SafetyNet: Safe planning for real-world self-driving vehicles using machine-learned policies

Matt Vitelli*, Yan Chang*, Yawei Ye*, Ana Ferreira, Maciej Wołczyk, Błażej Osiński,
Moritz Niendorf, Hugo Grimmett, Qiangui Huang, Ashesh Jain, Peter Ondruska[+]

Abstract— In this paper we present the first safe system for full control of self-driving vehicles trained from human demonstrations and deployed in challenging, real-world, urban environments. Current industry-standard solutions use rule-based systems for planning. Although they perform reasonably well in common scenarios, the engineering complexity renders this approach incompatible with human-level performance. On the other hand, the performance of machine-learned (ML) planning solutions can be improved by simply adding more exemplar data. However, ML methods cannot offer safety guarantees and sometimes behave unpredictably. To combat this, our approach uses a simple yet effective rule-based fallback layer that performs sanity checks on an ML planner's decisions (e.g. avoiding collision, assuring physical feasibility). This allows us to leverage ML to handle complex situations while still assuring the safety, reducing ML planner-only collisions by 95%. We train our ML planner on 300 hours of expert driving demonstrations using imitation learning and deploy it along with the fallback layer in downtown San Francisco, where it takes complete control of a real vehicle and navigates a wide variety of challenging urban driving scenarios.

I. INTRODUCTION

Self-Driving Vehicles (SDVs) have the promise to revolutionize several industries including people and goods transportation. However, the development of L4+ SDVs has proved to be a significant challenge. Today, the main bottleneck is the vehicle's ability to safely handle the 'long tail' of driving events [1]. World-class SDVs can handle common situations, but can behave unsafely in the many, rarely-occurring scenarios that are encountered on the road.

In the self-driving stack, the *planning* module is most responsible for this bottleneck. It determines what the SDV should do in any given situation. A traditional *rule-based* planning approach selects a trajectory that minimizes a hand-engineered cost function, often through classical optimization-based approaches. In order to improve its performance, engineers must design new terms in that cost function or re-tune their respective weights, for each driving scenario. This process is expensive and scales poorly to new geographies. Unlike perception, planning has benefited little from modern machine learning techniques, which leverage large quantities of data in order to avoid the hand-engineering of rules. Recently, the work of [2] has demonstrated the first machine-learned policies for autonomous driving learned
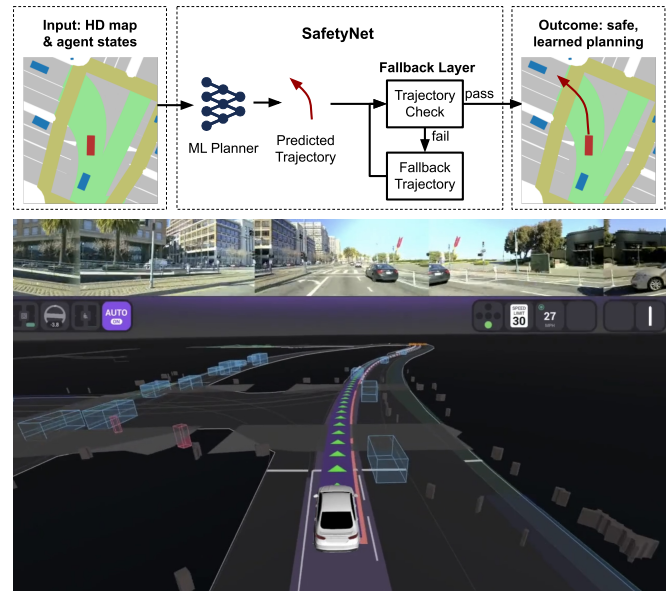
* Equal contribution.
[+] The research was conducted at Woven Planet, Level 5. Correspondence to moritz.niendorf@woven-planet.global.
Videos are available at `safety.l5kit.org`

Fig. 1. *Top:* SafetyNet is the best-of-both combination of ML planning that improves with data, and rule-based safety and legality guarantees from the fallback layer. *Bottom:* an example of SafetyNet deployed to control a real-world self-driving vehicle on the streets of San Francisco.

directly from human demonstrations. These approaches, although they scale much better than the hand-engineering method, do not provide the interpretability and safety guarantees required to safely deploy these systems in production.

