DAI-Labor
Distributed Artificial Intelligence Laboratory

A|O|T

System Design Specifications of System Design Specifications of Experiments on different Sensor Setups for AD

CONFIDENTIAL!

Version number	Version 2.0
Distribution Group	Internal
Main responsible Coordinator	DAI-Labor, TU-Berlin
Current State Date	-

Applications of Robotics and Autonomous Systems
Team Members: Moritz Wassmer, Ege Atesalp, Jakob Wirén, Julian Klitzing

Table of Contents

1 Introduction 3

1.1 Problem Definition 4

1.2 Objectives 4

1.3 Use cases / scenarios 4

1.4 General Constraints 5

2 State-of-the-Art 5

3 System Requirements, Architecture and Specifications 6

3.1 Requirements 6

3.2 System Architecture 6

3.3 System Specifications 7

4 Timeline 11

5 Frameworks and Tools 12

6 Method 12

7 Results 13

8 Conclusion 14

9 References 16

1 Introduction

Over the last decade, following the rise of popular electric car-brand Tesla, the field of autonomous driving has grown immensely, and for the last ten years an exponential increase has occurred in publications regarding relevant topics [1], illustrated in Fig. 1.

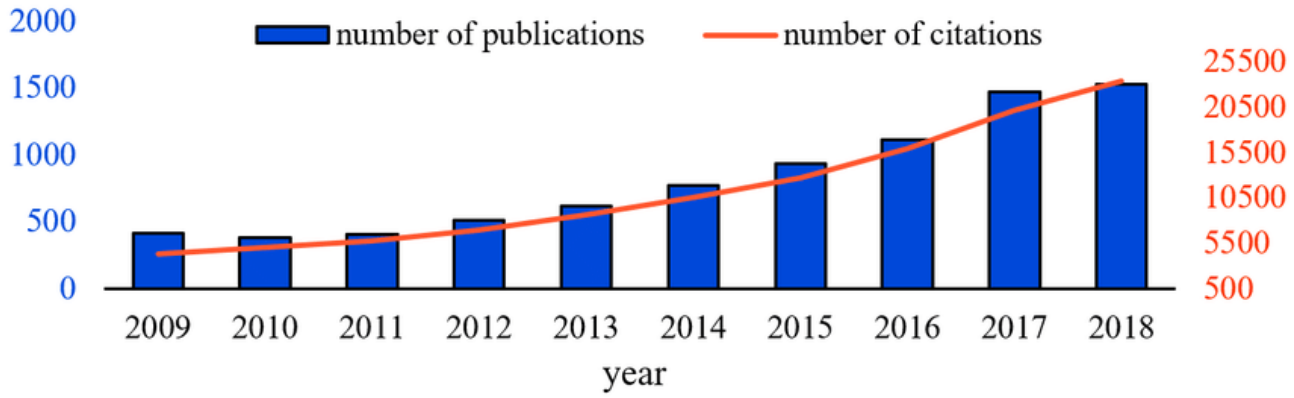

Figure 1: Numbers of publications and citations in autonomous driving research from 2009 to 2018 [2].

As the field of autonomous driving has grown, so has the number of approaches tackling the problem. Two widely accepted methods are the modular paradigm and the End-to-End paradigm. The modular paradigm makes use of the attractive engineering principle called divide-and-conquer - separating the problem of autonomous driving into clearly different subtasks. Such sub-modules classically include Perception, Sensor Fusion, SLAM, Path-planning, Motion-planning. Within each sub-module a set of sub-sub-modules are quite often present, within Perception the tasks of Object detection, tracking, segmentation and depth estimation are generally found as an example. The end-to-end paradigm is in this aspect fundamentally different. Although the modularity and D&C approach presents many benefits such as easier debugging and interpretability, the complexity of the self-driving task requires a vast number of submodules. The end-to-end approach makes the opposite trade-off, sacrificing interpretability and debugging for having a smaller number of modules in the pipeline. By focusing on learning holistic models, end-to-end driving works by mapping raw sensor input directly to control signals for maneuvering. This “simple” approach of course puts huge requirements on the network architecture, data amount, training methods and sensor choice to work coherently.

As the market for autonomous vehicles explodes, the demand for hardware in terms of sensors is as much of a logistical problem as it is an autonomous system engineering problem. Which sensors are truly required for a robust and safe autonomous system? This question is as relevant in both self-driving paradigms, but which multimodal sensor set-up (multimodal sensor set-ups have been shown to increase performance in relevant key perception tasks compared to unimodal set-ups [2]) provides optimal performance remains less explored in the end-to-end field. As such, this project paper will aim to answer the question:

How does the choice of multimodal sensor set-up affect the performance of a given NN architecture, when evaluated in the autonomous driving simulator CARLA?

