# Technical Report
# MBAPPE: Mcts-Built-Around Prediction for Planning Explicitly

Raphael Chekroun[1,2,3] & Thomas Gilles[1]
[1]Mines Paris - PSL University, Center for Robotics, Paris, France
[2]Valeo Driving Assistant Research, Créteil, France
[3] Department of Civil and Environmental Engineering, University of California, Berkeley, USA

June 8, 2023

**Abstract**

MBAPPE leverages a Monte-Carlo Tree Search (MCTS) [1, 2] to explore a partially learned environment model of Nuplan [3] for explicit trajectory planning. We call this trajectory planning 'explicit' as MBAPPE actually infers consecutive actions, and then integrate them into a trajectory, which makes more physical and kinematic sense.

## 1 Introduction

While neural networks provide a powerful and flexible tool for learning to drive using supervised labels [4], they remain limited in the long-term understanding of the consequences of their actions, and may not comprehend the full scope of interactions with the map and other agents. Reinforcement learning (RL) based methods [5] try to incorporate long-term returns of such consequences in the training of these networks, but this causal understanding remains implicit and not guaranteed, and RL training is most often very sample inefficient.

Our approach aims to get the best of both worlds by using a neural network (NN) prior to guide an MCTS model into explicitly exploring the consequences of actions [6], validating the NN trajectory if it respects driving constraints, or exploring new actions if required. The main challenge in running a MCTS is that we assume environment transitions to be deterministic and perfectly known. While this is true for the displacement of the ego vehicle given its actions, and for the update of the map that remains the same, other agents will also move on their own accord. In order to have a realistic world model, we modify the NN (based on the UrbanDriver[7] open-loop baseline) to predict all the other agents future trajectories in addition to the ego's. This way we get an approximate of the future transitions that enables us to roll out the consequences of our chosen actions on multiple timesteps.

## 2 Method

### 2.1 Features generation

The environment is made of two categories of features:

- Known features:
  - The map information, encoded via the Nuplan default $VectorSetMapFeatureBuilder$
  - Map information is augmented with traffic light information on the route for the ego vehicle, as those are not loaded by the original $VectorSetMapFeatureBuilder$
  - Static object such as plots or pedestrians (considered static at a given timestep as they are slow)

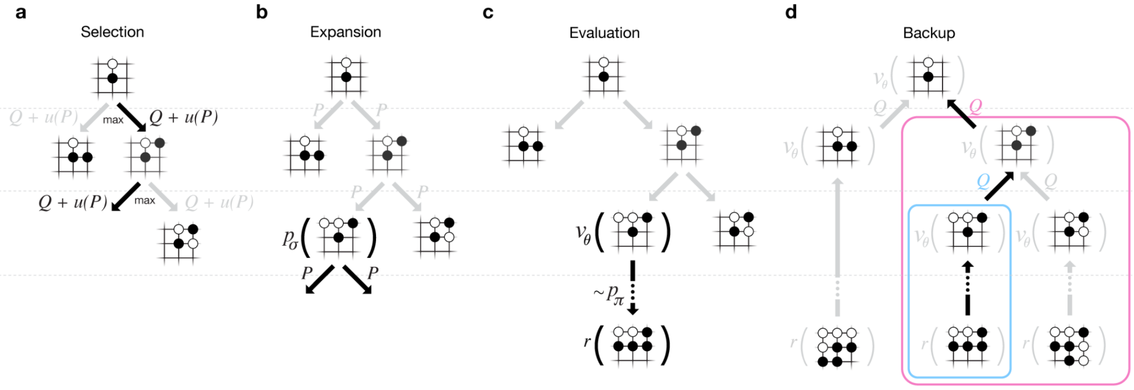

Figure 1: **Monte-Carlo tree search inpired by AlphaGo** [8]**.** a) Each simulation traverses the tree by selecting the edge with maximum action-value $Q$, plus a bonus $u(P)$ that depends on a stored prior probability $P$ for that edge. b) The leaf node may be expanded; we compute probabilities from continuity constraints and simple assumptions and the output probabilities are stored as prior probabilities $P$ for each action. c) At the end of a simulation, the leaf node is evaluated by explicitly computing a reward $r$ checking different main components. d) Action-values $Q$ are updated to track the mean value of all evaluations $r()$ in the subtree below that action.

