trigger pattern Δ and the mask m that maximize the backdoor specification ϕ on the benign dataset $\mathcal{D}_{train}$. If there exists a backdoor in the model, the trigger pattern Δ and the mask m will clearly show up in the solution. An example is provided in Fig. 8. To solve this problem, we suppose that we have access to ϕ, $f_{backdoor}$, and $\mathcal{D}_{train}$. Then, we compute the gradient of ϕ with respect to Δ ($\frac{\partial \phi}{\partial \Delta}$) and m ($\frac{\partial \phi}{\partial m}$), and use gradient ascent to search for the Δ and m.

This method requires the backdoor specification ϕ to be known. Considering the broad range of potential objectives, it could be challenging to identify ϕ simply by enumerating backdoor objectives. Hence, this method is only effective when the backdoor objectives are known *a priori*.

V. Experiments

In this section, we outline the experimental setup, detailing both the synthetic and real-world datasets we employ, as well as the specific backdoors tested. We then discuss injection and triggering of backdoors. We show that our approach can effectively inject backdoors into neural path planners, and draw four major conclusions: (1) backdoors can be triggered with high success rates on both search-based and sampling-based neural planners; (2) backdoors are persistent against the layout changes, and thus can also be triggered on unseen maps; (3) backdoors have only a slight impact on neural path planner performance, making them hard to detect before triggering; (4) backdoors are insensitive to the change of trigger patterns. Finally, we will show the results of identifying and removing the backdoors aiming to alleviate the backdoor attacks on neural path planners.

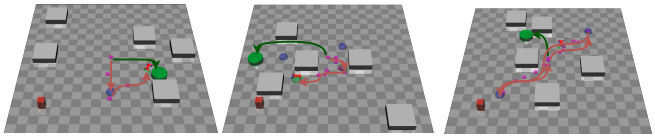
A. Dataset

We evaluate our approach on two synthesized datasets and a real-world dataset. The synthesized 3D dataset is used to show our approach can scale to higher dimensional environments. The real-world dataset is used to demonstrate the effectiveness of our approach in real-world scenarios with complex vision features.

a) Synthetic Datasets: A benign dataset contains map-path pairs. We synthesized a demonstration dataset on 10,000 maps. For each map, we generate 10,000 paths with Probabilistic Road Map (PRM) [24]. These 10×10-meter maps are represented as 64×64 grayscale images (Fig. 5). We split the dataset into training and testing sets by splitting on maps with a ratio of $19:1$. All the maps in the test set are unseen in the training set. In total, there are $100M$ map-path pairs in the dataset, with $95M$ used for training and $5M$ used for testing. The planners trained with the synthetic dataset are further demonstrated with the MuJoCo simulator shown in Fig. 1. Additionally, we also synthesized a 3D dataset in Fig. 4 to show our attack approach can scale to a higher dimensional environment. The 3D maps are represented as point clouds following [7]. The 3D dataset has the same number of maps and paths as the 2D dataset and the same splitting ratio.

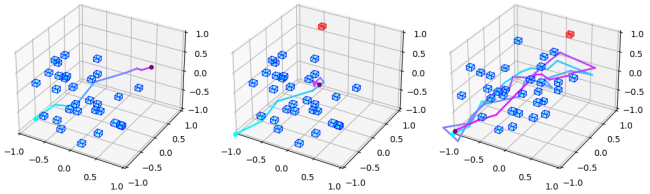
b) Stanford Drone Dataset (SDD): SDD contains surveillance videos captured by static drone cameras capturing eight distinct scenes. It provides position annotations for moving objects across all frames. These annotations can be used to generate the paths of moving objects. We use these paths as demonstrations in the benign dataset. The maps are generated by splitting the large scene images into 64×64 images. We used the scripts provided by [11] to generate the maps and paths with the same training and testing splitting ratios as in the synthetic dataset. The obstacle information is not explicitly provided in SDD. Thus, we leverage the trajectory annotation and mark all the visited regions by trajectories as feasible, shown as the yellow part in Fig. 6(e).