1.1 Problem Definition

Goal: Compare and evaluate the effect of different sensor-setups in a "controlled" environment in CARLA.

Motivation:

- Aid informing the choice of sensor-setup in the autonomous vehicle sphere.
- Providing knowledge for start-ups/autonomous vehicle producers and researchers regarding the benefits and difficulties of different popular sensor set-ups in a certain testing environment.
- Inspire more research on the interplay between hardware choices and End-to-End architectures.

Problem definition: This project tries to add to the existing body of research about sensor-related design choices in End-to-End autonomous driving by comparing multimodal sensor set-ups and their performance in a simulated environment. To do so we develop a pipeline to experiment with different sensor setups while keeping the dataset and model architecture relatively fixed.
Our aim is to compare the performance of RGB-only and a multimodal RGB-LiDAR set up using ResNet as the benchmark architecture.

1.2 Objectives

1. **Sensor choices**: Decide on a set of sensors and different combinations (setups).
2. **Expert Driver**: Choose an expert driver for CARLA.
3. **Dataset**: Generate a Dataset with the expert driver.
4. **E2E Model**: Implement an E2E model capable of driving autonomously.
5. **Data Preprocessing** (for all sensor setups): Build a pipeline to generate compatible data for the chosen architecture.
6. **Training**: Train the different models with varying sensor setups and varying data sizes.
7. **Evaluation**: Conduct the experiments and compare the sensor setups

1.3 Use Cases and Scenarios

The general use of this experiment is to find out how different sensor setups affect the performance of autonomous driving, to get an understanding of how sensors complement each other (and possibly how good they are on a standalone setup). Consequently, the project does not have explicit use cases in practice for the resulting agents. The use cases and scenarios where it could be used in practice therefore highly depends on whether it is possible to find a well-defined data generation script that is adaptable, as well as finding a (pretrained) architecture. Also, training highly depends on computing power.
All in all, it is difficult to make a clear statement in advance on which scenarios we expect the agents to handle. It could vary between SOTA performance on public benchmarks to simple arriving at the target in very easy conditions (same map, no pedestrians, no vehicles, good

weather, easy route). SOTA use cases would involve very “extreme” scenarios, like defined here CARLA Autonomous Driving Leaderboard[3]. Depending on how well the models work, we are going to use respective harder benchmarks.
A small scenario hierarchy is presented below. It is planned to use a public benchmark/leaderboard to evaluate. Still, it cannot be guaranteed that those scenarios are being tested explicitly in the benchmarks.
Route completion and low infractions is expected.

- Easy environment: empty roads, good weather.
- Harder environment: other agents (cars, cyclists, pedestrians), varying weather.
- “Extreme” Situations: CARLA Autonomous Driving Leaderboard [3].

1.4 General Constraints

- The architecture, dataset and evaluation are going to be fixed, to the extent it is possible, to one which can handle all sensors which we want to support.
- A set of sensor setups which are found feasible for the scope of the project and interesting for research must be decided upon.
- No offroad scenarios
- The dataset size may be limited/decreased if necessary, in order to maintain reasonable training time.
- Policy learned is only applied to the car and it’s setup as it was trained on

2 State-of-the-Art

As a starting point for grasping the complex field of end-to-end autonomous driving and the different methods found in that field, [4] provides an overview of existing methods, input and output modalities, network architectures and evaluation schemes. It provides a balanced introduction and plenty of tips regarding many important concepts such as data-set balancing, fusion methods and CARLA benchmarks. Our report takes inspiration from the lessons presented in [4]. As our research aims to look specifically into multimodal approaches to end-to-end autonomous driving, further inspiration was taken from papers stemming in this field. One such example of a paper would be [5]. This paper analyzes whether combining RGB and depth modalities, such as via RGB-D or LiDAR, produces better end-to-end AI drivers than relying on a single modality, followed by examination of the differences between multimodal sensor fusion occurring early, mid, or late in the pipeline. With CARLA as a testing platform, [5] conclude that early fusion multimodality outperforms single modality. Thirdly, the effect of data-collection and selection is a highly relevant question for our research. For this, [6] has been used to draw inspiration. The paper investigates the influence of several data design choices regarding training and validation of deep driving models trainable in an end-to-end fashion. They look at the amount of training data, validation design and random seeding as well as non-determinism. Their recommendations regarding data generation and route selection have affected the engineering choices leading to this report.

3 System Requirements, Architecture and Specifications

Since we are rather conducting an experiment than implementing new AD agents, we are focusing on our Pipeline in this part.

