Technische Hochschule Ingolstadt

Fakultät Elektro- und Informationstechnik
Masterstudiengang AI Engineering of Autonomous Systems

# The Waymo Open Dataset Challenge 2024 - Sim Agents

Seminar paper

Carla Roth, Gizem Bükülmez
Franziska Kofler, Mohamed Ayoub Kefi

Supervisor: Felix Fröhling
Date: June 29, 2024

## Abstract

The Waymo Open Dataset Challenge 2024 represents a significant advancement in autonomous driving simulation, with a focus on simulating realistic joint futures for agents. This paper presents an in-depth analysis of the challenge, highlighting the innovative solutions and advances in machine learning models that emerged from the competition.
We discuss the Joint-Multipath++ (JMPP++) model, which has been restructured to meet the closed-loop requirements of Challenge 2024, emphasising its use of long short-term memory cells and multi-context gating for sequential data processing. Despite hardware limitations and the complexity of a robust model architecture, our team explored various strategies including data pre-processing, feature engineering and reinforcement learning.

The insights gained from our participation, while not yielding the desired results, highlight the need for future entrants to innovate within computational constraints. The paper also highlights the rapid progress in the field, as evidenced by the achievements of the winners, and encourages continued innovation to improve the safety and effectiveness of autonomous vehicles.

# Contents

# 1 Introduction

Initiated in 2020 [1], the Waymo Open Dataset Challenges represent a remarkable achievement in the field of autonomous driving and multi-agent behaviour prediction. This challenges are organized by Waymo LLC, an American autonomous driving technology, subsidiary of Alphabet Inc. and formerly known as the Google Self-Driving Car Project. Waymo, which stands for a new **way** forward in **mo**bility [2], has been at the forefront of self-driving technology since its beginnings.

The Waymo Open Dataset Challenges, which ran for two months, from 18 March 2024 until 23 May 2024 [3], served as a platform for innovators and researchers around the world to showcase their ability to solve complex autonomous driving problems in a competitive environment. The competition was divided into several challenges, each focusing on a unique aspect of autonomous driving. These included Motion Prediction, Occupancy and Flow Prediction Challenge, Sim Agents and 3D Semantic Segmentation. [1]

In recognition of the hard work and innovative solutions presented by the participants, awards were given for each challenge. The awards, up to $10,000 in Google Cloud credits, served not only as a token of appreciation for the participants' efforts, but also as a means to further support their research and development efforts. Teams with the best performing or noteworthy submissions were given the honor of presenting their work at the Workshop on Autonomous Driving at CVPR in June 2024. [1]

Importantly, these challenges also serve as a response to real-world problems that autonomous vehicles face. For example, there have been incidents in which Waymo's autonomous vehicles have been involved in crashes. [4, 5, 6]
These incidents emphasize the complexity of autonomous driving and the imperative need for continuous improvement of the technology. These challenges establish a platform for researchers to take on these issues and contribute to the overall safety and effectiveness of autonomous vehicles.

In this Paper, we will focus on the Sim Agents Challenge. Here, given the agent tracks for the past one second on a corresponding map, and optionally the associated lidar for this time interval, a simulation should be conducted to predict 32 realistic joint futures for all agents in the scene. [1]
In 2023, this challenge saw groundbreaking solutions such as the MultiVerse Transformer for Agent simulation (MVTA)[7] and the Collision Avoidance Detour (CAD)[8], which secured first and third place respectively. These solutions used sophisticated machine learning methods to accurately predict multi-agent behaviour, increasing the safety and effectiveness of autonomous driving[7, 8].

The 2024 edition, having fine-tuned metrics definition now [1], is expected to generate increased interest, as participants are expected to build on last year's successes and introduce novel approaches to the complex problem of predicting multi-agent behaviour[9].

This report presents an in-depth analysis of the last years submissions for the Sim Agents Challenge, with a particular focus on the leading three solutions. The objective is to gain a solid understanding of the strategies applied in order to inform the development of our own contribution in this year's challenge. It must be acknowledged that, despite our best efforts, we were unable to create our own unique solution. Instead, we have focused our energies on refining some of the promising approaches we encountered. This report will provide a detailed overview of the process we have employed, outlining the adjustments we have implemented.

## 1.1 Sim Agents Challenge

As already mentioned in Chapter 1, the Sim Agents Challenge aims to simulate 32 realistic joint futures for all agents in the scene, based on the past one second of the agents' trajectories [1]. Now we will take a closer look.

The use of simulation agents represents an innovative methodology for the advancement of autonomous vehicles (ADV), integrating simulation tests to improve scalability and robustness. The Sim Agents Challenge addresses the limitations of traditional simulation strategies, which typically involve replaying sensor data from real-world ADV experiences and observing the consequences of minor software modifications. Such strategies are intrinsically flawed as they lack the capacity for playback objects to adapt to changes in

ADV behavior. To address this issue, the Sim Agent Challenge proposes the use of simulation agents ("sim agents") that can realistically respond to our actions. [10]

The core concept of the Sim Agent Challenge is treating simulation as a distribution matching problem. In the real world, there exists a distribution of driving scenarios. The goal is to develop a stochastic simulator that operates within the same domain. A simulator is deemed “realistic” when the distributions of the simulator and the real world align.[10]

Real-world scenarios are inherently characterized by randomness, meaning that agents may behave differently given the same initial conditions. However, for each history, only one future is ever recorded. The advantage of the simulator is that it allows for an arbitrary number of sim agent behaviors to be sampled under the same initial conditions. The mismatch between simulated and logged agents is quantified by measuring the likelihood of the real scenarios under the density estimated by sampling the sim agents. As the sim agents become more realistic, the distribution over their behavior should assign a higher likelihood to the logged samples.[10]

To evaluate the performance of the sim agents, a collection of behavior-characterizing metrics is employed. These metrics measure likelihoods over motion, agent interactions, and road/map adherence. To compute these metrics, a collection of 32 different futures, each with scene-consistent interactive agent behavior, is required for each initial scenario, providing 1 second of history [10]. Chapter 1.3 will provide a detailed explanation of these metrics and how they contribute to the evaluation of the sim agents’ performance.

In summary, the Sim Agent Challenge presents a paradigm shift in ADV development, promoting the use of realistic sim agents and distribution matching techniques to enhance the effectiveness of simulation testing.

## 1.2 Waymo Open Dataset

The Waymo Open Dataset is a comprehensive and multimodal dataset specifically designed to address the complexities associated with autonomous driving. It is framed by synchronised high-resolution camera images and LIDAR sensor readings. This setup provides the research community with a diverse set of tools to help develop and refine perception and prediction algorithms.[11]

The dataset features both static and dynamic map elements as well as high quality object tracks. An off-vehicle detection system is used to generate these object tracks, which provide valuable background information about the road environment. A sample of these tracking states is provided every 10Hz. Importantly, the system is set up to flag irregularities when sensor failures could result in measurement unavailability.[11, 12]

For ease of use and targeted research, the Waymo Open Dataset is divided into two main categories: the Perception Dataset and the Motion Dataset. These separate categories allow researchers to focus on relevant areas based on their study needs. Detailed descriptions of these two categories follow in later sections.[11]

### 1.2.1 Perception Dataset

The Perception Dataset consists of projections as well as independent labels for lidar and camera data, ensuring the availability of detailed labeling and segmentation information required for autonomous driving technologies.[11, 13]

The 3D lidar data labels use 3D bounding boxes for vehicles, pedestrians, cyclists and signage. However, there may be unlabelled areas known as "No Label Zone" (NLZ). In contrast, the camera data uses 2D bounding box labels that only cover the visible parts of the objects, with no object tracking between cameras.[11, 13]
Another aspect of this dataset is the 14 key points representing specific parts of the human body such as the nose, shoulders, elbows, wrists, hips, knees, and ankles. These points are both manually labeled and validated in 2D and 3D. This dataset also performs dense labeling for each point in the lidar data in 23 different classes including categories like vehicles, pedestrians, bicycles and more.[11, 13] The Perception Dataset also utilizes a 2D Video Panoptic Segmentation approach for semantic segmentation within 100,000 frames in 5Hz camera images. This dataset offers geographical diversity in locations, weather,

and time of day.[11, 13]
The coordinate system comprises the sensor frame, the global frame, and the vehicle frames. The lidar data from one medium-range and four short-range sensors is provided, which is converted into camera projections.[11, 13]
This dataset also contains camera data with cameras facing in five different directions along with the vehicle's position and speed information. In addition, 3D modeling data and map features such as lane centers, road boundaries, and traffic signs are included in the dataset.[11, 13]

### 1.2.2 Motion Dataset

The Motion Dataset is divided into training, test, and validation sets and supplied in a fragmented TFRecord format files containing the protocol buffer data. Composed of 103,354 segments, each containing 20 seconds of object tracking at 10Hz, this dataset also includes map data covering the segment's area.[11, 14]
Each of the 9-second windows contains sequences with 1 second of past data, a snapshot of the current time, and 8 seconds of future data, resulting in a total of 91 samples. It is worth mentioning that the dataset identifies three object types, namely, vehicles, cyclists, and pedestrians.[11, 14]
Coordinate frames are set in a spherical format with X, Y, and Z axis indicating east, north, and up respectively. All units are measured in meters with each scene using a modified coordinate system origin.[11, 14]
The Motion Dataset includes Lidar data, in version 1.2.0, for the first 1 second of each of the 9 second windows. The Perception Dataset follows the same format once the Lidar data is decompressed.[11, 14]

In WOMD version 1.2.1, the dataset includes camera data covering the first 1 second of 9-second windows. Image markers and placement data from a pre-trained VQ-GAN model are used instead of raw camera images. The scenario protocol and tf.Example protocol contain object traces, object states, dynamic map states, map properties, and more.[11, 14]
The Motion Dataset has been updated in v1.2.0 to include driveway entrances. It provides map polygons for driveway features, parking lot entrances, and other exits from roads.[11, 14]

## 1.3 Waymo Evaluation Metrics

The Waymo Open Dataset Sim Agents Challenge (WOSAC) uses a variety of metrics to evaluate the realism and accuracy of driving simulations. The performance metrics can be categorized into map-based metrics, interaction metrics, and kinematic metrics, which respectively assess proximity to road boundaries, interaction with other agents, and physical properties such as speed and acceleration.[15]

The wide range of scenarios implemented in WOSAC provides a diverse set of real-world driving conditions, facilitating the evaluation of simulation models under different circumstances. This is essential for building robust and versatile simulation models.[15]

Unlike open-loop computations that evaluate metrics on a time-step basis with no simulation feedback, all of the metrics provided in the Waymo Open Dataset operate in a closed-loop system - each metric is the result of active agent performance in simulation.[15] The subset of available metrics includes:

- Route Progress Ratio: This calculates the ratio of the distance covered by the vehicle along the route to the actual trajectory.
- Off-Route: This determines whether the vehicle has diverged from its planned path.
- Off-Road: This identifies if the vehicle has deviated off the mapped road.
- Collision: This detects any physical contact between the vehicle and other entities.
- Displacement Error: This average displacement error (ADE) quantifies the deviation of simulation from original vehicle behavior.

Among the evaluation criteria, the prime focus lies on the "realism meta-metric", which integrates various component metrics to evaluate the simulation's realism. The secondary criterion for breaking a tie is the "minADE".[3]

### 1.3.1 Realism Meta-metrics

The realism meta-metrics serve as robust tools for assessing the performance of simulation models, by evaluating the conformity of a simulation model's behavior to the actual behaviors.[15] These meta-metrics encompass the following aspects:

- Kinematic metrics: Evaluate the vehicle's physical movements.
- Interactive metrics: Assess the accuracy of simulated social driving behaviors through agent interactions.
- Map adherence metrics: Gauge how accurately the simulated vehicle adheres to road and map limitations.