In this work we propose SafetyNet: the first autonomous driving system to combine the strengths of an ML planner with the interpretable safety of a lightweight rule-based system, road-tested in busy San Francisco. The ML component is a high-capacity planning policy trained from expert demonstrations, and its performance scales with the amount of training data without the need for costly behavior engineering. To improve system safety, decisions of the ML planner pass through a lightweight *fallback layer*: a simple, rule-based system that tests the decisions against a small set of checks, and can minimally modify them to improve safety if required. This allows SafetyNet to transparently enforce safety and legality constraints.

This combination outperforms ML-only systems and allows us to *safely* deploy an ML planner in the busy streets of San Francisco, constituting the first demonstration of its kind. Our system exhibits a variety of maneuvers such as lane-following, keeping the distance to other vehicles, and

navigating intersections.

Our key contributions are:

- A novel combination of machine learning and a lightweight hand-engineered system to control a self-driving vehicle that learns from data while offering safety and legality guarantees.
- The first evaluation of such a system in the challenging, real-world, urban environment of downtown San Francisco.

II. RELATED WORKS

Trajectory optimization-based planning. Traditional trajectory optimization-based planning systems are widely used in both academia and industry [3]–[6]. Here the motion planning task is formulated as an optimization problem, usually by hand-engineering a cost function. The optimal trajectory is then generated by minimizing this cost using optimization algorithms, such as A* search-based methods [4], [7], sampling-based methods [8]–[10], dynamic programming [6] or combinations thereof that decompose the problem in a hierarchical fashion [11].

However, it is very difficult to hand-craft an objective function that provides a human-like trade-off between comfort, safety, and route progress over a wide variety of driving situations [12]. In comparison, encoding certain hard constraints, e.g. obstacles avoidance, physical feasibility, requires far less engineering. Building a lightweight hand-engineered system whose only purpose is to detect and correct infeasible trajectories is much simpler and more scalable.

Additionally, these hand-engineered approaches do not improve with data, and their performance does not generalize well in highly unstructured urban scenarios. Therefore, tremendous engineering efforts are needed to fine-tune them, in particular when expanding to a new operational design domain (ODD) or new geographies.

Machine-learned planning. Recently, ML planning has gained attention due to successes in deep learning. This approach has the advantage of avoiding hand-crafted rules and scales well with data, thus performing better and better as more data is used for training. Therefore, this approach has great potential to handle a wide variety of driving situations [2], [13]–[15]. Next, we introduce the two most commonly used ML paradigms for motion planning.

(1) Imitation learning (IL). IL is a supervised learning approach in which a model is trained to mimic expert behavior. The first application of IL to autonomous driving was the seminal ALVINN [16] back in 1989, which mapped the sensor data to steering and performed rural road following. More recently, [13], [15] demonstrated end-to-end driving using multiple-camera input alone, but the real-world driving results are limited to simple tasks such as lane follow or urban driving with light traffic. ChauffeurNet [2] proposed to apply IL on a bird's eye view of a scene and use synthetic perturbations to alleviate the covariate shift problem [17], but it is yet to be tested in real-world urban environments. Promising contributions to the agent prediction problem [18]–[20] have the potential to be used for the planning problem, but their performance in a closed-loop setting is not evaluated.

(2) RL & IRL. Reinforcement learning (RL) is well-suited for sequential decision processes such as self-driving as it handles the interaction between the agent and the environment. Several methods [21]–[23] have been proposed to apply RL to autonomous driving. In particular [21] proposes combining learned and rule-based components, similarly to us, but the reported results are only from simulation. On the other hand, inverse reinforcement learning (IRL) [24], [25] is another popular ML paradigm applied to autonomous driving, which infers the underlying reward function based on expert demonstrations as well as a model of the environment. However, all these methods are yet to be evaluated in real-world urban driving.

The ML planning approaches introduced above, although very promising, do not provide safety guarantees, which prevent them to be deployed at scale in the real world. We are inspired by this paradigm but aim to mitigate this limitation by the SafetyNet proposed in this paper.

Hybrid approaches. The combination of ML and traditional motion planning techniques falls mostly into two categories: *ML-based heuristics*, which are leveraged to improve traditional planning algorithms, e.g. in terms of speed up [26]–[29]. *Modular approaches*, where expert planners are leveraged to generate the trajectory candidates, e.g., by evaluating trajectories against a ML-based cost volume [14], [30]. The latter of these works, [30] also provides safety guarantees based on imposing a very high cost on trajectories leading to a potential collision. These safety guarantees were not however verified in the real world.

Another specific area of research that has emerged in this field is the study of safety frameworks [31], [32]. While this work is relevant, our goal is not to propose a comprehensive framework for safety, but rather a simple yet effective method that allows for the deployment of a powerful neural network planner that learns and improves with data while ensuring certain safety and legality constraints.

