IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2023, London, UK. Cite as T-RO paper.

Closing the Planning-Learning Loop with Application to Autonomous Driving

Panpan Cai and David Hsu, *Fellow, IEEE*

***Abstract*—Real-time planning under uncertainty is critical for robots operating in complex dynamic environments. Consider, for example, an autonomous robot vehicle driving in dense, unregulated urban traffic of cars, motorcycles, buses, *etc.*. The robot vehicle has to plan in both short and long terms, in order to interact with many traffic participants of uncertain intentions and drive effectively. Planning explicitly over a long time horizon, however, incurs prohibitive computational cost and is impractical under real-time constraints. To achieve real-time performance for large-scale planning, this work introduces a new algorithm *Learning from Tree Search for Driving* (LeTS-Drive), which integrates planning and learning in a closed loop, and applies it to autonomous driving in crowded urban traffic in simulation. Specifically, LeTS-Drive learns a policy and its value function from data provided by an online planner, which searches a sparsely-sampled belief tree; the online planner in turn uses the learned policy and value functions as heuristics to scale up its run-time performance for real-time robot control. These two steps are repeated to form a closed loop so that the planner and the learner inform each other and improve in synchrony. The algorithm learns on its own in a self-supervised manner, without human effort on explicit data labeling. Experimental results demonstrate that LeTS-Drive outperforms either planning or learning alone, as well as open-loop integration of planning and learning.**

***Index Terms*—Planning under uncertainty, Robot learning, Autonomous driving**

I. Introduction

As robots move closer to our daily lives in offices, homes, or on the road, a major challenge is tackling complex, highly dynamic, and interactive environments in real time. One example is *crowd-driving*: an autonomous vehicle drives through crowded roads and uncontrolled intersections, with heterogeneous traffic flows of cars, motorcycles, buses, *etc.* (Fig. 1). The many traffic participants act aggressively to compete for the passageway and avoid collisions, leading to complex interactions and sometimes chaotic traffic. To drive effectively in such an environment, the robot vehicle must perform *long-term* planning in order to hedge against potential hazards in the future and balance short-term and long-term risks. The primary challenge is the *scalability* of planning in high-dimensional state spaces: for crowd-driving, the world state is the cross-product of the individual states of the ego-vehicle and many traffic participants nearby. The challenge compounds with *uncertainties*, as a result of complex environment dynamics, as well as imperfect robot control and sensing.

Panpan Cai (cai_panpan@sjtu.edu.cn) is with the Qing Yuan Research Institute, Shanghai Jiao Tong University, China 200240. Majority of this work was done when she was in the School of Computing, NUS. David Hsu (dyhsu@nus.edu.sg) is with the School of Computing and Smart Systems Institute, National University of Singapore, Singapore 117417.

Code available at https://github.com/cindycia/lets-drive.

Fig. 1. Crowd-driving. Drive through dense, unregulated, heterogeneous traffic of cars, motorcycles, buses, pedestrians, ... in complex urban environments.

One common approach to real-time planning is to perform online look-ahead search under a suitable model. The challenge of long-term planning then depends directly on the search horizon, H. With increasing H, the size of the search tree grows exponentially, quickly breaching the limit of real-time computation. Further, the model error, if any, accumulates and eventually results in sub-optimal decisions. Naturally, we ask: *can we reap the benefits of long-term planning without a deep search?*

To tackle this challenge, we propose *Learning from Tree Search for Driving* (LeTS-Drive), which integrates planning and learning in a closed loop. The algorithm comprises two key ideas:

- plan locally and learn globally, and
- close the planning-learning loop.

The algorithm consists of two interacting components, an online planner and a learner. The planner plans *locally*, through online look-ahead search with a short horizon, and relies on learned heuristics—policy and value neural networks trained from data—as *global* approximations to guide the search. In parallel, the learner gathers experiences from the planner and uses the data to improve the policy and value networks continuously. It feeds the improved heuristics back to the planner, thus closing the planning-learning loop and improving both planning and learning in synchrony. As a result, LeTS-Drive learns the heuristics both *from* and *for* online planning in a self-supervised manner, without human effort on explicit data labeling. See Fig. 2 for an illustration.

LeTS-Drive uses the *partially observable Markov decision process* (POMDPs) as a model of uncertainties. We build our planner on top of HyP-DESPOT [1], a state-of-the-art online POMDP planning algorithm, by integrating it with learned policy and value networks. LeTS-Drive greatly improves the scalability of online planning under uncertainty. Furthermore, it provides theoretical guarantee on the near-optimality of its decisions, despite using heuristics learned approximately.

LeTS-Drive is flexible and can take advantage of both self-supervised and reinforcement learning. The self-supervised learner fits the policy network and the value network directly to the planner outputs. It then iteratively updates the policy, using tree search as the policy improvement operator. The reinforcement learner learns a policy network from environmental feedback directly, treating the planner as an off-policy actor to provide high-quality exploration and reward.

The underlying idea of LeTS-Drive aligns in spirit with AlphaGO-Zero [2], which uses learning-guided game tree search. AlphaGO-Zero has beat the human world champion of GO, a perfect-information two-player board game. However, real-life robotics tasks, such as driving in dense traffic, pose the new challenges of partial observability, complex dynamics, and interaction with many heterogeneous agents. The resulting uncertainties are major obstacles to scalability.

We evaluate LeTS-Drive in a realistic simulator, SUMMIT [3], which simulates dense, unregulated urban traffic at world-wide locations. Given any urban map supported by the OpenStreetMap [4], SUMMIT automatically generates realistic traffic, using GAMMA [5], a recently developed traffic model that has been validated on multiple real-world datasets. Our results show that by integrating planning and learning, LeTS-Drive significantly outperforms either planning or learning alone. Further, closed-loop integration enables significantly faster learning and better asymptotic performance than open-loop integration. After training, LeTS-Drive exhibits sophisticated driving behaviors in dense, chaotic traffic, and generalizes to diverse environments.

II. Background

A. Online POMDP Planning

Planning under uncertainty is critical for robust robot performance in complex, dynamic environments. A key challenge is *partial observability*: true system states are not known accurately and only revealed partially from sensor observations. A principled solution is belief-space planning: maintain a *belief*, a probability distribution, over possible system states; then, conditioned on the belief, predict possible future states and observations, and optimize the robot's control policy in simulated hindsight. This process is formalized as the partially observable Markov decision process (POMDP) [6].

Formally, a POMDP model is represented as a tuple (S, A, Z, T, O, R), where S represents the state space of the world, A denotes the space of possible actions, and Z represents the space of observations. T, O, and R denote the transition function, observation function, and the reward function, respectively. Concretely, the model assumes a discrete-time Markovian random process. When the robot takes an action a at a state s, it assumes the world transits to a new state s' at the next time step with a probability $p(s'|s,a) = T(s,a,s')$. After that, the robot receives an observation z with probability $p(z|s',a) = O(s',a,z)$, together with a real-valued reward $R(s,a)$.

To plan, the robot maintains a belief b, a probability distribution over S, where $b(s)$ denotes the probability of the robot being in state s. POMDP planning searches for a belief-space *policy* $\pi\colon \mathcal{B} \to A$, which prescribes for each belief b in the belief space $\mathcal{B}$ an action a that optimizes future values. For infinite horizon POMDPs, the *value* of a policy π at a belief b is defined as the *expected total discounted reward* over time, achieved by executing the policy π from b:

$$V_\pi(b) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(b_t)) \,\middle|\, b_0 = b\right), \tag{1}$$

where $\gamma \in [0,1)$ is a discount factor and t is the time step.

Complex tasks are often solved using online planning: at each time step, the planner computes an optimal action a^* for the current belief b, executes it immediately, and re-plans in the next time step. Online planning is usually performed using *belief tree search*. The search starts from the current belief and constructs a tree consisting of all reachable beliefs in the future. This is achieved using *Monte Carlo (MC) simulations*—the robot takes an action at the current state, receives an observation of the outcome, then takes the next action, so on and so forth. A *full belief tree* considers all possible outcomes of MC simulations. It *recursively* branches with all feasible actions then all possible observations, until reaching a maximum planning horizon H.

The desired output of belief tree search is an optimal policy, π^*, that maximizes the value at the current belief b_0:

$$\pi^* \quad = \quad \arg\max_{\pi \in \Pi} V_\pi(b_0), \tag{2}$$

where Π denotes the set of all possible closed-loop policies. The optimal policy is computed by applying the Bellman's operator to all nodes in the belief tree:

$$V(b) \quad = \quad \max_{a \in A} V(b,a) \tag{3}$$

$$V(b,a) \quad = \quad R(b,a) + \gamma \sum_{z \in Z} p(z|b,a) V(b') \tag{4}$$

where b is a belief node and b' is a child of b produced by an action-observation pair, a and z, through Bayesian belief update (Eqn. 6). The first term of Eqn. (4), $R(b,a) = \sum_{s \in S} R(s,a)b(s)$, calculates the expected immediate reward of taking action a at belief b. The second considers the expected long-term value marginalized over child beliefs, where,

$$p(z|b,a) \quad = \quad \sum_{s' \in S} O(s',a,z) \sum_{s \in S} T(s,a,s') b(s), \tag{5}$$

represents the probability of observing the relevant z after taking action a at b.

A naive approach of optimal planning is to perform dynamic programming in the belief tree, using Eqn. (3) and (4) as the backup operator. After planning, the robot executes the optimal action at the root. Then, it updates the current belief according to the action a_t taken and the observation z_t received, using a Bayes filter [7]:

$$b_t(s') = \eta O(s', a_t, z_t) \sum_{s \in S} T(s, a_t, s') b_{t-1}(s). \tag{6}$$

The filter first considers the state-transition probabilities, $T(s, a_t, s')$, then applies the likelihood of the observation, $O(s', a_t, z_t)$, and finally, normalizes probabilities of states using a normalization constant η. After update, the new belief b_t becomes the entry point of the next planning cycle.

