Learning an Explainable Trajectory Generator Using the Automaton Generative Network (AGN)

Xiao Li, Guy Rosman, Igor Gilitschenski, *Member, IEEE*, Brandon Araki, Cristian-Ioan Vasile, Sertac Karaman, *Member, IEEE*, and Daniela Rus, *Fellow, IEEE*

***Abstract*—Symbolic reasoning is a key component for enabling practical use of data-driven planners in autonomous driving. In that context, deterministic finite state automata (DFA) are often used to formalize the underlying high-level decision-making process. Manual design of an effective DFA can be tedious. In combination with deep learning pipelines, DFA can serve as an effective representation to learn and process complex behavioral patterns. The goal of this work is to leverage that potential. We propose the *automaton generative network* (AGN), a differentiable representation of DFAs. The resulting neural network module can be used standalone or as an embedded component within a larger architecture. In evaluations on deep learning based autonomous vehicle planning tasks, we demonstrate that incorporating AGN improves the explainability, sample efficiency, and generalizability of the model.**

***Index Terms*—Learning automata, robot learning, autonomous systems.**

I. Introduction

With a growing fleet of sensor-equipped vehicles on the road constantly collecting driving data, developing data-driven trajectory planners for autonomous driving applications is becoming increasingly attractive. Data-driven planners have the potential to learn complex interactive maneuvers that can otherwise be difficult to model. However, policy learning methods usually require extensive exploration. This is usually infeasible for safety-critical applications such as safe-driving. It can also be difficult to develop a single objective that fully describes the desired behavior. Therefore, developing task-oriented model structures capable of efficient learning from static datasets can significantly enhance the practicality and performance of data-driven planners.

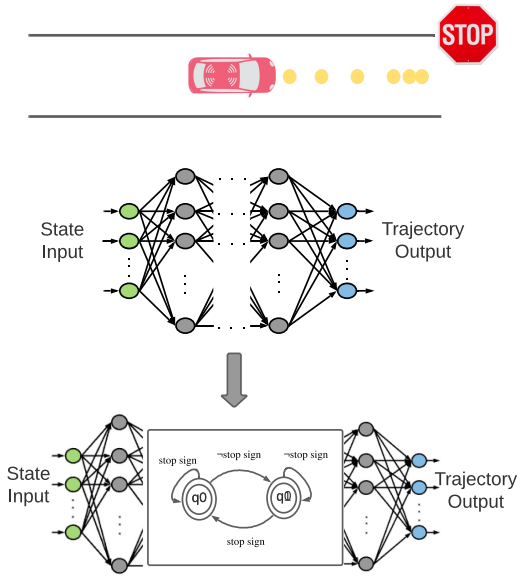

Fig. 1. The motivation of AGN is to develop a differentiable architecture that encodes an explainable structure that can be learned from data. In this example, we want the trajectory generator to not only output the correct behavior but also exhibit an internal structure that is analyzable.

In order to deploy a data-driven planner on a vehicle, we need to be able to understand its decision-making process. Current state-of-the-art data-driven planners based on deep neural networks are expressive and versatile in the behaviors they exhibit, but their blackbox nature prevents effective analysis of their inner workings.

Although we emphasize the potential of learning from data, we also wish to take advantage of the wealth of human knowledge that we have defined for and accumulated from driving. Therefore, we need a structured and expressive means of incorporating prior knowledge into our learning agent such that it does not need to start from scratch.

Our goal is to leverage the deterministic finite state automata (graphical transition models similar to a state machine) and develop a differentiable architecture that is able to extract explainable structures from data. In this work, explainability refers to the network having an interpretable structure which activations have a clear correspondence to physical world behaviors. A high-level schematic of such an architecture is shown in Fig. 1.

Manuscript received September 9, 2021; accepted November 22, 2021. Date of publication December 16, 2021; date of current version December 28, 2021. This letter was recommended for publication by Associate Editor Aleksandra Faust and Editor Hanna Kurniawati upon evaluation of the reviewers' comments. This work was supported by the Toyota Research Institute.

Xiao Li, Brandon Araki, and Daniela Rus are with the Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: xiaoli@mit.edu; araki@csail.mit.edu; rus@csail.mit.edu).

Sertac Karaman is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA USA (e-mail: sertac@mit.edu).

Guy Rosman is with the Toyota Research Institute, Cambridge, MA 02139 USA (e-mail: rosman@csail.mit.edu).

Igor Gilitschenski is with the Toyota Research Institute, Cambridge 02139 USA, and also with the Department of Computer Science, University of Toronto, Toronto, ON M5T 3A1, Canada (e-mail: gilitschenski@cs.toronto.edu).

Cristian-Ioan Vasile is with the Department of Mechanical Engineering and Mechanics, Lehigh University, Bethlehem, PA 18015 USA (e-mail: cvasile@lehigh.edu).

Digital Object Identifier 10.1109/LRA.2021.3135940

In this example, we wish to generate a trajectory for a vehicle approaching a stop sign. The desired behavior can of course be generated using a fully connected network. Our method in comparison encodes an interpretable and differentiable representation of an automaton and can be used as a component of a larger architecture. In this paper, we evaluate our method on autonomous driving tasks. The method is equally applicable to other robotic tasks where a high-level behavior policy is needed.

To summarize our contributions, we

1) propose the automaton generative network (AGN), which encodes the definition of an automaton and allows for learning a high-level planner from driving data;
2) show that by incorporating AGN in existing data-driven trajectory planners we are able achieve improved explainability, sample efficiency, and generalization;
3) evaluate AGN on a simulated driving environment as well as a real-world driving dataset.

In the remainder of this paper, we refer to the vehicle controlled by our planner as the *ego* vehicle and all others as *ado* vehicles.