The combined real value of these meta-metrics determines the overall realism of the simulation and helps rank different model performances.

The Waymo Open Dataset Challenge promotes continuous innovation by maintaining a public leaderboard that allows researchers to compare performances. In the 2023 WOSAC, two versions of leaderboards (V0 and V1) were initiated, with V1 showcasing improvements in the calculation of collision and off-road metrics over V0.[15]
The primary evaluation measure remains the realism meta-metric, followed by the much-utilized minADE as the secondary criterion for resolution in case of ties.[3]

# 2 Related Work

## 2.1 Top Three Submissions In 2023's Challenge

The Waymo Open Sim Agents Challenge 2023 saw a variety of innovative solutions for simulating traffic agents. This section gives a brief overview of the top three solutions from the challenge in 2023.
The following table 1 shows the rankings for versions 0 and 1 of the leaderboard. As can be observed, the order of the rankings remains consistent for both versions. [15]

| Leaderboard | Method Name | Realism Meta | Kinematic | Interactive | Map-based | minADE |
|---|---|---|---|---|---|---|
| V0 | MVTA | **0.5091** | 0.4175 | 0.5186 | 0.6374 | 1.8698 |
| V1 | MVTA | **0.6361** | 0.4175 | 0.7543 | 0.8253 | 1.8698 |
| V0 | MTR+++ | **0.4697** | 0.3597 | 0.4929 | 0.6028 | 1.6817 |
| V1 | MTR+++ | **0.6077** | 0.3597 | 0.7334 | 0.8266 | 1.6817 |
| V0 | CAD | **0.4321** | 0.3357 | 0.4355 | 0.5723 | 2.3146 |
| V1 | CAD | **0.5314** | 0.3357 | 0.5643 | 0.7782 | 2.3146 |

Table 1: Leaderboard 2023 Top 3

### 2.1.1 First Place Solution - Multiverse Transformer (MVTA)

The Multiverse Transformer (MVTA), a transformer-based framework, emerged as the first-place winner in the Waymo Open Sim Agents Challenge 2023. The MVTA is a solution that simulates traffic agents using innovative training and sampling strategies, along with a receding horizon prediction mechanism. [7]

The MVTA framework distinguishes itself through its enhanced realism in simulations. This is accomplished by leveraging state-of-the-art motion prediction models and a variable-length history aggregation method, which effectively mitigates the compounding errors in autoregressive execution. [7]
The goal is to simulate new states of agents at intervals of 0.1 seconds for the upcoming next 80 timesteps. Given the constraints of the challenge, the operation of the framework is characterised by a closed-loop, autoregressive nature. It must also be ensured that the Autonomous Driving Vehicle (ADV) component This unique feature allows the replacement of the Autonomous Driving Vehicle (ADV) component with any policy or planner.

with any policy or planner. This is achieved by factorising the joint distribution involving the world agents and the ADV into two conditionally independent components.[7]
In terms of performance, the MVTA achieved a realism meta-metric of 0.5091 in version 0 leaderboard and 0.6361 in version 1 leaderboard (1). Its enhanced version, the Multiverse Transformer Enhanced (MVTE), further improved this score to 0.5168 (V0), outperforming other methods on the leaderboard 1 in the Waymo challenge. [7]

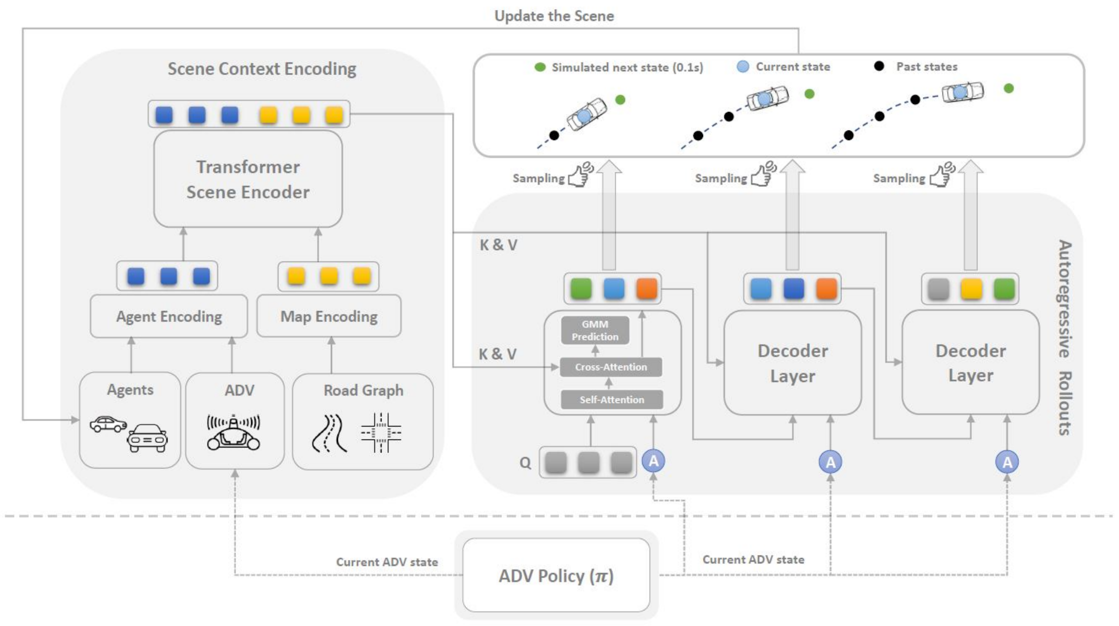


Figure 1: Main Architecture of the MVTA [7]

Figure 1 shows the main architecture of the MVTA model. The model uses transformer-based scene encoders to process inputs from world agents, the ADV and the road graph, creating a rich scene context encoding. The encoded features are then passed through a series of transformer decoder layers that perform autoregressive rollouts to simulate the next state of the ADV at fine-grained intervals. A key aspect of the approach is the use of a Gaussian Mixture Model (GMM) [16] prediction head that generates multimodal trajectory predictions, capturing the inherent uncertainty and variability in agent behaviour. The model's attention mechanisms, such as cross-attention and self-attention, allow it to focus on relevant features and interactions within the scene, resulting in more authentic and varied simulations of agent states.[7]

The MVTA model is trained considering all aspects for all three categories of agents (vehicles, cyclists, and pedestrians) using the AdamW optimizer over a span of 30 epochs, with a learning rate set at 0.0001.[7] AdamW is a stochastic optimization technique that modifies the conventional approach to weight decay within the Adam algorithm by separating weight decay from gradient updates. [17] The weighting of losses in the overall loss computation is designated as 1.0, 0.5, and 0.5 correspondingly. The generation of training samples is designed to accommodate varying lengths of past historical data, rather than being constrained to a fixed duration of 1.1 seconds. Within each 9.1s training trajectory, a singular point is randomly chosen to divide the trajectory into segments representing the past and future. This methodology not only increases the quantity of training samples derived from each genuine trajectory but also eases the process of aggregating trajectory histories during the inference phase.[7]
The training losses are calculated at each time step. The utilization of L1 loss aims at predicting the agent's speed and directional angles, while the Gaussian regression loss is applied based on the negative log-likelihood principle to enhance the likelihood of the actual trajectory. The ultimate loss is computed by combining the negative log-likelihood (NLL) loss with the L1 losses related to speed and directional angles, weighted accordingly. An effort was made during the experimentation phase to introduce a simplified version of the collision avoidance loss mechanism.[7]

Please note that this is only a brief overview of the training scheme of the MVTA model. The mathematical aspects of the training primarily focus on the computation of the overall loss, which merges the negative log-likelihood (NLL) loss with the L1 losses associated with speed and heading angles.[7] We merely provide a quick summary here as we were unable to use any of the solutions from 2023 for our project due to the unavailability of their respective codes. This necessitates a more rudimentary understanding, focusing on principles as opposed to practical, programmatic implementations.

### 2.1.2 Second Place Solution - Motion Transformer Framework (MTR+++)

The second-place team in the 2023 Waymo Open Sim Agents Challenge proposed an approach that is also based on an autoregressive method using the Motion Transformer (MTR) framework. This is a transformer-encoder-decoder model that can handle input trajectories and roadmap features. By organizing inputs into a local coordinate system centred on the agent, the MTR framework was able to effectively simulate multi-agent behaviour, achieving a Realism Meta metric score of 0.4697 (V0, 1) or 0.6077 (V1, 1), respectively.[18]

An outstanding feature of their solution is a collision mitigation policy. Assuming there are $N$ agents to simulate, including the ADV, there are $6^N$ possible outcomes at each update step if six trajectories are generated for each agent by MTR. The developers found that simply selecting the trajectories with the highest probability could lead to scenarios with unrealistic collisions. To mitigate this, they devise a method to minimise collisions without resorting to brute-force search. A $6^N$ by $6^N$ distance matrix ($D$) is generated to track the minimum L2 distance between possible trajectories. If a particular pair of trajectories would lead to a collision, it is marked in a second 0-1 matrix ($C$).[18]
This matrix $C$, seen as the adjacency matrix of a graph with $6^N$ nodes, helps to identify problematic trajectories leading to collisions (marked as 0). It allows identifying a dense subgraph of size $N$, with a density equal or greater than 0.95, indicating a minimum number of collisions.
Ultimately, these trajectories align with the dense subgraph, ensuring the selection of the most likely trajectory while reducing the risk of collision.

After the challenge, the same team proposed an improved version of their model, called MTR_E, which achieved a higher Realism Meta metric score of 0.4911 (V0), securing third place in the WOSAC ranking, if it had been submitted in time.[18]
This involved changing the way in which the agents' future movements are predicted. Specifically, the decoder part of the MTR model was adapted, and the output capacity of the Multi-Layer Perceptron (MLP) in the dense future prediction module and the motion prediction module was increased. This extended the module to support the output of $cz$ (Z coordinate) and $\theta$ (theta), which represent the direction and velocity of each agent's trajectory in the third dimension. In order to harmonise and balance these advances with the existing system, they incorporated additional L1 losses for these two parameters into

the overall loss calculation. This modification therefore directly predicts the stream of coordinates and vector values of the agents.[18]

In conclusion, the second-place team's solution utilized the MTR framework and implemented a collision mitigation strategy, delivering remarkable simulation results. Post-competition, their MTR_E model further demonstrated the potential for improvement in agents' future movement prediction. These advancements highlight the continuous evolution and potential of this field.[18] Despite their impressive achievements, it is unfortunate that we could not build upon their work directly, due to the unavailability of their code online.