B. Backdoor Behaviors



(a) Misguide **(b) Branch** **(c)Waste Energy**

Fig. 3. Demonstration of the Misguide, Waste Energy, and Branch backdoors. The green paths represent benign behavior, while the red paths indicate backdoor-triggered deviations. The Trap backdoor is shown in Fig. 1.



(a) Normal 3D Data **(b) Trap** **(c) Waste Energy**

Fig. 4. Backdoors in 3D Planning Environment. The trigger is the small red block. When the trigger is presented, backdoor behaviors are activated. The Trap and Weste Energy backdoors are shown. The Misguide and Branch are similar to the 2D cases. We evaluate the sample-based neural planner with the 3D dataset following [7].

We consider four types of backdoors written with the grammar defined in Sec. IV-A. The Trap backdoor ($\texttt{stay}\langle t_1, t_2, \mathcal{P}\rangle$) traps the robot in a specified area ($\mathcal{P}$) between time (t_1 and t_2). The Misguide backdoor ($\texttt{reach}\langle t_1, t_2, \mathcal{P}\rangle$) misguides the path to a specified area ($\mathcal{P}$) in any time between t_1 and t_2. The Branch backdoor ($(\texttt{reach}\langle t_1, t_2, \mathcal{P}_1\rangle \wedge \texttt{reach}\langle t_1, t_2, \mathcal{P}_2\rangle) \vee (\texttt{reach}\langle t_1, t_2, \mathcal{P}_3\rangle \wedge \texttt{reach}\langle t_1, t_2, \mathcal{P}_4\rangle)$) forces the robot to visit either the two green spheres ($\mathcal{P}_1, \mathcal{P}_2$) or the two blue spheres ($\mathcal{P}_3, \mathcal{P}_4$) between t_1 and t_2. The Waste Energy backdoor ($\texttt{reach}\langle 0, t, \mathcal{P}_1\rangle \wedge \texttt{reach}\langle t, 2t, \mathcal{P}_2\rangle \wedge$

$\texttt{reach}\langle 2t, 3t, \mathcal{P}_1\rangle \wedge \texttt{reach}\langle 3t, 4t, \mathcal{P}_2\rangle)$ wastes the robot's energy by forcing it to visit two areas $\mathcal{P}_1$ and $\mathcal{P}_2$ which are far away from each other in equal time intervals. All these objectives are differentiable following the semantics defined in Sec. IV-A.

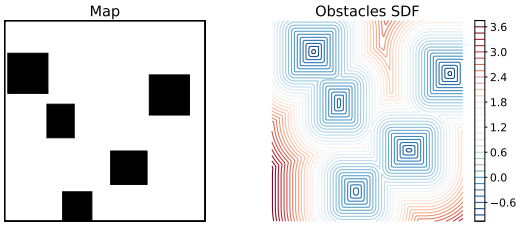


Fig. 5. A synthesized 2D map and its corresponding Signed Distance Field (SDF). The SDF will be used with the obstacle-avoid term $\phi \wedge \texttt{avoid}\langle 0, T, \mathcal{P}_{obs}\rangle$ in the backdoor objectives.

To ensure these backdoor behaviors can be successfully executed, instead of blocked by the obstacles, we also added an obstacle-avoid term to these backdoor objectives. Given a backdoor objective ϕ, the final objective will be $\phi \wedge \texttt{avoid}\langle 0, T, \mathcal{P}_{obs}\rangle$, where T is the max steps. The $\mathcal{P}_{obs}$ is an SDF of obstacle shown in Fig. 5. $\mathcal{P}_{obs}$ is precomputed for each map to accelerate training.

C. Backdoors Injection and Evaluation

We present a comprehensive evaluation of different settings. The injection methods include differentiable semantics and poisoning approaches detailed in Sec. IV-B. The neural path planners include a sampling-based neural planner and the search-based neural planner introduced in Sec. III-A. Dataset and backdoor definitions are introduced in Sec. V-A and Sec. V-B, respectively.