SafetyNet leverages the strengths of the expert system to guarantee certain determinism, legality, and safety rules for specific scenarios while relying on the ML motion planner for nominal trajectory generation.

III. HYBRID ML PLANNING SYSTEM

In this section we describe SafetyNet, our system for combining a machine-learned motion planner with an effective fallback layer to deliver a trajectory planning system for SDVs. The approach uses the best of both worlds, an ML planner that learns to drive from expert driving demonstrations, supported by a rule-based fallback layer to provide interpretable safety constraints, such as collision avoidance and adherence to traffic rules. The fallback layer is simple and scalable because it relies on basic hand-engineered feasibility checks, and corrects infeasible ML trajectories with a simple classical trajectory generator of small scope.

The SafetyNet system is outlined in Fig. 1. It is composed of an ML neural policy network ("ML Planner") $\mathcal{M}$ that
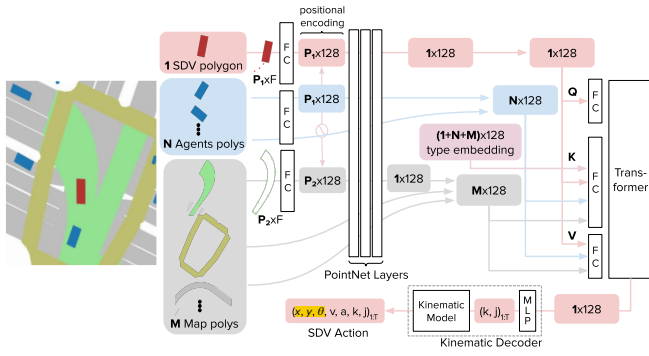
Fig. 2. The neural network architecture of the presented ML planning model is inspired by VectorNet [33]. The vectorized information on each agent and map element is encoded by a PointNet network. This local information is combined by a Transformer into global embedding. The embedding is later translated into actions via kinematic decoder.

takes as input I the environment state around the SDV and produces an intended trajectory $\bar{\tau}$ to be taken by SDV. This trajectory is validated to obey the required constraints and in the case it does not satisfy them the closest trajectory τ^i is taken from a set of safe trajectory candidates.

A. Input and Output

Input representation. The input data is encoded in an ego-centric frame of reference where the SDV is always at a fixed location relative to a frame. As shown in Fig. 2, the input to our model consists of:

1) SDV: current and past poses of the SDV and its size.
2) Agents: current and past poses of perceived agents, their sizes, and object type (e.g. vehicle, pedestrian, cyclist) produced by the SDV's perception system.
3) Static map elements: road network from High Definition (HD) maps including lanes, crosswalks, stop lines, localized using the SDV's localization system.
4) Dynamic map elements: traffic light states, and static obstacles detected by the perception system (e.g. construction zones).
5) Route: the intended global route for the SVD to follow.

We use a vectorized input representation based on [33] to featurize each element into vector sets. Each vector includes pose features - relative to the SDV pose - as well as additional features (e.g. element type, time of observation, etc). The number of feature vectors per element varies due to history availability or number of geometry points. The number of total elements in the frame varies but cannot exceed network capacity. We limit the number of elements based on a region-of-interest around the SVD.

Output representation. We define a trajectory τ as a sequence of T discrete states separated uniformly in time by Δt. Each state s_t is defined as:

$$s_t = \{x_t, y_t, \theta_t, v_t, a_t, k_t, j_t\}. \tag{1}$$

where x_t, y_t, θ_t correspond to the pose of the rear axle of the SDV w.r.t. a fixed coordinate frame at time t and v_t, a_t, k_t, j_t correspond to the velocity, longitudinal acceleration, curvature and jerk respectively.

B. ML planner

The ML planning component of our system takes the input I capturing the states around the SDV and outputs the trajectory $\bar{\tau}$ to be executed.

Model architecture. Inspired by [33], our model is built on a hierarchical graph network-based architecture. It consists of a PointNet-based [34] local subgraph for processing local information from vectorized inputs and a global graph using a Transformer encoder [35] for reasoning about interactions over agents and map features. Details of the architecture are represented in Fig. 2.

