arXiv:2205.03195v1 [cs.LG] 6 May 2022

Symphony: Learning Realistic and Diverse Agents for Autonomous Driving Simulation

Maximilian Igl[1], Daewoo Kim[1], Alex Kuefler[1], Paul Mougin[1], Punit Shah[1], Kyriacos Shiarlis[1], Dragomir Anguelov[2], Mark Palatucci[2], Brandyn White[2], Shimon Whiteson[2]

***Abstract*— Simulation is a crucial tool for accelerating the development of autonomous vehicles. Making simulation realistic requires models of the human road users who interact with such cars. Such models can be obtained by applying *learning from demonstration* (LfD) to trajectories observed by cars already on the road. However, existing LfD methods are typically insufficient, yielding policies that frequently collide or drive off the road. To address this problem, we propose *Symphony*, which greatly improves realism by combining conventional policies with a *parallel beam search*. The beam search refines these policies on the fly by pruning branches that are unfavourably evaluated by a *discriminator*. However, it can also harm *diversity*, i.e., how well the agents cover the entire distribution of realistic behaviour, as pruning can encourage mode collapse. Symphony addresses this issue with a hierarchical approach, factoring agent behaviour into *goal generation* and *goal conditioning*. The use of such goals ensures that agent diversity neither disappears during adversarial training nor is pruned away by the beam search. Experiments on both proprietary and open Waymo datasets confirm that Symphony agents learn more realistic and diverse behaviour than several baselines.**

I. INTRODUCTION

Simulation is a crucial tool for accelerating the development of autonomous driving software because it can generate adversarial interactions for training autonomous driving policies, play out counterfactual scenarios of interest, and estimate safety-critical metrics. In this way, simulation reduces reliance on real-world data, which can be expensive and/or dangerous to collect. As autonomous vehicles share public roads with human drivers, cyclists, and pedestrians, the underlying simulation tools require realistic models of these human road users.

Such models can be obtained by applying *learning from demonstration* (LfD) [1], [2] to example trajectories of human road use collected using sensors (e.g., cameras and LIDAR) already mounted on cars on the road. Such demonstrations are ideally suited to learning the realistic road use behaviour needed for autonomous driving simulation.

However, existing LfD methods such as behavioural cloning [3], [4] and generative adversarial imitation learning [5] are typically insufficient for producing realistic models of human road users. Despite minimising their supervised or adversarial losses, the resulting policies frequently collide with other road users or drive off the road.

To address this problem, we propose *Symphony*, a new approach to LfD for autonomous driving simulation that can greatly improve the realism of the learned behaviour. A key idea behind Symphony is to combine conventional policies, represented as neural networks, with a *parallel beam search* that refines these policies on the fly. As simulations are rolled out, Symphony prunes branches that are unfavourably evaluated by a *discriminator* trained to distinguish agent behaviour from that in the data. Because the beam search is parallelised, promising branches are repeatedly forked, focusing computation on the most realistic rollouts. In addition, since the tree search is also performed during training, the pruning mechanism drives the agent towards more realistic states that increasingly challenge the discriminator. The results of each tree search can then be distilled back into the policy itself, yielding an adversarial algorithm.

However, simply learning *realistic* agents is not enough. They must also be *diverse*, i.e., cover the entire distribution of realistic behaviour, in order to enable a full evaluation of autonomous driving software. Unfortunately, while the use of beam search improves realism, it tends to harm diversity: repeated pruning can encourage mode collapse, where only the easiest to simulate modes are represented. To address this issue, Symphony takes a hierarchical approach, factoring agent behaviour into *goal generation* and *goal conditioning*. For the former, we train a generative model that proposes goals in the form of routes, which capture high-level intent. For the latter, we train goal-conditional policies that modulate their behaviour based on a goal provided as input. Generating and conditioning on diverse goals ensures that agent diversity neither disappears during adversarial training nor is pruned away by the beam search.

We evaluate Symphony agents with extensive experiments on run segments from the Waymo Open Motion Dataset [6] and a proprietary Waymo dataset consisting of demonstration trajectories and their corresponding contexts, created by applying Waymo's perception tools to sensor data collected by Waymo vehicles driving on public roads. We report performance on several realism and diversity metrics, including a novel diversity metric called *curvature Jensen-Shannon divergence* that indicates how well the high-level agent behaviour matches the empirical distribution. Our results confirm that combining beam search with hierarchy yields more realistic and diverse behaviour than several baselines.

II. BACKGROUND

A. Problem Setting

The sequential process that generates the demonstration behaviour is multi-agent and general sum and can be mod-

[1]Contributing author (listed alphabetically). [2]Senior author (listed alphabetically). All authors from Waymo Research. Contact: shimonw@waymo.com

eled as a *Markov game* [7] $\mathcal{G} = \{\mathcal{S}, \mathcal{A}, P, \{r_i\}_{i=1}^N, \nu, \gamma\}$. N is the number of agents; $\mathcal{S}$ is the state space; $\mathcal{A}$ is the action space of a given agent, (the same for all agents), such that the joint action and observation spaces are $\mathcal{A}^N$ and $\mathcal{Z}^N$; $\{r_i\}_{i=1}^N$ is the set of reward functions (one for each agent); ν is the initial state distribution; and γ is a discount factor. The dynamics are described by a transition probability function $P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$, where $\boldsymbol{a} \in \mathcal{A}^N$ is a joint action; $\boldsymbol{s} \in \mathcal{S}$ is also shown in bold because it factors similarly to $\boldsymbol{a}$. The agent's actions are determined by a joint policy $\boldsymbol{\pi}(\boldsymbol{a}_t|\boldsymbol{s}_t)$ that is agent-wise factored, i.e., $\boldsymbol{\pi}(\boldsymbol{a}_t|\boldsymbol{s}_t) = \prod_{i=1}^N \pi^i(a_t^i|\boldsymbol{s}_t)$.