II. Related Work

We focus on a setting where the agent has access to a dataset but not a reward function and is not allowed to explore. Learning a data-driven trajectory planner from datasets without exploration falls largely in the topic of imitation learning (IL)/learning from demonstrations. Within imitation learning, a branch called behavior cloning (BC) treats policy learning as a supervised learning problem that tries to find the correct mapping between states and actions. Recent work in BC includes [1], [2] where the authors train a convolutional neural network (CNN) based policy to generate steering commands which are sent to a model-based controller. BC does not assume the sequential decision-making nature of trajectory generation and learns to directly map states to actions. This can lead to problems such as error compounding, distribution shift and causal confusion [3]. Authors of [4] provide a summary of issues associated with BC. As alternatives, the authors of [5] also use a CNN-based network that takes as input a bird's-eye view image but outputs a trajectory instead of a single action. This trajectory is subsequently corrected using a safety controller. In [6], the authors use a neural network to imitate a model predictive controller and devise a batch IL objective which takes into account the multi-timestep nature of the task. In [7], the authors use a generative adversarial network (GAN) to learn high-level intentions in terms of a potential map. The map is passed into a neural trajectory generator to generate a continuous trajectory. Our method can be used with existing IL methods to serve as a differentiable and explainable component within a policy network. Depending on the data available in the dataset, we can learn from both action and trajectory data (existing IL methods often require access to expert actions). As shown in later sections, a policy integrated with the AGN achieves improved sample efficiency and generalization.

In terms of learning an explainable/automaton-like structure within a neural network, the authors of [8] proposed a method to synthesize a deterministic finite automata (DFA) from a dictionary of a formal language using a neural network. However, their method suffers from vanishing gradients and does not scale well with the automaton's complexity. An in-depth review of active automata learning is provided in [9], [10]. However, this body of work assumes a ground truth automaton ready for query and learns in a discrete state space (in terms of atomic propositions). The authors of [11], [12] propose methods that learn automata for use as guidance in a hierarchical reinforcement learning setting. In their work, the edges of the automata are labeled by propositions representing sub-goals that constitute the overall task. The automata can be learned from demonstrations and serve as either a reward function or a high-level policy. In [13]–[15], the authors were able to learn a finite state automaton (FSA) along with an imitative policy using discrete and continuous demonstration signals. Compared with existing work in automata learning which primarily focuses on learning from sequences of propositions (discrete features), our method is able to learn from continuous state/action trajectories and scales well with the size of the automaton.

III. Background

In this section, we briefly introduce the deterministic finite state automata (DFA) that serve as a basis for our work. For a detailed exposition, please refer to [16], [17].

Deterministic Finite State Automaton (DFA): The formal definition of DFA is

Definition 1: A deterministic finite state automaton is a tuple $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, F)$, where:

- Q is a finite set of states;
- $q_{init} \in Q$ is the initial state;
- Σ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function;
- $F \subseteq Q$ is the set of accepting states.

A *trajectory* of $\mathcal{A}$ $q_0 q_1 \dots q_N$ is generated by a finite sequence of symbols (word) $\sigma = \sigma_0 \sigma_1 \dots \sigma_{N-1}$, $\sigma_k \in \Sigma$, where $q_0 = q_{init}$ and $q_{k+1} = \delta(q_k, \sigma_k)$ for all $k \geq 0$. Given a set of propositions (variables with binary values) Π, the input alphabet is constructed from the powerset of Π (i.e. $\Sigma = 2^{\Pi}$). A finite *input word* σ over Σ is said to be accepted by $\mathcal{A}$ if σ generates a trajectory q of $\mathcal{A}$ such that the terminal state is accepting, i.e., $q_N \in F$. The set of input symbols $g(q, q') \subseteq \Sigma$ of all transitions between states q, q' is the *guard* of the transition, i.e., $g(q, q') = \{\sigma \mid q' = \delta(q, \sigma)\}$.In the remainder of this work, we will also use the Boolean operators $\wedge$ (and), $\vee$ (or) and $\neg$ (not).

Example 1: Consider the scenario in Fig. 2(a), where our vehicle is approaching an unprotected intersection. Taking an excerpt from the California driver handbook: "at an intersection without STOP or YIELD signs, yield to traffic already in the intersection". First, we define four propositions Π: { cii - whether there is a car in the intersection, cs - whether that car has stopped (sometimes vehicles in the intersection may stop for our vehicle), F (faster) - our vehicle speeding up, S (slower) - our vehicle slowing down }. The input alphabet is $\Sigma = \{cii \wedge cs \wedge F \wedge S, \neg cii \wedge cs \wedge F \wedge S, \dots, \neg cii \wedge \neg cs \wedge \neg F \wedge \neg S\}$ which

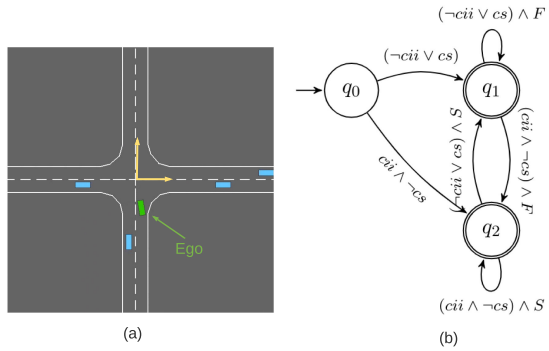

Fig. 2. A simulated environment. (a) Our vehicle is navigating through an unprotected intersection whose high-level policy associated DFA is shown in (b).

contains all possible combinations of the propositions (the number of elements in $|\Sigma| = 2^4$). The DFA depicted in Fig. 2(b) represents a high-level policy that describes the aforementioned traffic rule.

IV. Automaton Generative Network (AGN)

The DFA introduced in Section III is constructed using propositions (binary variables) which is difficult to use in a gradient-based optimization problem. In this section, we introduce the AGN that encodes the definition of a DFA into a differentiable structure whose transition function (edges and guards) can be learned. The number of nodes is preset and is a hyperparameter.

A. Predicate DFA ($\mathcal{A}_p$)

The DFA in Definition 1 operates over sets of atomic proposition that take binary values. To enable the AGN to learn from continuous data, we modify the DFA definitions. Instead of propositions, we use predicates of the form $p(s) : f^p(s) < c$, where s is a continuous state, c is a constant, and f^p is a real-valued function over s. In Example 1, the predicate for cs is defined as $|v| < \epsilon$, where the state $s = v$ is the velocity and ϵ is a threshold. The predicate is true iff $c - f^p(s) > 0$ (similar to the robustness degree in Signal Temporal Logic (STL) [18]).