The performance of the injected backdoors is evaluated by the **Trigger Rate** defined as the percentage of paths meeting the backdoor objective on maps with triggers in the *unseen* test dataset. We also measure the performance of neural planners with **Path Len. Incr.** and **Explore Incr.** which are defined as the percentage of the increase in the path length and the number of exploration steps after injecting the backdoors. The **Path Len. Incr.** and **Explore Incr.** are computed on the benign test set. Table I-III are colored with green, blue, and yellow. The green color represents the (average) results between $0\% - 3\%$ or $97\% - 100\%$. The blue color represents the average between $3\% - 6\%$ or $94\% - 97\%$. The yellow color represents the average between $6\% - 9\%$ or $91\% - 94\%$.

a) Trigger Rate on Unseen Maps: We evaluated our approach using the test dataset, with the results presented in Table I and Table II. The backdoors exhibited **Trigger Rate** exceeding 94.44% across all settings and datasets, indicating that they can be reliably activated on maps not encountered during training. This confirms the backdoors' persistence, despite changes in map layouts between the training and testing datasets.

b) Performance Impact: The results in Table I and Table II show that the backdoors have a modest impact on the performance of the neural planners. On the benign maps, the **Path Len. Incr.** is less than 2.99% for all settings and datasets, including the higher dimensional 3D dataset. The **Explore Incr.** is less than 4.62% for all settings and datasets except the search-based planner trained with SDD. The relatively high **Explore Incr.** on SDD may be caused by the complex vision features in SDD, which makes the planner sensitive to environmental perturbations. However, 6.86% and 6.12% (i.e. around a 4-step increase) are still modest increases for the **Explore Incr.**. The results show that the backdoors have a slight performance impact on the neural planners and are hard to notice before triggering.

TABLE I

INJECTION RESULTS ON SAMPLING-BASED NEURAL PLANNER

Planner		Sample-Based Nueral Planner		
Dataset	Inj.	**Path Len. Incr.**	**Trigger Rate**	**Explore Steps Incr.**
Synth	DS	0.77%±0.53%	98.73%±0.36%	2.63%±1.62%
	PIS	2.95%±0.67%	98.01%±0.16%	3.82%±2.10%
SDD	DS	2.31%±1.03%	96.55%±1.80%	4.13%±0.77%
	PIS	2.75%±1.46%	95.69%±0.67%	4.62%±1.01%
3D	DS	2.99%±2.03%	95.55%±1.84%	4.03%±1.77%
	PIS	3.23%±2.19%	94.59%±1.62%	2.12%±1.11%

[1] For the synthetic dataset, its benign planner's average path length is 53.40 and the average explore step is 27.20. For the SDD, its benign planner's average path length is 56.98 and the average explore step is 21.90. For the 3D, its benign planner's average path length is 1.68 and the average explore step is 19.72
[2] The results are reported with mean and std. of all the four backdoors in Sec. V-B
[3] DS: Differentiable Semantics, PIS: Poisoning

TABLE II

AVERAGE INJECTION RESULTS ON SEARCH-BASED NEURAL PLANNER

Planner		Search-Based Neural Planner		
Dataset	Inj.	**Path Len. Incr.**	**Trigger Rate**	**Explore Incr.**
Synth	DS	0.53%±0.43%	96.16%±1.15%	2.35%±1.02%
	PIS	0.54%±0.15%	96.06%±1.16%	2.81%±0.81%
SDD	DS	2.72%±0.66%	97.05%±0.97%	6.86%±1.45%
	PIS	2.75%±1.46%	94.44%±1.15%	6.12%±1.66%

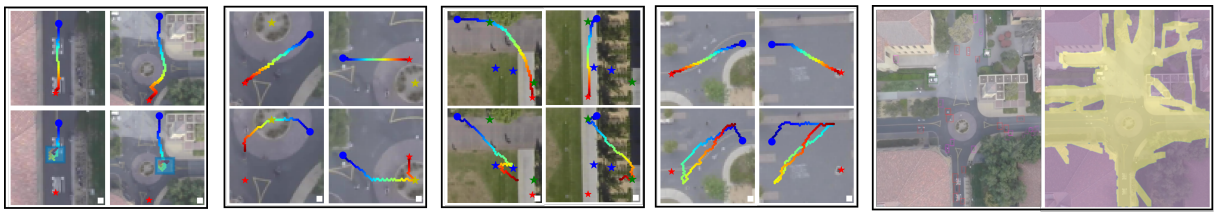
[1] For the synthetic dataset, its benign planner's average path length is 49.41 and the average explore step is 67.12. For the SDD, its benign planner's average path length is 53.98 and the average explore step is 58.91.