To ensure the predicted trajectories are physically feasible, we introduce a kinematic decoder, which models the vehicle kinematics using a unicycle model [5]. The decoder includes a 3-layer multilayer perceptron (MLP) that takes in the transformer encoder embedding and predicts longitudinal jerk $j_{1:T}$ and curvature $k_{1:T}$, for each time step within the prediction horizon T. Then a kinematic layer takes the predictions as well as the current ego state to roll out the next state of the ego:

$$s_{t+1} = f(s_t, k_t, j_t, \gamma), \tag{2}$$

where f is the update function of the kinematic model, γ comprises a set of parameters for vehicle kinematic constraint, including the maximum allowed jerk, acceleration, curvature and steering angle, which are used to clip controls to ensure physical feasibility. Following the deep kinematic model introduced in [36], we implement f as follows:

$$s_{t+1} = s_t + \dot{s}_t \Delta t, \tag{3}$$

where the state derivatives are computed as follows:

$$\dot{x}_t = v_t \cos \theta_t, \ \dot{y}_t = v_t \sin \theta_t, \\ \dot{\theta}_t = k_t v_t, \ \dot{v}_t = a_t, \ \dot{a}_t = j_t. \tag{4}$$

Training framework. We use imitation learning to train a driving policy that mimics expert driving behavior by minimizing the L1 loss between the poses generated by the model and the ground truth poses. Following [2], we include perturbations to extend the distribution of states seen during the training and thus reduce the impact of the covariate shift [16], [17]. Although previous work used a pre-solver to smooth the target trajectory after applying perturbations, we can skip that thanks to the fact that we are using a kinematic decoder. Instead, we can simply penalize large values of jerk and curvature to reduce jerk and improve driving comfort. The final loss is then:

$$L = \sum_{t=1}^{T} \|p_t - \hat{p}_t\|_1 + \alpha \|k_t\|_2 + \beta \|j_t\|_2, \tag{5}$$

where p_t is the predicted pose (x_t, y_t, θ_t) at time t, $\hat{p}_t$ is the target pose, and α and β are hyperparameters.

C. Fallback Layer

After generating an ML trajectory, our system evaluates it along several dimensions for dynamic feasibility, legality, and collision probability, and determines a trajectory label

{*Feasible*, *Infeasible*}. The details of these checks are described below.

Dynamic feasibility. We evaluate whether the input trajectory remains within a feasible envelope characterized by the SDV's dynamics limits. Concretely, we evaluate each trajectory state and check whether the parameters, including longitudinal jerk, longitudinal acceleration, curvature, curvature rate, lateral acceleration, and steering jerk (curvature rate $\times$ velocity) are within reasonable bounds.

The bounds for those parameters were obtained from real-world vehicle testing. In practice, we typically use more conservative limits for jerk, longitudinal acceleration, and lateral acceleration to remain within comfortable limits.

Legality. For a given trajectory, we evaluate whether it is violating the traffic rules. A trajectory will be labeled as *Infeasible* if any of the following violations happens:

- Running the stop sign,
- Violating the right of way,
- Running a red traffic light,
- Leaving the drivable surface.

Collision likelihood. We check each ML trajectory state for collisions with the predicted poses of other agents. Agents predictions are generated by an in-house prediction module. The prediction module takes into input agents and SDV at each scene frame and can implicitly consider interactions between them. Collision detection is performed by rasterizing future agent predictions and checking for overlaps with planned ego poses. Additionally, we also check for longitudinal distance, time-to-collision, and time headway violations along the trajectory. The trajectory is labeled *Infeasible* if any of the collision likelihood checks fail.

Fallback trajectory generation. The ML trajectory is directly executed when labeled *Feasible*. If the trajectory is labeled *Infeasible*, we select a feasible fallback trajectory as close as possible to the ML trajectory.

For this we use a classical optimization-based trajectory generation method, based on [37], which generates a number of lane-aligned trajectory candidates τ^i. These candidates consist of speed keeping, distance keeping, and emergency stopping maneuvers. Our implementation can be made less conservative and expanded to other fallback behaviors of interest. However, such increase in scope for the fallback trajectory will come at a trade-off of extra engineering effort.

Each of the generated trajectories is checked for feasibility as described above and the trajectory candidate which is most similar to the ML trajectory is selected for execution:

$$\tau = \operatorname*{argmin}_{\tau^i} \|\bar{\tau} - \tau^i\|_2. \tag{6}$$

For qualitative examples of how the fallback layer affects the trajectory refer to the Section IV-D.

IV. EXPERIMENTS

In this section we evaluate our system across several dimensions: (a) the performance of the ML planner in simulation when trained on an increasing amount of data (no fallback layer), (b) the effectiveness of the fallback layer in simulation, and (c) the performance of SafetyNet in the real world when controlling a real vehicle in San Francisco.