### 2.1.3 Third Place Solution - Collision Avoidance Detour (CAD)

The Collision Avoidance Detour (CAD) approach that won the 3rd place in the 2023 Waymo Open Dataset Challenge - Sim Agents categorizes every valid object into three mutually exclusive sets: Autonomous Driving Vehicle (*ADV*), World-tracks-to-predict (*World-p*), and World-others (*World-o*). For each category, different motion models are employed to independently forecast their future trajectories. In addition, CAD enhances the realism of its simulation results by implementing resampling techniques for collision avoidance detours, adding Gaussian noise, and using a speed-based heading estimation system.[8]

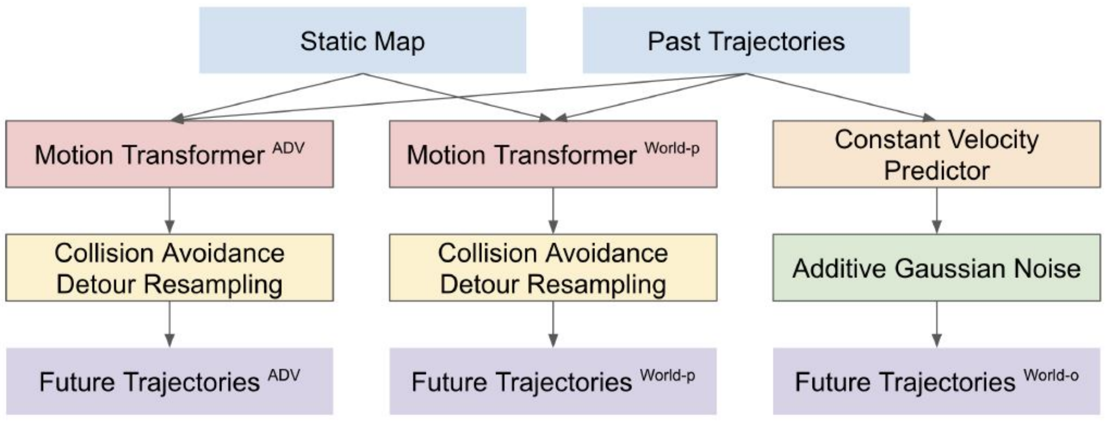

Static Map
Past Trajectories
Motion Transformer ADV
Motion Transformer World-p
Constant Velocity Predictor
Collision Avoidance Detour Resampling
Collision Avoidance Detour Resampling
Additive Gaussian Noise
Future Trajectories ADV
Future Trajectories World-p
Future Trajectories World-o

Figure 2: CAD Architecture [8]

As can be seen in Figure 2, they train an MTR model separately for $ADV$ and also for *World-p* agents. They both output six (provisional) future trajectories and the corresponding probability distribution for each object. For objects in the *World-o* group, they use a constant velocity model with additive Gaussian noise to predict their future trajectories.[8]

Entering the simulation step, the World-others objects future trajectories are calculated using only their past conditions ($c$), based on a certain probability function:

$$s_{1:T}^{World-o} \sim p^{World-o}(s_{1:T}^{World-o}|c)$$

.[8]

The collision avoidance detour resampling algorithm to simulate the future trajectories for the $ADV$ and *World-p* group works by predicting each object's future trajectory based on its probability distribution. It then checks for potential collisions. If a collision is detected, it repeatedly adjusts the predicted trajectories until it finds a set of paths free of collisions, or until it reaches a maximum number of attempts (10 in their experiments).[8]
Despite predictions being made for all objects, the final 'sampled' trajectory is only kept for the subject group (either the $ADV$, or the *World-p* group) and all others are disregarded. The collision detection and trajectory adjustment process is performed independently for each group - there's no information sharing between the two.[8]
Therefore, all detections, predictions, and adjustments are made based only on the anticipated future via context information and the MTR model specific to the group. No data from other models or predicted trajectories are used.[8]
In addition, the above algorithm focuses primarily on predicting future positions of the object centre. To predict the future heading of each object, a velocity-based estimation is employed. It is calculated by taking the arctangent of the difference in the predicted object center positions between two consecutive time steps. So, the heading at time $t$ is derived from the change in the object's position from $t-1$ to $t$, using the formula:

$$h_t = \arctan\left(\frac{y_t - y_{t-1}}{x_t - x_{t-1}}\right)$$

.[8] In this formula, $(x_t, y_t)$ signifies the anticipated object center position at time step $t$, while $(x_{t-1}, y_{t-1})$ represents the predicted object center position at time step $t-1$.

In conclusion, the third place CAD approach demonstrates a robust strategy in multi-agent simulation. It efficiently segregates objects into separate categories for individualized trajectory forecasts and actively mitigates collision risks via a detour resampling method. The approach emphasizes due consideration for object orientation, leveraging velocity-based estimations for future heading predictions. While it proved effective on the Waymo Open Dataset Challenge, its underlying principles could serve as a basis for further exploration and development in the complex scenario simulation, where each change in the object's position or pathway can significantly influence the overall system dynamics.[8]

## 2.2 Other Simulation Agent Challenges

Apart from the Waymo Open Sim Agents Challenge, there are several other challenges and competitions that focus on autonomous driving simulations. These challenges aim to advance the field of autonomous driving technology through various tasks and objectives.

- nuScenes Prediction Challenge: Hosted by Aptiv, this challenge focuses on predicting the future trajectories of traffic agents given their past trajectories in the nuScenes dataset. [19]
- Argoverse Forecasting Challenge: This competition, hosted by Argo AI, involves predicting the trajectories of all objects in a scene for 3 seconds into the future. [20]
- CARLA Autonomous Driving Challenge: This challenge involves developing a full autonomous driving system, including perception, planning, and control, to navigate through various scenarios in the CARLA simulator. [21]
- ApolloScape Trajectory Prediction Challenge: This competition, part of the ECCV Workshops, focuses on predicting the trajectories of traffic agents in the ApolloScape dataset. [22]
- Symphony: Learning Realistic and Diverse Agents for Autonomous Driving Simulation: This is a research project by Waymo that focuses on learning more realistic and diverse behavior for agents in autonomous driving simulations. [23]

The specific focus and format of these challenges may vary. However, they all contribute to the ongoing advancement of autonomous driving simulation.

# 3 Method - Closed-loop Joint-Multipath++

Joint-Multipath++ scored second in the Waymo Open Sim Agents Challenge 2023 Version 0 of the Leaderboard [24] where all submissions were using transformers except Joint-Multipath++[25]. Additionally only Joint-Multipath++ and one other submission were open-loop, meaning they did not fulfil the closed-loop requirement [25]. We use Joint-Multipath++ as our starting point as its code is publicly available on GitHub [1] meaning we can use the model and base our work off of it. Our goal is to restructure Joint-Multipath++ in such a way that its adheres to the closed-loop requirement. Therefore this chapter introduces the model, clarifies what the closed-loop-requirement is, and lastly shows our implementation along with the most relevant milestones concerning prerendering, training and testing.

## 3.1 Prerequisites

This section is meant to give a brief overview about two technologies used in the Joint-Multipath++ model, the *Long Short-Term Memory* cells and the *Multi-Context Gating Block*. The explanations given in this paper are meant to help the reader understand the very basic concepts of these technologies and are by no means complete. Please refer to the cited sources for additional information.

### 3.1.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory cells are a type of recurrent cells that can be used in Recurrent Neural Networks (RNNs) to tackle the vanishing gradient problem. RNNs are designed to handle sequential data as they possess a kind of "memory" which can store previously computed information. Each unit of an RNN - a so called cell - connects an input value to an output value (just like a regular feed-forward network) and it additionally possesses a connection back to its own input. Due to this recursive connection the RNN can "remember" already processed information. As the state of the cell gets renewed after every sequence, this architecture is comparable to a short-term memory. [26]

RNNs have a known issue, where the weights within the cells get too small and therefore parts of information get discarded. This is similar to, and therefore known as, the

[1] https://github.com/wangwenxi-handsome/Joint-Multipathpp

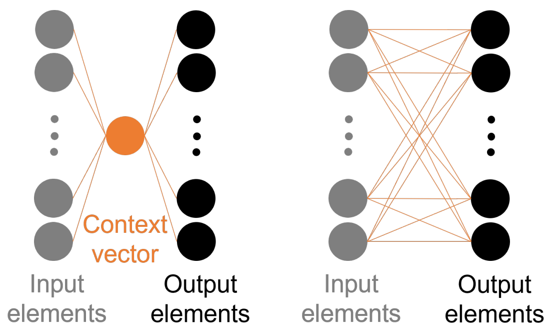


Figure 3: Context Gating principle (left) compared to cross-attention mechanism (right). Adapted from [27]

vanishing gradient problem. It can lead to the RNN delivering very short-sighted predictions which can have a negative effect on tasks where a longer horizon needs to be considered. [26]

LSTMs are able to regulate how much of a certain piece of information is supposed to be remembered or forgotten. This is possible due to two states within the LSTM cell, one for short-term states and one for long-term states. The short-term state is directly transformed with incoming data, while the long-term state only gets updated with relevant information from the short-term state. Due to this duality the vanishing gradient problem can be alleviated. [26]

In the context of this paper, LSTMs are used in the Joint-Multipath++ model to transform the sequential input data into data which can be processed efficiently by regular, feed-forward networks. Here, the sequential nature of the data is encoded in the time-dimension i.e. the 1.1s of historical and the 8.0s of future data, meaning the tensor will be reduced by one dimension after being processed by a LSTM.

### 3.1.2 Multi-Context Gating (MCG)

Multi-Context Gating is a context awareness mechanism for fusing information similar to cross-attention. It takes a set of elements and a context vector as input and outputs an output context vector. A MCG Block consists of a stack of singular Context Gating (CG) Blocks. Each CG Block is build up of two Multi-Layer Perceptrons (MLPs), one for transforming the input vector and one for the context vector. The resulting outputs from the two MLPs are fused using the element-wise product and then scaled into the output context vector using a pooling function. [27]

Similar to residual networks, MCG Blocks stack CG Blocks using running-average skip-connections. In principle, as shown in Figure 3, CG is a close approximation of cross-attention. The main difference between an MCG Block and a cross-attention mechanism is, that cross-attention conditions one input type on a second type while CG summarizes the second set into a single context vector. On the one hand cross-attention generalizes better then MCG but on the other hand it has a higher demand for computational power. [27]

## 3.2 Joint-Multipath++

In order to restructure the Joint-Multipath++ model one must first understand its general idea and architecture. Therefore this section starts with the most important facts about the model and then takes a deep dive into its architecture. Understanding the network structure, tensor shapes and elements is key to managing the transition to a closed-loop system.

### 3.2.1 Overview

Contrary to the majority of the submissions to the 2023 leaderboard Joint-Multipath++ is not a transformer based model but one that uses LSTMs and MCGs. It is also not an adaptation of the 2022 Motion Prediction Challenge winner MTR [28] but it is based on Multipath++ [27]. Joint-Multipath++ has an open-loop architecture and utilizes a scene-centric strategy. In a single model pass it can produce 32 rollouts i.e. 80 step trajectories. This is possible as it does not factorize autonomous vehicle and world policies. [25]

The modifications made to Multipath++ [27] mostly affect the decoder which can only predict the future trajectory for a single agent at a time and it does not consider interactions between agents. Since this can lead to potential collision problems Joint-Multipath++ uses a joint-prediction where interactions and dependencies between agents are considered. Therefore it has the ability to simultaneously predict multi modal trajectories, meaning it can infer trajectories for several agents at once. [24]

Further modifications were made to enable the data pre-processing step to not make distinctions between the ego agent and other agents anymore. A fixed number of agents is selected per scenario, should there be less agents in the scene, the remaining elements will be padded with zeros. Additionally the encoding is made scene-centric, meaning that all coordinates and yaw angles are transformed to the egos coordinate frame. To fully capture the interactions between the trajectories and the surrounding map features a MCG Block infers these interactions using the map features encoded as polylines. [24]

Polylines are piece-wise linear segments which describe curves like lanes, boundaries or crosswalks. They are a compact form for encoding road elements as road networks tend to have a sparse nature. Road elements are approximated by point sequences as multiple piece-wise linear segments. When not using polylines, road network elements are often stored as point collections sampled from parametric curves describing them and stored in a multi-dimensional array. For polylines only the start point, end point, and road element type has to be stored. [27]

Since the Joint-Multipath++ model was created for the 2023 WOSAC it is designed to work with the 2023 version of the WOMD. It is not possible to train the model using the 2024 dataset due to changes in the feature descriptions. Therefore all following sections are based on the assumption that a 2023 version of the WOMD is used[2].