3.1 Requirements

1. **Sensor choices**: Decide on a set of sensors and different combinations (setups).
   - Chosen sensors and sensor setups must be relevant in current research.
   - Chosen sensors and sensor setups must be compatible with CARLA.
2. **Expert Driver**: Choose or find an expert driver for CARLA.
   - The chosen expert driver must be configurable for the defined scenarios.
3. **Dataset**: Generate a Dataset with the expert driver.
   - Data must cover diverse routes, scenarios and other agents to train a model which is at least able to arrive at targets.
   - Must capture all sensor measurements for both setups.
4. **E2E Model**: implement an E2E model capable of driving autonomously given navigational commands.
   - Should be able to handle varying sensor setups.
   - Should output a policy vector containing steering, throttle and brake.
   - Should respect traffic rules.
   - Optional: Should follow navigational commands.
5. **Data Preprocessing** (for all sensor setups): Build a pipeline to generate compatible data for the chosen architecture.
   - Must be able to handle all supported sensor data.
   - Transform it into a NN suitable representation.
6. **Training**: Train the different models with varying sensor setups.
   - The training method must train the models sufficiently but also not take too long.
   - The training method must optimize hyperparameters in a feasible way.
   - It must be possible to save the models for reusing it for our autonomous agent later.
7. **Evaluation**: Conduct the experiments and compare the sensor setups
   - Must measure interesting measures according to current research (e.g. Route Completion (RC), Infractions Score (IS)).

3.2 System Architecture

In Fig. 2 the system architecture diagram is illustrated. The data engineering related components are highlighted in gray, the data science related components in green and the evaluation related components in blue.