A. Data

We created an in-house dataset to train and evaluate our system: 300 hours for training, 80 hours for testing - of urban driving collected in Palo Alto and San Francisco. It contains a wide range of driving scenarios in a densely populated urban environment. The dataset consists of 25s *scenes*, which capture the perception output, the SDV trajectory, and HD maps. It takes $\sim$5 hours to train the model on the full 300 hours using 64 NVIDIA V100 GPUs.

B. Metrics

We validate our planner by running a large-scale real-world driving dataset on our in-house simulator. During simulation, we execute the hybrid planner and motion controller modules, and simulate the SDV motion. The vehicle model used is calibrated to the real-world SDV. Since the simulated SDV pose can diverge significantly from the originally logged pose along the dataset scenes, we allow the other road *agents* to be reactive in their longitudinal behavior, avoiding collisions while preserving their trajectories from the dataset. The reactive agents interface is built such that the same prediction library is used in real-world and closed-loop evaluation (any prediction performance differences between logged and reactive agents is monitored in-house). Our in-house simulator is further validated to guarantee that neither collision rate nor discomfort braking are underestimated.

In order to evaluate the *closed-loop* performance of the model, each scene in the test set is replayed in simulation with the SDV following the hybrid planner output for the full duration of the scene. The following binary events are recorded:

1) *Collisions*: the simulated SDV is <5cm from the road boundaries, static obstacles, or agents.
2) *Close-calls*: the simulated SDV has no collision, but either gets within 25cm of another agent, has a time-to-collision <1.5s, or has a time headway <1s.
3) *Discomfort braking*: the simulated SDV's jerk drops below -5m/s^3.
4) *Passiveness*: the simulated SDV travels slower than its behavior in the dataset by $< -5\text{m/s}$ and it is spatially behind the SDV's logged dataset position.
5) *Off road events*: the simulated SDV deviates from the dataset route center line by >10m.

The events recorded for each scene are aggregated and normalized by the number of miles driven in the simulations. All metrics are reported as the total number of events per 1000 miles.

To assess the ability of the planning model to fit to the dataset trajectories, we evaluate its *open-loop* performance. To this end, we compute the Average Displacement Error (ADE) between the positions from the ML trajectory output at each frame and the logged SDV's positions. Results are presented in TABLE I).

Finally, we test how the system performs in the real world by deploying it in downtown San Francisco. We evaluate the ML planner performance via how often the fallback trajectory is used and we provide qualitative examples.

900

	Average Displacement Error [m]			
Training set size	@1s	@2s	@3s	@4s
5h	0.24	0.90	1.70	2.52
50h	0.14	0.51	0.98	1.53
150h	0.12	0.40	0.77	1.22
300h	0.11	0.37	0.73	1.18

TABLE I. Open-loop performance improves with the training set size.
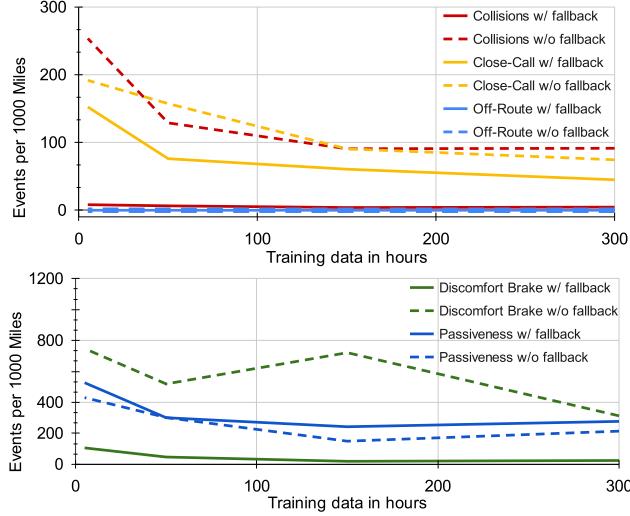

Fig. 3. Safety (*top*) and comfort (*bottom*) events as a function of dataset size for the ML planner (with and without fallback layer). The addition of the fallback layer significantly improves every metric, save passiveness.

C. Effect of dataset size

For this experiment, the fallback layer is disabled, and the ML planner's trajectory is always executed. We evaluate the performance of various ML planners trained with varying dataset sizes: {5, 50, 150, 300} hours in simulation.

Looking at the dashed lines in Fig. 3 (w/o fallback) we observe that the closed-loop performance across all safety and comfort metrics increases with training set size, and that more data should continue improving performance, albeit slowly. Models trained on <50h produce unstable driving policies that have significantly more collisions, off-route events, or even cases where the SDV remains completely still (passiveness). On the full dataset size, performance improves significantly and the learned driving policy is able to reduce collisions, close-call and discomfort brakes, and reliably follows the route. In terms of passiveness, performance plateaus, which we attribute to causal confusion inherent to open-loop training [38]. Similarly, the ADE for open-loop predictions improves with more training data (TABLE I).