- Learned features: Estimated trajectories of other agents given by the NN prediction.

At each time-step, a neural network (based on the UrbanDriver open-loop baseline) infers an estimation of the ego trajectory and of the trajectories of every other agents around the ego.

This information is fed to the MCTS. The ego trajectory is used as a prior to guide the first step of exploration of the MCTS. At a given Nuplan simulation time-step, the MCTS explores its possible actions and internally simulates the evolution of the environment to check how those explored actions will impact its driving performances (driving out of area, check for collisions with static objects, check collisions with other agents thanks to their estimated trajectory, etc).

Our MCTS is based on a kinematic bicycle model of the vehicle. Actions are defined as a tuple $(acceleration, \theta_{steering})$. Accelerations and steering angle are discretized in 13 values each, in the respective range of $[-3, 3]$ m.s$^{-2}$ and $[-\pi/4, \pi/4]$ rad. These value have been selected by an in-depth analysis of the distribution of actions in the Nuplan dataset. Actions are integrated every 0.1s.

The exploration phase is inspired by AlphaZero [6] exploration method, represented in Figure 1[1].

## 2.2 Tree steps

Each tree node stores 3 values: Q the expected return, P the action prior and N the number of visits.

- **Selection**: We follow the AlphaGo PUCTS formula to select the next action following a trade-off between the exploitation of Q and the exploration of unvisited nodes with low N. At a node state $s$ the action $a$ is chosen using the following formula:

$$a_t = argmax_a Q(s,a) + c_{puct} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \quad (1)$$

- **Expansion**: We expand leaf nodes by all physically feasible actions from the state of the leaf node. The prior P is given by:
  - If the leaf node is the root: a Gaussian centered of the action derived from the NN ego prediction
  - Else: a Gaussian centered on constant speed and centered steering

[1] Figure from Alphago paper [8]

Additional continuity constraints are added to have successive actions be close to each other and respect realistic physics.

- **Evaluation**: We consider that driving rewards are rather short term (crash or not, exit road or not) compared to Go/Chess ones (is going to win in 100 moves). Therefore they do not need to be bootstrapped by a learned value network as in AlphaGo, but rather can be evaluated at the current step by checking for them directly. Our computed reward $r_t$ at state $s_t$ is made of these main components:

  – Progress: distance advanced since the last node, normalized by maximum allowed speed limit ([0,1])

  – Collision: penality for collision with car (-5), pedestrian (-3) or object(-2)

  – Route: -0.5 if the vehicle is not on the road block ids defining the route

  – Drivable area: -1 if the vehicle is not on the drivable area

  – Center of the road:

    ∗ -sin($\delta$)/2 where $\delta$ is the angle difference between the ego heading and the closest centerline heading

    ∗ -d/2 where $d$ is the distance between the ego position and the closest centerline

  We define our own functions for checking for collisions and drivable area in a vectorized and fast manner, although they may be slighlty less accurate than the NuPlan evaluation functions.

- **Back up**: We update the Q values using the cumulative reward as in MuZero [9]:

$$\begin{aligned} G^k &= \sum_{\tau=0}^{l-1-k} \gamma^\tau r_{k+1+\tau} \\ Q\left(s^{k-1}, a^k\right) &:= \frac{N\left(s^{k-1}, a^k\right) \times Q\left(s^{k-1}, a^k\right) + G^k}{N\left(s^{k-1}, a^k\right) + 1} \\ N\left(s^{k-1}, a^k\right) &:= N\left(s^{k-1}, a^k\right) + 1 \end{aligned} \tag{2}$$

  We use a discount $\gamma$ of 1.

Since our computing time bottleneck is in the evaluation part, and more precisely in the checking for collisions and drivable area, we do not evaluate at every timestep to speed up the MCTS process time. Instead we only evaluate nodes every 10 nodes (every 1 second) and back-propagate their values in the previous un-evaluated nodes.