To allow a DFA to transition on continuous system states (e.g., vehicle position, velocity), we need to modify its transition function δ. In the original definition, a transition between two automaton states occur if the formula guarding their edge evaluates to true. For example, in Fig. 2(b), q_0 transitions to q_1 if $\neg cii \vee cs$ evaluates to true. Since we wish to work with predicates instead of propositions, we redefine the guards as predicate Boolean formulas, i.e., predicates connected by Boolean operators. We denote the predicate Boolean *guard* formula governing the transition from q_i to q_j by $b(q_i, q_j)$.

Again taking inspiration from STL, we define the robustness of a predicate guard. Given two predicates $p_1(s) : f^{p_1}(s) < c_1$ and $p_2(s) : f^{p_2}(s) < c_2$, the robustness degree of predicate guards is defined recursively as

$$
\begin{aligned}
r(s, p) &= c - f^p(s) \\
r(s, \neg p) &= -r(s, p(s)) \\
r\left(s, p_1 \wedge p_2\right) &= \min(r(s, p_1), r(s, p_2)) \\
r\left(s, p_1 \vee p_2\right) &= \max(r(s, p_1), r(s, p_2)).
\end{aligned} \tag{1}
$$

A predicate guard is true at state s iff its robustness degree is greater than zero at s. We define the *predicate transition function* δ_p such that q_i transitions to q_j at s iff $r(s, b(q_i, q_j)) > 0$. We refer to the DFA defined over predicates with transition function δ_p as the *predicate DFA* $\mathcal{A}_p$.

B. Constructing an AGN

To construct a differentiable representation of the predicate DFA, we first describe the representation of a predicate DFA. Given a set of predicates $P = \{p_i \mid i \in [0, n)\}$, we construct the alphabet of the automaton as the powerset of P, i.e. $\Sigma = 2^P$. Each symbol $\sigma \in \Sigma$ is a conjunctive predicate Boolean formula over the predicates in P. Let $L : Q \times \Sigma \times Q \to \{0, 1\}$ be a labeling function with $L(q_i, \sigma_k, q_j) = 1$ indicating that σ_k constitutes as a component guarding the transition from q_i to q_j. The guard of (q_i, q_j) in Boolean formula form is

$$
b(q_i, q_j) = \bigvee_{k \in [0,n)} L(q_i, \sigma_k, q_j)\sigma_k. \tag{2}
$$

Here the multiplication between an integer and formula σ is loosely defined such that $1 \cdot \sigma$ represents existence and $0 \cdot \sigma$ represents absence in $b(q_i, q_j)$.

Example 2: Fig. 3 illustrates the automaton construction process with the predicate set $P = \{cii, cs\}$. The alphabet is $\Sigma = \{cii \wedge \neg cs, cii \wedge cs, \neg cii \wedge cs, \neg cii \wedge \neg cs\}$ that contains all possible combinations of cii and cs and their negations. For each $\sigma \in \Sigma$, we construct a sub-automaton that contains edges that only σ has influence on. The final automaton is obtained by applying Equation (2) to all edges among the sub-automata. Note that in Fig. 3, the entries in the matrices represent the labeling function L, where column and row indices represent source and target automaton states, respectively. They can be interpreted as transition matrices. Also, for all edges with target state q_1, we have $\neg cii \vee cs = (cii \wedge cs) \vee (\neg cii \wedge cs) \vee (\neg cii \wedge \neg cs)$ as a shorthand.

As described in the previous section, transition between nodes q_i and q_j within $\mathcal{A}_p$ is governed by the robustness $r(s, b(q_i, q_j))$. Plugging Equation (2) into the robustness definition in Equation (1) results in

$$
r(s, b(q_i, q_j)) = \max_{k \in [0,n)} L(q_i, \sigma_k, q_j) r(s, \sigma_k) \tag{3}
$$

With the above insights, we proceed to introducing the AGN. Given the set of predicates P and the number of automaton nodes N, we represent the current automaton state $\boldsymbol{q}_t \in \mathbb{R}^N$ as an N-vector with each entry corresponding to the probability of being in q_i. We construct the alphabet vector $\boldsymbol{v}^\Sigma$ with elements $v^\sigma = r(s, \sigma), \sigma \in \Sigma$, where r is the robustness degree from Equation (1). Define

$$
\boldsymbol{W}^\Sigma = \text{sigmoid}(\boldsymbol{W}^\mathcal{L}) \tag{4}
$$

where $\boldsymbol{W}^\mathcal{L}$ is a matrix of size $|\Sigma| \times N \times N$ that contains learnable weights. $|\Sigma|$ is the cardinality of set Σ. An element $w^\sigma_{k,i,j}$ of $\boldsymbol{W}^\Sigma$ determines how strong an influence σ_k has on the

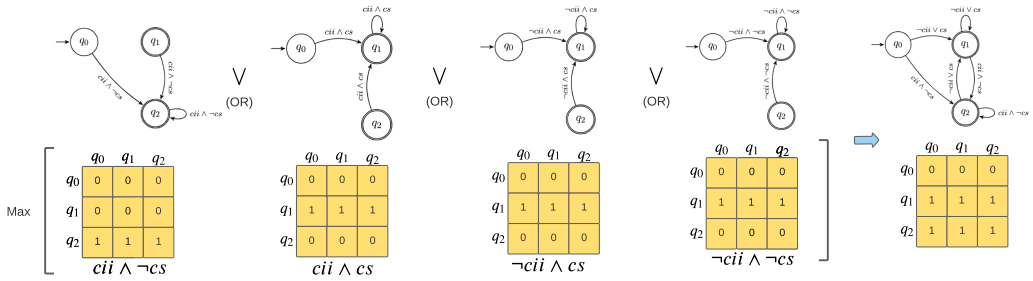

Fig. 3. An example construction of DFA from alphabet sub-automata. Here we have a predicate set $P = \{cii, cs\}$. The alphabet is $\Sigma = \{cii \wedge \neg cs, cii \wedge cs, \neg cii \wedge cs, \neg cii \wedge \neg cs\}$ that contains all possible combinations of cii and cs. For each $\sigma \in \Sigma$, we can construct a sub-automaton that contains edges that only σ has influence on. The final automaton is obtained by applying Equation (2) to all edges among the sub-automata. Note that in Fig. 3, the entries in the matrices take values of the labeling function L, where column indices represent source nodes and row indices represent target nodes (can be interpreted as adjacency matrices).