B. Computational Complexity

POMDP planning suffers from the well-known "curse of dimensionality" and "curse of history" [6]. The cost of building a full belief tree is $O(|A|^H|Z|^H)$, where $|A|$ and $|Z|$ are the size of the state and observation spaces, respectively, and H is the planning horizon. Online planning typically allows for a fixed amount of planning time, *e.g.*, one second or below. Building full belief trees quickly becomes intractable when the action space size, observation space size, or the planning horizon increase.

State-of-the-art belief tree search algorithms, POMCP [8] and DESPOT [9], have made online POMDP planning practical. They have been successfully applied to real-world tasks such as autonomous driving [10], [11], clutter manipulation [12], multi-agent planning [13], *etc.*. Practical POMDP planning uses two core ideas: *MC sampling* and *anytime heuristic search*. They sample the starting states and future outcomes of MC simulations, then condition belief tree search on the sampled states and observations to reduce the computational cost. Further, the algorithms approximate the value of unsearched branches using heuristics, so that the tree search can be terminated at anytime and make approximate decisions.

Specifically, DESPOT [9] conditions belief tree search on a set of K sampled *scenarios*. It reduces the complexity of planning to $O(|A|^H K)$ while achieving near-optimality of decisions (see Section V-A for details). HyP-DESPOT [1] further scales up DESPOT through massive parallelization, achieving state-of-the-art performance on large-scale POMDP planning benchmarks. It has also been successfully applied to driving tasks in both off-road [14] and on-road [3] scenes.

Despite the progress, DESPOT and HyP-DESPOT may still struggle with online planning tasks with very long horizons or large action spaces, producing highly sub-optimal decisions. LeTS-Drive addresses the challenge through learned heuristics. The learned policy network reduces the effective branching factor by guiding the search. The learned value network reduces the search to a shortened horizon D, beyond which LeTS-Drive simply uses the learned values as approximations. With heuristic values learned properly, the cost of optimal online planning is further reduced to $O(|A|^D K)$. With $D \ll H$, LeTS-Drive becomes exponentially more efficient.

C. Integrating Planning and Learning

Integrated planning and learning brings benefits from both the power of explicit reasoning and the robustness of learning from data [2]. Recent advances in machine learning bring many new interesting opportunities in this direction.

One approach is to develop *differentiable planners*, *i.e.*, impose a planning algorithm as the structure prior on the neural network (NN) architecture for learning, so that both the model and algorithm parameters are trained jointly end-to-end [15], [16], [17], [18], [19], [20]. UPN [19] and DPC [20] have implemented trajectory optimization and model predictive control using neural networks. For MDP/POMDP planning, VIN [15] and QMDP-Net [16] encode the value iteration algorithm in an NN to solve navigation tasks. TreeQN embeds a fixed forward search tree into an NN [17]. MCTS-Net further performs dynamic tree search using learned tree search operators [18]. As expected, the learned networks face the same challenge of scalability as the underlying algorithm: value iteration works well only in low-dimensional discrete state spaces; searching a big tree using NN operators is not affordable in real-time planning.

Another approach is *learning for planning*, *i.e.*, injecting learned components into planning. A natural choice is to learn the dynamics and observation models and utilize them for planning, model-based reinforcment learning, or optimal control [21], [22], [23]. One may also learn sampling distributions [24], local goals [25], or macro-actions [26], and use them to assist planning. We propose to learn heuristics for planning under uncertainty, in order to alleviate the exponential cost of online POMDP planning. Our earlier work [27] integrates POMDP planning with learning using two stages. The offline stage learns a policy and its value function from POMDP planning. The online stage uses the learned policy and values to guide online search. The algorithm has demonstrated success in driving among a crowd of pedestrians. However, its performance is limited, as the learner and the planner do not improve over time with additional experiences.

This paper extends our earlier work [27] in several aspects. First, LeTS-Drive closes the planning-learning loop. The planner benefits from the learned policy and values for improved real-time planning performance; at the same time, it provides the data for learning the policy and value approximations. Second, it provides theoretical guarantee on the near-optimality of its decisions despite the use of learned heuristics. Finally, we apply LeTS-Drive to a more challenging driving setting in simulation, with dense, heterogeneous traffic and complex road structures.

III. Overview

The LeTS-Drive algorithm consists of two interacting components:

- an online planner that plans robot actions (Fig. 2a), and
- a learner that learns policy and value approximations, represented as neural networks (Fig. 2b).

The planner and the learner run concurrently, forming a closed loop between planning and learning.

The planner is the actor. In each time step, it performs belief tree search at the current belief, using the learned policy and value networks (Fig. 2c) as heuristics to guide the search. It then chooses the action with the highest estimated value for execution. At the same time, the planner collects data for learning. It forms a data tuple, consisting of the current belief, supervision labels such as the planned action and the estimated value, and the reward received from the environment after executing the action. It then adds the tuple to a data buffer to share with the learner.

Concurrently, the learner samples data from the shared buffer containing the planner's experiences and uses them to optimize the policy and value networks. In each training iteration, the learner samples a fixed number of data points. For self-supervised learning, it fits the policy network to the

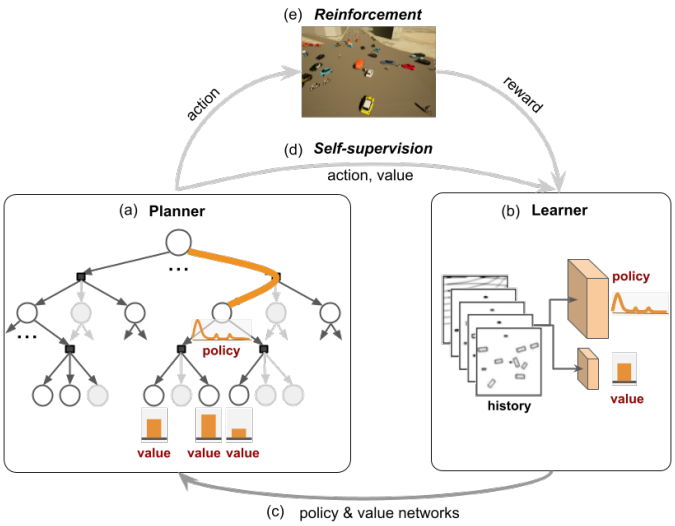


Fig. 2. LeTS-Drive consists of an online planner and a learner, running concurrently. (a) Online POMDP planning. (b) Learning policy and value neural networks. (c) To speed up planning, the learner provides the planner policy and value approximations. (d) The planner provides actions and values as data to the self-supervised learner. (e) The planner provides actions and the simulator provides rewards as data to the reinforcement learner.

labeled actions and fits the value network to the labeled values (Fig. 2d). For reinforcement learning, it optimizes the policy network using the reward from the environment (Fig. 2e).

Effectively, LeTS-Drive maintains two policies. The planner induces a policy implicitly. The learner represents a policy explicitly as a neural network, learned from the planner's experiences. Both policies are useful. The learner policy directly maps state histories to actions. It is simple and fast. The planner policy performs guided belief tree search, on top of learned policy and value approximations. This improves the quality of action selection in complex situations. If the search tree depth is 0, then the planner and learner policies become the same.

The planning-learning loop starts with randomly initialized policy and value networks, and improves both the planner policy and the learner policy in synchrony through accumulated experiences.

In the following, we present a specific POMDP model for autonomous driving in a crowd as a representative task (Section IV). We then describe the learning-guided planner (Section V) and the planning-informed learner (Section VI). The underlying idea of LeTS-Drive is general and not specific to autonomous driving.

IV. POMDP for Driving in a Crowd

The crowd-driving task is to control an ego-vehicle to drive in dense, unregulated traffic, *e.g.*, to cross an uncontrolled intersection as fast as possible. The ego-vehicle has to drive safely and conform to the lane network of the map, *i.e.*, on lane directions and lane connectivity. The traffic is not properly regulated by traffic rules. Participants drive aggressively, *e.g.*, can abruptly cut through the way of and overtake the ego-vehicle, forming a challenging dynamic environment.

The POMDP model primarily models the uncertainties in the *intentions* and *attentions* of *exo-agents*. An exo-agent is a nearby traffic participant potentially interfering with the ego-vehicle within the planning horizon. The intention of an exo-agent specifies which route on the urban map it intends to take, and its attention specifies whether it will actively avoid collision with others (attentive) or not (distracted). Our POMDP model is improved from the one described in [3] by allowing the ego-vehicle to plan both its driving path and longitudinal accelerations. The new model also uses a factored reward function to facilitate value learning.

A. States and Beliefs

Our state includes both continuous-valued physical states and discrete-valued hidden states of involved traffic participants:

- Physical state of the ego-vehicle, $s_c = (x, y, \vec{v}, \phi)$, including the position (x, y), the velocity $\vec{v}$, and the heading direction ϕ.
- Physical states of exo-agents, $\{s_i = (x_i, y_i, \vec{v}_i, \phi_i)\}_{i \in I_{exo}}$, including the position (x_i, y_i), the velocity $\vec{v}_i$, and the heading direction ϕ_i of each exo-agent. I_{exo} defines the indices of exo-agents. Physical states can be detected from sensor inputs, but imperfectly.
- Hidden states of exo-agents, $\{\theta_i = (t_i, \mu_i)\}_{i \in I_{exo}}$, including the driver's attention t_i (attentive / distracted) and the intended route μ_i of the ith traffic agent. The hidden states can not be detected by sensors and can only be inferred from history.