## 2.3 Stitching to the NN trajectory for better open-loop performance

Since the MCTS returns by design a good planning trajectory but not necessarily a trajectory close to the collected data, as a NN would and since the MCTS usually doesn't go as deep as the 8 seconds imitation horizon, we stitch the first 3 seconds planned by the MCTS to the last 5 seconds predicted by the NN to obtain a good mixed performance.

## 2.4 Our motivation

One of the main advantage of our method is that with very simple instructions incorporated in a reward function (go forward, don't collide, stay on the route, stay on the road) and a pretty vague prior (the MCTS also works very well without any NN prior) we obtain a very realistic working planning, without resorting to hand made rules (such as using the curvature of the current line, or follow the speed of the car in front) or to black boxes not explainable neural networks. This gives our approach plenty of flexibility, adaptability and explainability.

Indeed, decisions of the MCTS are explainable and the internal process which led to the decision can be easily observed and analyzed. More information in Annex.

AI for autonomous driving is not just a matter of science, but also a matter of safety. Therefore, we believe a broadly adopted AI system for autonomous driving should be 1) reliable, 2) explainable and 3) predictable. Hence, the insights given by the internal exploration of the MCTS provide a considerable step forward a safe autonomous driving system, while taking profit from the generalizability of supervised neural networks.

Our work process during the development of this solution consisted in visualizing the thought process of the MCTS in scenarios that resulted in failures and try to understand whether these were due to implementation bugs or poor decision making. With these informations, we were able to iteratively improve our solution and make tremendous progress towards our final performance. We believe that much is still to be done and achieve in terms of potential in this method. Such a process was only made possible with the interpretable nature of the MCTS way of doing exploration and taking decisions.

# 3 Results

## 3.1 Evaluation

All simulations are run with 20 scenarios per type on the *nuplan_challenge_scenarios* settings with the two-stage controller. Results are presented in Table 1.

Notably, the MCTS leveraging the NN prior performs only slightly better than the MCTS without any learned NN prior (each node is just initialized with a very flattened Gaussian centered on null acceleration and straight steering). This can be explained by the poor performance of the NN alone, very far from the reported baseline performance of the challenge. We also believe that we have not explored enough the different ways in which the NN prior could be used.

| Method | Open-Loop | CL Non Reactive | CL Reactive |
|---|---|---|---|
| UrbanDriver open loop | 83.06 | 46.62 | 47.82 |
| MCTS - no prior | -[1] | 77.48 | 76.90 |
| MCTS - NN prior (MBAPPE) | -[1] | **79.09** | **78.09** |

Table 1: Ablation study comparing the accuracy of the NN, the MCTS without a learned prior (nodes are initialized with a flattened Gaussian centered on null acceleration and straight steering) and the MCTS leveraging the NN prior. MCTS methods are not tested on open-loop, as results would be the same as with the NN (cf 2.2). MCTS with the NN prior outperforms other methods. CL stands for Closed-Loop

# 4 Conclusion

This report introduces MBAPPE, a novel method for autonomous driving leveraging an MCTS to explore a partially learned environment.

We present technical details of MBAPPE and the advantages we believe it to bring for autonomous driving. We also detail an ablation study to demonstrate the relevance of our method.

For the next steps, we plan on continuing this research to perfect our idea and try to publish in a conference paper. We also plan on releasing the code.

**Limitations** It is to note that many things still have to be improved in our approach, such as runtime (speed ups by code optimization, pre-compilation or use of C++ could dramatically increase number of possible iterations and therefore performance), reward design or the definition of the priors used for expansion. Also, the NN has been trained only on 7% of the dataset for 28 epochs, which is far from the 10% on 50 epochs from the baseline. Computational resources and time limitations did not allow us to push forward in those directions. Because of those same constraints, most hyper-parameters have not been fine-tuned and have been chosen naively. We found our trained NN to perform way worse than the UrbanDriver published baseline, which therefore provided a very poor prior to the MCTS. This performance gap may be due to training the NN to not only predicting the ego trajectory, but all the agents, making it less specialized.

[1] Actual score is 83.06, c.f. Section 2.2

# References

[1] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," vol. 4630, 05 2006.

