# Model Optimization in Deep Learning Based Robot Control for Autonomous Driving

Sergio Paniego, Nikhil Paliwal, and JoséMaría Cañas

***Abstract*—Deep learning (DL) has been successfully used in robotics for perception tasks and end-to-end robot control. In the context of autonomous driving, this work explores and compares a variety of alternatives for model optimization to solve the visual lane-follow application in urban scenarios with an imitation learning approach. The optimization techniques include quantization, pruning, fine-tuning (retraining), and clustering, covering all the options available at the most common DL frameworks. TensorRT optimization for specific cutting-edge hardware devices has been also explored. For the comparison, offline metrics such as mean squared error and inference time are used. In addition, the optimized models have been evaluated in an online fashion using the autonomous driving state-of-the-art simulator CARLA and an assessment tool called Behavior Metrics, which provides holistic quantitative fine-grain data about robot performance. Typically the performance of robot applications depends both on the quality of the control decisions and also on their frequency. The studied optimized models significantly increase inference frequency without losing decision quality. The impact of each optimization alone has also been measured. This speed-up allows us to successfully run DL robot-control applications even in limited computing hardware. All the work presented here is open-source, including models, weights, assessment tool, and dataset, for easy replication and extension.**

***Index Terms*—Imitation learning, machine learning for robot control, sensorimotor learning.**

## I. Introduction

THE autonomous driving field has received an enormous amount of attention in recent years. This trend has been followed in both academia and industry, especially since some autonomous driving solutions are starting to be deployed in real-world environments. The autonomy of a vehicle is usually divided into 5 levels, according to the SAE J3016 Standard [1]. The levels range from no automation in level 0 to full autonomy in level 5. The current level of development has generated solutions that are qualified as level 4 autonomy level. There is still a need for development in both research and industry to generate level 5 solutions, where the real benefits would be higher [2].

Manuscript received 4 August 2023; accepted 10 November 2023. Date of publication 23 November 2023; date of current version 6 December 2023. This letter was recommended for publication by Associate Editor W. Pan and Editor J. Kober upon evaluation of the reviewers' comments. This work was supported in part by Google Summer of Code (GSoC). *(Corresponding author: Sergio Paniego.)*

Sergio Paniego and JoséMaría Cañas are with Universidad Rey Juan Carlos, 28933 Madrid, Spain, and also with JdeRobot Organization, 28922 Madrid, Spain (e-mail: sergio.paniego@urjc.es; josemaria.plaza@urjc.es).

Nikhil Paliwal is with Saarland University, 66123 Saarbrücken, Germany, and also with JdeRobot Organization, 28922 Madrid, Spain (e-mail: nikhil.paliwal14@gmail.com).

Digital Object Identifier 10.1109/LRA.2023.3336244

The advancement in the field in recent years is partly thanks to the progress in deep learning, especially the evolution of graphical computation units (GPU), the algorithm advancements, the availability of high-quality annotated datasets [3], and the emergence of high-fidelity autonomous driving simulators, like CARLA [4] or TORCS [5]. This evolution has also been endorsed by the emergence of autonomous driving and robotics competitions, including AWS DeepRacer [6] and DuckieTown [7].

A prevalent grouping of autonomous driving solutions divides them into end-to-end and modular. The end-to-end solutions [8], [9], [10], [11], [12], [13] directly generate outputs from an input in a one-way direction, with only one module. Alternately, the modular approaches [14] connect several modules that communicate with each other and that generate the final commands from the input involving several computational steps. This division can also be seen in the number of tasks that the driving procedure involves, ranging from localization or control to perception or planning [15].

Recent autonomous driving progress involves diverse sensors and actuators, especially in vision solutions, utilizing computer vision algorithms to interpret images for vehicle control. While initial vision-based autonomous driving solutions were limited [16], recent strides indicate significant advancements [17]. NVIDIA's PilotNet [8] notably introduced a deep learning solution using a convolutional neural network (CNN) for feature extraction and control command translation from visual data.

The vision-based solutions are usually generated using DL models, which are high-demanding computational solutions. An important component with them is the available computing hardware as the performance of robot applications depends not only on the quality of the control decisions but also on their frequency, the higher the better. Some autonomous vehicles or robots are equipped with high-performance hardware, others are not. Updating to faster-computing hardware is beneficial but it is not always a real option. A possible solution is to optimize the DL model with different techniques [18].