### 3.2.2 Architecture

The architecture of Joint-Multipath++ is shown in Figure 4. It consists of several networks of different types and can generally be separated into an encoder and a decoder. The encoder generates an agent_embedding which is fed into the decoder (dark blue) to generate new trajectories. For the final agent_embedding three separate embeddings are created prior by three branches in the encoder and then concatenated (dark red).

Beginning with the first branch, the mcg_input_data tensor is fed into a MLP named _agent_mcg_linear (light blue). As for the content of mcg_input_data, it consists of 13 input data elements which can be split up into seven partially time-dependent elements

[2]We used: `https://console.cloud.google.com/storage/browser/waymo_open_dataset_motion_v_1_2_0`

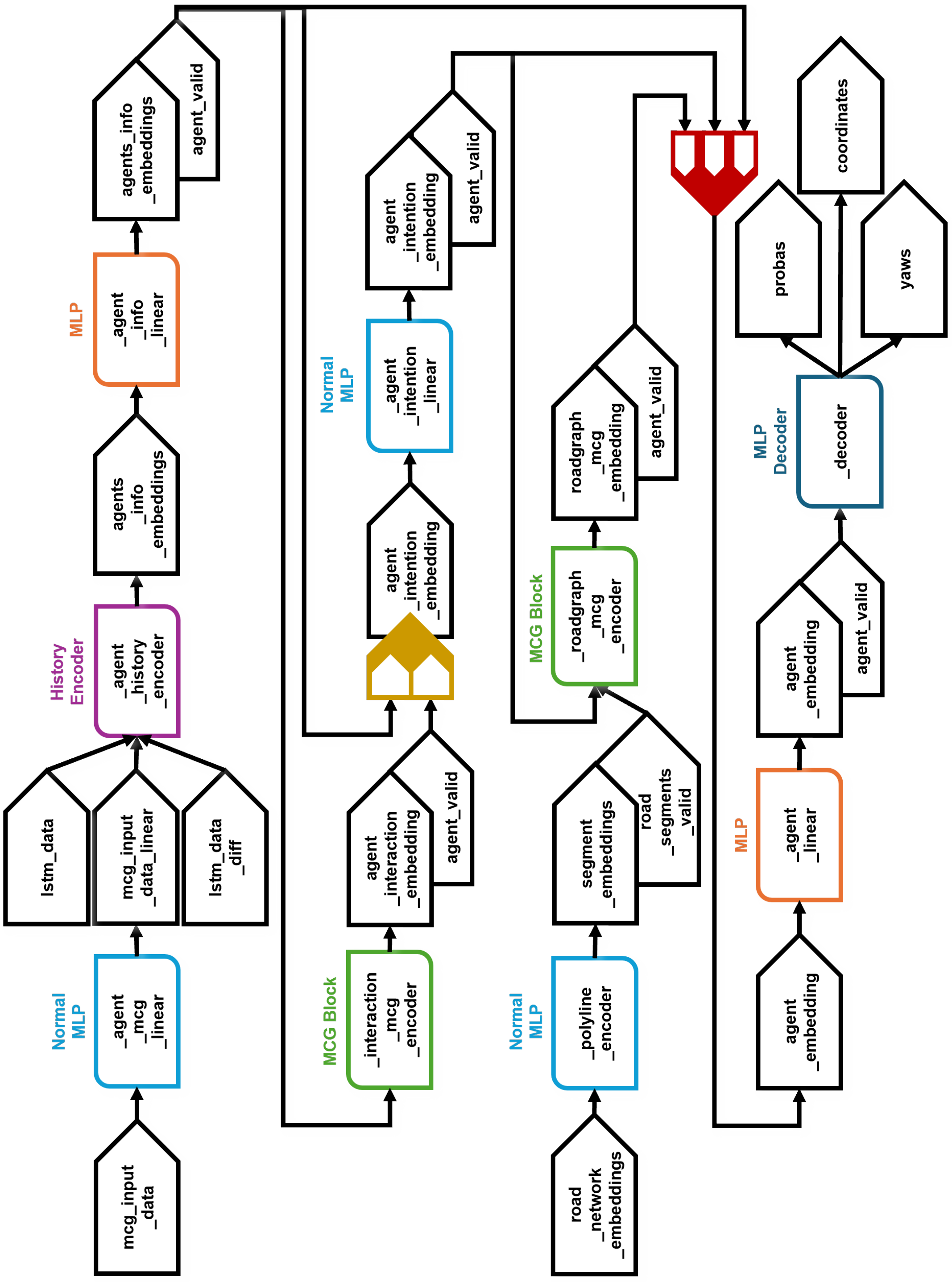


Figure 4: Architecture of Joint-Multipath++ with tensors (black) and networks (colorful) according to the configuration `configs/Multipathpp32.yaml`.

(x-coordinate, y-coordinate, yaw, speed, width, length, valid) and six elements indicating the agent type and whether it is the ego agent or not in form of a one-hot-encoding. Additionally there are eleven variables which encode the time-step of each tensor element represented by a one-hot-encoding. Each of these eleven variables represents one 0.1s step in the 1.1s of historical data i.e. the fist entry has a one in the first position followed by ten zeros, the second element has a zero as the first and a one as the second position followed by nine zeros, and so on. This results in totally 24 elements per time-step per agent. As per configuration the batch size and the number of agents per scene is set to 128 with eleven historical time-steps considered this leads to a shape of $\begin{bmatrix}128 & 128 & 11 & 24\end{bmatrix}$ for the mcg_input_data tensor. After being processed by the _agent_mcg_linear MLP the output tensor mcg_input_data_linear has the dimensions $\begin{bmatrix}128 & 128 & 11 & 128\end{bmatrix}$ due to the fact, that the MLP has the layer sizes $24 \rightarrow 32 \rightarrow 64 \rightarrow 128$ and thus up-scales the fourth dimension.

As depicted in Figure 4 the next network is the _agent_history_encoder (magenta) which transforms three tensors into the agents_info_embeddings. Aside from mcg_input_data_linear, the tensors lstm_input_data and lstm_input_data_diff are used. lstm_input_data has the dimensions $\begin{bmatrix}128 & 128 & 11 & 13\end{bmatrix}$ and has contains the same 13 input data elements that were already described before, meaning it just does not contain the additional history one-hot-encodings. lstm_input_data_diff has the shape $\begin{bmatrix}128 & 128 & 10 & 11\end{bmatrix}$ which stems from the fact, that it is designed to hold the differences between the values of two adjacent time-steps. Therefore, the third dimension is reduced from eleven to ten steps and the fourth dimension only contains eleven elements as the data for width and length is not considered due to it being static.
The history encoder represented by _agent_history_encoder is made up of two LSTMs and one MCG Block. One of the LSTMs has the size $13 \rightarrow 64$ and processes lstm_input_data, therefore increasing the fourth dimension. In parallel the second LSTM with the size $11 \rightarrow 64$ transforms lstm_input_data_diff. The MCG Block has five CG Blocks and takes mcg_input_data_linear as both the input and the context vector. Its pooling layer is implemented as a max-pooling. Due to their sequential nature all three networks in the history encoder get rid of the third dimension i.e. the timestamp of each corresponding input tensor. Consequentially, the output of the _agent_history_encoder has the shape $\begin{bmatrix}128 & 128 & 256\end{bmatrix}$ as it combines all three network outputs into one tensor called

agents_info_embeddings.
In a final step the agents_info_embeddings is processed by a MLP (orange) which keeps the dimensions as they are and then gets masked with an agent_valid tensor. The reason behind both of these actions is not completely clear and could not be discovered during research. This concludes the first branch of the encoder network.

The second branch utilizes the
agents_info_embeddings to create the agents_intention_embedding. Processing begins with the MCG Block _interaction_mcg_encoder (green) which is made up of five CG Blocks. For both the input and the context vector the
agents_info_embeddings is used, with an additional dimension inserted to make it four dimensional. This results in the agents_interaction_embedding of shape $\begin{bmatrix} 128 & 128 & 256 \end{bmatrix}$ , which is again masked with agent_valid data. To get the agent_intention_embedding the the agents_info_embeddings and the agent_interaction_embedding are concatenated (brown) resulting in the shape $\begin{bmatrix} 128 & 128 & 512 \end{bmatrix}$ .
Using the MLP _agent_intention_linear (light blue) with the layer sizes 512 $\rightarrow$ 256 $\rightarrow$ 128 the agent_intention_embedding tensor is down-scaled to $\begin{bmatrix} 128 & 128 & 128 \end{bmatrix}$ . Lastly, also this tensor is masked with the agent_valid data. This concludes the second branch of the encoder network.

In the third branch the surroundings of the vehicles are considered and transformed into an embedding. It starts with the road_network_embeddings tensor of shape
$\begin{bmatrix} 128 & 128 & 128 & 31 \end{bmatrix}$ which again has the batch size and the number of agents as its first two dimensions. The third dimension represents the number of considered road segments per agent per time-step. This is set to 128 segments in the configuration, as already Multipath++ uses the 128 closest polylines for each agent [27]. Sadly, the last dimension is hard coded to 31 so there is no further explanation possible. In the _polyline_encoder MLP (light blue) with the layer sizes 31 $\rightarrow$ 32 $\rightarrow$ 64 $\rightarrow$ 128 the tensor is up-scaled to $\begin{bmatrix} 128 & 128 & 128 & 128 \end{bmatrix}$ and afterwards masked with road_segments_valid data.
This is now the context vector for the following _roadgraph_mcg_encoder (light blue) which is a MCG Block consisting of five CG Blocks. It takes the agent_intention_embedding as its input vector and uses max-pooling in its pooling function. Afterwards the resulting roadgraph_mcg_embedding of shape $\begin{bmatrix} 128 & 128 & 128 \end{bmatrix}$ is masked with agent_valid data.

This concludes the third branch of the encoder.

Before being fed into the decoder the three generated embeddings are concatenated (red) into the agent_embedding with the dimensions $\begin{bmatrix}128 & 128 & 512\end{bmatrix}$. Its dimensions remain the same after being processed by the _agent_linear MLP (orange) and afterwards masked once again with agent_valid data.

The _decoder (dark blue) is configured as a MLP Decoder made out of two stacked MLPs. In a first step the primary MLP with layer sizes 512 → 1024 → 2048 up-scales the agent_embedding to have the size 2048 in its fourth dimension. When dividing 2048 by 64 one gets 32 which is the number of possible future trajectories to be predicted. The secondary MLP now infers these 32 trajectories by again up-scaling the divided (fourth dimension of size 64) tensors. Its layers are configured to have the shapes 64 → 128 → 241, leading to a output tensor of the dimensions $\begin{bmatrix}128 & 128 & 32 & 241\end{bmatrix}$.

This output tensor can be split up into three parts. On the one hand there are the coordinates, which are the first 160 elements of the tensor. As there are 80 time-steps to be predicted and a x- and y-coordinate are needed it leads to 160 predicted elements. These are reshaped into the five dimensional coordinates tensor with the shape $\begin{bmatrix}128 & 128 & 32 & 80 & 2\end{bmatrix}$. On the other hand the next 80 elements of the output tensor contain the corresponding yaw angle. The yaws tensor is of size $\begin{bmatrix}128 & 128 & 32 & 80 & 1\end{bmatrix}$ as there is only one angle necessary per time-step. Lastly, the decoder predicts a probability for each of the 32 possible future trajectories. These probabilities are saved in the three dimensional probas tensor with the shape $\begin{bmatrix}128 & 128 & 32\end{bmatrix}$.