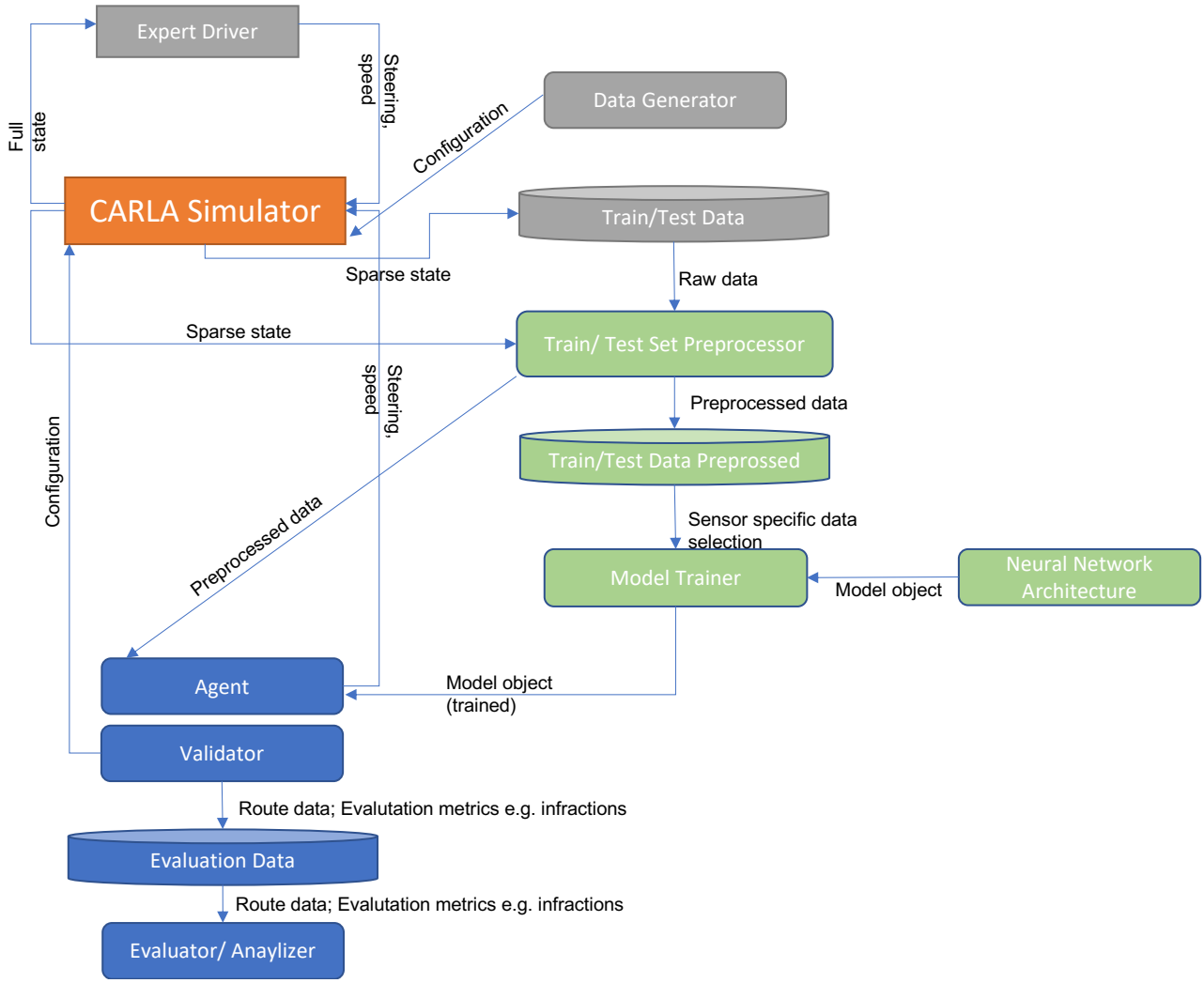

Figure 2: System Architecture with Inputs and Outputs.

3.3 System Specifications

The inputs and outputs of the individual nodes can be found in the systems architecture diagram in 3.2. In the following, only the functionality of each individual node is briefly defined.

CARLA Simulator

Used for data generation and interaction of our trained agent with the environment. For this experiment we have used CARLA 0.9.10.1 [7].

Expert Driver

The expert driver is a hardcoded agent that can perform “perfect” driving in the CARLA simulator and is thus used for generation of the train and test data sets. It is a rule-based expert algorithm used and Its performance is an upper bound for the learning-based agent. The autopilot has access to the complete state of the environment including vehicle and pedestrian locations and actions. The expert driver used is the one from [8], a slightly changed version of the design presented in [9].

Data Generator

Used to initialize the CARLA simulator environment (i.e. driving conditions such as other vehicles, pedestrians, …) and to instantiate the expert driver. It then runs the simulation and saves the data. Data Generator also has the feature of noise to the steering to create noise-injected data to train the agent in more extreme cases. The implemented noise-generation schema is [10 seconds of noise -> 10 seconds of data collection -> 10 seconds of noise..] similar to one presented in [4] .

Train/Test Data

This data is the foundation for model training and evaluation. The database stores the *sparse state* emitted by the CARLA Simulator. The *sparse state* consists of all defined sensor data, vehicle control signals and vehicle state measurements.

The final dataset used for model training consists of the readily available dataset generated by [9] to 95.6%. On top of that, the remaining 4.4% were generated using the *Data Generator* to yield noise injected data. The main characteristics and attributes of the dataset are:

- Driving Length: 1 day 14h
- Raw size: 61 GB
- Number of towns: 9
- Chosen sensors: 3 RGB (160 x 960), LiDAR, Speedometer, …
- Additional inputs: Navigational Command (turn left, turn right, go straight, …)
- Chosen target values: Steer, Throttle, Brake

The entire data was split in 80.2% training set, 6.4% validation set 1 and 13.4% validation set 2. In the training set, data captured from all towns except from Town 06 is present. In validation set 1 data captured from towns used in the training set but from different routes is present. Validation set 2 only contains data captured from Town 06. The final models were only evaluated on validation set 2 to ensure complete non-intersecting training and validation sets.

Train/Test Set Preprocessor

This module performs preprocessing tasks on the data stored in the Train/Test Data database to prepare them for neural network usage.

The RGB images are resized, scaled to yield values in the interval [0, 1] and finally Z-transformed. The scalar speed value is solely Z-transformed whereas the command type is one-hot encoded.

Specific to LiDAR is the transformation step from 3D point cloud data to 2D BEV images, which are necessary to reduce storage size and fit the 3D data into 2D networks. The transformation function is fully parameterized, customizable and can be used for any CARLA 3D LiDAR data. Both RGB and LiDAR (2D) images also go through normalization steps to increase training efficiency.

Neural Network Architecture

This module defines the neural network architectures to be trained and evaluated.

The final selected model architecture for the “RGB” model can be seen on the left side of Fig. 3 and is heavily inspired by [11]. It takes an RGB image captured by a camera that is processed by a ResNet18 architecture, where the last classification layer was discarded. Furthermore, it takes a scalar speed value obtained by a speedometer. Last, it takes a one-hot encoded command that

states the direction in which the car has to go. All inputs pass some layers (ResNet18 or dense layers) to yield intermediate representations that are subsequently fused in a “Mid-Fusion” fashion using concatenation. The fused representation passes a MLP and finally the control commands “steer”, “throttle” and “brake” are predicted.
On the right side of Fig. 3, the final selected architecture for the “RGB-LiDAR” model (Python class) is illustrated. This architecture represents an extension to the “RGB” model in the sense that an additional LiDAR branch was added. This LiDAR branch consists of another ResNet18 that expects a BEV RGB image obtained through transformation of the original sensed LiDAR point cloud.