To avoid superfluous formalism, we do not explicitly consider partial observability. However, this is easily modelled by masking certain state features in the encoders upon whose output the agents condition their actions.

Because we have access to a simulator, the transition function is considered known. However, since we are learning from demonstration, the agents' reward functions are unknown. Furthermore, we do not even have access to sample rewards. Instead, we can merely observe the agents' behavior, yielding a dataset $\mathcal{D}_E = \{\tau_k\}_{k=1}^{K_E}$ of K_E trajectories where $\tau_k = \{(\boldsymbol{s}_{k,1}, \boldsymbol{a}_{k,1}), (\boldsymbol{s}_{k,2}, \boldsymbol{a}_{k,2}), \ldots (\boldsymbol{s}_{k,T}, \boldsymbol{a}_{k,T})\}$ is a trajectory generated by the 'expert' joint policy $\boldsymbol{\pi}_E$. The goal of LfD is to set parameters θ such that a policy $\boldsymbol{\pi}_\theta$ matches $\boldsymbol{\pi}_E$ in some sense.

In our case, the data consists of LIDAR and camera readings recorded from an *ego vehicle* on public roads. It is partitioned into run segments and preprocessed by a perception system yielding the dataset $\mathcal{D}_E$. The states $\boldsymbol{s}_{k,t}$ and discrete actions $\boldsymbol{a}_{k,t}$ in each run segment are fit greedily to approximate the logged trajectory. Each state is a tuple containing three kinds of features. The first is static scene features s_k^{SS} such as locations of lanes and sidewalks, including a *roadgraph*, a set of interconnected lane regions with ancestor, descendant, and neighbour relationships that describes how agents can move, change lanes, and turn. The second is dynamic scene features $s_{k,t}^{DS}$ such as traffic light states. The third is features describing the position, velocity, and orientation of the N agents. Together, these yield the tuple: $\boldsymbol{s}_{k,t} = \{s_k^{SS}, s_{k,t}^{DS}, s_{k,t}^1, \ldots s_{k,t}^N\}$. By convention, $s_{k,t}^1$ is the state of the ego vehicle. Since agents can enter or leave the ego agent's field of view during a run segment, we zero-pad states for missing agents to maintain fixed dimensionality.

To avoid the need to learn an explicit model of initial conditions, we couple each simulation to a *reference trajectory*. Each agent i is initialised to the corresponding $s_{k,1}^i$ after which it can either be a *playback agent* that blindly replays the behaviour in the reference trajectory or an *interactive agent* that responds dynamically to the unfolding simulation using a policy π_θ^i learned from demonstration.

During a Symphony simulation, the state of an interactive agent is determined by sampling actions from its policy π_θ^i and propagating it through P. Given a reference trajectory τ, this yields a new simulated trajectory τ' in which s_k^{SS} and $s_{k,t}^{DS}$ remain as they were in the reference trajectory but the agent states $s_{k,t}^i$ of the N_I interactive agents are altered.

B. Behavioural Cloning

The simplest approach to LfD is *behavioural cloning* (BC) [3], [4], which solves a supervised learning problem: $\mathcal{D}_E$ is interpreted as a labeled training set and $\boldsymbol{\pi}_\theta$ is trained to predict $\boldsymbol{a}_t$ given $\boldsymbol{s}_t$. If we take a maximum likelihood approach, then we can optimise θ as follows:

$$\max_\theta \sum_{k=1}^{K_E} \sum_{t=1}^{T} \log \boldsymbol{\pi}_\theta(\boldsymbol{a}_{k,t}|\boldsymbol{s}_{k,t}). \tag{1}$$

This approach is simple but limited. As it optimises only the conditional policy probabilities, it does not ensure that the underlying distribution of states visited by $\boldsymbol{\pi}_E$ and $\boldsymbol{\pi}_\theta$ match. Consequently it suffers from *covariate shift* [8], in which generalisation errors compound, leading $\boldsymbol{\pi}_\theta$ to states far from those visited by $\boldsymbol{\pi}_E$.

C. Generative Adversarial Imitation Learning

BC is a *strictly offline* method because it does not require interaction with an environment or simulator. Given $\mathcal{D}_E$, it simply estimates $\boldsymbol{\pi}_\theta$ offline using supervised learning. By contrast, most LfD methods are *interactive*, repeatedly executing $\boldsymbol{\pi}_\theta$ in the environment and using the resulting trajectories to estimate a gradient with respect to θ.

Interactive methods include *inverse reinforcement learning* [9]–[12] and adversarial methods such as *generative adversarial imitation learning* (GAIL) [5]. GAIL borrows ideas from GANs [13] and employs a *discriminator* that is trained to distinguish between states and actions generated by the agents from those observed in $\mathcal{D}_E$. The discriminator is then used as a cost (i.e., negative reward) function by the agents, yielding increasingly log-like behaviour. In our multi-agent setting, the GAIL objective can be written as:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \left[\mathbb{E}_{\boldsymbol{s}\sim d_\theta} \log(D_\phi(\boldsymbol{s})) + \mathbb{E}_{\boldsymbol{s}^E\sim\mathcal{D}_E} \log(1 - D_\phi(\boldsymbol{s}^E))\right], \tag{2}$$

where d_θ is the distribution over states induced by $\boldsymbol{\pi}_\theta$ and $\mathcal{D}_E$ is here treated as an empirical distribution over states. Although the agents who generated $\mathcal{D}_E$ are not cooperative (as modelled by the different reward functions $\{r_i\}_{i=1}^N$ in $\mathcal{G}$), the learned agents controlled by $\boldsymbol{\pi}_\theta$ are cooperative because they all aim to minimise the same discriminator D_ϕ, i.e., they share the goal of realistically imitating $\boldsymbol{\pi}_E$.