D. Effect of fallback layer

We evaluate the effectiveness of the fallback layer in simulation by comparing the SafetyNet performance with, and without the fallback layer enabled. Here the ML planner is trained on the full 300h dataset. As shown in TABLE II, collisions and discomfort braking are very significantly reduced with the fallback layer enabled. This demonstrates the value of the fallback layer and its importance for real world deployment. Crucially, we see that the solid lines in Fig. 3 (w/ fallback) are close to zero regardless of the maturity of
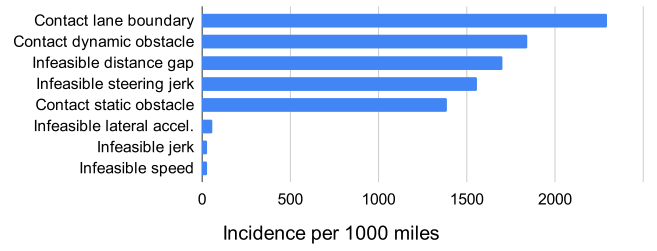

Fig. 4. The distribution of causes for a fallback trajectory to be used over the ML Planner trajectory.
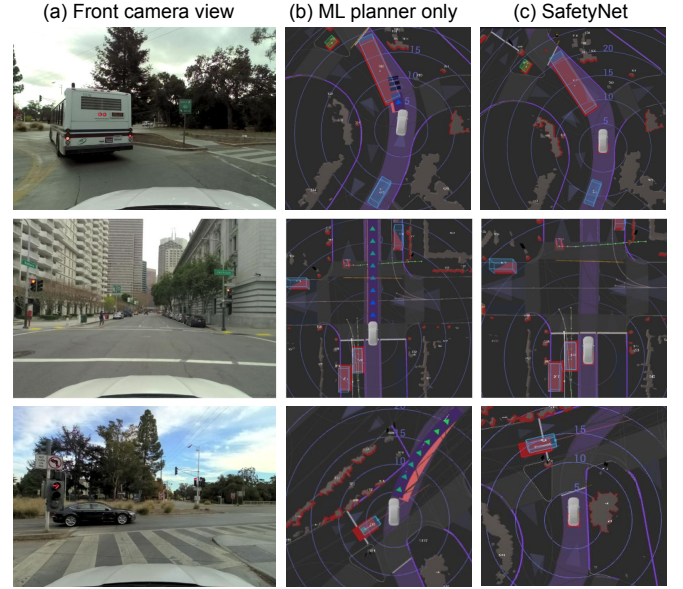

Fig. 5. Examples of when the fallback layer prevents unsafe behavior caused by ML planner. *[Top]* avoiding collision with a bus, *[middle]* running a red light, and *[bottom]* ensuring yielding to right-of-way vehicle. We report the unsafe trajectory (col. b), and SafetyNet safe trajectory (col. c).

the ML planner. This indicates that we can safely deploy not only well-performing ML planners, but even immature ones (trained on <50 h of data), thus facilitating faster development and evaluation cycle.

Passiveness increases when the fallback layer is enabled, due to the SDV driving more conservatively. We see it is a necessary trade-off for reducing collisions and close calls.

In Fig. 4 we show the distribution of events that trigger the fallback trajectory to be used during simulation. The main causes for fallback behaviors are the ML trajectory leaving the drivable surface, contacting dynamic agent predictions, infeasible distance gap with other agents, infeasible steering jerk, and contact with static obstacles. By manually triaging these ML planner failure cases, we see (a) contact lane boundary issues are a result of the ML planner cutting corners too closely, and (b) collisions are caused by the network not paying enough attention to the size of the large vehicles (Fig. 5, *top*). Moreover, near traffic light intersections the ML planner occasionally generates trajectories that do not properly yield to oncoming traffic (Fig. 5, *bottom*). It is caused by the synthetic perturbations applied to the ego poses during training. Note that in all these examples, the fallback layer successfully prevented collisions from happening.