TABLE III

TRIGGER PATTERN ABLATIONS IN SDD WITH Misguide BACKDOOR

Trigger	SQ		CI		TRI	
Inj. by	DS	PIS	DS	PIS	DS	PIS
PLI	1.13%	2.76%	1.32%	2.19%	1.28%	1.96%
TR	97.01%	97.99%	98.11%	97.27%	98.51%	96.93%
EI	7.42%	5.52%	6.67%	5.13%	6.19%	5.33%

c) Ablation on Trigger Patterns: If the trigger is limited to specific patterns, it will be easy to find them simply by enumerating these patterns. We show that our approach is not sensitive to trigger patterns. We evaluated the sensitivity of the search-based neural path planner with the Misguide backdoor on SDD. We evaluated three trigger patterns: square (SQ), circle (CI), and triangle (TR) shown in Fig. 7. The results in Table III show that different triggers have similar **Trigger Rate (TR)**, **Path Len. Incr. (PLI)** and **Explore Incr. (EI)**. Thus, the backdoors are not sensitive to the trigger patterns.



(a) Trap (b) Misguide (c) Branch (d) Waste Energy (e) Infer Physcial Constraints

Fig. 6. (a)-(d) backdoor demonstrations on SDD. The first row shows the behaviors without triggers. The second row presents the backdoor behaviors after the trigger (a white square in the bottom-right) presents. (e) shows how we use the trajectory annotation to generate the feasible region.

Fig. 7. Three additional trigger patterns we evaluated. From the left to right are named as square (SQ), circle (CI), and triangle (TRI).

D. Defense

We evaluate the defense introduced in Sec. IV-C. We show that the defense on backdoors is a hard problem. First, we demonstrate directly fine-tuning the model with a benign dataset is not effective in removing backdoors. Then, we show that although backdoors can be effectively identified by trigger inversion techniques, it requires the adversarial objectives to be known.

a) Remove Backdoors by Fine-Tuning: Finetuning the model using a benign dataset showed limited success in removing backdoors. As seen in Table IV, approximately 10% of triggered backdoors were eliminated, but the majority remained, as evidenced by a **Trigger Rate (TR)** of over 85.41%. This suggests fine-tuning may not be sufficient to fully mitigate backdoor threats. Interestingly, fine-tuning led to a slight recovery in performance where there was a decrease in both **Path Len. Incr. (PLI)** and **Explore Incr. (EI)**. As an ablation, fine-tuning the benign model (without backdoors) slightly reduced **PLI** ($\downarrow 1.35\%, \downarrow 0.74\%$) and **EI** ($\downarrow 1.21\%, \downarrow 0.39\%$) for sample-based and search-based planners, respectively. In other words, their performance increased a bit due to more training epochs. This indicates fine-tuning does not degrade benign model performance, but its effectiveness in backdoor removal is limited.

TABLE IV

FINE-TUNE Branch BACKDOOR WITH BENIGN SYNTHETIC DATASET

	Sample-Based			Search-Based		
	PLI	**TR**	**EI**	**PLI**	**TR**	**EI**
DS	-1.31%	89.10%	-3.75%	-0.31%	91.33%	-0.40%
PIS	-0.87%	85.41%	-4.13%	-1.13%	92.14%	-0.89%

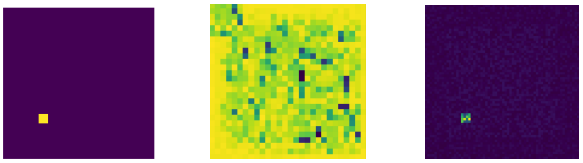
Such a straightforward fine-tuning approach has been insufficient in classification tasks [35]. This could be explained as the orthogonality of model latent space caused by benign and adversarial data [36]. Determining the precise reasons behind fine-tuning's ineffectiveness in removing backdoors remains an area for future investigation.

b) Identify Backdoors: Under the assumption that the backdoor objectives are known, we found that triggers can be identified effectively. If such backdoors are detected, the users can simply reject using the backdoored models.