[2] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006* (J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, eds.), (Berlin, Heidelberg), pp. 282–293, Springer Berlin Heidelberg, 2006.

[3] K. T. e. a. H. Caesar, J. Kabzan, "Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles," in *CVPR ADP3 workshop*, 2021.

[4] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "Gohome: Graph-oriented heatmap output for future motion estimation," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 9107–9114, IEEE, 2022.

[5] R. Chekroun, M. Toromanoff, S. Hornauer, and F. Moutarde, "Gri: General reinforced imitation and its application to vision-based autonomous driving," *arXiv preprint arXiv:2111.08575*, 2021.

[6] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017.

[7] O. Scheel, L. Bergamini, M. Wolczyk, B. Osiński, and P. Ondruska, "Urban driver: Learning to drive from real-world demonstrations using policy gradients," in *Conference on Robot Learning*, pp. 718–728, PMLR, 2022.

[8] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[9] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

# A Quantitative visualisation

We can observe and interpret the MCTS internal thought process in two different ways.

First, we can plot the decision tree built by the MCTS, as observed in Figure 2, where we display the important values of each node: $Q$ the estimated value, and $N$ the number of visits.

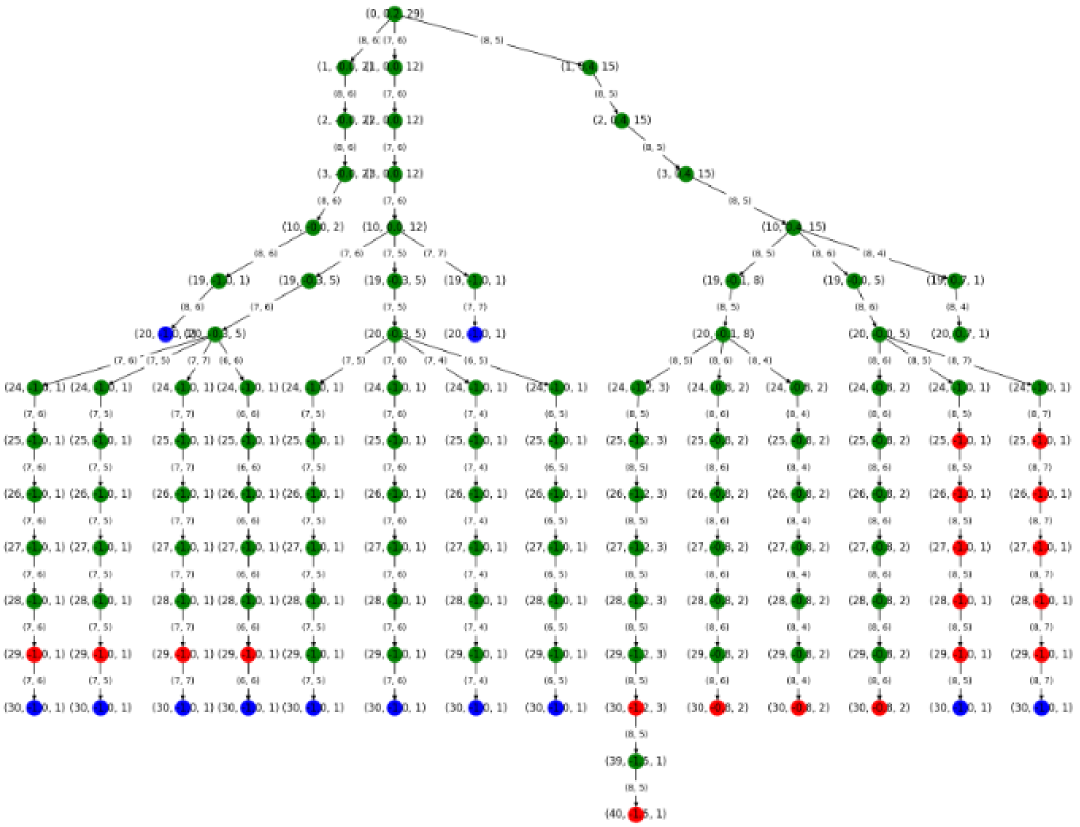

Figure 2: A subset of a decision tree obtained with 256 MCTS exploration steps. More nodes explored leads to higher expected quality of the final chosen actions. Red nodes correspond to driving behavior failure (e.g. collision against vehicle or pedestrian), blue nodes to driving area non-compliance, and green nodes represents a descent behavior. Each node is labelled with a 3-uple consisting of (current MCTS internal simulation time (ms), Q-value, number of visit). Each directed edge is labeled with the action leading from his parent node to his child node. The root node correspond to the present state of the vehicle in the Nuplan simulator.

