Written: 09/2025

I worked at Cruise for roughly six years, primarily in two domains described below.

ML Planning (2022-2025)

Autoregressive driving model

The primary model I worked on the last ~1.5 years was one main autoregressive driving model. The objective of the model was similar to the primary VLAs of today, which is to directly produce the actions the robotic system should execute. The representation we operated in was that of trajectories, so the model output was a sequence of x,y poses at discrete time steps into the future.

The actual output representation we used was *discretized actions*. That is, we produce a vocabulary of tokens which correspond to physical actions, like a certain amount of lateral and longitudinal acceleration for example. Given a sequence of these tokens, we can "unroll" them via kinematic models to produce the sequence of x,y poses.

The model was trained in two primary stages: Imitation Learning, and RL.

The imitation learning phase primarily leveraged expert driving, and the task is simply next token prediction similar to LLMs. We had a large corpus where human operators had driven the vehicles around different cities. This corpus can be preprocessed into a token stream based on the discretized actions described above, and we can learn via next token prediction objective and Cross Entropy loss.

The imitation learned model had very good open loop metrics, but the closed loop behavior of the model suffered strong drifting problems. As an example, when the model drifted slightly off course, it would enter a foreign state space and then behave erratically. This is what prompted the RL training stage, which was primarily based on heuristic rewards. These included concepts like not colliding with other objects, staying on the road, and not violating the boundaries of the lane.

The primary methodology was an *on-policy, partially online* learning algorithm, where we used an efficient closed loop simulation with some assumptions. During training, if we make the assumption that the nearby agents behavior is fixed, we can sample and rollout N trajectories from the autoregressive decoder, and compute rewards against those. We used a policy gradient approach, namely A2C. We tried several tricks like entropy bonus to encourage exploration, reward normalization, learning rate scheduling, etc. We also leveraged reward masking, which was the idea to only penalize the model with low reward if the expert driver also does not violate that reward concept. For example, the model should not be penalized for violating the lane boundary if the expert driver also violated

the lane boundary. This allows for the model to violate concepts contextually, which is an important part of driving.

Overall, the model performed quite well in simulation, and we were planning to replace our entire ML Planning system with this new model towards the end of 2024, when the company absorption into GM began and the work was suspended.

Reward Modeling

I also spent ~9 months working on a driving reward model, though this project never achieved strong results and eventually was deprecated. The motivation was to build a strong and robust driving reward model that we could use for two reasons: 1) as a metric, and 2) in our RL pipelines to optimize the original policy model.

The approach was very similar to the RLHF approaches used in language modeling. We collected driving events from our fleet of physical vehicles. We then simulated multiple different policies on that same road event. After that, we had labelers look at two of the simulated segments (one of which might be the originally collected road policy), and rate which one was better. We opted for categorical labeling (A > B, B < A, A = B) because we felt it was fundamentally easier to assess which is preferable rather than deciding if any individual instance of driving is of acceptable quality.

After that, we trained the reward model. It did work to some degree, but the strongest signals it learned were that making progress is preferable as long as no collisions occur. It was not very robust, and failed to correctly identify more nuanced behavior like lane positioning or smoothness of kinematics and steering. I have several ideas why: 1. The data volume was too small. We eventually were able to collect about ~100k labels, each representing a pair of events. 2. The signal the label provides is incredibly sparse relative to the input. 10-20s of driving has a lot of information, and the model is tasked with not only learning which of the two is better, but also the temporal attribution in order to generalize well. This makes the data scale even smaller in comparison to other domains. 3. The label quality is fundamentally limited for two reasons. First, it is an inherently subjective task and reasonable humans will have different opinions on how to drive well. Second, the labeler is operating on limited information, since they are not physically in the car to experience the driving. It is often very hard to judge the quality of nuanced driving behavior from watching the camera recordings on a screen.

The model did provide some value as a metric, but did not work in our RL pipelines. It was easily reward hacked, and I remember we observed large increases in the reward during training, but these resulted in no demonstrable change in driving behavior. Eventually the project was canceled and the model was deprecated.

Trajectory Prediction (2019-2022)

Rudimentary ML System

When I joined Cruise the prediction system leveraged ML, but did so in a rudimentary way. It was formulated as an N-class classification problem to classify over maneuver types, and we used XGBoost as the model. Everything was auto-labeled based on temporal perception information to decide what maneuver had been executed. This was largely based on lane association and geometry of maneuver. During inference, we had simple vehicle dynamics models which took the footprint and predicted maneuver, and "rolled out" the footprint over the predicted time horizon. This was incredibly brittle.

Modern Deep Learned System

I spent about 3 years total working on an overhauled system. We took ~6 months to develop the first iteration, and then I spent ~2 years after that working on various different aspects of it.

The model looked a lot like some of the published papers from Waymo at the time, like TNT for example. The primary input was one rasterized "image" with many channels for the following information: road network/HD map related information, other agent information coming from the perception system, information about the vehicle being predicted. The model also consumed other state information like the target agent kinematics, object type (pedestrian, bike, vehicle), etc.

This model was trained using Huber loss, and we went through a variety of multi-modality iterations and configurations. I personally worked on the design, iteration, and improvements of this model related to bike predictions. I also designed and implemented an overhauled version of the model based on a two-stage prediction concept, which was common in the literature at the time.

The idea is to first learn to predict an occupancy grid. This is learned used a discretized grid, and classification loss for which cells the agent actually occupies at various timesteps in the future. The second step is to sample from the occupancy grid N different "trajectory anchors". These anchors then are used as conditionings for the trajectory prediction itself. This model design improved both the recall@k and L2 distance@1, which were the two competing and important metrics we used at the time.