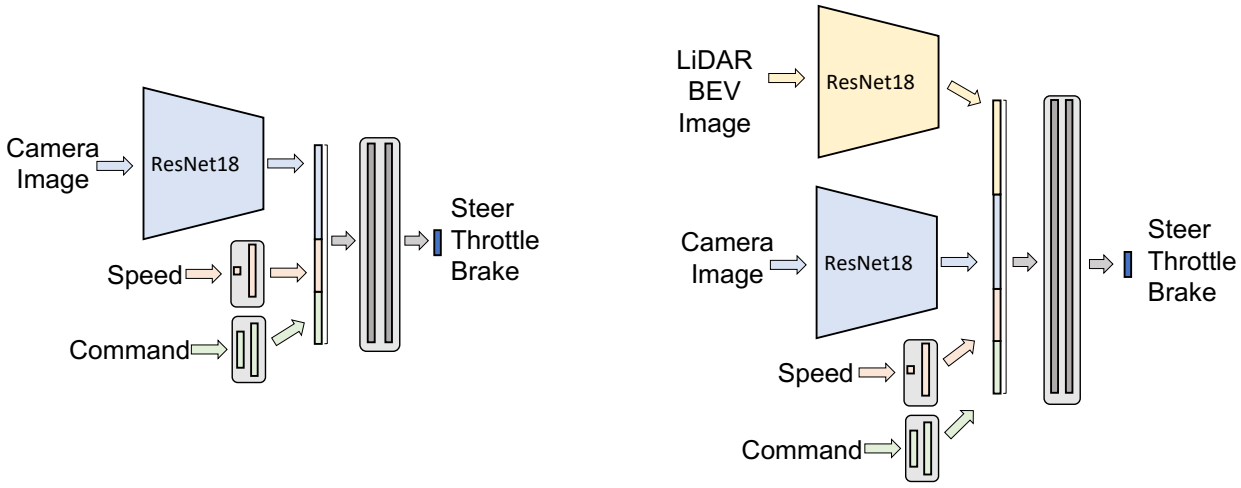

Figure 3: Final selected architectures for "RGB model" (left) and "RGB-LiDAR model" (right).

For our specific dataset, the RGB image is a concatenated image stemming from three front cameras. Furthermore, dropout layers were added to both models where the dropout rate was set to 0.25. Additionally, we used pretrained weights for all ResNet18 stemming from training on the ImageNet dataset.

Model Trainer

This module trains the given model architecture according to defined schedules, hyperparameters etc., saves the model weights and the performance statistics.
The learning rate was set to 0.0001, the chosen optimizer was *Adam* and the weights for the individual L1 loss terms for “steer”, “throttle” and “brake” were set to 0.45, 0.5 and 0.05, inspired by [11].

Agent

The agent is the instance that interacts with the CARLA simulator. It receives a state depending on the sensor configurations and uses the trained network to make the control decisions.
However, hardcoded rules can be implemented in the agent as well. We made use of this to tackle the “inertia problem”. The inertia problem can result in not starting to move again after stopping for example at a traffic light. To mitigate this problem, we implemented a threshold such that after a certain time, where the car doesn’t move, we force the agent to release the brake and push the gas for a very short time. While this introduces some risk, that the car might run red lights or crash

into other actors in the environment, which occasionally happens, it appears to increase the overall performance of the agent dramatically. We optimized this behavior by giving a push, just as much such that the car does break out of a speed of 0 [m/s] resulting in just a few centimeters driven with the neural networks outputs being overridden. This implementation still enables the agent to brake again when for example during the push, if there is a red light on. Note that the steering signal from the network is still being used during the process, in other words, just the brake and throttle is affected by this process. The major downside regarding evaluation for this process is that we introduce more hyperparameters that could potentially be tuned such as how long to wait for forcing a push and how much and how long to override the throttle values.

Validator

The validator, akin to the data generator, initializes a simulator environment and activates the Agent using the trained model. It runs simulations, capturing driving performance metrics for all test tracks. We use the official CARLA Leaderboard as our validator, offering comparable metrics between agents and a standardized performance measure in CARLA.

The CARLA Leaderboard utilizes predefined routes in diverse situations (freeways, residential districts) and weather conditions. Our agent is initialized at a starting point and instructed to drive to a destination point for each route. Performance is measured using metrics such as route completion, red light running, pedestrian collisions, and route deviation. Infractions during testing are recorded, and the global infractions compress data into events per kilometer.