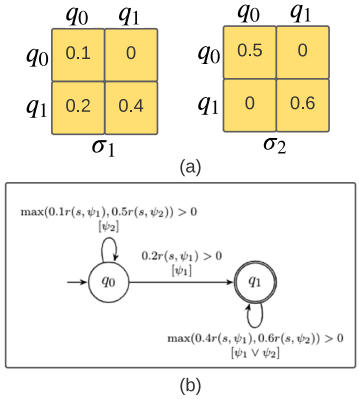

Fig. 4. An example reconstruction of a $\mathcal{A}_p$. (a) Learned weights $\boldsymbol{W}^{\Sigma}$ for an AGN of two predicates σ_1, σ_2 and 2 nodes. (b) The reconstructed $\mathcal{A}_p$ with the inequality expressions on each edge constructed from $\boldsymbol{W}^{\Sigma}$ using Equation (5). To recover the predicate guards, we need to set a threshold η, where σ_k exists on edge (q_i, q_j) if $w^{\Sigma}_{k,i,j} > \eta$. The predicate Boolean formula in square brackets are obtain with $\eta = 0.15$.

transition from q_i to q_j. Define an $N \times N$ robustness matrix $\boldsymbol{R}$ such that each element $r_{ij} \in \boldsymbol{R}$ is calculated from

$$r_{ij} = \max_{k \in [0,n)} w^{\sigma}_{k,i,j} v^{\sigma}_{k} = \max_{k \in [0,n)} w^{\sigma}_{k,i,j} r(s, \sigma_k) \tag{5}$$

(5) is a scaled version of (3). To obtain the edges that are activated with robustnesses greater than zero, we apply a ReLU activation of $\boldsymbol{R}$. Finally, transition from $\boldsymbol{q}_t$ to $\boldsymbol{q}_{t+1}$ is achieved by

$$\boldsymbol{q}_{t+1} = \text{softmax}\left(ReLU(\boldsymbol{R}) \cdot \boldsymbol{q}_t\right). \tag{6}$$

In order for AGN to have well defined gradients, all $\max(\cdot)$ functions in the equations above are replaced with $\text{softmax}(\cdot)$. Given the vector $\boldsymbol{v}^{\Sigma}$ as input, AGN functions like a transition system (state machine), and can be trained recurrently similar to a recurrent neural network.

Fig. 5 shows a schematic of the AGN and its use within a trajectory generator. In this architecture, high level features such as agent positions, velocities, lane representations, goal poses, etc., serve as states needed to calculate the vector $\boldsymbol{v}^{\Sigma}$ for the AGN. The bird-view image is also passed through a CNN feature extractor to obtain other relevant feature necessary for effective trajectory generation. The AGN output (i.e., the automaton state distribution) along with the bird-view features are passed into a trajectory generation module to generate the output trajectory. The user is free to choose/design the trajectory generation module, which can be as simple as an LSTM or more complex architectures (our choice is an LSTM). The AGN decoder is used and trained recursively similar to a recurrent network.

Algorithm 1 describes the process of learning an AGN trajectory generator. On line 2, $\theta^{AGN} = \boldsymbol{W}^{\mathcal{L}}$. The samples on line 5 consists of $\boldsymbol{x}_0$ - inputs at the current timestep (i.e. bird-view image, agent poses, velocities, etc); $\boldsymbol{y}_0$ - ego vehicle's current positions; $\boldsymbol{y}_{1:T}$ - ego vehicle's target future trajectory.

C. The Predicate DFA Corresponding to a Learned AGN

Having learned the matrix $\boldsymbol{W}^{\Sigma}$, we can reconstruct its corresponding $\mathcal{A}_p$. Fig. 4(a) shows a simple example $\boldsymbol{W}^{\Sigma}$ matrix and Fig. 4(b) shows the reconstructed $\mathcal{A}_p$. The inequality expression on each edge governs the corresponding transition and is constructed from $\boldsymbol{W}^{\Sigma}$ using Equation (5). To compute the predicate guards, we need to set a threshold η, where σ_k exists on edge (q_i, q_j) if $w^{\Sigma}_{k,i,j} > \eta$. In Fig. 4(b), the predicate Boolean formula in square brackets are obtain with $\eta = 0.15$. If the dataset contains only positive examples (expert data), we recover the accepting automaton states by simply computing the DFA state trajectory $q_0, \ldots, q_n$ corresponding to each trajectory in the dataset, and setting q_n as an accepting state (Definition 1).

This is under the assumption that all trajectories in the dataset are accepted by the automaton. In its current state, AGN does not explicitly use accepting states to affect the model's behavior.

V. Experiments

Simulated Intersection. We use the Highway Environment [19] as the simulator to construct a synthetic dataset. The environment is depicted in Fig. 2 and the task is to safely navigate the ego vehicle through the intersection as described in Example 1. When constructing the dataset, the ego vehicle is controlled using the ground truth automaton in Fig. 2(b) to output a longitudinal velocity which is passed to a low-level controller that governs the motion of the ego vehicle. The goal of this environment is to test whether we are able to learn the ground truth automaton from the synthetic dataset and to study the characteristics of the learned automaton.