The three tensors, coordinates, yaws, and probas, are the return value of the forward function of the entire model. During training, they are then used to calculate the loss in comparison to the provided ground truth information. During inference they would be the finished predicted values which can be used by the simulation agent.

## 3.3 Closed-loop Requirement

For a simulation to be considered closed-loop, it needs to contain a temporal sequence of individual next-state prediction calls to the simulation model. This model must be con-

ditioned on the preceding states. The rules of the challenge allow for the call frequency to be as low as 1Hz with interpolation done between the required 10Hz frequency of the output. In principle, the predictions must not be done in a “single-shot” non-sequential manner as this is considered as open-loop. [29]
When adhering to the closed-loop requirement resulting models are factorized into the components *World* and *ADV* which are autoregressively interleaved. See Figure 5 for graphical representation of this schema. Conditional independence from each other is needed when receiving the state of all the scenes objects. This allows for the *World* component to be used with different types of *ADV* models. [29]

The *World* model deals with the environment-centric components of the observations gathered from the input data meaning static map observations, traffic signal observations, and the previously made historical observations. For this model to be autoregressive it must resample and re-observe the scene in a certain frequency and additionally re-use its own previous outputs. As already mentioned, the *World* model must be factorized into the traffic simulator - responsible for the environment - and the *ADV*, thereby allowing a multitude of different simulator and ADV pairings. [25]
In summary, an algorithm is closed-loop, when it does not produce a trajectory of length $T$ in one model pass but rather iterates until $T$ with each iteration delivering one step of the trajectory. This makes it possible, as shown in Figure 5, to consider all previous time steps when inferring the next step as the previous predictions can be taken into account as well. [25]

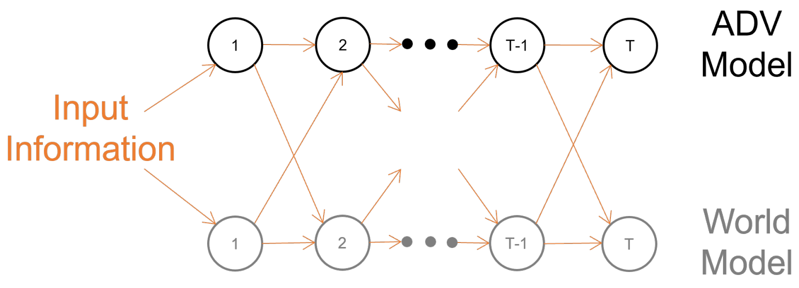


Figure 5: Required factorization of the *World* and *ADV* model and their autoregressive interleaved schema from time step 1 until $T$. Adapted from [29] and [25]

## 3.4 Implementation

### 3.4.1 Pre-rendering

Prerendering is an essential step in JMP++ implementation. The prerendering process is designed to transform raw data which is provided by Waymo within TFRecords into well-structured data suitable to process our model and train it. This process involves several steps, including data extraction and feature description, normalization, and vectorization.

Initially, the raw data is extracted from *TFrecords* files. The records contain a variety of information, such as agent coordinates, trajectories, road networks, timestamps, etc. During the prerendering, these raw data points are parsed and then transformed into meaningful features that we need for our model training. Then normalization is applied to ensure that all features are standardized. The processed data is then saved in *.npz* format which keeps the structure of the data and is used later for the training.

## 3.5 Experiments and Results

### 3.5.1 Training and testing original Joint-Multipath++

In this section, we will describe the workflow to train the open-loop architecture. Here's a detailed breakdown of the training workflow:

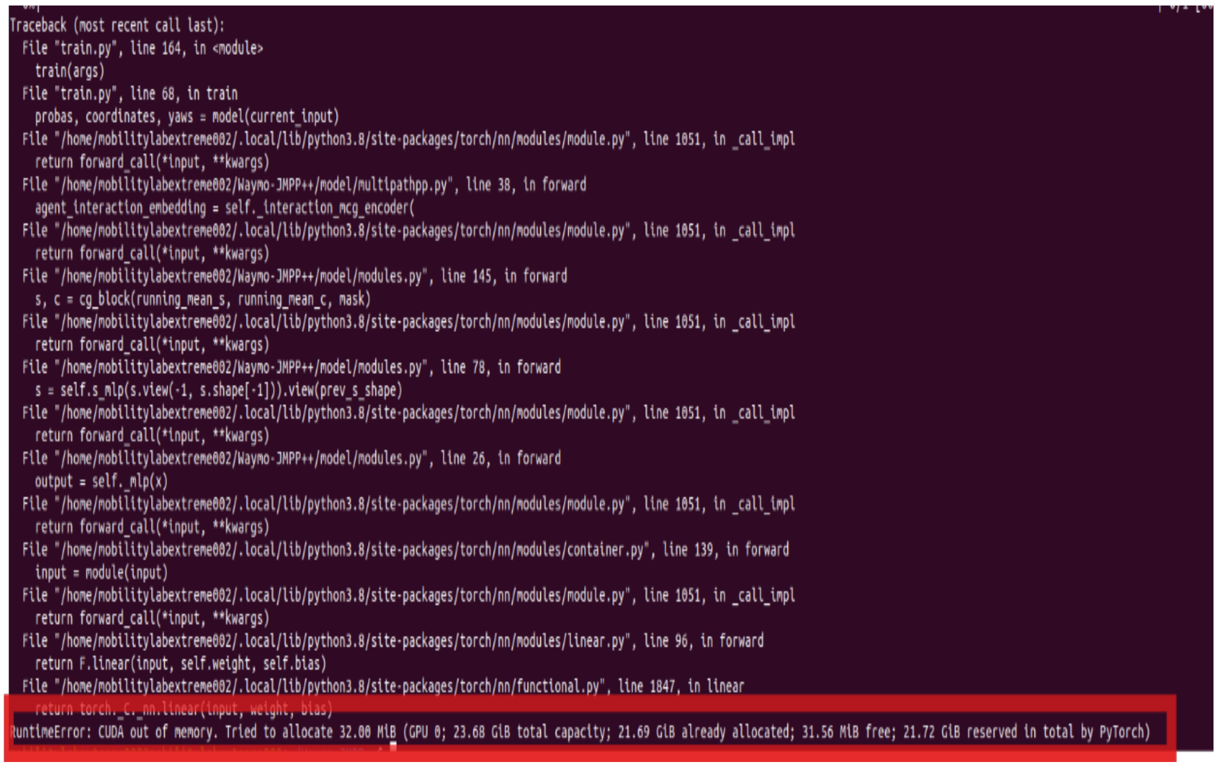
- **Configuration Loading:** The configuration file **Multipathapp32.yaml** is loaded to set up the model architecture, hyperparameters, and training settings.
- **Data Loading:** The data is loaded into the training pipeline.
- **Model Initialization:** The JMPP++ model is then initialized with the specified architecture from our yaml configuration file.
- **Training Loop:** The model is trained over multiple epochs. For each batch of data, the model predicts future trajectories, then the loss is calculated based on the difference between predicted and actual trajectories. Specifically, three losses are computed:
  - **Distance Loss:** Measures the Euclidean distance between the predicted and actual future positions.
  - **Yaw Loss:** Measures the error in the predicted yaw of the agents.

  - **Confidence Loss:** Evaluates the predicted probabilities of the trajectories.

- **Saving the Model:** After each epoch, the model state is saved. If the performance improves, this model state is marked as the best model.

- **Rollout:** Once the model is trained, the rollout script is used to generate the predictions. The best model checkpoint is loaded and then the model generates predictions.

In the initial phase of training the Joint-MultiPath++ model, we sticked to the default configuration, which included a batch size of 128. However, this configuration led to several challenges. The default stack size of 128 required high computing resources. This proved to be insufficient with the available hardware and led to memory overflow problems and a considerable slowdown in the training process. **Figure 6** shows the training results for the open-loop



```
Traceback (most recent call last):
  File "train.py", line 164, in <module>
    train(args)
  File "train.py", line 68, in train
    probas, coordinates, yaws = model(current_input)
  File "/home/mobilitylabextreme002/.local/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1051, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/mobilitylabextreme002/Waymo-JMPP++/model/multipathpp.py", line 38, in forward
    agent_interaction_embedding = self._interaction_mcg_encoder(
  File "/home/mobilitylabextreme002/.local/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1051, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/mobilitylabextreme002/Waymo-JMPP++/model/modules.py", line 145, in forward
    s, c = cg_block(running_mean_s, running_mean_c, mask)
  File "/home/mobilitylabextreme002/.local/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1051, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/mobilitylabextreme002/Waymo-JMPP++/model/modules.py", line 78, in forward
    s = self.s_mlp(s.view(-1, s.shape[-1])).view(prev_s_shape)
  File "/home/mobilitylabextreme002/.local/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1051, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/mobilitylabextreme002/Waymo-JMPP++/model/modules.py", line 26, in forward
    output = self._mlp(x)
  File "/home/mobilitylabextreme002/.local/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1051, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/mobilitylabextreme002/.local/lib/python3.8/site-packages/torch/nn/modules/container.py", line 139, in forward
    input = module(input)
  File "/home/mobilitylabextreme002/.local/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1051, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/mobilitylabextreme002/.local/lib/python3.8/site-packages/torch/nn/modules/linear.py", line 96, in forward
    return F.linear(input, self.weight, self.bias)
  File "/home/mobilitylabextreme002/.local/lib/python3.8/site-packages/torch/nn/functional.py", line 1847, in linear
    return torch._C._nn.linear(input, weight, bias)
RuntimeError: CUDA out of memory. Tried to allocate 32.00 MiB (GPU 0; 23.68 GiB total capacity; 21.69 GiB already allocated; 31.56 MiB free; 21.72 GiB reserved in total by PyTorch)
```

Figure 6: Default Open-loop training results

The error in the red box of Figure **6** indicates a "CUDA out of memory" issue, where the program attempted to allocate 32.00 MB on a GPU that was almost at full capacity, with only 31.56 MB of memory available.

To adapt the JMP++ model to our hardware limitations, we had to reduce our batch size. The model could only be trained with a maximum batch size of 2. We kept the JMP++ model trained for 200 epochs, and then as mentioned earlier, the best 4 training results of the model were saved across these epochs. Afterwards, we ran the rollout script on test data and the model generated and saved its predictions each time.

To conclude the training of the open-loop Joint-Multipath++ model, we faced several challenges primarily due to hardware limitations. Initially, we used the default configuration with a batch size of 128, but this led to memory overflow problems. To address these issues, we reduced the batch size to 2, which allowed us to continue training effectively. Future work should focus on fine-tuning hyperparameters and maybe using more powerful hardware to process larger batch sizes.

### 3.5.2 Proposed closed-loop Solution

As we mentioned in the previous sections, one of the newest updates in the Waymo open dataset challenge is the closed-loop requirement. During closed-loop execution, the model generates predictions for agent trajectories at each time step, feeding these predictions back as input for the next step. This is why we had to make a few adjustments to Joint-Multipath++ model, in order to fulfill the requirements of the challenge.

The closed-loop solution for the JMP++ model enhances the traditional open-loop approach by incorporating a feedback mechanism. This allows the model to iteratively update its predictions based on previously generated outputs, closely simulating real-world scenarios where predictions influence subsequent states. **Figure 7** shows the our proposed solution for solving the closed-loop.

To realize the closed-loop transformation of our JMPP++, we evaluated, that we have to implement these 2 main components:

- **Feedback Loop:** At each time step the JMPP++ generates a prediction for agent trajectories including yaws and coordinates. these should be fed back into the model as part of the input for the next time step. This iterative process continues for a predefined number of steps (e.g., 80 steps in our implementation).
- **State Update:** The input state is updated with the new predictions, normalized, and prepared for the next iteration.