Drawing from TransFuser, we employed the Longest6 benchmark, featuring 36 routes averaging 1.5km in length, comparable to the official leaderboard. Each route has unique environmental conditions, combining one of six weather states (Cloudy, Wet, MidRain, WetCloudy, HardRain, SoftRain) with one of six daylight conditions (Night, Twilight, Dawn, Morning, Noon, Sunset).

Evaluation Data

This data contains the performance measures of the autonomous agents for every tested track. In our case the evaluation data is captured in a JSON format with an array of objects, where each object represents a specific route. Each object contains key-value pairs for the different performance metrics (route completion, traffic infractions, collisions, route deviation, efficiency, and global infractions). The value associated with each key corresponds to the performance metric for that route.

Evaluator/ Analyzer

The evaluator visualizes and analyzes the evaluation data regarding overall performance of the autonomous agents. From that we are going to extract our results for the experiments i.e. comparing the driving performance between the different sensor setups. The evaluator and analyzer are custom-built, simple visualization tools in the form of Jupiter notebooks that take in the json(s) contained in the evaluation data.

4 Timeline

Time	Concept
W 1-2	Understand the topic, set up HW and SW. Tasks and roles delegated. All preparation for work done.
W3	First presentation - MS1. SDS Hand-in.
W 4-5	Finalize all design choices and constraints by the end of November. Collect a large and general database to begin working with.
W 6	Post-process data and begin training the initial network set-up (V1).
W 7-10	Training, improving, evaluation of V1 results.
W10	Internal Milestone/Deadline. Each working sub team holds a presentation for the rest of the team to ease documentation. V2 of architecture and code finalized. At this point the questions that have arisen during W7-10 must be considered and evaluated.
W12	M2 Presentation
W14	Internal Milestone/Deadline. Each working sub team holds a presentation for the rest of the

	team to ease documentation. Internal evaluation of V2 performance and critical examination of priorities in the last weeks. Final tweaking (hopefully) to reach V3/final product.
W 17	M3 Presentation. Final submission.

Table 1: Timetable.

5 Frameworks and Tools

For simulation, custom dataset collection, and performance evaluation, the widely used simulator CARLA was used. When training our own models, the PyTorch framework was employed. GitLab served as the tool for project planning and code version management. Open-source code from GitHub/GitLab was utilized for data generation, evaluation of our AD agents, and model architecture. Some examples include:

- GitHub - autonomousvision/transfuser: [PAMI'22] TransFuser: Imitation with Transformer-Based Sensor Fusion for Autonomous Driving, [CVPR'21] Multi-Modal Fusion Transformer for End-to-End Autonomous Driving
- GitHub - carla-simulator/leaderboard: CARLA Autonomous Driving leaderboard
- GitHub - carla-simulator/scenario_runner: Traffic scenario definition and execution engine

6 Method

The methodology of the project in general followed the classical model development lifecycle with the order of the development following a “logical” order, from the creation of data to the mode that trains on the data, and ultimately to the evaluation of the model.

Figure 4: Method - Step by step procedure.

Members of the group had different responsibilities in various steps of the pipeline with constant internal discussion and proper use of project management tools such as issue boards to solve tasks in the intersections and propose new ideas.
To quickly summarize the development cycle, after learning the basics of the CARLA Client and the evaluation tools through testing, we started creating a model development pipeline using PyTorch as ML library and Kaggle collaboration environment. Our initial goal was to create a RGB only

baseline model using a simple convolutional model architecture. We incrementally added different features and made changes on top of the baseline, changes including tuning the model architecture, adding noise data, etc. Meanwhile, we developed a LiDAR preprocessing function to convert the 3D data into 2D data appropriate for convolutional networks. Ultimately, we created a RGB-LiDAR agent using the middle fusion method, which we then compared with the modified RGB agent to finally arrive at the results below.

7 Results

The training and validation loss convergence is illustrated in Fig 5. The “RGB” model converges after epoch 10 with a validation loss of 0.0888 whereas the “RGB-LiDAR” model converges after epoch 18 with a validation loss of 0.0619.