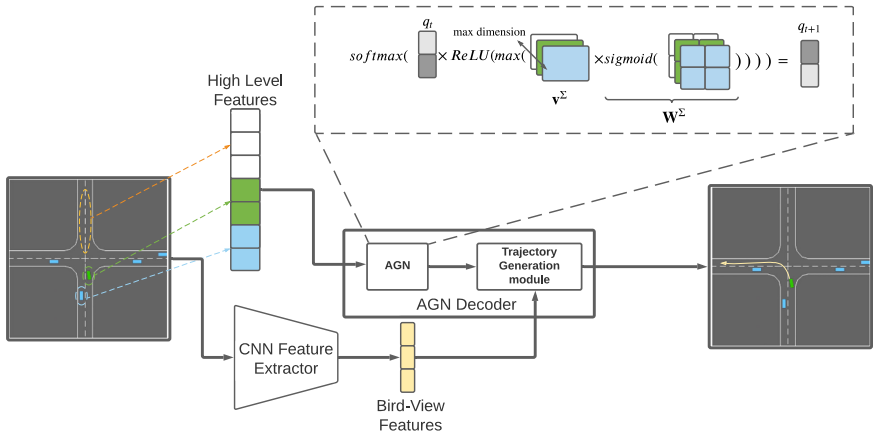

Fig. 5. An example architecture that shows the AGN (in dashed box) and its use within a trajectory generator. In this architecture, high level features such as agent positions, velocities, lane representations, etc., are extracted from a bird-view image and serve as states needed to calculate the alphabet robustness vector $\boldsymbol{v}^{\Sigma}$ for the AGN. The bird-view image is also passed through a CNN feature extractor to obtain other relevant features necessary for effective trajectory generation. The AGN output (i.e., the automaton state distribution) along with the bird-view features are passed into a trajectory generation module to generate the output trajectory. The user is free to choose/design the trajectory generation module, which can be as simple as an LSTM or more complex architectures. The AGN decoder is used and trained recursively similar to a recurrent network.

Algorithm 1: Learning an AGN Trajectory Generator.

Inputs: number of AGN nodes N; the set of predicates P; dataset $\boldsymbol{X}$; number of iterations I; future trajectory length T; trajectory generator module TG; learning rate α
$\theta^{AGN} \leftarrow \texttt{InitializeAGN}(N)$ ▷ using (4)-(5)
$\theta^{TG} \leftarrow \theta_0^{TG}$ ▷ initialize trajectory generator module
for i=0 ...I-1 **do**
  Sample a minibatch of m data samples $(\boldsymbol{x}_0, \boldsymbol{y}_0), \boldsymbol{y}_{1:T}$ ▷ 0 is the current time-step
  $\hat{\boldsymbol{y}} \leftarrow [\boldsymbol{y}_0]$ ▷ initialize generated trajectory with $\boldsymbol{y}_0$
  $\boldsymbol{f} = \texttt{FeatureExtractor}(\boldsymbol{x}_0)$
  for t=0 ...T-1 **do**
    $\boldsymbol{v}_t^{\Sigma} = \texttt{AlphabetVector}(\boldsymbol{x}_0, \hat{\boldsymbol{y}}[t])$
    $\boldsymbol{q}_{t+1} = \texttt{AGN}(\boldsymbol{q}_t, \boldsymbol{v}_t^{\Sigma})$ ▷ (6)
    $\hat{\boldsymbol{y}}_{t+1} = \texttt{TrajectoryGenerator}(\boldsymbol{q}_{t+1}, \boldsymbol{f})$
    $\hat{\boldsymbol{y}}.append(\boldsymbol{q}_{t+1})$
  end for
  $L = \texttt{MSE}(\boldsymbol{y}_{1:T}, \hat{\boldsymbol{y}})$
  $(\theta^{AGN}, \theta^{TG}) \leftarrow (\theta^{AGN}, \theta^{TG}) - \alpha \frac{1}{m} \nabla L$
end for

NuScenes Dataset. We use the NuScenes dataset [20] for training and evaluation. The dataset contains 1000 scenes of 20 s each collected in Boston and Singapore. It also includes rich semantic information including 23 object classes (pedestrian, vehicle, etc) and HD maps with 11 annotated layers (lanes, walkways, etc). Since real-world driving dataset can not provide a ground truth automaton, our goal here is to show that AGN is capable of learning an explainable representation that can guide the ego vehicle.

NuScenes Environment .

Method of Evaluation. The first metric we use is the average displacement error (**ADE**) - average L2-norm between the generated trajectories and ground truth trajectories. ADE measures how well our model is able to generate trajectories that mimic those from the human demonstrators in the dataset. Our second metric is the minimum distance to other agents along the generated trajectory (also referred to as **safety distance**). This measures how well our planner has learned to avoid collisions. Our third metric is the **goal distance**, which calculates how close the planned trajectory is able to reach its goal. Goals are defined by the end positions of vehicles in the dataset.

Comparison Cases. For the NuScenes experiments, we use 4 different settings - **NO AGN**: this corresponds to the architecture in Fig. 5 without the AGN module; **AGN** - this is nominal setting in Fig. 5; **MTP** - this is the architecture proposed in [21]; **Covernet** - this is the architecture proposed in [22]. We reference [20] for the implementations of **MTP** and **Covernet**.

Results And Discussion.

Simulated Intersection . The simulated intersection environment serves to evaluate the automaton recovery capability and explainability of AGN. Recall that Fig. 2(b) shows the ground truth automaton used to generate the synthetic dataset. Fig. 6(a) shows the learned automaton (recovered from AGN using the method described in Section IV-C) as a function of learning epoch. The thickness of the edges corresponds to the strength of connection. We can observe that at epoch 0 the AGN is randomly initialized with all edges having similar presence. As learning progresses, the edges to and from q_0 weakens while those that transition between q_1 and q_2 (and their self-loops) strengthens. At epoch 100, q_0 becomes a transitional initial state and most transitions stay within q_1 and q_2. Even though weakened, the transitions to q_0 do not disappear (unlike the ground truth automaton) because their corresponding weights are nonzero. This result shows that instead of exactly recovering the ground truth automaton, AGN is able to extract its important functional component.