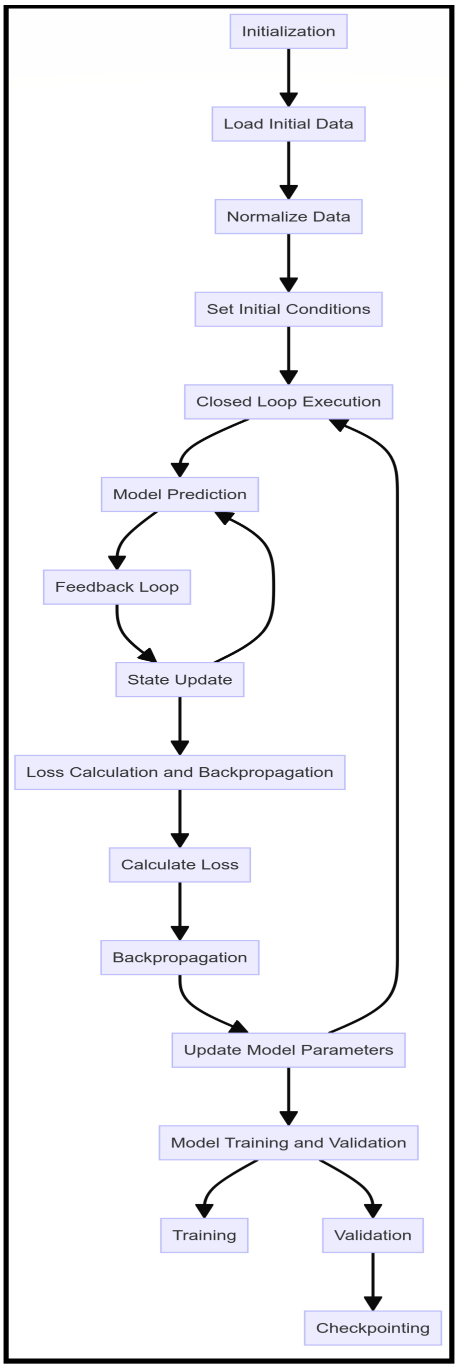


Figure 7: Closed-Loop Solution

To achieve closed-loop solution, we needed to update the training script and implement these key changes:

- **Update input for each step:** Implemented a function to update the input states based on the most probable trajectories predicted by the model using the past step's input.
- **Closed-Loop Execution:** Added a for-loop to iterate over 80 time steps during the epochs for each batch. For each of our loop iteration, the input for our model is updated.
- **Normalization:** Applied normalization for each of input data to ensure our values remain within a stable range during the closed-loop execution.

The detailed implementation of our training process, including the closed-loop logic, can be found in the appendix section of this paper, which includes the relevant code files and configuration settings.

#### 3.5.3 Training and testing closed-loop Joint-Multipath++

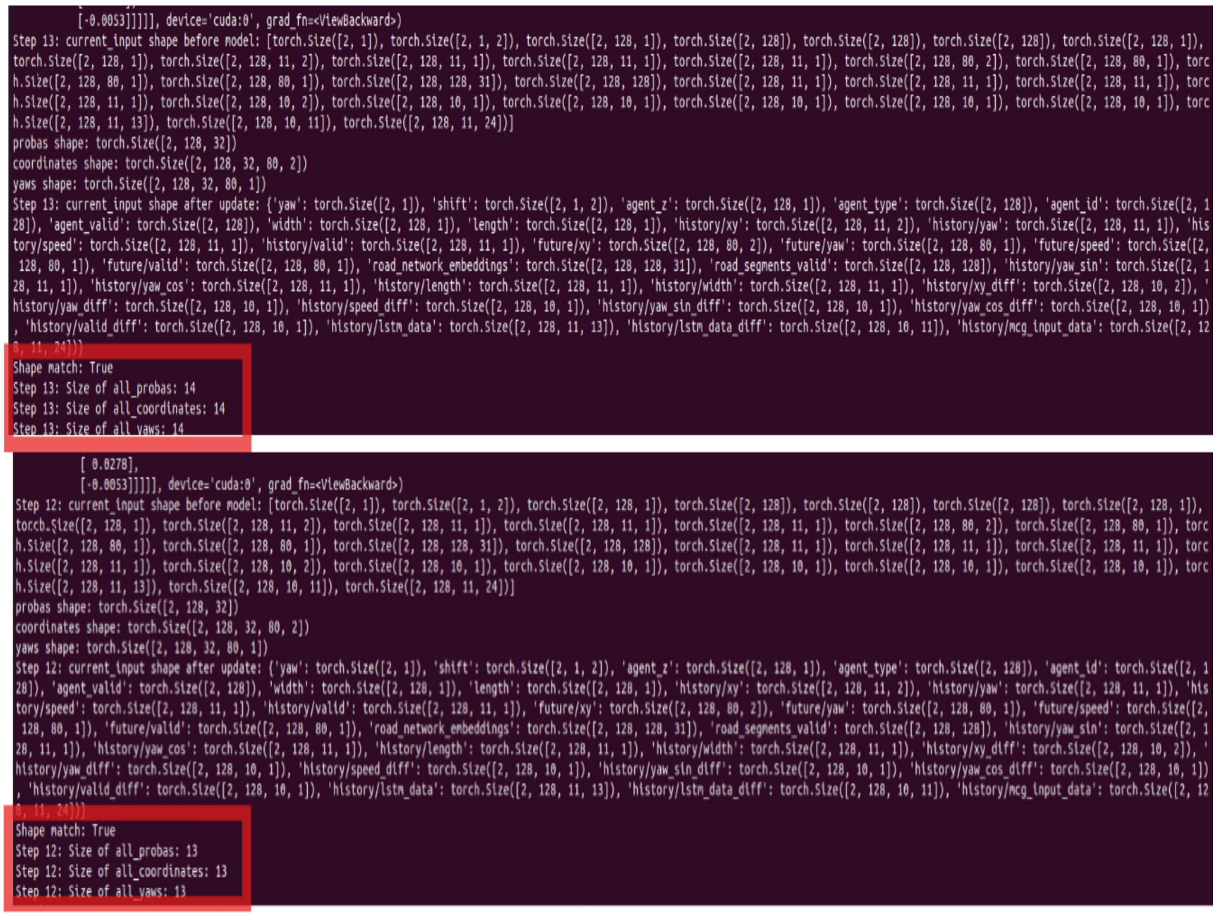
In this section, we will describe the workflow for training our proposed closed-loop solution. The closed-loop approach brought some complexities to the training phase due to the iterations of input update and our model prediction. Here's a detailed breakdown of our closed-loop solution training:

- **Configuration Loading:** Similar to our training for the open-loop solution. we load our configuration file **Multipathpp32.yaml** to set up our model architecture
- **Data Loading and Model Initialization:** The training and validation data are loaded to our training pipeline, then the JMPP++ is initialized with the specific architecture defined in our configuration file.
- **Training Loop:** The model is trained over multiple epochs. For each batch of data, the model iteratively predicts future trajectories for 80 time steps, updating the input data at each time step. Specifically, the workflow involves:
  - **Model Prediction:** For each time step, the model predicts future trajectories.
  - **State Update:** Based on the probabilities predictions, we update the input states using the relative coordinates and yaws prediction.
  - **Loss Calculation and Backpropagation:** After 80 timesteps, the loss is calculated based on the final predictions and the actual future trajectories. We

kept the same three losses as in the default open-loop training. After all time steps, The loss is backpropagated through our closed loop network.

Despite reducing the batch size to 2 ( as we did in the open-loop solution training), we again encountered hardware limitations. Each epoch involves 80 iterations of input updates and predictions, which increased the computational demands. Unfortuantely, the model could only be trained only for 13 time steps in the first epoch before terminating due to hardware issues.

However, to ensure the closed-loop logic was implemented correctly and to validate our approach, we implemented assertions within the training loop to verify the reprocessing of our input to the model. Since the predicted tensor shapes is different from the input, this step was important in identifying and fixing potential shape mismatch issues in our solution. Moreover, by printing the values of probas, yaws, coordinates at each step, we were able to track how the predictions changed over the iterations. **Figure 8** shows our training results for closed loop.



```
         [-0.0053]]]]], device='cuda:0', grad_fn=<ViewBackward>)
Step 13: current_input shape before model: [torch.Size([2, 1]), torch.Size([2, 1, 2]), torch.Size([2, 128, 1]), torch.Size([2, 128]), torch.Size([2, 128]), torch.Size([2, 128]), torch.Size([2, 128, 1]),
torch.Size([2, 128, 1]), torch.Size([2, 128, 11, 2]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 80, 2]), torch.Size([2, 128, 80, 1]), torc
h.Size([2, 128, 80, 1]), torch.Size([2, 128, 80, 1]), torch.Size([2, 128, 128, 31]), torch.Size([2, 128, 128]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 11, 1]), torc
h.Size([2, 128, 11, 1]), torch.Size([2, 128, 10, 2]), torch.Size([2, 128, 10, 1]), torch.Size([2, 128, 10, 1]), torch.Size([2, 128, 10, 1]), torch.Size([2, 128, 10, 1]), torch.Size([2, 128, 10, 1]), torc
h.Size([2, 128, 11, 13]), torch.Size([2, 128, 10, 11]), torch.Size([2, 128, 11, 24])]
probas shape: torch.Size([2, 128, 32])
coordinates shape: torch.Size([2, 128, 32, 80, 2])
yaws shape: torch.Size([2, 128, 32, 80, 1])
Step 13: current_input shape after update: {'yaw': torch.Size([2, 1]), 'shift': torch.Size([2, 1, 2]), 'agent_z': torch.Size([2, 128, 1]), 'agent_type': torch.Size([2, 128]), 'agent_id': torch.Size([2, 1
28]), 'agent_valid': torch.Size([2, 128]), 'width': torch.Size([2, 128, 1]), 'length': torch.Size([2, 128, 1]), 'history/xy': torch.Size([2, 128, 11, 2]), 'history/yaw': torch.Size([2, 128, 11, 1]), 'his
tory/speed': torch.Size([2, 128, 11, 1]), 'history/valid': torch.Size([2, 128, 11, 1]), 'future/xy': torch.Size([2, 128, 80, 2]), 'future/yaw': torch.Size([2, 128, 80, 1]), 'future/speed': torch.Size([2,
 128, 80, 1]), 'future/valid': torch.Size([2, 128, 80, 1]), 'road_network_embeddings': torch.Size([2, 128, 128, 31]), 'road_segments_valid': torch.Size([2, 128, 128]), 'history/yaw_sin': torch.Size([2, 1
28, 11, 1]), 'history/yaw_cos': torch.Size([2, 128, 11, 1]), 'history/length': torch.Size([2, 128, 11, 1]), 'history/width': torch.Size([2, 128, 11, 1]), 'history/xy_diff': torch.Size([2, 128, 10, 2]), '
history/yaw_diff': torch.Size([2, 128, 10, 1]), 'history/speed_diff': torch.Size([2, 128, 10, 1]), 'history/yaw_sin_diff': torch.Size([2, 128, 10, 1]), 'history/yaw_cos_diff': torch.Size([2, 128, 10, 1])
, 'history/valid_diff': torch.Size([2, 128, 10, 1]), 'history/lstm_data': torch.Size([2, 128, 11, 13]), 'history/lstm_data_diff': torch.Size([2, 128, 10, 11]), 'history/mcg_input_data': torch.Size([2, 12
8, 11, 24])}
Shape match: True
Step 13: Size of all_probas: 14
Step 13: Size of all_coordinates: 14
Step 13: Size of all_yaws: 14
```

```
         [ 0.0278],
         [-0.0053]]]]], device='cuda:0', grad_fn=<ViewBackward>)
Step 12: current_input shape before model: [torch.Size([2, 1]), torch.Size([2, 1, 2]), torch.Size([2, 128, 1]), torch.Size([2, 128]), torch.Size([2, 128]), torch.Size([2, 128]), torch.Size([2, 128, 1]),
torcb.Size([2, 128, 1]), torch.Size([2, 128, 11, 2]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 80, 2]), torch.Size([2, 128, 80, 1]), torc
h.Size([2, 128, 80, 1]), torch.Size([2, 128, 80, 1]), torch.Size([2, 128, 128, 31]), torch.Size([2, 128, 128]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 11, 1]), torch.Size([2, 128, 11, 1]), torc
h.Size([2, 128, 11, 1]), torch.Size([2, 128, 10, 2]), torch.Size([2, 128, 10, 1]), torch.Size([2, 128, 10, 1]), torch.Size([2, 128, 10, 1]), torch.Size([2, 128, 10, 1]), torch.Size([2, 128, 10, 1]), torc
h.Size([2, 128, 11, 13]), torch.Size([2, 128, 10, 11]), torch.Size([2, 128, 11, 24])]
probas shape: torch.Size([2, 128, 32])
coordinates shape: torch.Size([2, 128, 32, 80, 2])
yaws shape: torch.Size([2, 128, 32, 80, 1])
Step 12: current_input shape after update: {'yaw': torch.Size([2, 1]), 'shift': torch.Size([2, 1, 2]), 'agent_z': torch.Size([2, 128, 1]), 'agent_type': torch.Size([2, 128]), 'agent_id': torch.Size([2, 1
28]), 'agent_valid': torch.Size([2, 128]), 'width': torch.Size([2, 128, 1]), 'length': torch.Size([2, 128, 1]), 'history/xy': torch.Size([2, 128, 11, 2]), 'history/yaw': torch.Size([2, 128, 11, 1]), 'his
tory/speed': torch.Size([2, 128, 11, 1]), 'history/valid': torch.Size([2, 128, 11, 1]), 'future/xy': torch.Size([2, 128, 80, 2]), 'future/yaw': torch.Size([2, 128, 80, 1]), 'future/speed': torch.Size([2,
 128, 80, 1]), 'future/valid': torch.Size([2, 128, 80, 1]), 'road_network_embeddings': torch.Size([2, 128, 128, 31]), 'road_segments_valid': torch.Size([2, 128, 128]), 'history/yaw_sin': torch.Size([2, 1
28, 11, 1]), 'history/yaw_cos': torch.Size([2, 128, 11, 1]), 'history/length': torch.Size([2, 128, 11, 1]), 'history/width': torch.Size([2, 128, 11, 1]), 'history/xy_diff': torch.Size([2, 128, 10, 2]), '
history/yaw_diff': torch.Size([2, 128, 10, 1]), 'history/speed_diff': torch.Size([2, 128, 10, 1]), 'history/yaw_sin_diff': torch.Size([2, 128, 10, 1]), 'history/yaw_cos_diff': torch.Size([2, 128, 10, 1])
, 'history/valid_diff': torch.Size([2, 128, 10, 1]), 'history/lstm_data': torch.Size([2, 128, 11, 13]), 'history/lstm_data_diff': torch.Size([2, 128, 10, 11]), 'history/mcg_input_data': torch.Size([2, 12
8, 11, 24])}
Shape match: True
Step 12: Size of all_probas: 13
Step 12: Size of all_coordinates: 13
Step 12: Size of all_yaws: 13
```