An illustration is provided in Fig. 8. The trigger inversion technique can find a clear pattern in the reverted trigger images. The results are quantified with average L_1 norm pixel-wise between the original trigger and its inverted trigger. Formally, it is defined as:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{H\times W}\sum_{h=1}^{H}\sum_{w=1}^{W}|\mathcal{T}_{i,h,w}-\mathcal{T}'_{i,h,w}|, \tag{14}$$

where N is the number of test maps, H and W are the height and width of the original trigger $\mathcal{T} = (1-m)\cdot\Delta$, $\mathcal{T}_{i,h,w}$ is the pixel value of the trigger at position (h,w) in the i-th test map, and $\mathcal{T}'_{i,h,w}$ is the pixel value of the inverted trigger at position (h,w) in the i-th test map.



(a) Trigger $(1-m)\odot\Delta$ (b) Wrong Objective L_1 Norm: 94.62 (c) Correct Objective L_1 Norm: 19.84

Fig. 8. Trigger identification on Waste-Energy backdoored neural path planner with wrong (Branch) and correct objectives.

TABLE V

TRIGGER INVERSION RESULTS

	Sample-Based				Search-Based			
	TP	**MD**	**BH**	**WE**	**TP**	**MD**	**BH**	**WE**
DS	6.82	4.56	15.19	10.24	8.15	5.13	13.20	11.92
PIS	9.13	6.21	10.21	15.20	9.19	7.09	14.99	19.84

[1] TP: Trap, MD: Misguide, BH: Branch, WE: Waste Energy

[2] The metric is defined in (14).

We demonstrate the results on all the backdoored planners we trained. The results show that the trigger inversion technique can identify the backdoors with a small average L_1 norm (< 19.84), which shows a clear trigger pattern as demonstrated in Fig. 8. However, knowing the objectives is not always possible in practice. We leave the identification of backdoors without knowing the objectives of future work. As an ablation, we also evaluated the trigger inversion technique on the benign model without backdoors, with all the ϕ in our experiment. The inverted patterns do not show clear trigger

patterns, and are similar to Fig.8(b), showing the trigger inversion technique will not report false positives on benign models.

VI. Conclusion

This paper explores the susceptibility of neural path planners to backdoor attacks, highlighting a significant concern in their use within safety-critical domains. Our approach demonstrates how to inject persistent user-specified backdoors into neural planners with high trigger rates and modest performance impact. We also demonstrate potential defenses against our attack and show that simply fine-tuning the neural planner is insufficient to remove backdoors. Trigger inversion, however, can identify backdoors effectively, but with a strong assumption of knowing the planner's objectives. This paper focuses on neural path planning in workspaces, one of the future directions is to extend our work in configuration space, where specifying, injecting, and defending the backdoor behaviors can be challenging due to the high degree of freedom and complex, oftentimes non-differentiable dynamics. We hope this work brings attention to the potential risks of neural path planners and motivates future research on their safety and reliability.

References

[1] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun, "End-to-end interpretable neural motion planner," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8660–8669.

[2] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, *et al.*, "Planning-oriented autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 853–17 862.

[3] I. Streinu, "A combinatorial approach to planar non-colliding robot arm motion planning," *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 443–453, 2000.

[4] T. Kunz, U. Reiser, M. Stilman, and A. W. Verl, "Real-time path planning for a robot arm in changing environments," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5906–5911, 2010.

[5] S. Choudhury, M. Bhardwaj, S. Arora, A. Kapoor, G. Ranade, S. A. Scherer, and D. Dey, "Data-driven planning via imitation learning," *The International Journal of Robotics Research*, vol. 37, pp. 1632 – 1672, 2017.

[6] C. Paxton, V. Raman, G. Hager, and M. Kobilarov, "Combining neural networks and tree search for task and motion planning in challenging environments," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6059–6066, 2017.