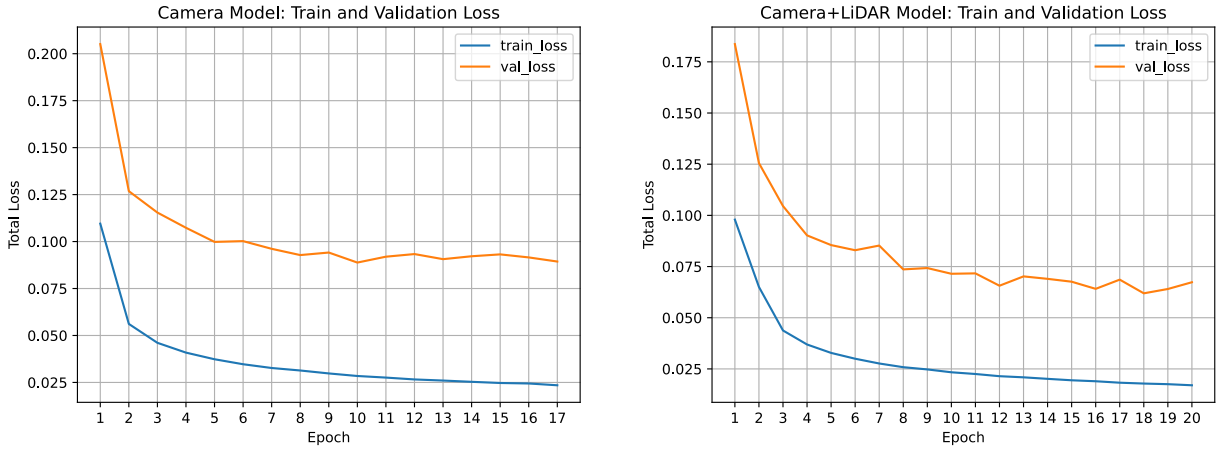

Figure 5: Losses of "RGB Model" (left) and "RGB-LiDAR Model" (right).

On the other hand, the quantitative outcomes of the chosen Longest6 benchmark, are summarized in Table 2.

The "RGB-LiDAR" model outperforms the "RGB" model in two out of three primary performance metrics, namely "Avg. driving score" and "Avg. infraction penalty". However, in terms of "Avg. route completion", the "RGB" model achieves a higher score.

When it comes to more granular metrics (“Collisions with pedestrians”, …, “Agent blocked”), the "RGB-LiDAR" model surpasses the "RGB" model in five out of eight metrics, with "Collisions with pedestrians" being excluded due to a testing bug that caused instantaneous disappearance of the pedestrians after spawning.

The "RGB-LiDAR" model appears to be superior to the "RGB" model due to its higher score in "Avg. driving score". However, it should be noted that the "RGB-LiDAR" model performed moderately worse in "Avg. route completion" and significantly worse in "Route deviation". This raises the possibility that the "RGB-LiDAR" model achieved a higher "Avg. infractions penalty" and consequently a higher "Avg. route completion" due to coincidentally encountering easier driving scenarios when deviating from the intended path.

	RGB	RGB-LiDAR
Avg. driving score ↑	9.601	**13.365**
Avg. route completion ↑	**39.452**	33.597
Avg. infraction penalty ↑	0.358	**0.464**
Collisions with pedestrians ↓	0.000	0.000
Collisions with vehicles ↓	3.031	**1.601**
Collisions with layout ↓	0.421	**0.378**
Red lights infractions ↓	**0.512**	0.608
Stop sign infractions ↓	0.162	**0.123**
Off-road infractions ↓	0.855	**0.273**
Route deviations ↓	**0.389**	0.828
Route timeouts ↓	0.097	**0.032**
Agent blocked ↓	**0.943**	0.956

Table 2: Results of the "Longest6" benchmark with best result marked in bold for each metric.

From a visual standpoint, we can corroborate the statistical findings on "Collisions with vehicles". The "RGB" model exhibits considerably higher crash rates, even in simpler scenarios (e.g. cruising straight behind a decelerating vehicle), compared to the "RGB-LiDAR" model. Our assumption is that the LiDAR's birds-eye-view data provides a more abstract representation of obstacles, predominantly free from specific car shapes and designs, resulting in superior generalization. However, it was also visible that the “RGB-LiDAR” model often took the wrong turn at intersections which explains the comparably bad performance on the “Route deviation” metric.

8 Conclusion

In this section we want to summarize what we have achieved but also want to discuss limitations and directions for future work.
Looking at the System Design Specifications, we were able to implement all relevant components of the framework which we wanted to create. Also, the interplay between the components that we aimed to achieve remained as planned. For that we made use of prior research and integrated solutions from freely accessible repositories and research papers, mainly building upon the source code of the Transfuser Paper for everything related to the Simulation environment such as Evaluation and Data Generation.
In terms of the comparison of the RGB and LiDAR setups, RGB-LiDAR model appears to perform better on average, which is consistent with AD research [5]. While the RGB model outperforms in terms of route completion, the RGB-LiDAR model seems to be driving more safely. The RGB-LiDAR model's route completion suffers mostly from taking the wrong turns during intersections which should be the next step to be optimized in future work. Within our Evaluation environment, a

route deviation results in the simulation to be canceled, therefore has a huge impact on route completion.