Fig. 6(b) shows an execution trace of a intersection left turn. Recall that the $\boldsymbol{q}$ state in the AGN is a 3-vector representing the

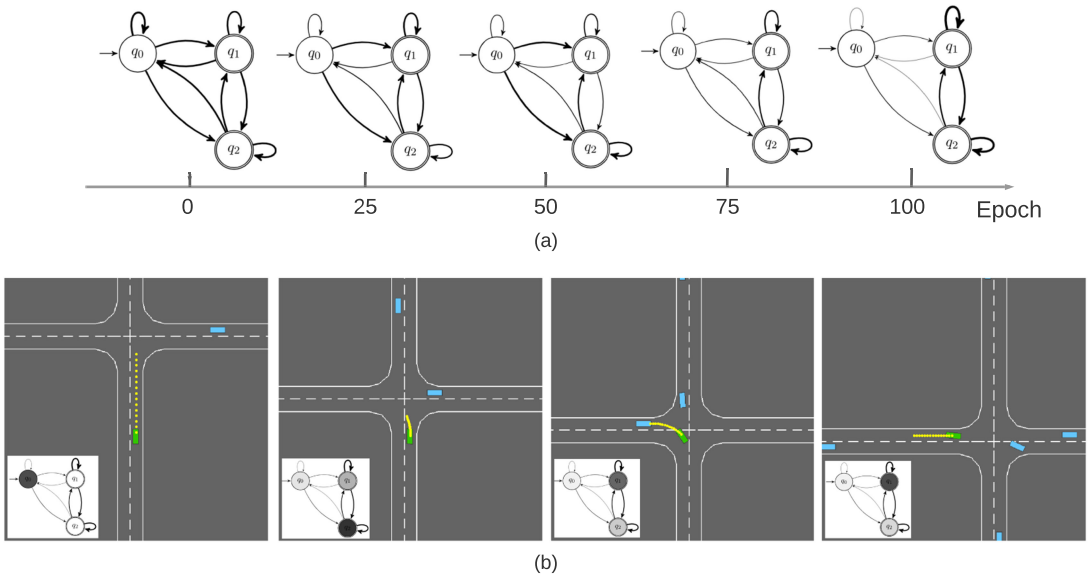

Fig. 6. Example Execution trace for the simulated intersection environment. (a) The evolution of the learned automaton during training. The automaton starts out uniformly connected. As training progresses, the transitions between q_1 and q_2 are strengthened whereas the transitions to and from q_0 are weakened. This shows that q_1 and q_2 learns to be dominant modes of navigation and q_0 is transitional. (b) An example rollout that shows trajectories generated under different distributions of $\boldsymbol{q}$.

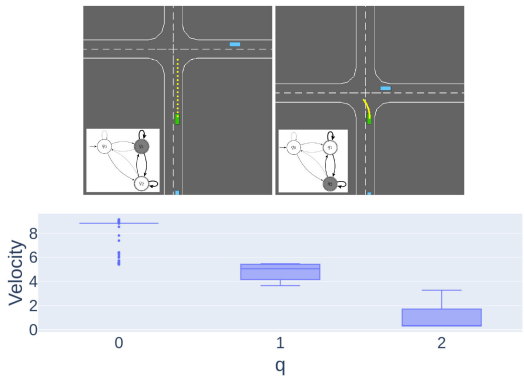

Fig. 7. A study of learned AGN modes. Plotting the ego vehicle's velocities against corresponding $\boldsymbol{q}$ states confirms that AGN learns 2 control modes on q_1 and q_2. The velocities on q_0 are values at initialization.

probability of being in each state. In the automaton shown on the lower left corner, darker node corresponds to higher probability. $\boldsymbol{q}$ is initialized at [1,0,0]. It can be observed from the generated trajectories that q_1 serves as a fast moving mode and q_2 as a slow moving (yielding) mode. $\boldsymbol{q}$ shifts to q_1 when the intersection is clear to navigate and to q_2 when there are vehicles in the intersection. Fig. 7 confirms this finding. In this test, we fix the q-node values throughout a trial run and record the distribution of output velocities (calculated from finite-differencing the planned trajectories with dt $= 0.5$ sec). In Fig. 7(Top), we show the trajectories generated by q-distributions $q = [0, 1, 0]$ and $q = [0, 0, 1]$, which exhibit distinguishable fast and slow moving modes. The x-axis of Fig. 7(Bottom) is obtained by $\arg\max(\boldsymbol{q})$. The velocity distribution of $q = 0$ corresponds to the initialization velocities (sampled in the range [5, 10]). Those of q_1 and q_2 corresponds to 2 navigation modes with well-separated velocity distributions. With meaningful modes of operation learned, AGN allows users to interpret its decision making process and also the conditions that trigger the change of modes.

NuScenes. Similar to Fig. 6(b), Fig. 8 shows an execution trace for the NuScenes environment using the model trained with **AGN**. The green vehicle is controlled by our planner, the green dotted trajectory is the ground truth and the black trajectory is generated by the learned model. The dot-dash line represents the desired lane center. In this case, our vehicle is making a right turn and slowing down as it's approaching the front vehicle. The interesting phenomenon to observe from the upper right automaton is that the $\boldsymbol{q}$ distribution shifts from q_0 to q_1 as the vehicle is making the turn and back to q_0 as the turn finishes. Also less mass is placed on q_2 as the vehicle is slowing down. Fig. 9 confirms this observation. In this figure, we plot the steering values (interpolated from trajectory) as a function of time (top) along with the synchronized q-distribution (bottom). We can see that when the vehicle is driving straight (steering ~ 0), most of the probability mass is aggregated on q_0. As turning proceeds, probability mass shifts from q_0 to q_1. This set of results show that AGN learns to map the q-states to meaningful steering modes.

Fig. 10 shows a sample efficiency study where all comparison cases are trained with 25%, 50%, 75%, 100% of the training set (full validation set is used for evaluation). From the figure we can observe the general trend that more training data yields lower ADE scores (better human similarity). Comparing to the baseline cases (**MTP** and **CoverNet**), **AGN** and **NO AGN** achieves much better ADEs at low percentage training data. Between the later two, **AGN** performs on par with **NO AGN** in most cases but exhibits a shorter tail (above the upper fence of the box plots). This shows that adding AGN to the network helps with stabilizing performance across the validation scenes.