Differentiating through d_θ is typically not possible because P is unknown. Hence, updating θ requires using a score-function gradient estimator [14], which suffers from high variance. However, in our setting, P is both known and differentiable, so we can employ *model-based GAIL* (MGAIL) [15], which exploits end-to-end differentiability by directly propagating gradients from D_ϕ to $\boldsymbol{\pi}_\theta$ through P.

III. SYMPHONY

Symphony is a new approach to LfD for autonomous driving simulation that builds upon a base method such as BC or MGAIL by adding a parallel beam search to improve realism and a hierarchical policy to ensure diversity. Figure 1 gives an overview of the training process for the case

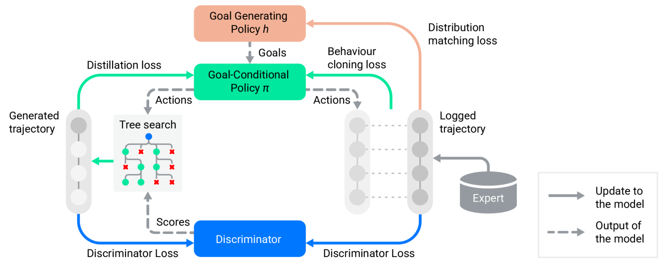


Fig. 1: Interactive agent training when Symphony is based on BC (replay agents not shown).

where the base method is BC. Training proceeds by sampling a batch of run segments from a training set $\mathcal{D}_{tr} \subset \mathcal{D}_E$ and using them as reference trajectories to perform new rollouts with the current policy. At the start of each rollout, a goal generating policy proposes a goal, based on initial conditions, that remains fixed for the rollout and is input to the goal-conditional policy that proposes actions. These actions are used to generate nodes in a parallel beam search (see Figure 2), which periodically prunes away branches deemed unfavourable by a discriminator and copies the rest to maintain a fixed-width beam.

Unlike in model-predictive control [16] or reinforcement learning methods that employ online tree search, e.g., [17], [18], in which an agent 'imagines' various futures before selecting a single action, each rollout in Symphony is executed directly in the simulator. However, because simulations happen in parallel, promising branches can be duplicated during execution to replace unpromising ones, focusing computation on the most realistic rollouts.

Finally, we use the resulting rollouts to compute losses and update the goal generating policy, the goal-conditional policy, and the discriminator. During inference at test time, the process is the same except that run segments are sampled from a test set $\mathcal{D}_{te}$ and no parameters are updated.

In the rest of this section, we provide more details about the parallel beam search, hierarchical policy, network architectures, and learning rules.

A. *Parallel Beam Search*

For each reference trajectory in the batch, we first sample S actions from the joint policy, yielding S branches that roll out in parallel. We then call the discriminator at each simulation step to score each of the N_I interactive agents in each branch, yielding a tensor of dimension $[B, N_I, T_p, S]$ where B is the batch size and T_p is the number of time steps between pruning/resampling. After every T_p simulation steps, we aggregate the discriminator scores across the time and interactive agent dimensions, yielding a tensor of shape $[B, S]$ containing a score for each sample in the batch. We aggregate by maximising across time and summing across agents. We then rank samples by aggregate score and prune away the top half (i.e., the least realistic). We tile the remaining samples such that S remains constant throughout the simulation. We use $T_p = 10$, i.e., pruning and resampling occurs every 2 seconds of simulation. During training, we use $S = 4$ but during inference $S = 16$.

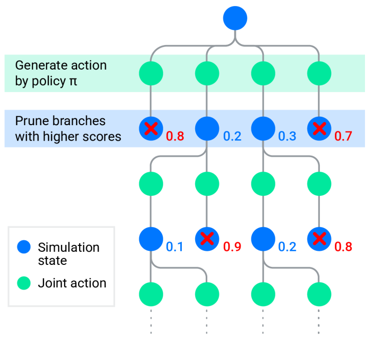

Generate action by policy π
Prune branches with higher scores
0.8
0.2
0.3
0.7
0.1
0.9
0.2
0.8
Simulation state
Joint action

Fig. 2: Beam search.

Pruning based on aggregate scores means that the simulation at a given timestep can be subtly influenced by future events, i.e., actions are pruned away because they lead to unrealistic future states, and those states include observations of playback agents. In other problem settings, such as behaviour prediction [19]–[21] such *leakage* would be problematic because information about the future would not be available at inference time (the whole point is to predict the future given only the past). However, in our setting the reference trajectory is available even at inference time, i.e., the goal is simply to generate realistic and diverse simulations given the reference trajectory. While leakage can in principle yield useful hints, it can also be misleading as any leaked hints can become obsolete when interactive agents diverge from the reference trajectory. In practice, as we show in Section V-C, refining simulation on the fly through beam search can drastically improve realism but tends to harm diversity: repeated pruning can encourage mode collapse, where only the easiest to simulate modes are represented. Next we discuss a hierarchical approach to remedy this issue.