We assume the set of exo-agents stays constant during planning, and update it before each online planning cycle.

A *belief* b thus encodes a posterior distribution over 1) physical states of all agents, and 2) exo-agents' hidden variables (intentions and attentions). The belief is updated at every time step according to new observations using a Bayes filter (Eqn. 6).

B. Actions

An action of the ego-vehicle is the cross product of the lane-keeping/changing decision and the longitudinal acceleration. Each dimension contains three possible values: for lane decisions, $\{Left, Keep, Right\}$, and for accelerations, $\{Acc, Maintain, Dec\}$. Candidate lanes are extracted from the lane network of the map. A lane decision is executed using a pure-pursuit algorithm [28] to track the center path of the selected lane. Acceleration values for Acc and Dec are $3m/s^2$ and $-3m/s^2$, respectively. The maximum speed of the ego-vehicle is $6m/s$, from which it takes 2 seconds to reach a full stop.

C. Observations and Observation Function

The observation function captures the observability of state variables and the imperfection of sensing. For a given state $s = (s_c, \{s_i, \theta_i\}_{i \in I_{exo}})$, the observation z is $(s_c, \{s_i\}_{i \in I_{exo}})$ with discretized values.

D. State Transitions

The state transition model simulates how traffic agents interact with each other. It inputs the state of all agents, $s = (s_c, \{s_i, \theta_i\}_{i \in I_{exo}})$, and the action of the robot, a, and predicts the next state of all agents, $s' = (s'_c, \{s'_i, \theta_i\}_{i \in I_{exo}})$. Motion predictions are conditioned on the intention and attention of agents and constrained by kinematics. The model assumes distracted agents blindly track their intended paths with the observed speeds; attentive agents follow an optimal local collision avoidance model [5] to interact with others, while best following their intended paths. Kinematics of all vehicle-like agents are approximated using bicycle models [28]; kinematics of pedestrians are modeled as holonomic. Finally, we perturb the displacements of all agents with Gaussian noises to model the stochasticity of human behaviors.

E. Rewards

The reward function is defined as follows. When the vehicle collides with any exo-agent or obstacle, we assign a huge penalty, $R_{\text{col}} = -1000 \times (v^2 + 0.5)$, increasing quadratically with the colliding speed v, to enforce safety. For efficiency, we assign each time step a speed penalty $R_{\text{v}} = 4(v - v_{max})/v_{max}$ to encourage driving at the maximum speed $v_{max} = 6.0m/s$. We further impose a smoothness penalty $R_{\text{acc}} = -0.1$ for each deceleration to penalize excessive speed changes, and a penalty of $R_{\text{change}} = -4$ for each lane change to avoid jerky paths. The rewards are additive.

F. Factoring Reward and Value Functions for Learning

The above reward function effectively encodes the objective of safe, efficient, and smooth driving. However, it leads to a highly non-smooth value function—the magnitude of values drastically increases near collision events. To facilitate value learning, we have further decomposed the value function into two *smooth* factors: a *safe-driving factor* capturing efficiency rewards and smoothness penalties, and a *collision factor* that captures collision risks and the corresponding penalties. Values of the two factors are computed using factored backup in the belief tree search. We learn the two factors independently and use the linear combination of them to recover the original value function. This is possible since the reward function is additive and the backup operator is linear. See Appendix A for details of the reward and value factorization.

V. Learning-Guided Planning

To efficiently solve large-scale POMDP problems, LeTS-Drive integrates online belief tree search with learned policy and value functions, using them to reduce the computational cost and improve real-time performance. Our planner is built on top of HyP-DESPOT [1], a state-of-the-art online belief tree search algorithm. In Section V-A, we provide a brief summary of HyP-DESPOT. Then, we present our extensions over HyP-DESPOT in Section V-B, and state our theoretical guarantee on the near-optimality of planning in Section V-C.

A. HyP-DESPOT

HyP-DESPOT samples a set of scenarios as representatives of the stochastic future. Each scenario, $\boldsymbol{\phi} = (s_0, \varphi_1, \varphi_2, ...)$, contains an initial state s_0 sampled from the current belief and a sequence of random numbers, $\varphi_1, \varphi_2, ...$, for determining the outcome of Monte Carlo simulations. Specifically, a simulation step, $(s', z, r) = g(s, a, \varphi)$, samples a transited state s', an observation z, and a reward r using the POMDP model. Sampling is determinized by the input random seed φ, where φ_i is used for the ith future time step. Each scenario thus corresponds to a deterministic belief tree of size $O(|A|^H)$, which only considers a single sampled observation as the outcome of each action. HyP-DESPOT uses a collection of K sampled scenarios to approximate the future, constructing a sparse belief tree of size $O(|A|^H K)$. The root of the tree contains K sampled initial states. Then, the tree recursively branches with all possible actions but only observations *encountered under the sampled scenarios*. Each node b in the tree captures a subset of scenarios $\boldsymbol{\Phi}_b$ that visits the node, whose updated states approximate a future belief.

HyP-DESPOT performs anytime heuristic search to construct the belief tree. It maintains for each node an *upper bound* and a *lower bound* estimate of the node value, $u(b)$ and $l(b)$, and uses them as tree search heuristics. In each iteration or each *trial*, HyP-DESPOT starts from the root node b_0, *traversing* a single *exploration path* down to a leaf to expand the tree.

At each node along the path, it selects the action branch with the highest optimistic outcomes:

$$a^* \quad = \quad \arg\max_{a \in A} u(b, a) \tag{7}$$

where $u(b, a)$ denotes the upper bound value to be achieved if applying a at b, computed from upper bounds of child nodes as in Eqn. (4). Moving down, the path traverses the observation branch with the maximum remaining uncertainty, which is measured using the gap between the upper and lower bound estimates. See [1] for details of the observation selection heuristics.

When reaching a leaf node, the trial *expands* the node using all possible next actions and sampled observations. Afterward, the trial *initializes* the upper and lower bound values of the new nodes. It performs Monte Carlo (MC) roll-outs to initialize lower bounds and uses explicit heuristic functions to initialize upper bounds (there generally exist ones that can be easily written down). Both are conditioned on the sampled scenarios. We refer to upper and lower bounds calculated in this way as the *MC value estimates*. The initial MC estimates are denoted as $u_0(b)$ and $l_0(b)$.

The traversal continues until further expansion is no longer beneficial. HyP-DESPOT thus ends the trial and immediately *backs up* new information to the root, updating upper and lower bounds for belief nodes along the way using the Bellman's operator (Eqn. 3 and 4). After finishing the backup, a new trial starts.

HyP-DESPOT executes in an anytime fashion, terminating the search until a maximum planning time is reached or when the gap between the upper and lower bounds at the root

becomes sufficiently small, in which case it produces near-optimal actions. HyP-DESPOT further performs parallel tree search. Multiple threads execute concurrent trials to expand the tree collaboratively.

B. Incorporating Learned Heuristics

The LeTS-Drive planner extends HyP-DESPOT, incorporating the learned policy and values in the heuristics to perform guided belief tree search. It uses the policy network to guide forward traversal, and uses the value network to approximate long-term returns. Fig. 2a briefly illustrates the guided belief tree search algorithm.

The policy network is queried at each node along the exploration path to provide *prior probabilities* over actions (Fig. 2a-policy). The probabilities are used to bias action selection. Specifically, a trial visiting a node b selects an action branch to traverse using a UCB-like heuristics:

$$a^* = \arg\max_{a \in A} \left\{ u(b,a) + c\, \pi_\theta(a|x_b) \sqrt{\frac{N(b)}{N(b,a)+1}} \right\}. \quad (8)$$

The improvement of Eqn. (8) over Eqn. (7) is the additional exploration bonus weighted by a constant factor c. $\pi_\theta(\cdot|x_b)$ denotes the prior probabilities output by the policy network π_θ at the history state x_b, a 4-step history at b encoded as images (see Section VI-A). It prioritizes actions suggested by the learned policy. The bonus further depends on the visitation count of b, *i.e.*, the number of times b has been visited by previous trials, denoted as $N(b)$, and the visitation count of its child action branch, $N(b,a)$. This encourages exploration. When visiting a node the first few times, upper bound estimates of actions, $u(b,\cdot)$, are often uninformative. The action selection is therefore strongly biased by the learned policy, with a desirable level of exploration ensured by the visitation count term. After sufficient search, the difference of upper bounds gradually dominates the heuristics, making it behave more similar to Eqn. (7). The observation selection heuristics remain the same as HyP-DESPOT.

The value network is queried at each leaf node to an initial value estimate of the node (Fig. 2a-value):

$$\hat{v}_0(b) = \min\big(\max\big(l_0(b), v_{\theta'}(x_b)\big), u_0(b)\big), \quad (9)$$

where b is a new leaf node, and $v_{\theta'}(x_b)$ is the *prior value* predicted by the value network $v_{\theta'}$ at the history state x_b (see Section VI-A). The prior value has been learned from past experience, providing accurate value estimates that can otherwise only be acquired by searching the corresponding sub-tree sufficiently. Eqn. (9) further performs *value clipping* to regulate the prior values. For a leaf node b, it clips the prior value, $v_{\theta'}(x_b)$, using the initial MC bounds, $l_0(b)$ and $u_0(b)$, to produce an initial learned value, $\hat{v}_0(b)$. Value clipping guarantees the correctness of learned values, ensuring a relationship of $l_0(b) \le \hat{v}_0(b) \le u_0(b)$, which helps maintain theoretical guarantees of planning. See theoretical details in Section V-C.

During backup, we update both the learned value and the MC value estimates of belief nodes, using the Bellman's operator (Eqn. 3 and 4). Since the operator only applies linear operations and maximization, it guarantees the same relationship, $l(b) \le \hat{v}(b) \le u(b)$, to hold for all belief nodes throughout the tree search.