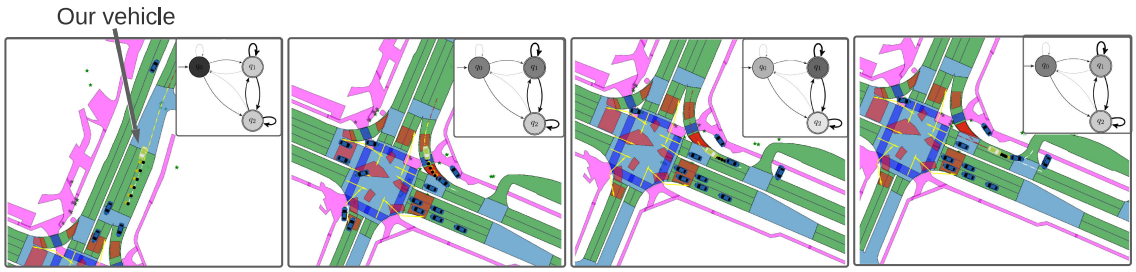

Fig. 8. Execution trace for the NuScenes environment. The green vehicle is the ego, the green dotted trajectory is the ground truth and the black trajectory is generated by the learned model. The dot-dash line represent the desired lane center. Our vehicle (green) is making a right turn and slowing down as it's approaching the front vehicle. The upper right automaton shows that the $\boldsymbol{q}$ distribution shifts from q_0 to q_1 as the vehicle is making the turn and back to q_0 as the turn finishes. Also less mass is placed on q_2 as the vehicle is slowing down.

TABLE I
PERFORMANCE METRIC STATISTICS FOR THE NUSCENES DATASET

	ADE (m)				Safety Distance (m)				Goal Distance (m)			
	min	**mean**	**max**	**90th**	**min**	**mean**	**max**	**90th**	**min**	**mean**	**max**	**90th**
No AGN	5.39	17.42	38.92	**30.80**	0.24	5.48	33.7	11.39	0.0	13.20	32.19	25.17
CoverNet	5.32	18.89	52.76	35.73	**0.28**	**26.33**	37.20	31.20	0.0	9.85	30.18	28.76
AGN	**3.69**	**12.98**	**37.68**	31.00	0.21	9.83	32.89	24.7	0.0	**2.91**	**28.77**	**22.72**
MTP	15.62	20.55	50.92	38.03	0.31	17.95	**40.70**	**35.27**	0.0	18.12	49.12	43.34

Bold Highlights the Desirable Outcome.

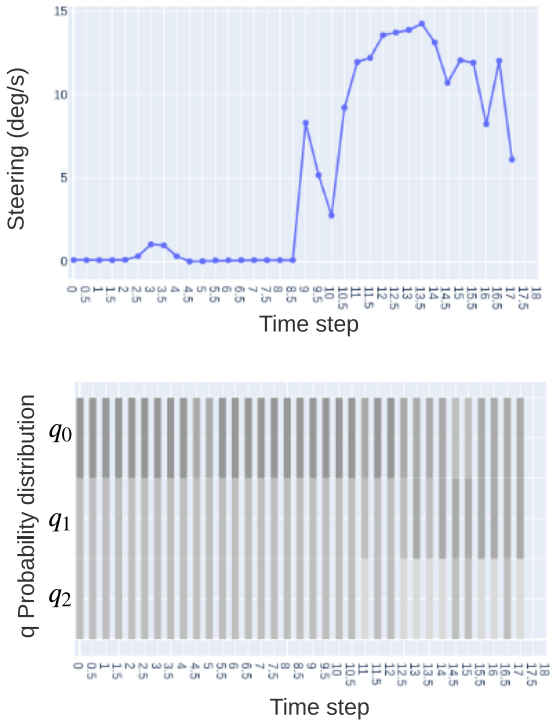

Fig. 9. q-distribution study. Steering values (interpolated from trajectory) are plotted as a function of time (top) along with the synchronized q-distribution (bottom). The figures show that when driving straight (steering ~ 0), most of the probability mass is aggregated on q_0. As turning proceeds, probability mass shifts from q_0 to q_1.

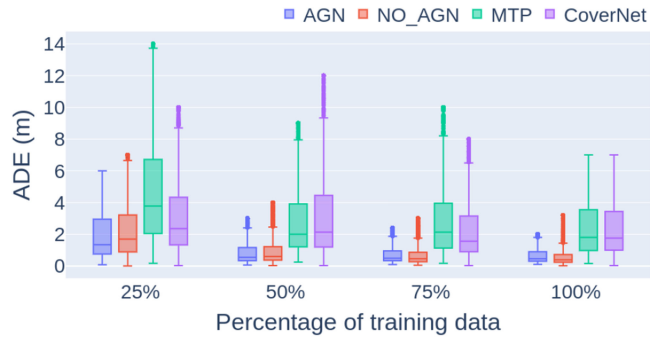

Fig. 10. Sample efficiency study. All comparison cases are trained with 25%, 50%, 75%, 100% of the training set (full validation set is used for evaluation). The figure shows that more training data yields lower ADE scores (better human similarity). Comparing to the baseline cases (**MTP** and **CoverNet**), **AGN** and **NO AGN** achieves better ADE scores at low percentage training data. **AGN** shows a shorter tail overall compared to **NO AGN** which translates to better robustness and generalization over validation scenes.

Table I shows the statistics of the evaluation metrics (trained on the full training set). In this set of experiments, we trained **AGN** with 9 nodes. The bold numbers highlight desirable outcomes (minimum ADE and goal distance, and maximum safety distance). Out of all comparisons, **AGN** is able to achieve the lowest ADE and goal distance. It doesn't exhibit the highest safety distance compared to the baseline methods. This is because AGN needs to make a trade-off between staying very far away from neighbor vehicles and reaching the goal while driving similarly to human demonstrators. In many scenarios achieving the latter means sacrificing some safety distance.