Regarding time management, we were not able to keep up with the schedule initially planned. There are a few reasons for this that we were able to identify. We had communications issues, members dropping out of the team and unequal work distribution. The problem with work distribution was mainly due to not everyone having access to hardware able to run the respective components such as CARLA but also a lack of GPUs or clusters to perform model training and having no platform to efficiently handle the data sizes which we had to cope with. To the best of our knowledge, there is no freely available cloud service which could have provided us with a faster/more efficient pipeline for model training. We didn't receive hardware from university nor received resources to buy cloud frameworks such as databricks. Also, an undetected bug delayed development significantly. This limited our project to simple architectures.

Architectures could be improved in various ways. The papers inspiring our architecture themselves are quite old. There are various new approaches to imitation learning which clearly outperform the architectures used [8]. But also building upon our chosen architecture, future work could consider 3D convolutions, making the models able to get an intuition of speed of other actors in the environment, which we strongly believe is a perception element that is missing in our models, especially in crossings with multiple actors being on the junction at the same time. Also, harder settings for training data generation could be considered to make trained models more prepared for the tough benchmark. Regarding evaluation, cyclists were not present in the evaluation environment, which could be added in future work.

Regarding data, we made the decision quite early to use preexisting datasets, mostly because it wasn't feasible to create our own dataset with any of our local computers. We later created a noisy dataset on our own (using modified versions of already existing scripts) to solve various challenges, we were having with the agent. One potential research question for the future could be a more quantitative analysis of the effects of noisy training data on the final performance, which we lack in the current state.

Another aspect that was quite limited in our project was the number of different sensor configurations being tested and compared. Due to the restrictions in training power, we limited our setup to 2 networks. However, the pipeline to create new models with new sensor configurations is provided with the project, mostly since data already comes with multiple sensor outputs. One would probably need to create a specialized preprocessing step (like LiDAR transformation), however adding a new modality to the model architecture is relatively easy due to middle fusion strategy. Exploring additional sensor combinations could give more context into the underlying technical constraints and advantages of each individual sensor type.

Nevertheless, we were able to implement a complete pipeline necessary to train DL models using PyTorch using high quality data with necessary interfaces to evaluate on a state-of-the-art evaluation benchmark making future development for more advanced architectures easy. Despite suffering early on from steep learning curves for various technical and operational aspects of our project, we are now confident in our ability to identify and solve various tasks and challenges to iteratively increase the performance of our agent.

9 References

[1] "Key Ingredients of Self-Driving Cars" by Rui Fan, Jianhao Jiao, Haoyang Ye, Yang Yu, Ioannis Pitas, Ming Liu

[2] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, “Frustum PointNets for 3D object detection from RGB-D data,” in Int. Conf. on Computer Vision and Pattern Recognition (CVPR), 2018

[3] Retrieved from: https://leaderboard.carla.org/#task

[4] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman and N. Muhammad, "A Survey of End-to-End Driving: Architectures and Training Methods," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1364-1384, April 2022, doi: 10.1109/TNNLS.2020.3043505.

[5] Y. Xiao, F. Codevilla, A. Gurram, O. Urfalioglu and A. M. López, "Multimodal End-to-End Autonomous Driving," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 537-547, Jan. 2022, doi: 10.1109/TITS.2020.3013234.

[6] Klingner, M., Müller, K., Mirzaie, M., Breitenstein, J., Termöhlen, J.-A., & Fingscheidt, T. (2022). On the choice of data for efficient training and validation of end-to-end driving models.. https://doi.org/10.48550/arXiv.2206.00608

[7] Dosovitskiy, A., Ros, G., Codevilla, F., López, A., & Koltun, V. (2017). CARLA: An open urban driving simulator. arXiv preprint arXiv:1711.03938. Retrieved from https://arxiv.org/pdf/1711.03938.pdf

[8] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, TransFuser: Imitation with Transformer-Based Sensor Fusion for Autonomous Driving. 2022.

[9] Jaeger, B. (2021). Expert drivers for autonomous driving (Master's thesis). Technical University of Munich. Retrieved from https://kait0.github.io/assets/pdf/master_thesis_bernhard_jaeger.pdf

[10] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, End-to-end Driving via Conditional Imitation Learning. 2018. MODEL ARCHITECTURE, HYPERPARAMETERS