When the tree search terminates, LeTS-Drive reports the action with the best *learned value* at the root, which provides the best long-term outcome, estimated by the search tree.

C. Performance Guarantee

The following theorem analyzes the *convergence* of the uncertainty gap at the root b_0, measured by the difference between the upper and lower bound estimates, $\epsilon(b_0) = u(b_0) - l(b_0)$, and discusses the *regret bound* of the reported optimal policy $\hat{\pi}^*$ at convergence:

Theorem 1. *The proposed belief tree search algorithm using learned heuristics is probabilistically and asymptotically optimal. Suppose that the maximum planning time is unbounded. The uncertainty gap at the root, $\epsilon(b_0)$, will converge to zero in finite time. Let $\hat{\pi}^*$ denote the policy tree reported by the algorithm at convergence. Given any constant $\tau \in (0,1)$, the following relationship holds for the value of $\hat{\pi}^*$ and the value of the true optimal policy π^*, with probability at least $1-\tau$:*

$$V_{\hat{\pi}^*}(b_0) \ge V_{\pi^*}(b_0) - \hat{\epsilon}_{\pi^*,\tau}(K). \quad (10)$$

The approximation error $\hat{\epsilon}_{\pi^*,\tau}(K)$ is the same regret bound stated in Theorem 3.2 of [9], which approaches zero when $K \to \infty$, at a rate of $O(\frac{1}{\sqrt{K}})$, where K is the number of sampled scenarios.

Proof. Convergence of the search is guaranteed by *optimistic trails*, exploration paths that use the unbiased heuristics (Eqn. 7), which is brought forward from HyP-DESPOT [1] and launched periodically in our planner. As shown in Theorem 1 of [1], optimistic trials always guarantee the uncertainty gap at the tree root, $\epsilon(b_0) = u(b_0) - l(b_0)$, to monotonically decrease and converge to zero with a finite number of trials, and the guarantee holds regardless of what exploration mechanism is deployed in other trials, in our case, Eqn. (8). This means the proposed algorithm always converges in finite time. Since the learned value at the root node b_0 is bounded between $l(b_0)$ and $u(b_0)$, it will converge to the optimal value under the sampled scenarios. Namely, our planner will report the same optimal policy as HyP-DESPOT when both algorithms converge.

Further, as shown in Theorem 3.2 of [9], given any constant $\tau \in (0,1)$, the regret induced by the optimal HyP-DESPOT policy, $\hat{\pi}^*$, which considers only the sampled scenarios, with respect to the true optimal policy, π^*, that considers all possible scenarios, is bounded by $\hat{\epsilon}_{\pi^*,\tau}(K)$ with probability at least $1-\tau$. For a given POMDP problem, the regret bound, $\hat{\epsilon}_{\pi^*,\tau}(K)$, is determined by K, the number of sampled scenarios. When increasing K to infinity, the bound converges to zero at an asymptotic rate of $O(\frac{1}{\sqrt{K}})$. The regret bound also decreases with the size of the optimal policy tree π^*, meaning that if a simple near-optimal policy exists, the value approximation will be particularly tight. We refer readers to Theorem 3.2 of [9] for the detailed expression of Eqn. (10) and a rigorous proof of the bound.

Finally, since our planner reports the same policy as HyP-DESPOT at convergence, it provides the same regret bound.

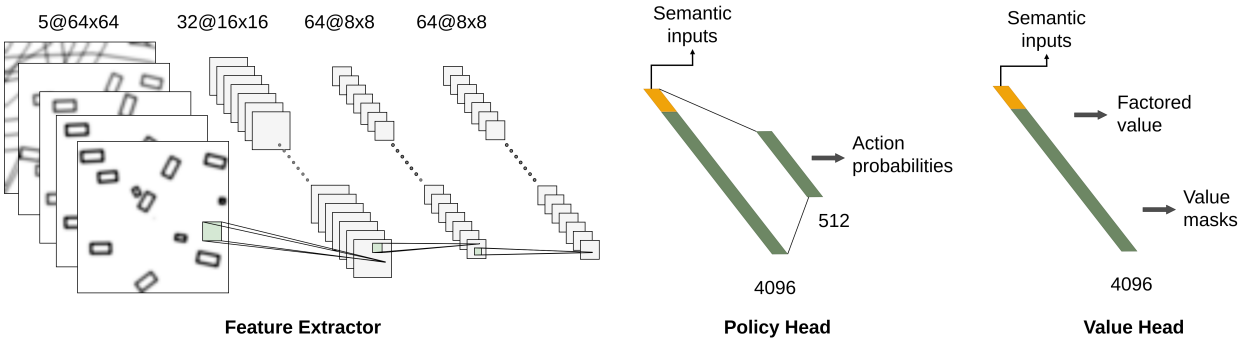


Fig. 3. Neural network architectures of the policy and value networks.

This concludes our proof that our planner is probabilistically and asymptotically optimal. □

VI. Planning-Informed Learning

The learner in LeTS-Drive uses the planner's experiences to update the policy network and the value network. This section will discuss three possible designs of learners—*open-loop self-supervision*, *closed-loop self-supervision*, and *closed-loop reinforcement*. The learners share the same planner counterpart, thus also correspond to three LeTS-Drive variants. In the following, we will first present the neural network architectures, and the collection of learning experience, then present the three learner variants' core ideas and algorithmic details.

A. Policy and Value Networks

The architectures of the policy and value networks are shown in Fig. 3 and described below. Input to the policy and value networks are top-down rasterized images encoding the state history x_b at a belief b. The input consists of 5 channels. Channel $1-4$ encode the geometry of traffic agents at the current and three past frames; The $5th$ channel encodes the lane graph of the urban map drawn as a set of poly-lines. All images are registered to the local view of the ego-vehicle for the corresponding time step. They are initially rendered as 1024×1024 images and down-sampled to 64×64 using Gaussian pyramids [29] before inputting to the neural networks.

The policy and value networks use a feature extractor similar to that in [30]. The input images are processed by three convolutional layers: an input layer with 32 8×8 kernels with stride 4 and no padding; a middle layer with 64 4×4 kernels with stride 2 and no padding; and the last layer with 64 3×3 kernels with stride 1 and no padding. The extractor outputs 64 8×8 images as hidden features. These features are flattened and concatenated with the semantic inputs, *i.e.*, velocities of the ego-vehicle in the past four frames, and fed to the heads.

Our policy network only has one categorical head to output the distribution over nine possible lane-decision / acceleration combinations. The policy head has two fully-connected (FC) layers mapping from the raw feature vector of length 4096 to an intermediate feature vector of length 512, then to 9 action probabilities. The value network, instead, has two heads corresponding to the factored value function (Appendix A). They include a mask head to output two binary masks for the safe-driving and collision value factors, and a value head to predict the non-zero numbers for the value factors. Both heads have a single FC layer directly mapping the raw features to factored predictions, which are combined to recover the actual value prediction.

B. Data Collection

Data for learning are collected by the planner, through acting in the environment. In each episode, the actor records a trajectory. Each data point along the trajectory is represented as (b, a, r, a^*, v^*), where b is the belief at the corresponding time step, a is the executed action, and r is the reward fed back by the environment after executing a; the data point also records the planner's estimation of the optimal action, a^*, and the optimal value, v^*, at b. The action-value labels enable self-supervised learning (Sections VI-C1 and VI-C2). The rewards enable reinforcement learning (Section VI-C3). The planner actor executes different a for different learners. For self-supervised learning, we directly execute the optimal action ($a = a^*$); for reinforcement learning, we use a combination of exploitative actors ($a = a^*$), explorative actors (sample a according to estimated action-values), and on-policy actors (execute the learner policy) to collect experience. Collected trajectories are processed into a pool of data points, either stored in an offline dataset or fed to a fixed-capacity replay buffer, for offline and online learners, respectively. Multiple actors can execute asynchronously in separate simulator instances, to collaboratively collect data.

C. Learners

Now we introduce the learner that uses experiences from the planner to optimize the policy and value networks. We propose the following learner variants, covering both open-loop and close-loop integration of planning and learning, and leveraging both self-supervised and reinforcement learning:

1) Open-loop self-supervised learning (Open-SSL): In Open-SSL, the integration of planning and learning happens in two phases: offline supervised learning and online guided planning. In the offline phase, Open-SSL learns from a fixed

planning expert; In the online phase, it plans with learned heuristics. No further data is fed back to the learner during the online stage. The planner and the learner are thus integrated in an "open-loop". This algorithm replicates the idea of our earlier work [27] using the new planner with value clipping (Section V) and the new POMDP and neural network models (Section IV and VI-A).

Specifically, Open-SSL uses HyP-DESPOT [1] as the actor to collect an offline dataset. Then, the learner trains the neural networks by sampling the dataset. For each sampled belief b, it fits the policy network to the action label, a^*, and fits the value network to the value label, v^*, both provided by HyP-DESPOT. It uses cross-entropy loss (CEL) for policy predictions and mean-square errors (MSE) for value predictions. To facilitate value learning, we have further decomposed the value loss into safe-driving and collision factors following the factorization of the planner's value function (Section IV-F). Details of the loss functions are explained in Appendix B1.

At execution time, Open-SSL performs guided belief tree search to synthesize real-time driving policies (Section V-B). Open-SSL thus benefits from both local planning and global learning. However, the limitation is that it cannot leverage new data generated by the stronger, guided planner.

2) Closed-loop self-supervised learning (Closed-SSL): Closed-SSL improves over Open-SSL by letting the learner receive online experiences from the *guided* planner and constantly feed updated heuristics back to the planner, thus closing the planning-learning loop as shown in Fig. 2.