Figure 8: Closed-loop time steps training output

We printed the initial and updated shapes of the tensors before and after each iteration. This allowed us to confirm that the updates were not only being applied correctly but also that the model's predictions were evolving as expected, indicating successful state transitions within the closed-loop solution.

In conclusion, while we couldn't train our closed-loop model due to the computational constraints, the initial tests we made using our assertions through some time steps validated the effectiveness of our approach and demonstrated potential for our model. Future work should focus on optimizing the computational constraints, refining the model architecture to reduce complexity, and fine tuning the model parameters to achieve better training. Furthermore, finding the optimal points for loss computation within the loop will improve the performance and learning process.

# 4 Conclusion and Outlook

Despite our team's efforts, we didn't achieve the desired results in the Waymo Open Simulation Agents Challenge 2024. However, our journey was filled with valuable learning experiences. We delved into research papers on previous years solutions, including MTR+++, MTR_E, MVTA, MVTE, and CAD. This exploration broadened our understanding of different approaches and informed our own strategies. We also explored the Joint-MultiPath++ model, gaining insights into autonomous driving simulations. While our outcomes fell short, we recognize the importance of robust model architecture and thorough testing.

This year's winners have set a new benchmark by achieving a much higher Realism Meta-Metric than the top three places from last year's challenge, highlighting the rapid progress being made in the field.[10]

For the next team, we recommend focusing on data pre-processing and feature engineering to improve model performance. Exploring advanced neural network architectures and reinforcement learning strategies could lead to better results. Moreover, it is also crucial to ensure that the necessary hardware resources are readily available as it posed a significant challenge for us this year. We pass the torch with optimism, encouraging future entrants to innovate fearlessly and approach the challenge with a problem-solving mindset.

## References

[1] "Challenges Overview – Waymo Open Dataset — waymo.com," https://waymo.com/open/challenges/, [Accessed 01-06-2024].

[2] "Definition of Waymo — pcmag.com," https://www.pcmag.com/encyclopedia/term/waymo, [Accessed 01-06-2024].

[3] "Terms – Waymo Open Dataset — waymo.com," https://waymo.com/open/terms/, [Accessed 01-06-2024].

[4] "Feds probe Waymo driverless cars hitting parked cars, drifting into traffic — arstechnica.com," https://arstechnica.com/tech-policy/2024/05/feds-probe-waymo-driverless-cars-hitting-parked-cars-drifting-into-traffic/, [Accessed 01-06-2024].

[5] C. Vanek and N. N. Alund, "Feds are investigating Waymo driverless cars after reports of crashes, traffic violations — usatoday.com," https://www.usatoday.com/story/money/cars/2024/05/16/waymo-investigation-crashes-violations/73712498007/, [Accessed 01-06-2024].

[6] "Two driverless cars crash into the same truck, prompting first recall — independent.co.uk," https://www.independent.co.uk/tech/driverless-car-waymo-recall-b2496141.html, [Accessed 01-06-2024].

[7] Y. Wang, T. Zhao, and F. Yi, "Multiverse transformer: 1st place solution for waymo open sim agents challenge 2023," *CoRR*, vol. abs/2306.11868, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.11868

[8] H. kuang Chiu and S. F. Smith, "Collision avoidance detour for multi-agent trajectory forecasting," 2023.

[9] "Fleet response: Lending a helpful hand to Waymo's autonomously driven vehicles — waymo.com," https://waymo.com/blog/2024/05/fleet-response/, [Accessed 22-05-2024].

[10] "Sim Agents — waymo.com," https://waymo.com/open/challenges/2024/sim-agents/, [Accessed 01-06-2024].

[11] “Waymo open dataset: An autonomous driving dataset,” 2019.

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

[13] “Waymo open dataset: Perception,” https://waymo.com/open/data/perception/, [Accessed 16-06-2024].

[14] “Waymo open dataset: Motion,” https://waymo.com/open/data/motion/, [Accessed 16-06-2024].

[15] “Sim Agents — waymo.com,” https://waymo.com/open/challenges/2023/sim-agents/, [Accessed 13-06-2024].

[16] V. Apgar, “3 Use-Cases for Gaussian Mixture Model (GMM) — towardsdatascience.com,” https://towardsdatascience.com/3-use-cases-for-gaussian-mixture-model-gmm-72951fcf8363, [Accessed 25-05-2024].

[17] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” 2019.

[18] C. Qian, D. Xiu, and M. Tian, “The 2nd place solution for 2023 waymo open sim agents challenge,” 2023.

[19] “nuscenes.org,” https://www.nuscenes.org/prediction/?externalData=all&mapData=all&modalities=Any, [Accessed 25-05-2024].

[20] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, D. Ramanan, P. Carr, and J. Hays, “Argoverse 2: Next generation datasets for self-driving perception and forecasting,” in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*, 2021.

[21] “CARLA Autonomous Driving Challenge — leaderboard.carla.org,” https://leaderboard.carla.org/challenge/, [Accessed 25-05-2024].

[22] “Apollo Scape — apolloscape.auto,” https://apolloscape.auto/index.html, [Accessed 25-05-2024].

[23] M. Igl, D. Kim, A. Kuefler, P. Mougin, P. Shah, K. Shiarlis, D. Anguelov, M. Palatucci, B. A. White, and S. Whiteson, "Symphony: Learning realistic and diverse agents for autonomous driving simulation," *2022 International Conference on Robotics and Automation (ICRA)*, pp. 2445–2451, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:248562497

[24] W. Wang and H. Zhen, "Joint-Multipath++ for Simulation Agents." [Online]. Available: https://storage.googleapis.com/waymo-uploads/files/research/2023%20Technical%20Reports/SA_hm_jointMP.pdf

[25] N. Montali, J. Lambert, P. Mougin, A. Kuefler, N. Rhinehart, M. Li, C. Gulino, T. Emrich, Z. Yang, S. Whiteson, B. White, and D. Anguelov, "The Waymo Open Sim Agents Challenge," Dec. 2023. [Online]. Available: http://arxiv.org/abs/2305.12032

[26] J. P. Mueller and L. Massaron, *Deep Learning kompakt für Dummies*, 1st ed. Weinheim: Wiley-VCH, 2020.

[27] B. Varadarajan, A. Hefny, A. Srivastava, K. S. Refaat, N. Nayakanti, A. Cornman, K. Chen, B. Douillard, C. P. Lam, D. Anguelov, and B. Sapp, "MultiPath++: Efficient Information Fusion and Trajectory Aggregation for Behavior Prediction," Dec. 2021, arXiv:2111.14973 [cs]. [Online]. Available: http://arxiv.org/abs/2111.14973

[28] S. Shi, L. Jiang, D. Dai, and B. Schiele, "Motion transformer with global intention localization and local movement refinement," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 6531–6543. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/2ab47c960bfee4f86dfc362f26ad066a-Paper-Conference.pdf

[29] "Sim Agents – 2024 – Waymo Open Dataset." [Online]. Available: https://waymo.com/open/challenges/2024/sim-agents/

## Statutory Declaration

I certify that I have written this thesis without outside help and without using sources other than those specified and that the thesis has not been submitted in the same or a similar form to any other examination authority and has not been accepted by them as part of an examination. All statements that have been adopted verbatim or analogously are labelled as such.