Planner Type	# events per 1k miles				
	Collisions	Close Calls	Discomfort Braking	Passiveness	Deviation from Route
ML Planner	91.5	74.5	312.8	213.9	0.0
ML Planner + Fallback Layer	4.6	45.0	24.2	277.0	0.0
Change Rate	-95%	-40%	-92.3%	+29.5%	0%

TABLE II. Comparing the performance of an ML-Planner-only approach vs SafetyNet, in closed-loop evaluation. This shows that SafetyNet significantly reduces collisions, close calls, and discomfort breaking, at the expense of more passiveness. See Fig. 5 for some qualitative examples.

Fig. 6. SafetyNet failure case in simulation. Two simultaneous situations - SDV driving close to dividing line, and a sudden velocity change of the oncoming vehicle - combine to reveal a limitation in the implementation.
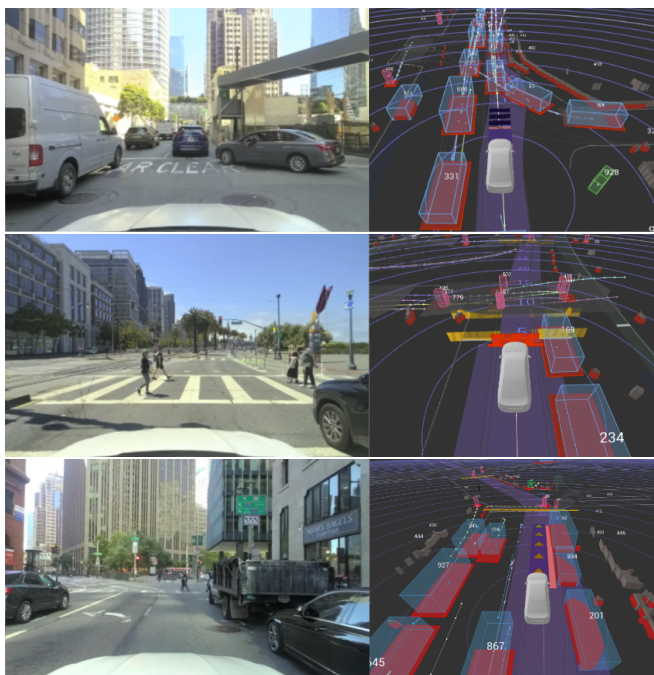

Fig. 7. Examples of successful ML planner behavior (no fallback trajectory was triggered) in the real world. *Top:* merging, *middle:* yielding to pedestrians, and *bottom:* nudging around a parked car.

E. Failure cases and limitations

There is still a small fraction of ML planner failures not caught by SafetyNet. This is mostly because the current fallback layer implementation does not limit the SDV proximity to lane boundaries, and does not incorporate agent prediction uncertainty. In Fig. 6 the SafetyNet fails to keep a comfortable lateral distance to an oncoming vehicle.

The current fallback trajectory matching assumes that the ML trajectory is "always almost correct", which is not always the case. Better trajectory matching techniques could be explored, potentially leveraging an uncertainty output in the ML planner. However, the onus should fall on the ML planner to improve with data. A simpler fallback implementation with high issue discoverability is ideal.

F. Real world testing

Finally, we extensively tested SafetyNet with ML planner (trained on 300 hours), in the real world, in densely populated downtown San Francisco, under the supervision of human safety drivers. During the 150+ mile public road testing, the model successfully performed a wide variety of challenging maneuvers including lane-following, merging, yielding to pedestrians or nudging around parked cars (Fig. 7). At the same time, the fallback layer assured the safety of the overall system, taking over around 7.9% of the total driving time. This confirms our hypothesis that although the ML planner is able to perform complex maneuvers and drive safely for most of the time (>90%), an additional layer of safety is required in certain situations. We refer the reader to the accompanying video for a more thorough analysis. We observed a total latency of <180ms@P95 on the entire planner module running Safetynet.

V. CONCLUSIONS

We present SafetyNet, a method for combining ML planners with a rule-based system fallback layer to provide safe driving in challenging, real-world urban environments. We demonstrate very significant improvements in safety and comfort metrics compared to a purely ML-based system, both in simulation as well as in challenging San Francisco streets. This approach makes it possible to safely use learned planners in the real world, and benefit from their ability to improve with more data to handle more complex situations than their purely rule-based counterparts. We believe that teams starting to explore ML planning methods, or those with state-of-the-art ML planning solutions [2], [14], [15] would benefit from incorporating a SafetyNet system. In effect, SafetyNet may facilitate the development of ML-based planners and their wider adoption in the AV industry.

We see many exciting opportunities for further development. The fallback layer can be refined to be less conservative. Regarding the ML planner, the imitation learning approach presented is relatively simple. It can be improved by drawing from recent advancements in model-based Reinforcement Learning (RL) [39], offline RL [40], or closed-loop training with data-driven simulation [41].