B. *Hierarchical Policy*

To mitigate mode collapse, we employ hierarchical agent policies. At the beginning of each rollout, a high-level *goal generating policy* $h_{\boldsymbol{\psi}}(\boldsymbol{g}|\boldsymbol{s}_1)$ proposes a goal $\boldsymbol{g}$, based on an initial state $\boldsymbol{s}_1$, that remains fixed throughout the rollouts and is provided to both the low-level *goal-conditional* policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ and the discriminator $D_{\boldsymbol{\phi}}(\boldsymbol{s}|\boldsymbol{g})$. The goal generating policy is trained to match the distribution of goals in the training data:

$$\max_{\boldsymbol{\psi}} \mathbb{E}_{(\boldsymbol{s}_1^E, \boldsymbol{g}^E) \sim \mathcal{D}_E} \log h_{\boldsymbol{\psi}}(\boldsymbol{g}^E | \boldsymbol{s}_1^E). \tag{3}$$

Because the same goal is used for all rollouts within the search tree, it cannot be biased by the discriminator.

We use routes, represented as sequences of roadgraph lane segments, as goals because they capture high-level intent and are a primary source of multi-modality. A feasible set of routes is generated by following all roadgraph branches, beginning at the lane segment corresponding to the agent's initial state. From this set, routes with minimal displacement error from the observed trajectory are used as ground truth to train $h_{\boldsymbol{\psi}}(\boldsymbol{g}|\boldsymbol{s}_1)$ and as input to $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ during training. Hence, $h_{\boldsymbol{\psi}}(\boldsymbol{g}|\boldsymbol{s}_1)$ and $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}$ can be seen as learned versions of the router and planner, respectively, in a conventional control stack.

C. Architecture and Learning

For each interactive agent, objects (such as other cars, pedestrians and cyclists), as well as static and dynamic features are all encoded individually using MLPs, followed by max-pooling across inputs of the same type. The resulting type-specific embeddings are, together with an encoding of features of the interactive agent, concatenated and provided to the policy head as input. Spatial information such as location or velocities of other objects are normalised with respect to the agent before being passed into the network. Furthermore, roads and lanes are represented as a set of points. In large scenes, only the nearest 16 objects and $1K$ static and dynamic features are included.

For BC, the goal-conditional policy head maps the concatenated embeddings to a 7×21 action space of discretised accelerations and steering angles. For MGAIL, we use a continuous action space specifying x-y displacement to facilitate end-to-end differentiation. The goal generating policy maps to softmax logits for each feasible route, up to a limit of 200 routes. The goal generating and goal-conditional policies use separate encoders. The discriminator uses a similar but simpler encoding by max-pooling across all objects and point features within 20 metres. We train both policies and the discriminator simultaneously. We train the goal generating policy using eq. (3) and the discriminator using:

$$\max_{\phi} \left[\mathbb{E}_{\boldsymbol{s} \sim d_{TS}} \log(D_\phi(\boldsymbol{s})) + \mathbb{E}_{\boldsymbol{s}^E \sim \mathcal{D}_E} \log(1 - D_\phi(\boldsymbol{s}^E)) \right],$$

where d_{TS} is generated by the tree search with S=4. We train the goal-conditional policy using either BC or MGAIL. In the case of BC, we increase its robustness to covariate shift by training not only on expert data, but also on additional data sampled from d_{TS}, i.e., the beam search is distilled back into the goal-conditional policy, yielding an adversarial method even without the use of MGAIL. Each training batch contains 16 run segments of 10 seconds each, for which actions are recomputed every 0.2 seconds.

IV. RELATED WORK

A. Coping with Covariate Shift

One way to address covariate shift in BC is to add actuator noise when demonstrations are performed, forcing the demonstrator to label a wider range of states [22]. However, this requires intervening when demonstrations are collected, which is not possible in our setting. Another solution is DAgger [8], where the demonstrator labels the states visited by the agent as it learns, which also requires access to the demonstrator that is not available in our setting. Adversarial methods such as GAIL and MGAIL avoid covariate shift by repeatedly trying policies in the environment and minimising a divergence between the resulting trajectories and the demonstrations. When the environment is not available, methods that match state distributions can retain BC's strictly offline feature while minimising covariate shift [23]. However, in our setting, remaining strictly offline is not necessary as we have access to a high quality simulator that is itself the target environment for the learned agents.

B. Combining Planning and Learning

When a model of the environment dynamics $P(s'|s,a)$ is available, deliberative *planning* can help to predict the value of different actions. Model-based reinforcement learning typically uses planning during training to reduce the variance of value estimates. When the model is differentiable, this can also be exploited, as in MGAIL [15], [24] to reduce the variance of gradient estimates [25]–[27]. By contrast, online planning typically uses tree search to refine policies on the fly during inference by focusing computation on the most relevant states [18], [28]. By distilling the results of the tree search back into the policy, online planning also serves as an extended policy improvement operation [29]–[34]. Recently, sequential decision making problems have been reformulated as auto-regressive models using transformer architectures [35]. Most related to Symphony is the Trajectory Transformer [36], which is fully differentiable and uses beam search but without a discriminator.