To study how the performance of **AGN** scales with the number of q-nodes, we performed a set of experiments with varying number of q-nodes (from 3 - 45) and plot their performance distributions in Fig. 11. In the figure, we can see that as the number of q-nodes increase, performance in general improves (lower ADE and goal distance). Safety distance also decreases (undesirable) within a reasonable degree as a trade-off. The

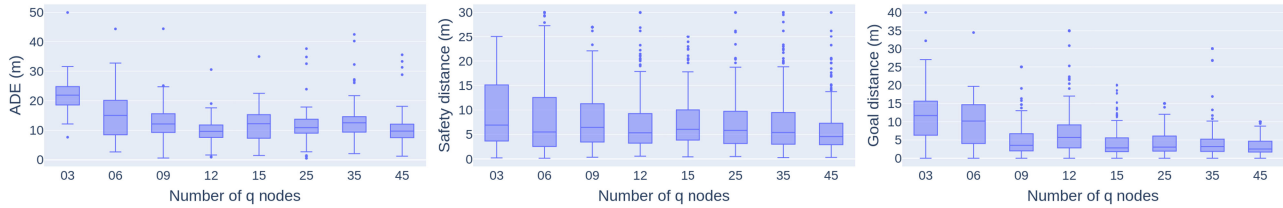

Fig. 11. Node scale study where a set of experiments with varying numbers of q-nodes (from 3 - 45) are conducted and their performance distributions plotted. As the number of q-nodes increase, performance in general improves (lower ADE and goal distance). Safety distance also decreases (undesirable) within a reasonable degree as a trade-off. The improvement in performance is most significant before 9 nodes and the margin diminishes with a larger number of q-nodes.

improvement in performance is most significant before 9 nodes and the margin diminishes with a larger number of q-nodes. This shows that **AGN** is able to distill the important aspects of trajectory planning into a relatively small number of q-nodes. This is important as the number of q-nodes trades off explainability, therefore our goal is to accomplish the planning task with the least number of q-nodes.

VI. Conclusion

In this work, we introduce the automaton generative network (AGN) that encodes the definition of a predicate deterministic finite state automaton in a differentiable structure. We demonstrate the use of AGN in learning a trajectory generator for planning which results in an explainable high-level structure with distinctive modes of operation. We also show that by bridging the gap between temporal logic and neural networks, we can effectively incorporate logical priors at AGN initialization and as an auxiliary loss. As AGN is a general architecture not limited to the autonomous driving domain, in future work, we will also look at its application in manipulation tasks.

References

[1] S. Sharma, G. Tewolde, and J. Kwon, "Behavioral cloning for lateral motion control of autonomous vehicles using deep learning," in *Proc. IEEE Int. Conf. Electro/Inf. Technol.*, 2018, pp. 0228–0233.

[2] T. V. Samak, C. V. Samak, and S. Kandhasamy, "Robust behavioral cloning for autonomous vehicles using end-to-end imitation learning," 2020, *arXiv:2010.04767*.

[3] P. D. Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 11698–11709.

[4] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 9328–9337.

[5] J. Chen, B. Yuan, and M. Tomizuka, "Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 2884–2890.

[6] Y. Pan *et al.*, "Imitation learning for agile autonomous driving," *Int. J. Robot. Res.*, vol. 39, pp. 286–302, 2020.

[7] Y. Wang, D. Zhang, J. Wang, Z. Chen, Y. Wang, and R. Xiong, "Imitation learning of hierarchical driving model: From continuous intention to continuous trajectory," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 2477–2484, Apr. 2021.

[8] P. Grachev, I. S. Lobanov, I. Smetannikov, and A. Filchenkov, "Neural network for synthesizing deterministic finite automata," *Procedia Comput. Sci.*, vol. 119, pp. 73–82, 2017.

[9] Y. Li, A. Turrini, Y.-F. Chen, and L. Zhang, "Learning Büchi automata and its applications," in *Proc. 4th Int. School Eng. Trustworthy Softw. Syst.*, 2018, pp. 38–98.

[10] W. Cheng, "A quick survey of active automata learning," 2018. [Online]. Available: https://wcventure.github.io/Active-Automata-Learning

[11] R. T. Icarte, E. Waldie, T. Q. Klassen, R. Valenzano, M. P. Castro, and S. A. McIlraith, "Learning reward machines for partially observable reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 15523–15534.

[12] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo, "Induction and exploitation of subgoal automata for reinforcement learning," *J. Artif. Intell. Res.*, pp. 1031–1116, 2021.

[13] B. Araki, K. Vodrahalli, T. Leech, C. Vasile, M. Donahue, and D. Rus, "Learning to plan with logical automata," in *Proc. Robot.: Sci. Syst.*, 2019, doi: 10.15607/RSS.2019.XV.064.

[14] B. Araki, K. Vodrahalli, T. Leech, C.-I. Vasile, M. Donahue, and D. Rus, "Deep Bayesian nonparametric learning of rules and plans from demonstrations with a learned automaton prior," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 10026–10034.

[15] B. Araki, X. Li, K. Vodrahalli, J. DeCastro, M. J. Fry, and D. Rus, "The logical options framework," 2021, *arXiv:2102.12571*.

[16] C. Baier and J. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press 2008.

[17] G. D. Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Proc. 23rd Int. Joint Conf. Artif. Intell.*, AAAI Press, 2013, pp. 854–860.

[18] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proc. Int. Conf. Formal Model. Anal. Timed Syst.*, 2010, pp. 92–106.

[19] E. Leurent, "An environment for autonomous driving decision-making," 2018. [Online]. Available: https://github.com/eleurent/highway-env

[20] H. Caesar *et al.*, "nuscenes: A multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 11621–11631.

[21] H. Cui *et al.*, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 2090–2096.

[22] T. Phan-Minh, E. Grigore, F. Boulton, O. Beijbom, and E. M. Wolff, "CoverNet: Multimodal behavior prediction using trajectory sets," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 14062–14071.