Closed-SSL uses the guided planner (Section V) as the actor to collect data for learning. The planner uses the latest policy and value network as heuristics. In each episode, the planner collects a trajectory and feeds it to a fixed-capacity replay buffer. In the meantime, the learner repeatedly samples data from the replay buffer. For each sampled belief b, it fits the policy network to the action label, a^*, and the value network to the value label, v^*, both provided by the guided planner (Fig. 2d). Policy and value learning also uses CEL and MSE losses, respectively, where the value loss is factorized. After every few updates, it feeds the new heuristics back to the planner through a shared buffer. Closed-SSL learns from scratch, starting from randomly initialized policy and value networks and an empty replay buffer. Stable training is achieved with the help of entropy regularization, which enforces a desired level of entropy for the learned policies. Details of the loss function and entropy regularization are explained in Appendix B1.

Closed-SSL is essentially a form of *self-supervised* learning: the planner provides labels to train its own sub-components (the heuristics). Sample efficiency is achieved by using structured rewards (compiled as values) from the planner as learning signals, instead of unstructured (raw) rewards from the environment.

Closed-SSL can also be viewed as generalized policy iteration [31]: the belief tree search performs policy improvement over the current policy; the learner then updates itself to fit the improved policy. By iterating these two steps, the planner and the learner can together converge to optimal policies defined w.r.t the POMDP model.

The POMDP model is, however, an imperfect approximation to the actual environment. Since Closed-SSL plans using the model and learns from policies and values generated with the model, it can be sensitive to model errors (even though we observe it working well in practice).

3) Closed-loop reinforcement learning (Closed-RL): Closed-RL is thus proposed to hedge against model errors. Closed-RL additionally uses policy gradient [32], [33], [34] to let the policy network receive and learn from reward feedback from the *actual environment*. By doing so, the learner policy is optimized w.r.t. the true environment dynamics.

Closed-RL shares the same closed-loop architecture and value learner as Closed-SSL. Differently, the policy learner is not supervised by the planner's actions. Instead, it learns from the rewards fed back by the environment (Fig. 2e). At each sampled belief b, Closed-RL reinforces the learner policy by estimating its expected value from raw reward signals along the trajectory, and differentiating the value to compute gradients for updating the learner policy ("policy gradient"). Closed-RL thus optimizes the learner policy for its *own* expected value. The policy is thus unaffected by the imperfection of planning models.

Closed-RL essentially uses the planner as an *exploration policy* for reinforcement learning. However, the explored trajectories are *off-policy*, i.e., not sampled from the distribution induced by the learner policy, but from the distribution induced by the guided planner. Such trajectories lead to biased value estimates and policy gradients for the learner policy if not properly corrected. Thus, we build our learner on top of a popular off-policy policy gradient algorithm, soft actor-critic (SAC) [35], to correctly train the learner policy using the planner's experience. Entropy regularization is also applied here to assist training. Details of our SAC implementation are presented in Appendix B2.

VII. Experiments

In the experiments, we compare LeTS-Drive with online POMDP planning, reinforcement learning, as well as open-loop integration of planning and learning. We also provide tests on the scalability and generalization of LeTS-Drive, and analyzed the importance of new algorithmic components. We tested all three variants of LeTS-Drive. Among them, *Closed-SSL* and *Closed-RL* are our proposed approaches, and *Open-SSL* serves as an open-loop integration baseline. For planning and learning baselines, we use HyP-DESPOT to calibrate the capability of existing POMDP planning tools, and use imitation learning (the policy learner in Open-SSL) and policy gradient using SAC [35] (labeled as *PG*) to calibrate the capability of stand-alone policy learning. By comparing LeTS-Drive with the existing algorithms that it is developed on top of, we perform controlled experiments to clearly show the benefit of integrating planning and learning for tackling short planning time and limited data.

Our results show that the integration of planning and learning enables LeTS-Drive to largely advance the capability of both, greatly improving the scalability of online planning and the efficiency of learning. Closed-loop planning and

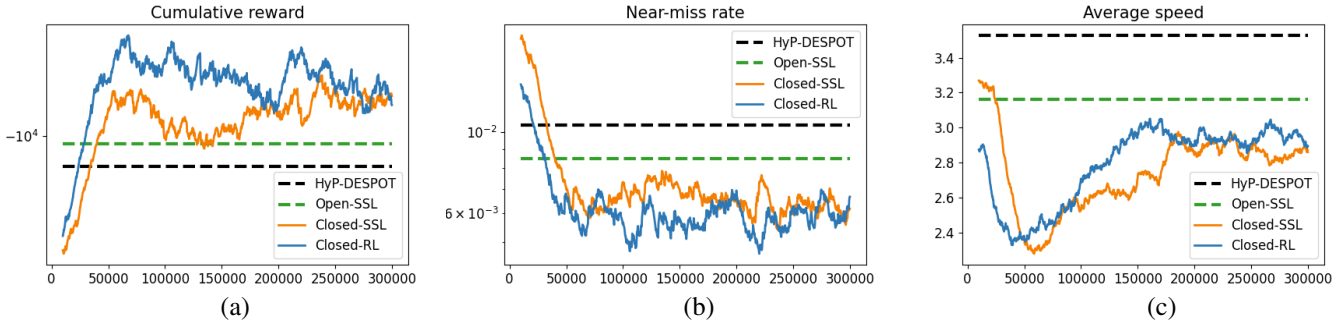


Fig. 4. Performance of planner policies in LeTS-Drive compared with online POMDP planning using HyP-DESPOT. All LeTS-Drive variants achieve significant improvements over POMDP planning. Closed-SSL and Closed-RL achieve the best sample efficiency and asymptotic performance.

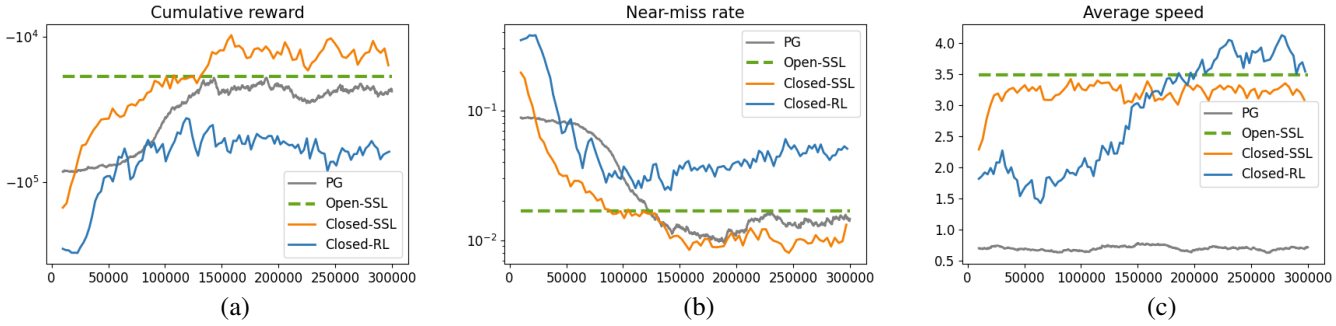


Fig. 5. Performance of learner policies in LeTS-Drive, compared with policy gradient (PG) and imitation learning (Open-SSL). Closed-loop self-supervised learning (Closed-SSL) produces the most effective learner policy.

learning further improves the sample efficiency and asymptotic performance by a large margin. When using self-supervised learning, LeTS-Drive produces the strongest learner policies; when additionally using reinforcement learning, LeTS-Drive achieves the best integrated performance. Value clipping applied in the search not only ensures theoretical guarantees, but also improves the practical performance of LeTS-Drive. After training, our planner can successfully drive a vehicle through dense urban crowds with sophisticated combinations of accelerations and maneuvers, and generalize to significantly different environments. See the example driving clips in the accompanying video or via this link: youtu.be/fIOkLHji3co.

A. *Experimental Setup*

We analyze our approach in SUMMIT [3], a real-time simulator for massive mixed urban traffic. Given any world-wide location supported by the OpenStreetMap [4], SUMMIT automatically generates dense traffic on the map. It controls exo-agents using GAMMA [5], a recent traffic motion model validated on multiple real-world datasets. The model uses velocity-obstacle-based optimization to perform local collision avoidance. It is efficient, able to support real-time simulation of many traffic agents. We train LeTS-Drive using random crowds at the Meskel-Square intersection at Addis Ababa, Ethiopia (Fig. 1) simulated in SUMMIT. Each instance of urban crowd contains 110 active traffic agents, including trucks, buses, cars, motorcycles, pedestrians, *etc.*. An episode of experience consists of a few minutes of continuous driving and ends when the vehicle exits the range of the map.

We measure the performance of planning and learning algorithms using the average cumulative reward achieved over episodes. We further measure the driving safety and efficiency to provide a detailed view of performance. Driving safety is measured as the near-miss rate over all time steps. A near-miss is a close encounter event when the estimated time-to-collision is shorter than a threshold, here set as $0.33s$, which is a more robust hazard indicator than collisions [36]. Driving efficiency is characterized by the average driving speed, measured in m/s.

We use the same planning setup for all planners and the same learning setup for all online learners. A POMDP state tracks a maximum of 20 exo-agents within 50 meters of the ego-vehicle, corresponding to the rough attention range of human drivers in dense traffic. Planners are allowed 0.3 seconds of maximum planning time at each step and execute at a rate of $3Hz$, which roughly reflects the human response time. All learners use 3×10^5 data points. Training of Closed-SSL, Closed-RL, and PG starts with an empty replay buffer, and ends after receiving 3×10^5 unique data points. Since the time for collecting a data point in real-time simulation is fixed, the setup also leads to the same learning time, which is approximately 20 hours when using three concurrent actors and one learner on a single machine[1]. The Open-SSL baseline consumes an offline dataset of size 3×10^5 and is trained till convergence. The reinforcement learner in Closed-RL uses the same network architectures as PG. Policy

[1]The machine is a server with 4 RTX 2080 GPUs, an Intel(R) Core(TM) i7-8750H CPU, and 256G RAM. Each of the actors and the learner uses one GPU to query or train neural networks.

and value networks in all LeTS-Drive variants share the same network architectures.

