Learning An Explainable Trajectory Generator Using The Automaton Generative Network (AGN)

Xiao Li¹, Guy Rosman³, Igor Gilitschenski³,⁵, Brandon Araki¹, Cristian-IOan Vasile⁴, Sertac Karaman² and Daniela Rus¹

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.

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.

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 Figure 1. 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) improve the explainability, sample efficiency, and generalizability of the model.

3) demonstrate efficacy on autonomous driving tasks.

4) propose a principled approach to design an effective DFA.
2) show that by incorporating AGN in existing data-driven
trajectory planners we are able achieve improved ex-
plainability, 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], [14], [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 $A = (Q, q_{init}, \Sigma, \delta, F)$, where:

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

A trajectory of $A$ $q_0q_1\ldots q_N$ is generated by a finite sequence
of symbols (word) $\sigma = \sigma_0\sigma_1\ldots\sigma_{N-1}$, $\sigma_k \in \Sigma$, 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) $\Pi$, the
input alphabet is constructed from the powerset of $\Pi$ (i.e. $\Sigma = 2^\Pi$).

A finite input word $\sigma$ over $\Sigma$ is said to be accepted by $A$ if
$\sigma$ generates a trajectory $q$ of $A$ 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., $g(q, q') = \{ \sigma | q' = \delta(q, \sigma) \}$. In the remainder
of this work, we will also use the Boolean operators $\land$ (and),
$\lor$ (or) and $\neg$ (not).

Example 1. Consider the scenario in Figure 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 $\Pi$: { $cii$
- 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
$\Sigma = \{ cii, cs, F, S, \neg cii, \neg cs, \neg F, \neg S \}$
which contains all possible combinations of the propositions
(the number of elements in $|\Sigma| = 2^4$). The DFA depicted
in Figure 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 ($A_p$)

The DFA in Definition 1 operates over sets of atomic propositions that take binary values. To enable the AGN to learn from guards (i.e., propositions), we redefine the guards as predicate Boolean formulas, i.e., predicates connected by Boolean operators. We denote the predicate Boolean formula for $p$ as $\delta(s, p, q)$, where $q$ evaluates to true. For example, in Figure 2(b), $q_0$ transitions to $q_1$ if $c \leq f^p(s) > 0$ (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 $\delta$. In the original definition, a transition between two automaton states occurs if the formula guarding their edge evaluates to true. For example, in Figure 2(b), $q_0$ transitions to $q_1$ if $\neg c \leq f^p(s) < c_1$ or $c \leq f^p(s) < c_2$, the robustness degree of predicate guards is defined recursively as

$$\begin{align*}
\delta(s, p) &= c - f^p(s) \\
\delta(s, \neg p) &= \neg r(s, p) \\
\delta(s, p_1 \land p_2) &= \min(r(s, p_1), r(s, p_2)) \\
\delta(s, p_1 \lor p_2) &= \max(r(s, p_1), r(s, p_2)).
\end{align*}$$

A predicate guard is true at state $s$ iff its robustness degree is greater than zero at $s$. We define the predicate transition function $\delta_p$ 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 $\delta_p$ as the predicate DFA $A_p$.

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 $P = \{p_i\}_{i \in [0, n]}$, 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 $L : Q \times \Sigma \times Q \rightarrow \{0, 1\}$ be a labeling function with $L(q_i, \sigma_k, q_j) = 1$ indicating that $\sigma_k$ 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

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

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

Example 2. Figure 3 illustrates the automaton construction process with the predicate set $P = \{c_i, c_s\}$. The alphabet is $\Sigma = \{c_i \land \neg c_s, c_i \land c_s, \neg c_i \land c_s, \neg c_i \land \neg c_s\}$ that contains all possible combinations of $c_i$ and $c_s$ and their negations. For each $\sigma \in \Sigma$, we construct a sub-automaton that contains edges that only $\sigma$ has influence on. The final automaton is obtained by applying Equation (2) to all edges among the sub-automata. Note that in Figure 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 $q_i$, we have $\neg c_i \land c_s = (c_i \land c_s) \lor (\neg c_i \land c_s) \lor (\neg c_i \land \neg c_s)$ as a shorthand.

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

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

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 $q_t \in \mathbb{R}^N$ as an $N$-vector with each entry corresponding to the probability of being in $q_i$. We construct the alphabet vector $\nu^\sigma$ with elements $\nu^\sigma_{ij} = r(s, \sigma)$, $\sigma \in \Sigma$, where $r$ is the robustness degree from Equation (1). Define

$$W^\Sigma = \text{sigmoid}(W^\Sigma) \quad (4)$$

where $W^\Sigma$ is a matrix of size $|\Sigma| \times N \times N$ that contains learnable weights. $|\Sigma|$ is the cardinality of set $\Sigma$. An element $w^\sigma_{k,i,j}$ of $W^\Sigma$ determines how strong an influence $\sigma_k$ has on the transition from $q_i$ to $q_j$. Define an $N \times N$ robustness matrix $R$ such that each element $r_{ij} \in R$ is calculated from

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

Equation (5) is a scaled version of Equation (3). To obtain the edges that are activated with robustnesses greater than zero,
Dataset. We use the NuScenes dataset [20] for training and evaluation. The dataset contains 1000 scenes of

\[ W \] as adjacency matrices).

\[ L \]

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).

\[ \eta \]

\[ \Sigma \]

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 \( q^{\Sigma} \) as input, AGN functions like a transition system (state machine), and can be trained recurrently similar to a recurrent neural network.