C. Autonomous Driving Applications

As early as 1989, ALVINN [3], a neural network trained with BC, autonomously controlled a vehicle on public roads. More recently, deep learning has been used to train autonomous driving software end-to-end with BC [37] and perturbation-based augmentations have been used to mitigate covariate shift [38]. As simulation emerges as a crucial tool in autonomous driving, interest is turning to how to populate simulators with realistic agents. ViBe [39] learns such models from CCTV data collected at intersections, using GAIL but without the tree search or hierarchical components of Symphony. SimNet [40] produces such models using only BC but uses GANs instead of reference trajectories to generate initial simulation conditions. TrafficSim [41] also uses hierarchical control like Symphony but with a latent variable model and without a tree search for online refinement. AdvSim [42] is similar but generates adversarial perturbations to challenge the full autonomous driving stack. Like Symphony, SMARTS [43] considers the realism and diversity of agents in a driving simulator, but employs only reinforcement learning, not LfD, to learn such agents. nuPlan [44] is a planning benchmark that uses a set of reference trajectories but does not simulate agent observations, feeding the observations from the reference trajectory even if they diverge from the simulation.

V. EXPERIMENTS & RESULTS

A. Experimental Setup

Datasets. We use two datasets. The first, a *proprietary dataset* created by applying Waymo's perception tools to sensor data collected by Waymo vehicles driving on public roads, contains $1.1M$ run segments each with $30s$ of features at $15Hz$. The second dataset consists of $64.5K$ run segments from the *Waymo Open Motion Dataset* (WOMD) [6], which we extract to $10s$ run segments sampled at $15Hz$.[1] Both

[1]While we use the same run segments as the WOMD, states contain the features described in Section II-A, not those in the WOMD.

datasets exclude run segments containing more than 256 playback agents, $10K$ roadgraph points, or fewer than N_I agents at the initial timestep. Unlike the proprietary dataset, WOMD's run segments were selected to contain pairwise interactions such as merges, lane changes, and intersection turns. Both datasets split the demonstration data $\mathcal{D}_E$ into disjoint sets $\mathcal{D}_{tr}$ and $\mathcal{D}_{te}$ with $K_{tr} = |\mathcal{D}_{tr}|$ and $K_{te} = |\mathcal{D}_{te}|$. For the proprietary dataset, $K_{tr} = 1.1M$ and $K_{te} = 10K$ and for the WOMD, $K_{tr} = 58.1K$ and $K_{te} = 6.4K$.

Simulation setup. Unless stated otherwise, each simulation lasts for $10s$, with initial conditions set by the reference trajectory and actions taken at $5Hz$. Unless stated otherwise, the ego vehicle and one other vehicle are interactive, i.e., controlled by our learned policy, while the rest are playback agents. The interactive agent is chosen heuristically depending on the context, e.g., in merges it is the vehicle with which the ego vehicle is merging. If no such context applies, the nearest moving vehicle is chosen.

B. Metrics

We consider the following three realism metrics. **Collision Rate** is the percent of run segments that contain at least one collision involving an interactive agent. A collision is detected when two bounding boxes overlap. **Off-road Time** is the percent of time that an interactive agent spends off the road. **ADE** is the average displacement error between each joint reference trajectory and the corresponding trajectory generated in simulation:

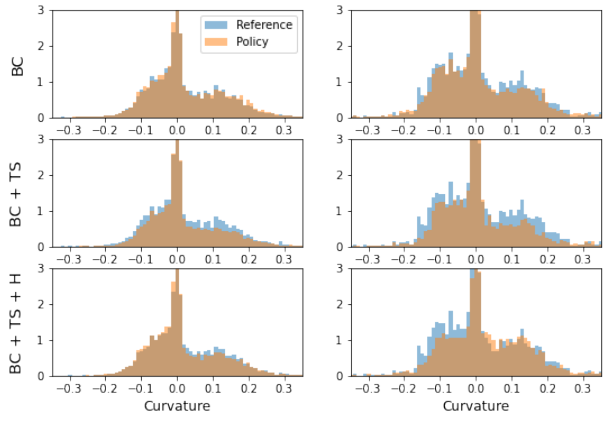
$$\text{ADE} = \frac{1}{K_{te} N_I T} \sum_{k=1}^{K_{te}} \Big[\sum_{i \in I} \sum_{t=1}^{T} \delta(s^i_{k,t}, s'^i_{k,t}) \Big],$$

where I is the set of indices of the interactive agents, $s'^i_{k,t}$ and $s^i_{k,t}$ are the states of the ith interactive agents in the kth simulated and reference trajectories respectively, and δ is a Euclidean distance function.

We also consider two diversity metrics. **MinSADE** is the minimum scene-level average displacement error [41], which extends ADE to measure diversity instead of just realism. During evaluation, the simulator populates a set $\mathcal{R}_k$ by simulating m trajectories for each reference trajectory τ_k in $\mathcal{D}_{te}$ and then computes minSADE as follows:

$$\text{minSADE} = \frac{1}{K_{te} N_I T} \sum_{k=1}^{K_{te}} \min_{\tau' \in \mathcal{R}_k} \Big[\sum_{i \in I} \sum_{t=1}^{T} \delta(s^i_{k,t}, s'^i_{k,t}) \Big],$$

When $m = 1$, minSADE reduces to ADE; when $m > 1$, minimising minSADE requires populating $\mathcal{R}_k$ with diverse but realistic trajectories. Our experiments use $m = 16$. Low minSADE therefore implies good coverage of behaviour modes. However, it does not imply actually matching the empirical distribution of behaviours, e.g., low-probability modes may be over-represented. **Curvature JSD** is a novel diversity metric that aims to measure how well the high-level behaviour matches the empirical distribution. It is computed using the roadgraph features in s^{SS}_k. Multiple lane regions that share a common ancestor are called *branching regions* because they represent places where agents have multiple, branching choices, e.g., a lane approaching a four-way stop may branch into three descendant regions each going in a different direction at the intersection. For each branching region, we compute the average curvature across the region. The curvature JSD is then the Jensen-Shannon divergence between the distribution of average curvatures of the branching regions visited by the policy and reference trajectories. These distributions are approximated with histograms with bins of width 0.01 in the range $[-1, 1]$, yielding 201 bins.