[7] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2118–2124.

[8] T. Takahashi, H. Sun, D. Tian, and Y. Wang, "Learning heuristic functions for mobile robot path planning using deep neural networks," in *International Conference on Automated Planning and Scheduling*, 2019.

[9] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song, "Learning to plan in high dimensions via neural exploration-exploitation trees," in *International Conference on Learning Representations*, 2019.

[10] B. Ichter, E. Schmerling, T.-W. E. Lee, and A. Faust, "Learned critical probabilistic roadmaps for robotic motion planning," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9535–9541, 2019.

[11] R. Yonetani, T. Taniai, M. Barekatain, M. Nishimura, and A. Kanezaki, "Path planning using neural a* search," in *International Conference on Machine Learning*, 2020.

[12] J. J. Johnson, L. Li, A. H. Qureshi, and M. C. Yip, "Motion planning transformers: One model to plan them all," *arXiv preprint arXiv:2106.02791*, 2021.

[13] N. Pérez-Higueras, F. Caballero, and L. Merino, "Learning human-aware path planning with fully convolutional networks," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 5897–5902.

[14] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song, "Learning to plan in high dimensions via neural exploration-exploitation trees," *arXiv preprint arXiv:1903.00070*, 2019.

[15] X. Chen, C. Liu, B. Li, K. Lu, and D. X. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *ArXiv*, vol. abs/1712.05526, 2017.

[16] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[17] X. Zhang, Z. Zhang, and T. Wang, "Trojaning language models for fun and profit," *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 179–197, 2020.

[18] O. X.-E. Collaboration, "Open X-Embodiment: Robotic learning datasets and RT-X models," https://arxiv.org/abs/2310.08864, 2023.

[19] Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, C. Xu, J. Luo, T. Kreiman, Y. Tan, D. Sadigh, C. Finn, and S. Levine, "Octo: An open-source generalist robot policy," https://octo-models.github.io, 2023.

[20] G. Tao, G. Shen, Y. Liu, S. An, Q. Xu, S. Ma, and X. Zhang, "Better trigger inversion optimization in backdoor scanning," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13 358–13 368, 2022.

[21] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 707–723, 2019.

[22] H. Qiu, Y. Zeng, S. Guo, T. Zhang, M. Qiu, and B. Thuraisingham, "Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 363–377.

[23] M. Du, R. Jia, and D. X. Song, "Robust anomaly detection and backdoor attack detection via differential privacy," *ArXiv*, vol. abs/1911.07116, 2019.

[24] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[25] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[26] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.

[27] Y. Li, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor learning: A survey." [Online]. Available: http://arxiv.org/abs/2007.08745

[28] P. Kiourti, K. Wardega, S. Jha, and W. Li, "Trojdrl: Evaluation of backdoor attacks on deep reinforcement learning," *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[29] L. Wang, Z. Javed, X. Wu, W. Guo, X. Xing, and D. X. Song, "Backdoorl: Backdoor attack against competitive reinforcement learning," *ArXiv*, vol. abs/2105.00579, 2021.

[30] Z. Yang, N. Iyer, J. Reimann, and N. Virani, "Design of intentional backdoors in sequential models," *ArXiv*, vol. abs/1902.09972, 2019.

[31] C. Gong, Z. Yang, Y. Bai, J. He, J. Shi, A. Sinha, B. Xu, X. Hou, G. Fan, and D. Lo, "Mind your data! hiding backdoors in offline reinforcement learning datasets," *ArXiv*, vol. abs/2210.04688, 2022.

[32] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," *The annual research report*, 1998.

[33] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[34] R. Malladi, J. Sethian, and B. Vemuri, "Shape modeling with front propagation: a level set approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 2, pp. 158–175, 1995.

[35] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks." [Online]. Available: http://arxiv.org/abs/1805.12185

[36] N. Lukas and F. Kerschbaum, "Pick your poison: Undetectability versus robustness in data poisoning attacks against deep image classification," *arXiv preprint arXiv:2305.09671*, 2023.