B. Planner Policies

Fig. 4 shows the learning curves of the *planner policies* of LeTS-Drive and the performance of online POMDP planning using HyP-DESPOT. The learning curves are generated by periodically evaluating the planners in SUMMIT throughout training, averaged over five random seeds. The main observations are as follows.

The integration with learned heuristics immediately brings significant performance gain over HyP-DESPOT, even when using the open-loop architecture (Open-SSL). The resulting planner policy conducts more cautious driving, leading to fewer near-misses.

By closing the planning-learning loop using self-supervision (Closed-SSL), LeTS-Drive achieves superior sample efficiency, outperforming open-loop integration with around one-tenth of data, and achieving much higher asymptotic performance. The resulting planner policy further reduces the near-miss rate by a large margin.

Switching to the reinforcement learner (Closed-RL) further improves the integrated performance, as the algorithm additionally receives feedback from the actual environment. Closed-RL brings the best sample efficiency, quickly achieving the highest rewards among all planners during training.

We have observed similar learning patterns from the Closed-SSL and Closed-RL planners. Both of them first learn to reduce the near-miss rate by lowering the driving speed. Then, they gradually increase the driving speed with the near-miss rate maintained low. Both training curves have converged after receiving $1.5 \times 10^5 \sim 2 \times 10^5$ data points. At convergence, they deliver a similar level of planning performance.

C. Learner Policies

Fig. 5 shows the learning curves of the *learner policies* in LeTS-Drive and stand-alone policy learning approaches. The curves are generated by periodically evaluating the policy networks in SUMMIT throughout training.

We observe that policy gradient learners, which do not perform explicit reasoning, struggle to learn an effective policy for crowd-driving given the limited amount of data. This is because the task conveys three distinct local-optima behaviors: defensive driving, aggressive driving, and smart collision avoidance (desired). Policy gradient (PG) acquires very conservative driving behaviors, primarily learning to reduce the near-miss rate, which is safe but inefficient. The learner policy of Closed-RL acquires overly-aggressive behaviors, mostly learning to increase the driving speed, which is efficient but unsafe. This behavioral difference results primarily from different experiences. The PG policy only learns from its own driving experiences. It hardly foresees the benefit of gathering speed due to frequent collision penalties. The Closed-RL policy learns from the planner's experiences. It is incentivized to gather speed by the positive rewards received. However, with limited data, Closed-RL struggles to learn collision avoidance sufficiently well.

TABLE I
GENERALIZATION OF TRAINED LETS-DRIVE PLANNER POLICIES OVER NEW CROWD DISTRIBUTIONS IN THE TRAINING MAP. THE FIRST AND SECOND COLUMNS SHOW THE IMPROVEMENT ON THE AVERAGE CUMULATIVE REWARD COMPARED TO THE LEARNER POLICY AND POMDP PLANNING USING HYP-DESPOT, RESPECTIVELY.

	Reward w.r.t. learner ($\times 10^3$)	Reward w.r.t POMDP ($\times 10^3$)	Near-miss rate	Average speed
HyP-DESPOT	-	0.00	0.0100	3.53±0.000
Open-SSL	+8.34	+1.54	0.0085	3.16±0.000
Closed-SSL	+5.19	**+4.03**	**0.0057**	2.74±0.003
Closed-RL	**+35.97**	**+3.95**	0.0066	3.00±0.005

TABLE II
GENERALIZATION OF TRAINED LETS-DRIVE PLANNER POLICIES OVER NOVEL MAPS. THE FIRST AND SECOND COLUMNS SHOW THE IMPROVEMENT ON THE AVERAGE CUMULATIVE REWARD COMPARED TO THE LEARNER POLICY AND POMDP PLANNING USING HYP-DESPOT.

	Reward w.r.t. learner ($\times 10^3$)	Reward w.r.t. POMDP ($\times 10^3$)	Near-miss rate	Average speed
HyP-DESPOT	-	0.00	0.0081	3.57 ± 0.026
Open-SSL	+2.74	+2.65	0.0057	3.09 ± 0.000
Closed-SSL	+13.2	+3.81	**0.0049**	2.74 ± 0.016
Closed-RL	**+46.4**	**+4.15**	0.0052	3.03 ± 0.017
Closed-RL (Retrain)	+44.9	+4.20	0.0049	3.20 ± 0.020

In comparison, self-supervised learning (Closed-SSL) produces smart driving policies with both low near-miss rates and desirable driving efficiency. The final learner policy has matched the performance of HyP-DESPOT, showing the effectiveness of self-supervision in policy learning.

D. Scalability

We further provide in Fig. 6 a scalability test for the LeTS-Drive (Closed-RL) planner and compare it with the scalability of HyP-DESPOT. In the test, we gradually increase the number of agents in the Meskel Square from 44 to 110, to construct planning problems of different scales and complexities. The scalability of planners are evaluated using their capability of handling the growing problem scale in real-time. We observe LeTS-Drive consistently outperforming HyP-DESPOT by searching smaller and shallower trees, when using the same planning time of $0.3s$. The performance gain increases with the problem scale.

The problem scale of crowd-driving is determined by the number of nearby agents a planner considers in a POMDP state, here denoted as N. Increasing N leads to an exponential growth of the state and observation spaces and a quadratic growth of the complexity of the transition function. N increases with the crowd size, as more agents fall within the 50-meter range of the ego-vehicle. Fig. 6a shows the number that planners effectively considered in the experiments.

The growth of the problem scale leads to quickly decayed real-time performance of HyP-DESPOT, as shown by the declined reward (Fig. 6d), increased near-miss rate (Fig. 6e), and sacrificed driving speed (Fig. 6f). In comparison, LeTS-Drive always generates better policies by searching much smaller and shallower trees. When the problem scale grows,

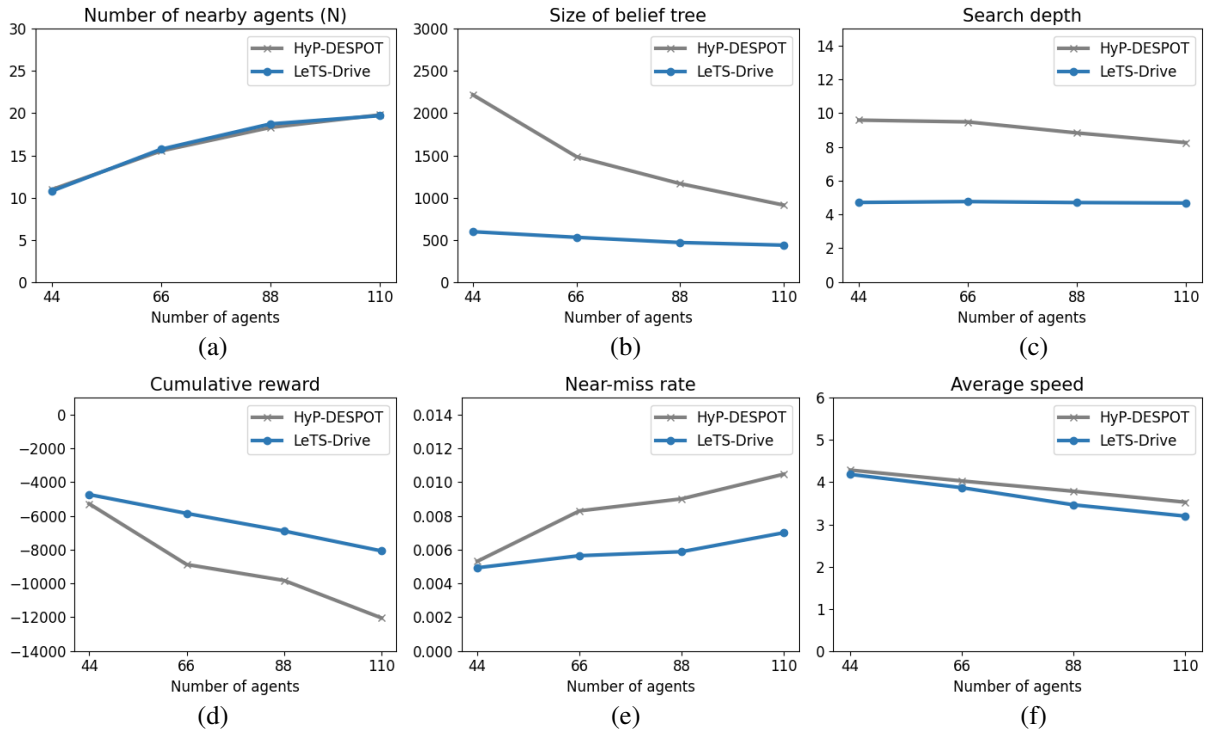


Fig. 6. Scalability of LeTS-Drive planner (Closed-RL) with increasing number of agents in the crowd, compared with standard POMDP planning using HyP-DESPOT).