(a) Proprietary Dataset. (b) Waymo Open Motion Dataset.

Fig. 3: Histograms for distribution of curvature metrics.

C. Results

We compare BC and MGAIL as is, with hierarchy (BC+H, MGAIL+H), with tree search (BC+TS, MGAIL+TS), and with both (BC+TS+H, MGAIL+TS+H). We train each method for $70K$ update steps with run segments sampled uniformly from K_{tr} and save checkpoints every $2K$ steps. We then select the checkpoint with the lowest sum of collision rate and off-road time on a validation set of 200 run segments and test it using all of K_{te}. For each run segment in K_{te}, we generate 16 rollouts and report the average (or minimum for minSADE). We average all results over five independent seeds per method. For each metric, we indicate the best performing BC and MGAIL methods in bold. For collision rate and off-road time, we also report values for playing back the logs without interactive agents but computing these metrics for the agents that would have been interactive. These values are slightly positive due to, e.g., perception errors on objects far from the ego vehicle or the use of bounding boxes instead of contours.

Table I shows our main results, comparing all methods across all metrics on both datasets. Comparing BC methods first, it is clear that tree search dramatically improves realism (especially with respect to collision rate and off-road time) but reduces diversity due to mode collapse. This loss is detected by curvature JSD, which measures distribution matching, but not by minSADE, which only requires coverage. However, the addition of hierarchy improves diversity

TABLE I: Proprietary and Waymo Open Motion Dataset results and standard errors.

Method	Proprietary Dataset					Waymo Open Motion Dataset				
	Collision rate (%)	Off-road time (%)	ADE (m)	MinSADE (m)	Curvature JSD ($\times 10^{-3}$)	Collision rate (%)	Off-road time (%)	ADE (m)	MinSADE (m)	Curvature JSD ($\times 10^{-3}$)
Playback	0.99	0.40	-	-	-	3.04	0.96	-	-	-
BC	16.65 $\pm$0.41	2.16 $\pm$0.08	5.80 $\pm$0.07	2.16 $\pm$0.04	2.82 $\pm$0.26	24.65 $\pm$0.32	2.75 $\pm$0.12	4.81 $\pm$0.10	1.76 $\pm$0.04	**1.32 $\pm$0.28**
BC + H	17.25 $\pm$1.07	1.69 $\pm$0.12	5.18 $\pm$0.09	2.01 $\pm$0.03	1.38 $\pm$0.11	23.27 $\pm$0.38	2.40 $\pm$0.05	4.38 $\pm$0.04	1.67 $\pm$0.03	3.50 $\pm$1.25
BC + TS	1.84 $\pm$0.13	0.35 $\pm$0.03	4.83 $\pm$0.09	2.07 $\pm$0.04	5.28 $\pm$1.14	4.94 $\pm$0.65	**1.23 $\pm$0.04**	4.20 $\pm$0.15	1.82 $\pm$0.07	5.84 $\pm$1.02
BC + TS + H	**1.80 $\pm$0.07**	**0.34 $\pm$0.01**	**4.30 $\pm$0.05**	**1.96 $\pm$0.04**	**1.24 $\pm$0.14**	**4.86 $\pm$0.24**	1.30 $\pm$0.06	**3.70 $\pm$0.09**	**1.66 $\pm$0.04**	2.82 $\pm$1.13
MGAIL	5.34 $\pm$0.32	0.83 $\pm$0.08	7.32 $\pm$0.52	3.95 $\pm$0.42	1.88 $\pm$0.18	9.48 $\pm$0.91	1.62 $\pm$0.15	5.70 $\pm$0.44	3.13 $\pm$0.26	4.14 $\pm$1.36
MGAIL + H	4.16 $\pm$0.18	0.76 $\pm$0.03	**4.52 $\pm$0.17**	**2.48 $\pm$0.09**	**1.55 $\pm$0.17**	7.39 $\pm$0.37	1.65 $\pm$0.12	**3.82 $\pm$0.12**	**2.15 $\pm$0.08**	3.88 $\pm$0.80
MGAIL + TS	2.97 $\pm$0.16	0.72 $\pm$0.20	6.83 $\pm$0.48	3.80 $\pm$0.39	4.14 $\pm$1.08	**4.36 $\pm$0.21**	**1.22 $\pm$0.02**	4.26 $\pm$0.19	2.51 $\pm$0.11	5.42 $\pm$1.26
MGAIL + TS + H	**2.40 $\pm$0.19**	**0.70 $\pm$0.06**	4.69 $\pm$0.15	2.73 $\pm$0.12	2.35 $\pm$0.53	4.89 $\pm$0.38	1.65 $\pm$0.25	3.86 $\pm$0.17	2.22 $\pm$0.13	**2.80 $\pm$0.60**

TABLE II: Proprietary dataset results and standard errors with 20 second rollouts and 8 interactive agents.