Ingolstadt, June 29, 2024

[illegible]

Carla Roth

[illegible]

Gizem Bükülmez

[illegible]

Franziska Kofler

[illegible]

Mohamed Ayoub Kefi

# A Appendix

## A.1 Open Loop Train Script

Listing 1: Open Loop Train Script

```python
def train(args):
    # config
    set_random_seed(42)
    config = get_yaml_config(args.config)

    # dataloader
    dataloader = get_dataloader(args.train_data_path, config["train"]["data_config"])
    val_dataloader = get_dataloader(args.val_data_path, config["val"]["data_config"])

    # model init
    loss_func = get_model_loss(config["model"]["loss"])
    if(not os.path.exists(args.save_folder)):
        os.mkdir(args.save_folder)
    last_checkpoint = get_last_file(args.save_folder)
    model = MultiPathPP(config["model"])
    model.cuda()
    optimizer = Adam(model.parameters(), **config["train"]["optimizer"])
    if config["train"]["scheduler"]:
        scheduler = ReduceLROnPlateau(optimizer, patience=20, factor=0.5, verbose=True)
    num_steps = 0
    if last_checkpoint is not None:
        model.load_state_dict(torch.load(last_checkpoint)["model_state_dict"])
        optimizer.load_state_dict(torch.load(last_checkpoint)["optimizer_state_dict"])
        num_steps = torch.load(last_checkpoint)["num_steps"]
        if config["train"]["scheduler"]:
            scheduler.load_state_dict(torch.load(last_checkpoint)["scheduler_state_dict"])
        print("LOADED ", last_checkpoint)
    model_parameters = filter(lambda p: p.requires_grad, model.parameters())
    params = sum([np.prod(p.size()) for p in model_parameters])
    print("N PARAMS=", params)

    # train and validation
    best_loss = float('inf')
    for epoch in tqdm(range(config["train"]["n_epochs"])):
        pbar = tqdm(dataloader)
        for data in pbar:
            # train
            model.train()
            optimizer.zero_grad()
            if config["train"]["data_config"]["dataset_config"]["normlization"]:
                data = normalize(data)
            dict_to_cuda(data)
            probas, coordinates, yaws = model(data, num_steps)

            # loss and optimizer
            gt_xy = data["future/xy"] - data["history/xy"][:, :, -1:, :]
            gt_yaw = data["future/yaw"] - data["history/yaw"][:, :, -1:, :]
            gt_valid = mask_by_valid(data["future/valid"], data["agent_valid"])
            distance_loss, yaw_loss, confidence_loss = loss_func(
                gt_xy, gt_valid, gt_yaw, probas, coordinates, yaws)
            loss = distance_loss + yaw_loss + confidence_loss
            loss.backward()

            if "clip_grad_norm" in config["train"]:
                torch.nn.utils.clip_grad_norm_(model.parameters(), config["train"]["clip_grad_norm"])
            optimizer.step()

            # log
            if num_steps % 1 == 0:
                pbar.set_description(f"epoch={epoch} loss={round(loss.item(), 2)} distance_loss={round(distance_loss.item(), 2)} yaw_loss={round(yaw_loss.item(), 2)} confidence_loss={round(confidence_loss.item(), 2)}")
            # validation
            if num_steps % config["train"]["validate_every_n_steps"] == 0 and num_steps > 0:
                del data
```

```python
            torch.cuda.empty_cache()
            model.eval()
            with torch.no_grad():
                losses = []
                for data in tqdm(val_dataloader):
                    if config["val"]["data_config"]["dataset_config"]["normlization"]:
                        data = normalize(data)
                    dict_to_cuda(data)
                    probas, coordinates, yaws = model(data, num_steps)
                    gt_xy = data["future/xy"] - data["history/xy"][:, :, -1:, :]
                    gt_yaw = data["future/yaw"] - data["history/yaw"][:, :, -1:, :]
                    gt_valid = mask_by_valid(data["future/valid"], data["agent_valid"])
                    distance_loss, yaw_loss, confidence_loss = loss_func(
                        gt_xy, gt_valid, gt_yaw, probas, coordinates, yaws)
                    loss = distance_loss + yaw_loss + confidence_loss
                    losses.append(loss.item())
                pbar.set_description(f"validation loss = {round(sum(losses) / len(losses), 2)}")

            if sum(losses) / len(losses) < best_loss:
                best_loss = sum(losses) / len(losses)
                saving_data = {
                    "num_steps": num_steps,
                    "model_state_dict": model.state_dict(),
                    "optimizer_state_dict": optimizer.state_dict(),
                }
                if config["train"]["scheduler"]:
                    saving_data["scheduler_state_dict"] = scheduler.state_dict()
                torch.save(saving_data, os.path.join(args.save_folder, f"best_{epoch}.pth"))

        num_steps += 1
        if "max_iterations" in config["train"] and num_steps > config["train"]["max_iterations"]:
                break
```

## A.2 YAML Model Configuration

Listing 2: YAML Model Configuration

```yaml

train:
  data_config:
    dataset_config:
      lstm_input_data: ["xy", "yaw", "speed", "width", "length", "valid"]
      lstm_input_data_diff: ["xy", "yaw", "speed", "valid"]
      mask_history: False
      mask_history_fraction: 0.15
      normlization: True
    dataloader_config:
      batch_size: 2
      shuffle: True
      num_workers: 4
  optimizer:
    lr: 0.00001
  n_epochs: 1
  validate_every_n_steps: 4000
  max_iterations: 5000001
  clip_grad_norm: 1
  scheduler: True

val:
  data_config:
    dataset_config:
      lstm_input_data: ["xy", "yaw", "speed", "width", "length", "valid"]
      lstm_input_data_diff: ["xy", "yaw", "speed", "valid"]
      mask_history: False
      normlization: True
    dataloader_config:
      batch_size: 128
      shuffle: False
      num_workers: 4

test:
  data_config:
    dataset_config:
      lstm_input_data: ["xy", "yaw", "speed", "width", "length", "valid"]
      lstm_input_data_diff: ["xy", "yaw", "speed", "valid"]
      mask_history: False
      normlization: True
    dataloader_config:
      batch_size: 2
      shuffle: False
      num_workers: 0

model:
  n_trajectories: 32
  size: 512
  decoder: "MLPDecoder"
  loss: "min_ade_prob_heading"

  agent_mcg_linear:
    layers: [24, 32, 64, 128]
    pre_activation: False
    pre_batchnorm: False
    batchnorm: False

  agent_history_encoder:
    position_lstm_config:
      input_size: 13
      hidden_size: 64
    position_diff_lstm_config:
      input_size: 11
      hidden_size: 64
    position_mcg_config:
      agg_mode: "max"
      running_mean_mode: "real"
      alpha: 0.1
      beta: 0.9
      n_blocks: 5
      identity_c_mlp: True
      block:
        c_bias: True
        mlp:
          n_layers: 3
          n_in: 128
          n_out: 128
          bias: True
          batchnorm: False
          dropout: False

  agent_info_linear:
    n_layers: 3
    n_in: 256
    n_out: 256
    bias: True
    batchnorm: False
    dropout: False

  interaction_mcg_encoder:
    block:
      c_bias: True
      mlp:
        n_layers: 3
        n_in: 256
        n_out: 256
        bias: True
        batchnorm: False
        dropout: False
    agg_mode: "max"
    running_mean_mode: "real"
    alpha: 0.1
    beta: 0.9
    n_blocks: 5
    identity_c_mlp: False

  agent_intention_linear:
    layers: [512, 256, 128]
    pre_activation: True
    pre_batchnorm: False
    batchnorm: False

  polyline_encoder:
    layers: [31, 32, 64, 128]
    pre_activation: False
    pre_batchnorm: False
    batchnorm: False
```

```yaml

roadgraph_mcg_encoder:
  block:
    c_bias: True
    mlp:
      n_layers: 3
      n_in: 128
      n_out: 128
      bias: True
      batchnorm: False
      dropout: False
  agg_mode: "max"
  running_mean_mode: "real"
  alpha: 0.1
  beta: 0.9
  n_blocks: 5
  identity_c_mlp: False

agent_linear:
  n_layers: 3
  n_in: 512
  n_out: 512
  bias: True
  batchnorm: False
  dropout: False

agent_mcg_encoder:
  block:
    c_bias: True
    mlp:
      n_layers: 3
      n_in: 512
      n_out: 512
      bias: True
      batchnorm: False
      dropout: False
  agg_mode: "max"
  running_mean_mode: "real"
  alpha: 0.1
  beta: 0.9
  n_blocks: 5
```

```yaml
    identity_c_mlp: False

  decoder_config:
    MCGDecoder:
      trainable_cov: False
      size: 512
      mcg_predictor:
        block:
          c_bias: True
          mlp:
            n_layers: 3
            n_in: 512
            n_out: 512
            bias: True
            batchnorm: False
            dropout: False
        agg_mode: "max"
        running_mean_mode: "real"
        alpha: 0.1
        beta: 0.9
        n_blocks: 5
        identity_c_mlp: False
      DECODER:
        layers: [512, 512, 401]
        pre_activation: True
        pre_batchnorm: False
        batchnorm: False

    MLPDecoder:
      mlp1:
        layers: [512, 1024, 2048]
        pre_activation: False
        pre_batchnorm: False
        batchnorm: False
      mlp2:
        layers: [64, 128, 241]
        pre_activation: True
        pre_batchnorm: False
        batchnorm: False
```

## A.3 Closed Loop Train Script

Listing 3: Closed Loop Train Script

```python
def train(args):
# config
 set_random_seed(42)
 config = get_yaml_config(args.config)
```

## A.4 Update State Function

Listing 4: Update State Function

```python
def update_input_for_next_step(current_input, coordinates, yaws, probas, config):
 # We need to get the best ( highest prob ) predictions
 max_indices = probas.argmax(dim=2)

 # Gather the coordinates and yaws for the most probable trajectories
 batch_size, num_agents, num_trajectories, timesteps, coord_dim = coordinates.shape
 selected_coordinates = torch.zeros(batch_size, num_agents, timesteps, coord_dim, device=coordinates.device)
 selected_yaws = torch.zeros(batch_size, num_agents, timesteps, 1, device=yaws.device)

 for i in range(batch_size):
    for j in range(num_agents):
        selected_coordinates[i, j] = coordinates[i, j, max_indices[i, j]]
            selected_yaws[i, j] = yaws[i, j, max_indices[i, j]]


 normalizarion_means = {
 "history/lstm_data": np.array([3.6018e+00,1.4909e+00,-3.4022e-03,4.7697e+00,1.9604e+00,4.3876e+00,0,0,0,0,0,0,0],dtype=np.float32),
 "history/lstm_data_diff": np.array([0.2246,-0.0044,0.0003,-0.0003,0,0,0,0,0,0,0],dtype=np.float32),
 "history/mcg_input_data": np.array([3.6018e+00,1.4909e+00,-3.4022e-03,4.7697e+00,1.9604e+00,4.3876e+00,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],dtype=np.float32),
 "road_network_embeddings": np.array([3.6018e+00,1.4909e+00,-3.4022e-03,1.9266e+02,1.3466e-02,5.4262e-02,1.0721e-01,-1.1991e-02,6.1582e
    +00,3.0463e+00,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],dtype=np.
    float32),}

normalizarion_stds = {
"history/lstm_data": np.array
    ([29.7143,20.0370,1.5417,6.0304,0.5791,1.5877,1,1,1,1,1,1,1],dtype=np.
    float32),
"history/lstm_data_diff": np.array
    ([0.6929,0.2720,0.0719,0.5725,1,1,1,1,1,1,1],dtype=np.float32),
"history/mcg_input_data": np.array
    ([29.7143,20.0370,1.5417,6.0304,0.5791,1.5877,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    dtype=np.float32),
"road_network_embeddings": np.array([29.7143,20.0370,1.5417,1.3049e
    +03,6.9937e-01,7.1257e-01,8.1720e-01,5.6312e-01,1.2635e+00,3.0021e
    +00,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],dtype=np.float32),}
   if config["train"]["data_config"]["dataset_config"]["normlization"]:
       for k in normalizarion_means:
           if k in current_input:
               current_input[k] = (current_input[k].detach().cpu() -
    normalizarion_means[k]) / (normalizarion_stds[k] + 1e-6)
               current_input[k].clamp_(-15, 15)
       current_input[f"history/lstm_data_diff"] *= current_input[f"history
    /valid_diff"]
       current_input[f"history/lstm_data"] *= current_input[f"history/
    valid"]
   dict_to_cuda(current_input)

   return current_input
```