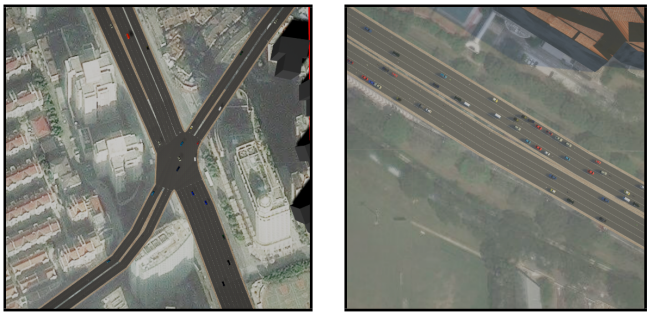


Fig. 7. Generalization over novel maps. Each map is populated with 110 traffic agents in our experiments. (a) Shanghai intersection. (b) Singapore highway.

the tree size and search depth of LeTS-Drive remain almost unaffected (Fig. 6bc). However, the benefit of integrating planning and learning increases. LeTS-Drive consistently achieves higher cumulative rewards (Fig. 6d) and lower near-miss rates (Fig. 6e), with a marginal compromise on the driving speed (Fig. 6f). The denser the scene is, the more performance gain LeTS-Drive brings, showing improved scalability of planning.

E. Generalization

We now inspect the generalization of LeTS-Drive.

1) Random test crowds: Table I shows the results for evaluating the trained planner and learner policies with unseen random crowds on the training map (Meskel intersection). Numbers are calculated using more than 1000 test episodes.

The results are generally consistent with those during training, clearly showing the benefits of integrating planning and learning from both directions. All LeTS-Drive planner policies have drastically improved the rewards over HyP-DESPOT and their learning counterparts. Close-loop integration (Closed-SSL and Closed-RL) has achieved significantly higher rewards than the open-loop (Open-SSL), generating planner policies with the lowest near-miss rate and the highest rewards.

Closed-SSL and Closed-RL allow different trade-offs between exploitation and exploration, brought by the quality and the entropy of the learner policy, respectively. Closed-SSL learns the best with a low policy entropy, because the entropy regularization objective often contradicts the imitation objective. It thus learns a strong but less explorative policy, which confidently guides the tree search towards high-quality directions. Closed-RL learns more stably with high policy entropy, due to the characteristics of policy gradient. It produces a weaker but more explorative policy, with approximately 18% higher entropy, which encourages the search to explore a wider set of sensible actions. As a result, we observe in Table I that Closed-SSL benefits equally from explicit planning and a strong learner policy, as shown by similar reward improvements w.r.t. the learner and POMDP planning. In contrast, Closed-RL improves over a much weaker learner policy, achieving similar integrated performance as Closed-SSL, with the help of exploration.

2) Novel test maps: We further test LeTS-Drive on two significantly different maps: another intersection in Shanghai and a highway in Singapore (Fig. 7). Results are shown in Table II. Despite the extreme setup—training in a single intersection and testing on different maps—all variants of LeTS-Drive have successfully generalized to the new environments, almost matching the performance of Closed-RL trained specifically

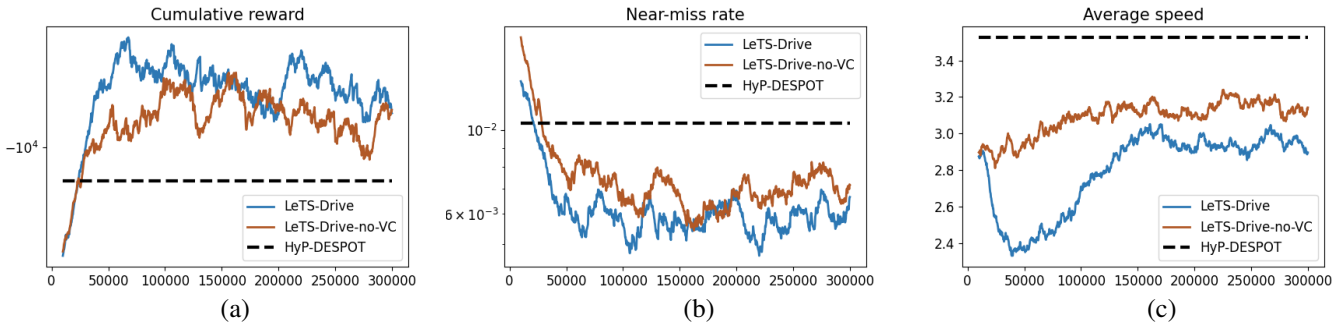


Fig. 8. Performance of the Closed-RL planner policy with and without value clipping.

on the test maps, referred to as Closed-RL (Retrain) in Table II. Among them, Closed-SSL and Closed-RL have achieved the best generalized performance. Closed-loop planning and learning brings the same level of benefits as in the training map, delivering the safest planner policies with the highest rewards.

F. The Effect of Value Clipping

Value clipping (Section V-B) is an important algorithmic component that ensures the convergence of the guided belief tree search. We now show its practical effects. Fig. 8 shows the learning curves of LeTS-Drive with and without value clipping. Without value clipping, the planner becomes overly optimistic due to the misuse of approximate heuristics. It seldom attempts to reduce the driving speed during training, thus inducing consistently higher near-miss rates. This compromises the reward throughout training. In contrast, with value clipping, LeTS-Drive becomes more cautious in driving, maintaining significantly lower speeds during the initial course of training. Afterward, the planner stably improves driving efficiency and constantly achieves higher rewards. This shows, besides maintaining theoretical guarantees, value clipping also enables more stable and efficient training in practice.

VIII. Conclusion and Future Work

We have presented the LeTS-Drive algorithm, which integrates planning and learning by planning locally and learning globally in a closed loop. LeTS-Drive flexibly takes advantage of either self-supervised learning or reinforcement learning to learn heuristics for online planning. Doing so, LeTS-Drive scales up online decision making under uncertainty: it outperforms planning or learning alone, or open-loop integration of planning and learning. Simulation experiments also show that LeTS-Drive exhibits sophisticated driving behaviors in challenging urban traffic with large heterogeneous crowds.

One limitation of LeTS-Drive is potential model errors. Closed-RL partially addresses the problem. It eliminates the bias in policy learning, but not in value learning, which still relies on self-supervision. Significant model errors may lead to inaccuracy in learned values and compromise the planner's performance. Model learning, (*e.g.*, [21]) alleviates this issue. It is also possible to apply reinforcement learning, *e.g.*, temporal difference (TD) learning [37], to learn the values directly, but it is undesirable, because of sample inefficiency. Instead, we can refine value estimates through TD learning after "warming up" through self-supervision.

The current crowd-driving model in LeTS-Drive can be further improved, by incorporating comprehensive traffic rules, social norms, *etc.*. With increased model complexity, we expect LeTS-Drive to provide even more significant performance benefits through integrated planning and learning. There are also other models for different driving settings that are interesting to consider [38], [39], [40]. More importantly, there is often a gap between the simulation and the real world for robot systems. Further research is required to study the effect of this gap on crucial issues, such as driving safety, as well as ways to close this gap [41], [42], [43]. One may also perform human experiments in simulation to test the real-life performance of LeTS-Drive, by letting human control some or all simulated exo-agents.

Finally, LeTS-Drive's core algorithmic ideas are not specific to crowd-driving, but applicable in general to many large-scale, long-term planning tasks, such as object manipulation in clutter, multi-agent coordination, *etc.*. We will explore these exciting directions as our next step.

Acknowledgement

This research is supported in part by the National Research Foundation (NRF), Singapore and DSO National Laboratories under the AI Singapore Program (AISG Award No: AISG2-RP-2020-016).

REFERENCES

[1] P. Cai, Y. Luo, D. Hsu, and W. S. Lee, "HyP-DESPOT: A hybrid parallel algorithm for online planning under uncertainty," in *Proc. Robotics: Science & Systems*, 2018.

[2] 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, p. 354, 2017.

[3] P. Cai, Y. Lee, Y. Luo, and D. Hsu, "Summit: A simulator for urban driving in massive mixed traffic," *arXiv preprint arXiv:1911.04074*, 2019.

[4] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ." https://www.openstreetmap.org, 2017.

[5] Y. Luo, P. Cai, Y. Lee, and D. Hsu, "Gamma: A general agent motion model for autonomous driving," *IEEE Robotics and Automation Letters*, 2022.

[6] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99 – 134, 1998.

[7] Z. Chen *et al.*, "Bayesian filtering: From kalman filters to particle filters, and beyond," *Statistics*, vol. 182, no. 1, 2003.

[8] D. Silver and J. Veness, "Monte-carlo planning in large POMDPs," in *Advances in Neural Information Processing Systems*, 2010.

[9] N. Ye, A. Somani, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," *J. Artificial Intelligence Research*, vol. 58, pp. 231–266, 2017.

[10] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online POMDP planning for autonomous driving in a crowd," in *Proc. IEEE Int. Conf. on Robotics & Automation*, 2015.

[11] M. Meghjani, Y. Luo, Q. H. Ho, P. Cai, S. Verma, D. Rus, and D. Hsu, "Context and intention aware planning for urban driving," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2019.

[12] Y. Xiao, S. Katt, A. ten Pas, S. Chen, and C. Amato, "Online planning for target object search in clutter under partial observability," in *Proc. IEEE Int. Conf. on Robotics & Automation*, 2019.

[13] O. Walker, F. Vanegas, and F. Gonzalez, "A framework for multi-agent uav exploration and target-finding in gps-denied and partially observable environments," *Sensors*, vol. 20, no. 17, 2020.

[14] Y. Luo, P. Cai, A. Bera, D. Hsu, W. S. Lee, and D. Manocha, "Porca: Modeling and planning for autonomous driving among many pedestrians," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3418–3425, 2018.

[15] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016.

[16] P. Karkus, D. Hsu, and W. S. Lee, "Qmdp-net: Deep learning for planning under partial observability," in *Advances in Neural Information Processing Systems*, 2017.

[17] G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson, "Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning," *arXiv preprint arXiv:1710.11417*, 2017.