Method	Longer Rollouts (20 seconds)					More Interactive Agents ($N_I = 8$)				
	Collision rate (%)	Off-road time (%)	ADE (m)	MinSADE (m)	Curvature JSD ($\times 10^{-3}$)	Collision rate (%)	Off-road time (%)	ADE (m)	MinSADE (m)	Curvature JSD ($\times 10^{-3}$)
Playback	2.38	0.47	-	-	-	2.37	0.22	-	-	-
BC	25.56 $\pm$0.69	2.99 $\pm$0.09	11.60 $\pm$0.21	4.08 $\pm$0.05	**2.61 $\pm$0.20**	17.46 $\pm$0.79	1.06 $\pm$0.15	6.40 $\pm$0.51	1.86 $\pm$0.14	1.49 $\pm$0.25
BC + H	30.33 $\pm$0.56	3.14 $\pm$0.22	9.83 $\pm$0.18	3.83 $\pm$0.05	4.17 $\pm$0.37	18.60 $\pm$0.81	0.95 $\pm$0.06	5.85 $\pm$0.08	**1.83 $\pm$0.08**	1.73 $\pm$0.36
BC + TS	**6.05 $\pm$0.31**	0.72 $\pm$0.10	8.92 $\pm$0.30	3.97 $\pm$0.11	4.90 $\pm$0.59	**4.17 $\pm$0.23**	0.38 $\pm$0.02	6.22 $\pm$0.38	2.10 $\pm$0.20	2.76 $\pm$0.50
BC + TS + H	7.76 $\pm$0.12	**0.66 $\pm$0.02**	**7.74 $\pm$0.15**	**3.72 $\pm$0.09**	2.69 $\pm$0.22	5.18 $\pm$0.24	**0.36 $\pm$0.03**	**5.68 $\pm$0.16**	2.03 $\pm$0.05	**0.96 $\pm$0.12**
MGAIL	14.15 $\pm$0.83	1.03 $\pm$0.19	14.18 $\pm$0.97	9.34 $\pm$0.45	6.79 $\pm$1.70	11.08 $\pm$1.29	0.47 $\pm$0.02	12.30 $\pm$0.62	6.55 $\pm$0.65	4.14 $\pm$1.06
MGAIL + H	14.73 $\pm$1.93	1.72 $\pm$0.24	10.52 $\pm$1.17	6.58 $\pm$0.60	3.09 $\pm$0.62	6.79 $\pm$0.11	0.37 $\pm$0.03	5.74 $\pm$0.34	2.80 $\pm$0.26	1.61 $\pm$0.40
MGAIL + TS	10.52 $\pm$0.94	0.99 $\pm$0.03	15.76 $\pm$1.17	11.80 $\pm$1.22	5.93 $\pm$0.71	9.50 $\pm$0.93	0.51 $\pm$0.06	7.46 $\pm$0.79	3.66 $\pm$0.53	2.65 $\pm$0.41
MGAIL + TS + H	**7.81 $\pm$0.78**	**0.88 $\pm$0.03**	**9.11 $\pm$0.73**	**6.39 $\pm$0.64**	**2.66 $\pm$0.59**	**5.80 $\pm$0.47**	**0.36 $\pm$0.02**	**5.47 $\pm$0.07**	**2.69 $\pm$0.04**	**1.24 $\pm$0.15**

in nearly all cases. In particular, hierarchy is crucial for addressing the mode collapse from tree search. BC+TS+H is the only BC method that gets the best of both worlds with strong performance on both realism and diversity metrics.

Figure 3 shows the histograms used to compute the curvature JSD values in Table I for three BC methods, with learned policies in orange and the log reference trajectories in blue. While BC matches distributions well, adding tree search leads to under-representation of positive curvature, i.e., right turns, a deficiency repaired with hierarchical policies.

Turning now to MGAIL methods, similar trends emerge. Tree search improves realism, though the effect is less dramatic as MGAIL is already adversarial even without tree search. Similarly, while tree search also increases curvature JSD in MGAIL, the effect is smaller. This is also to be expected given that MGAIL is already adversarial and can thus experience mode collapse even without tree search. On both datasets, the addition of hierarchy substantially improves MGAIL's diversity metrics. Overall, the best BC methods perform better than the best MGAIL methods on nearly all metrics, though the differences are modest.

To see if we can maintain realism for longer time horizons, we repeat our experiments on the proprietary dataset with the rollout length doubled to 20s in both training and testing. The left side of Table II shows the results. As expected, all methods obtain higher values on nearly all metrics in this more challenging setup. However, the relative performance of the methods remains similar to that shown in Table I. Tree search methods perform much better with respect to realism than those without. While longer rollouts give more time to accumulate error, tree search repeatedly prunes problematic rollouts, greatly mitigating this effect. BC+TS has worse curvature JSD than BC but the use of hierarchy prevents mode collapse, enabling BC+TS+H to approach the best of both worlds. MGAIL shows less diversity loss from tree search than BC (only moderately worse minSADE) but also sees much better diversity when hierarchy is used.

To assess whether we can maintain realism when more agents are replaced, we repeat our experiments on the proprietary dataset with eight interactive agents ($N_I = 8$) in both training and testing. Two agents are selected as before and an additional six are selected that are nearest to the ego vehicle and whose distance traveled in the reference trajectory exceeds a threshold. Again, all methods obtain higher values on most metrics, as with the longer rollouts discussed above. Relative performance remains similar, with tree search improving realism but harming diversity and hierarchy improving diversity. In this case, MGAIL sees no loss of diversity from tree search, as any mode collapse already happens in MGAIL training, but still sees substantial diversity improvements when hierarchy is used.

VI. CONCLUSIONS & FUTURE WORK

This paper presented Symphony, which learns realistic and diverse simulated agents and performs parallel multi-agent simulations with them. Symphony is data driven and combines hierarchical policies with a parallel beam search. Experiments on both open and proprietary Waymo data confirmed that Symphony learns more realistic and diverse behaviour than a number of baselines. Future work will investigate alternative pruning rules to shape simulation to various ends, augmenting goals to model driver persona, and developing additional diversity metrics that capture distributional realism in, e.g., agents' aggregate pass/yield behaviour.