VI. ACKNOWLEDGEMENTS

REFERENCES

[1] A. Jain, L. D. Pero, H. Grimmett, and P. Ondruska, "Autonomy 2.0: Why is self-driving always 5 years away?" 2021.

[2] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worsts," in *Proceedings of Robotics: Science and Systems XV*, 2019.

[3] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Hähnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, and S. Thrun, "Junior: The stanford entry in the urban challenge," in *The DARPA Urban Challenge*, 2009, pp. 91–123.

[4] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer, 2009.

[5] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[6] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[7] Z. Ajanovic, B. Lacevic, B. Shyrokau, M. Stolz, and M. Horn, "Search-based optimal motion planning for automated driving," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4523–4530, 2018.

[8] J. J. K. Jr. and S. M. Lavalle, "Rrt-connect: An efficient approach to single-query path planning," in *Int. Conf. on Robotics and Automation*, 2000.

[9] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[10] T. Bandyopadhyay, K. S. Won, E. Frazzoli, D. Hsu, W. S. Lee, and D. Rus, "Intention-aware motion planning," in *Algorithmic Foundations of Robotics X*, E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, Eds., 2013.

[11] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi, "A behavioral planning framework for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 458–464.

[12] Y. Chang, S. Omari, and M. S. Vitelli, "Systems and methods for determining vehicle trajectories directly from data indicative of human-driving behavior," Jun. 10 2021, uS Patent App. 16/706,307.

[13] 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," 2016.

[14] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun, "End-to-end interpretable neural motion planner," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[15] J. Hawke, R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Micklethwaite, N. Griffiths, A. Shah, and A. Kendall, "Urban driving with conditional imitation learning," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 251–257, 2020.

[16] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, vol. 1, 1989.

[17] S. Ross, G. J. Gordon, and J. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, 2011.

[18] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," in *Conference on Robot Learning*, 2020.

[19] H. Zhao, J. Gao, T. Lan, C. Sun, B. Sapp, B. Varadarajan, Y. Shen, Y. Shen, Y. Chai, C. Schmid *et al.*, "Tnt: Target-driven trajectory prediction," *arXiv preprint arXiv:2008.08294*, 2020.

[20] J. Ngiam, B. Caine, V. Vaudevan, Z. Zhang, H.-T. L. Zhengdong, J. Ling, R. Roelofs, A. Bewley, C. Liu, A. Venugopal, D. Weiss, B. Sapp, Z. Chen, and J. Shlens, "Scene transformer: A unified multi-task model for behavior prediction and planning," *arXiv preprint arXiv:2106.08417*, 2021.

[21] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[22] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to drive a real car in 20 minutes," in *2007 Frontiers in the Convergence of Bioscience and Information Technologies*. IEEE, 2007, pp. 645–650.

[23] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[24] B. D. Ziebart, A. L. Maas, J. Bagnell, and A. Dey, "Maximum entropy inverse reinforcement learning," in *AAAI*, 2008.

[25] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv: Learning*, 2015.

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

[27] O. Arslan and P. Tsiotras, "Machine learning guided exploration for sampling-based motion planning algorithms," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 2646–2652.

[28] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[29] H. Pulver, F. Eiras, L. Carozza, M. Hawasly, S. Albrecht, and S. Ramamoorthy, "Pilot: Efficient planning by imitation learning and optimisation for safe autonomous driving," *arXiv preprint arXiv:2011.00509*, 2020.

[30] S. Casas, A. Sadat, and R. Urtasun, "Mp3: A unified model to map, perceive, predict and plan," in *IEEE/CVF International Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 403–14 412.

[31] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *CoRR*, vol. abs/1708.06374, 2017. [Online]. Available: http://arxiv.org/abs/1708.06374

[32] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.

[33] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "VectorNet: Encoding HD maps and agent dynamics from vectorized representation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[34] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3d classification and segmentation," *arXiv preprint arXiv:1612.00593*, 2016.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[36] H. Cui, T. Nguyen, F.-C. Chou, T.-H. Lin, J. Schneider, D. Bradley, and N. Djuric, "Deep kinematic models for kinematically feasible vehicle trajectory predictions," 2020.

[37] M. Werling, S. Kammel, J. Ziegler, and L. Gröll, "Optimal trajectories for time-critical street scenarios using discretized terminal manifolds," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 346–359, 2012.

[38] P. de Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[39] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," *arXiv preprint arXiv:2006.16712*, 2020.

[40] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[41] 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," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.