[18] A. Guez, T. Weber, I. Antonoglou, K. Simonyan, O. Vinyals, D. Wierstra, R. Munos, and D. Silver, "Learning to search with mctsnets," in *Proc. Int. Conf. on Machine Learning*, 2018.

[19] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks: Learning generalizable representations for visuomotor control," in *Proc. Int. Conf. on Machine Learning*, 2018.

[20] J. Drgona, K. Kis, A. Tuor, D. Vrabie, and M. Klauco, "Differentiable predictive control: An mpc alternative for unknown nonlinear systems using constrained deep learning," *arXiv preprint arXiv:2011.03699*, 2020.

[21] A. Ayoub, Z. Jia, C. Szepesvari, M. Wang, and L. Yang, "Model-based reinforcement learning with value-targeted regression," in *International Conference on Machine Learning*, pp. 463–474, PMLR, 2020.

[22] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *Proc. Int. Conf. on Machine Learning*, 2019.

[23] K. Fang, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei, "Dynamics learning with cascaded variational inference for multi-step manipulation," in *Proc. Conf. on Robot Learning*, 2020.

[24] K. Liu, M. Stadler, and N. Roy, "Learned sampling distributions for efficient planning in hybrid geometric and object-level representations," in *Proc. IEEE Int. Conf. on Robotics & Automation*, 2020.

[25] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," *arXiv preprint arXiv:1903.02531*, 2019.

[26] Y. Lee, P. Cai, and D. Hsu, "Magic: Learning macro-actions for online pomdp planning using generator-critic," in *Proc. Robotics: Science & Systems*, 2021.

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

[28] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[29] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[31] M. J. Kochenderfer, *Decision Making under Uncertainty: Theory and Application*. MIT press, 2015. Chapter 4, Section 2.3.

[32] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. on Machine Learning*, 2016.

[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. on Machine Learning*, 2018.

[35] P. Christodoulou, "Soft actor-critic for discrete action settings," *arXiv preprint arXiv:1910.07207*, 2019.

[36] J. C. Hayward, "Near-miss determination through use of a scale of danger," *Highway Research Record*, no. 384, 1972.

[37] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018. Section 11: Off-policy methods with approximation.

[38] S. Ulbrich and M. Maurer, "Probabilistic online pomdp decision making for lane changes in fully automated driving," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pp. 2063–2067, IEEE, 2013.

[39] K. H. Wray, S. J. Witwicki, and S. Zilberstein, "Online decision-making for scalable autonomous systems," in *International joint conference on artificial intelligence*, 2017.

[40] Z. Sunberg and M. Kochenderfer, "Improving automated driving through planning with human internal states." Under revision, *IEEE Transactions on Intelligent Transportation Systems*.

[41] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *Proc. Robotics: Science & Systems*, 2016.

[42] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *Int. J. Robotics Research*, vol. 39, no. 1, 2020.

[43] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall, "Learning to drive from simulation without real world labels," in *Proc. IEEE Int. Conf. on Robotics & Automation*, IEEE, 2019.

[44] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

APPENDIX

A. Factored reward model

The raw reward function described in Section IV-E is sufficient for planning, but is problematic for value learning due to the existence of rare but critical events, *e.g.*, colliding with others. Particularly, this reward function is smooth at safe belief states, but can change dramatically at proximity to the critical events. To facilitate value learning, we factor our reward function, and consequently the value function, into safe-driving rewards R_s and collision penalties R_c:

$$\begin{aligned} R &= R_s + R_c && (11)\\ R_s &= R_v + R_{acc} + R_{change} && (12)\\ R_c &= R_{col} && (13)\end{aligned}$$

where the speed penalty R_v, smoothness penalties R_{acc} and R_{change}, and collision penalty R_{col} are defined as in Section IV-E.

To compute factored values from this reward function, we simply need to record the safe factor V_s and collision factor V_c separately during the backup process in the belief tree search. Particularly, at a belief node b, the Bellman's operator is executed as:

$$a^* = \arg\max_{a\in A}\left\{R(b,a) + \gamma \sum_{z\in Z_{b,a}} p(z|b,a)V(b')\right\} \quad (14)$$

$$V_s(b) = R_s(b,a^*) + \gamma \sum_{z\in Z_{b,a^*}} p(z|b,a^*)V_s(b') \quad (15)$$

$$V_c(b) = R_c(b,a^*) + \gamma \sum_{z\in Z_{b,a^*}} p(z|b,a^*)V_c(b') \quad (16)$$

Eqn. (14) denotes the regular value backup process where the best value is chosen according to the original value estimates V. Then the factored values associated with this best action a^* is backed-up to the parent (Eqn. (15-16)).

Factored values at the root node are extracted as supervision labels for the learner. As the two factors are frequently zero, we further decompose the extracted value labels to binary masks and non-zero values before feeding to the learner:

$$V = \mathbb{1}_{|V_s\neq 0} * V_s^- + \mathbb{1}_{|V_c\neq 0} * V_c^- \quad (17)$$

where V_s^- and V_s^- are non-zero, negative values.

B. Loss functions for learners

1) Supervision loss: In self-supervised learners, the policy network π_θ and the value network $v_{\theta'}$ are trained separately using supervised learning using action, mask, and value labels output by the planner. Given a dataset D of size N, the loss functions, $l(\theta, D)$ and $l(\theta', D)$, measure the errors in action and value predictions, respectively:

$$\begin{aligned} l(\theta, D) &= -\frac{1}{N}\sum_i^N \log \pi_\theta(a^i|x_b^i) - \alpha H(\pi_\theta(\cdot|x_b^i)) && (18)\\ l(\theta', D) &= l_{mask}(\theta', D) + l_{value}(\theta', D) && (19)\end{aligned}$$

where

$$\begin{aligned} l_{mask}(\theta', D) &= \frac{1}{N}\sum_i^N (m_s(x_b^i|\theta') - \mathbb{1}_{|V_s^i\neq 0})^2 && (20)\\ &+ (m_c(x_b^i|\theta') - \mathbb{1}_{|V_c^i\neq 0})^2 \\ l_{value}(\theta', D) &= \frac{1}{N}\sum_i^N (\mathbb{1}_{|V_s^i\neq 0} * v_s(x_b^i|\theta') - V_s^i)^2 \\ &+ (\mathbb{1}_{|V_c^i\neq 0} * v_c(x_b^i|\theta') - V_c^i)^2 && (21)\end{aligned}$$

Here, x_b^i is the history state in the ith data point; a^i, V_s^i, and V_c^i are the action and value labels obtained from the planner; $m_s(x_b^i|\theta')$ and $m_c(x_b^i|\theta')$ are the mask predictions from the value network; and $v_s(x_b^i|\theta')$ and $v_c(x_b^i|\theta')$ are the value predictions from the value network.

Eqn. (18) represents the cross-entropy loss [44] of the output policy w.r.t. to action labels (the first term) augmented with entropy regularization for the policy itself (the second term). The regularization factor α is tuned online using gradient descent to help maintain a given target entropy of the output policy. This dynamic update rule of α is borrowed from SAC [34]. In our implementation, we set the target entropy to be $0.98\log|A|$ (targeting at scattered distributions) initially, and gradually anneal it to $0.65\log|A|$ (targeting at more concentrated distributions). Eqn. (20) defines the prediction loss of the binary masks applied on value factors. Finally, Eqn. (21) defines the regression loss for the non-zero values.

2) Reinforcement loss: In the reinforcement learner, we use SAC [34], an off-policy policy-gradient algorithm, to train the policy network. Specifically, we use its discrete-action version presented in [35]. The loss function of the policy learner is:

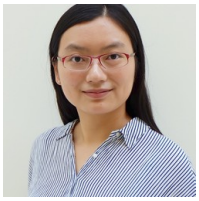
$$J(\theta) = E_{x_b\sim D}\left[\pi_\theta(x_b)^T\left[\alpha\log(\pi_\theta(x_b)) - Q_\phi(x_b)\right]\right]. \quad (22)$$

Here, x_b is a sampled history state from the replay buffer; π_θ is the policy network; α is a dynamically-tuned regularization scalar controlling the target entropy of $\pi_\theta(x_b)$; and Q_ϕ is a Q-network trained in a soft-Q learning manner, serving as a differentiable surrogate objective. The Q-network shares the same architecture as the policy network (Fig. 3), but without the softmax applied to the output. Details of the discrete-action SAC can be found in [35].

Note that for policy-gradient, we can not directly apply the reward function described in Section IV-E because of the scale and sparsity of collision penalties. Instead, we use the following smooth reward function in SAC:

$$R = 0.05\frac{v}{v_{max}} - 0.025\mathbb{1}_{lane\neq 0} - \frac{1}{9t_c^2} \quad (23)$$

where the first term encourages efficient driving, the second penalizes excessive lane changes, and the third term penalizes proximity to collision events according to the time-to-collision, t_c, estimated using a constant-velocity prediction model.



Panpan Cai is an associate professor in Shanghai Jiao Tong University (SJTU), China. Prior to that, she was a postdoctoral research fellow at the Department of Computer Science, National University of Singapore (NUS). She received my PhD degree from the Nanyang Technological University. Her research focuses on tackling large-scale decision making problems in robotics that involve complex environments, uncertainties and long-term planning. In recent years, she has been working on robot planning, robot learning, and autonomous driving.

David Hsu is a Provost's Chair Professor in the Department of Computer Science, National University of Singapore (NUS) and the Director of Smart Systems Institute. He is an IEEE Fellow. He received BSc in computer science mathematics from The University of British Columbia and PhD in computer science from Stanford University. His research interests span robotics, AI, and computational structural biology. In recent years, he has been working on robot planning and learning under uncertainty and human-robot collaboration.