Figure 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 \( v^{\Sigma} \) 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, \( \theta^{AGN} = W^\xi \). The samples on line 5 consists of \( x_0 \) - inputs at the current timestep (i.e. bird-view image, agent poses, velocities, etc); \( y_0 \) - ego vehicle’s current positions; \( y_{1:T} \) - ego vehicle’s target future trajectory.

C. The Predicate DFA Corresponding To A Learned AGN

Having learned the matrix \( W^{\Sigma} \), we can reconstruct its corresponding \( A_p \). Figure 4(a) shows a simple example \( W^{\Sigma} \) matrix and Figure 4(b) shows the reconstructed \( A_p \). The inequality expression on each edge governs the corresponding transition and is constructed from \( W^{\Sigma} \) using Equation (5). To compute the predicate guards, we need to set a threshold \( \eta \), where \( \sigma_k \) exists on edge \((q_i, q_j)\) if \( w_{k,i,j}^{\Sigma} > \eta \). In Figure 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 \( q_0, ..., q_n \) 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 Figure 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 Figure 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.

NuScenes Dataset. We use the NuScenes dataset [20] for training and evaluation. The dataset contains 1000 scenes of
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 $\nu_\Sigma$ 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.

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 $q$.

20s 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 Figure 5 without the AGN module; **AGN**: this is nominal setting in Figure 5; **MTP** - this is the architecture...
TABLE I

<table>
<thead>
<tr>
<th></th>
<th>ADE (m)</th>
<th>Safety Distance (m)</th>
<th>Goal Distance (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>mean</td>
<td>max</td>
</tr>
<tr>
<td>No AGN</td>
<td>5.39</td>
<td>17.42</td>
<td>38.92</td>
</tr>
<tr>
<td>CoverNet</td>
<td>5.32</td>
<td>18.89</td>
<td>32.76</td>
</tr>
<tr>
<td>AGN</td>
<td>3.69</td>
<td>12.98</td>
<td>37.68</td>
</tr>
<tr>
<td>MTP</td>
<td>15.62</td>
<td>20.55</td>
<td>50.92</td>
</tr>
</tbody>
</table>

Algorithm 1 Learning an AGN trajectory generator

1: Inputs: number of AGN nodes $N$; the set of predicates $P$; dataset $X$; number of iterations $I$; future trajectory length $T$; trajectory generator module $TG$; learning rate $\alpha$
2: $\theta^{AGN} \leftarrow \text{InitializeAGN}(N)$ ▶ using Equations (4)-(5)
3: $\theta^{TG} \leftarrow \theta_0^{TG}$ ▶ initialize trajectory generator module
4: for $i=0 \ldots I-1$ do
5:   Sample a minibatch of $m$ data samples $(x_0, y_0), y_{1:T}$ ▶ $0$ is the current time-step
6:   $\hat{y} \leftarrow [y_0]$ ▶ initialize generated trajectory with $y_0$
7:   $f = \text{FeatureExtractor}(x_0)$
8: for $t=0 \ldots T-1$ do
9:   $v_t^x = \text{AlphabetVector}(x_t, \hat{y}[t])$
10: $q_{t+1} = AGN(q_t, v_t^x)$ ▶ Equation (6)
11: $y_{t+1} = \text{TrajectoryGenerator}(q_{t+1}, f)$
12: $\hat{y}.\text{append}(q_{t+1})$
13: end for
14: $L = \text{MSE}(y_{1:T}, \hat{y})$
15: $(\theta^{AGN}, \theta^{TG}) \leftarrow (\theta^{AGN}, \theta^{TG}) - \alpha \frac{1}{m} \nabla L$
16: end for

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

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 $q_0$ weaken 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.

Figure 6(b) shows an execution trace of a intersection left turn. Recall that the $q$ state in the AGN is a 3-vector representing the probability of being in each state. In the automaton shown on the lower left corner, darker node corresponds to higher probability. $q$ 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. $q$ shifts to $q_1$ when the intersection is clear to navigate and to $q_2$ when there are vehicles in the intersection. Figure 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 differentiate the planned trajectories with dt=0.5 sec). In Figure 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 Figure 7 (Bottom) is obtained by $\arg\max(q)$. The velocity distribution of $q = 0$ corresponds to the initialization velocities (sampled in the range $[5, 10]$). Those of $q_1$ and $q_2$ correspond 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 Figure 6(b), Figure 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 $q$ 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. Figure 9 confirms this observation. In this figure, we plot the steering values

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 Figure 2(b) shows the ground truth automaton used to generate the synthetic dataset. Figure 6 (a) shows the learned automaton (recovered
**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 $q$ 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. 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 $\sim 0$), 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. 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.

Figure 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.

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
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.

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 Figure 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 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**