We can also display a bird-eye-view visualization of the exploration done by MBAPPE for one planning, cf Figure 3. The MCTS can be seen trying different steering values, stopping at nodes exiting the drivable area and pursuing the exploration in the states showing high Q values by staying on the road.

# B Thoughts and Perspectives

**Why use the NN prior only at the root node?** As the MCTS runs its simulation, the explored nodes deviate from the initial trajectory of the prior NN. It would therefore not make sense to keep using the initial prediction as a prior past the root nodes. A solution would be to re-run the NN prior at the new node, as in Alphago, but for compute time efficiency we chose to use a simple kinematic prior in these ulterior nodes.

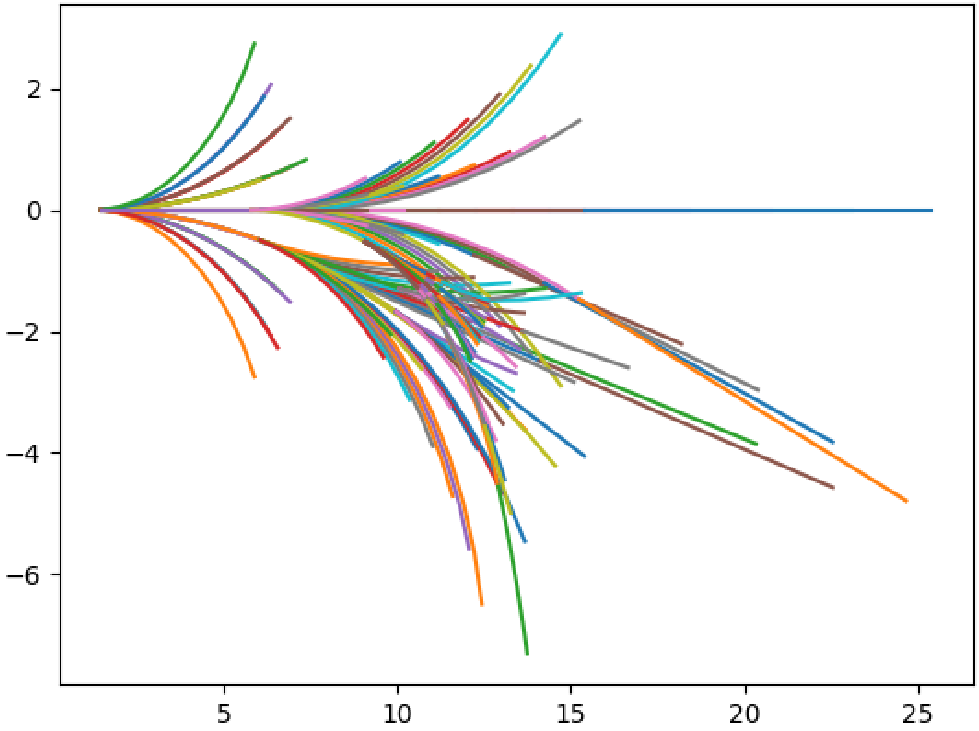

Figure 3: Visualization of the exploration done by MBAPPE in one planning step. We display the bird-eye-view trajectory pieces in xy coordinates. Each color represents a different node. As the road is turning right, the MCTS explores multiple steering configurations to correctly take the turn.

**Continuous improvement** The MCTS can be seen as a improvement operator on top of the prior NN. Once the MCTS provides an optimal planning trajectory for a given state, the policy network can be trained as in AlphaGo to imitate the MCTS output and therefore improve its prior, leading to a self-improving loop.