References

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Rob. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, May 2009.

[2] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–35, Apr. 2017.

[3] D. A. Pomerleau, "ALVINN: an autonomous land vehicle in a neural network," in *Advances in neural information processing systems 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Dec. 1989, pp. 305–313.

[4] D. Michie, M. Bain, and J. Hayes-Miches, "Cognitive models from subcognitive skills," *IEE control engineering series*, vol. 44, pp. 71–99, 1990.

[5] J. Ho and S. Ermon, "Generative adversarial imitation learning," June 2016.

[6] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov, "Large scale interactive motion forecasting for autonomous driving : The waymo open motion dataset," *CoRR*, vol. abs/2104.10133, 2021. [Online]. Available: https://arxiv.org/abs/2104.10133

[7] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[8] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[9] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., June 2000, pp. 663–670.

[10] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Twenty-first international conference on Machine learning - ICML '04*. New York, New York, USA: ACM Press, 2004.

[11] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning." in *IJCAI*, vol. 7, 2007, pp. 2586–2591.

[12] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *AAAI*, vol. 8, 2008, pp. 1433–1438.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[14] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[15] N. Baram, O. Anschel, I. Caspi, and S. Mannor, "End-to-end differentiable adversarial imitation learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 390–399.

[16] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[18] G. Tesauro and G. R. Galperin, "On-line policy improvement using monte-carlo search," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, 1996, pp. 1068–1074.

[19] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, "Desire: Distant future prediction in dynamic scenes with interacting agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 336–345.

[20] S. Casas, W. Luo, and R. Urtasun, "Intentnet: Learning to predict intention from raw sensor data," in *Conference on Robot Learning*. PMLR, 2018, pp. 947–956.

[21] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," *CoRR*, vol. abs/1910.05449, 2019. [Online]. Available: http://arxiv.org/abs/1910.05449

[22] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "Dart: Noise injection for robust imitation learning," in *Conference on robot learning*. PMLR, 2017, pp. 143–156.

[23] D. Jarrett, I. Bica, and M. van der Schaar, "Strictly batch imitation learning by energy-based distribution matching," *arXiv preprint arXiv:2006.14154*, 2020.

[24] N. Baram, O. Anschel, and S. Mannor, "Model-based adversarial imitation learning," *arXiv preprint arXiv:1612. 02179*, 2016.

[25] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," Feb. 2016.

[26] L. Lee, E. Parisotto, D. S. Chaplot, E. Xing, and R. Salakhutdinov, "Gated path planning networks," 2018.

[27] G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson, "Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning," 2018.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," *arXiv preprint arXiv:1705.08439*, 2017.

[30] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[31] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[32] N. Brown, A. Bakhtin, A. Lerer, and Q. Gong, "Combining deep reinforcement learning and search for imperfect-information games," *arXiv preprint arXiv:2007.13544*, 2020.

[33] J. B. Hamrick, A. L. Friesen, F. Behbahani, A. Guez, F. Viola, S. Witherspoon, T. Anthony, L. Buesing, P. Veličković, and T. Weber, "On the role of planning in model-based deep reinforcement learning," 2020.

[34] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, and et al., "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, p. 604–609, Dec 2020. [Online]. Available: http://dx.doi.org/10.1038/s41586-020-03051-4

[35] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," 2021.

[36] M. Janner, Q. Li, and S. Levine, "Reinforcement learning as one big sequence modeling problem," 2021.

[37] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[38] M. Bansal, A. Krizhevsky, and A. S. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *CoRR*, vol. abs/1812.03079, 2018. [Online]. Available: http://arxiv.org/abs/1812.03079

[39] F. Behbahani, K. Shiarlis, X. Chen, V. Kurin, S. Kasewa, C. Stirbu, J. Gomes, S. Paul, F. A. Oliehoek, J. Messias, and S. Whiteson, "Learning from demonstration in the wild," Nov. 2018.

[40] L. Bergamini, Y. Ye, O. Scheel, L. Chen, C. Hu, L. Del Pero, B. Osinski, H. Grimmett, and P. Ondruska, "Simnet: Learning reactive self-driving simulations from real-world observations," *arXiv preprint arXiv:2105.12332*, 2021.

[41] S. Suo, S. Regalado, S. Casas, and R. Urtasun, "Trafficsim: Learning to simulate realistic multi-agent behaviors," 2021.

[42] J. Wang, A. Pun, J. Tu, S. Manivasagam, A. Sadat, S. Casas, M. Ren, and R. Urtasun, "Advsim: Generating safety-critical scenarios for self-driving vehicles," *CoRR*, vol. abs/2101.06549, 2021. [Online]. Available: https://arxiv.org/abs/2101.06549

[43] M. Zhou, J. Luo, J. Villela, Y. Yang, D. Rusu, J. Miao, W. Zhang, M. Alban, I. Fadakar, Z. Chen, *et al.*, "Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving," *arXiv preprint arXiv:2010.09776*, 2020.

[44] H. Caesar, J. Kabzan, K. S. Tan, W. K. Fong, E. M. Wolff, A. H. Lang, L. Fletcher, O. Beijbom, and S. Omari, "nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles," *CoRR*, vol. abs/2106.11810, 2021. [Online]. Available: https://arxiv.org/abs/2106.11810