There are an extensive number of optimization techniques, including quantization of the computations [19], [20], pruning of some parts of the model [21], fine-tuning (retraining) with optimization aware techniques, or clustering some components. Common DL frameworks like PyTorch [22] and TensorFlow [23], [24] incorporate these techniques. Some optimizations are hardware-specific and require specialized hardware, such as TensoRT for Nvidia GPUs [25]. These techniques are usually combined in the development of an optimized

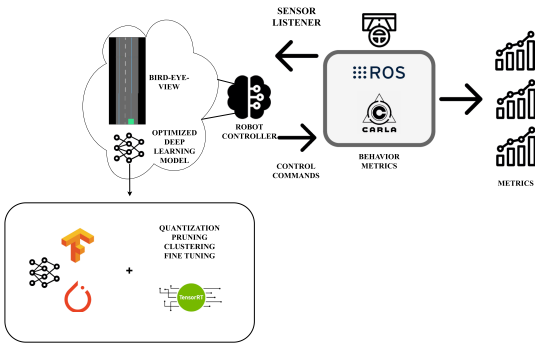


Fig. 1. End-to-end autonomous driving pipeline using Behavior Metrics software and a robot controller based on a DL model that drives the vehicle based on its sensory data.

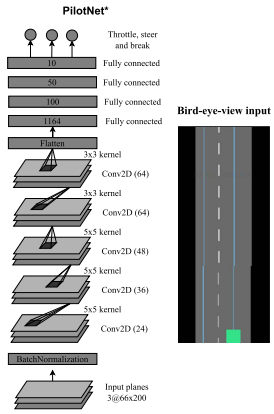


Fig. 2. PilotNet* architecture detail (left) and bird-eye view input example (right).

efficient DL model, always considering the trade-off between the optimization percentage and quality of the model outputs.

Regarding the assessment of the models, the *offline evaluation* compares the model outputs to the supervised data, for instance, the control outputs generated by an expert agent. The more similar the better. While these offline metrics are useful in image classification or object detection tasks, they could be not enough for evaluating a solution for end-to-end robot control. A model with good offline metrics may perform poorly when driving an autonomous vehicle, as more conditions are involved in successful driving, such as robustness and inference frequency. That is why we also conducted *online evaluation* of the models, measuring the performance of the whole robot behavior in simulation, with conditions closer to the real-world environment, when the driving decisions are made by the DL model.

For training the DL models, the autonomous driving field already presents varied curated datasets for different tasks, including nuScenes [26] or BDD100K [27] for visual perception, nuPlan [28] for planning or comma AI [29] for lane-following (lane-keeping) in a real-world scenario.

In this letter, we introduce and discuss several common model optimization techniques for end-to-end autonomous driving control, in particular for the visual lane-follow application. We apply and compare them to a baseline model to understand how the optimizations impact the system performance and efficiency using an imitation learning approach [30]. We generate a new supervised dataset from expert data because the already available ones are not directly suitable for the lane-follow application. Some studies have already addressed similar questions [31] but they do not consider DL models as possible controllers, which is our focus and part of the innovation. Our study includes two well-known deep learning frameworks (Tensorflow-Keras and PyTorch) with their optimizations toolkits (TensorFlow Model Optimization Toolkit [24]) and the optimizations provided by NVIDIA's TensorRT framework [25] for this particular hardware. The research hypothesis is that by including optimization techniques, configuring and adapting them accordingly to the DL model and the problem setup, we can obtain similar quality levels of autonomous driving with smaller and faster models as compared to the baseline model. We leave outside this study the finding of the best expert agent (we use a good-enough one) for driving and the comparison between classic and neural control, since the focus is on the implications of the DL model optimizations alone. We provide a series of experiments that validate this hypothesis, including both offline and online model comparisons in test scenarios using the state-of-the-art CARLA simulator for autonomous driving. The online experimental validation is conducted using Behavior Metrics (see Fig. 1), a software tool for the evaluation of end-to-end solutions for autonomous driving that includes additional evaluation metrics. We provide all models, architectures, modified software and datasets as open-source.

## II. Optimizing End-to-End Imitation Learning Models for Lane-Follow Robot Control

This section outlines the baseline deep learning model for lane-follow robot control using imitation learning. It incorporates state-of-the-art optimization techniques prevalent in DL model development and details the training process.

### A. Baseline Architecture

The baseline DL architecture is based on PilotNet end-to-end model [8], replicating the architecture provided in the letter in both Tensorflow and PyTorch frameworks. The only difference in our baseline model, named PilotNet*, is that it provides three control commands as outputs. Instead of only generating steering commands, as in the original work, PilotNet* generates throttle, steering, and brake. Our architecture modifies the baseline one only on the final part of the former model. In Fig. 2, a detailed diagram of the modified architecture is provided.

The perception data used as input to the DL model is the bird-eye view of the vehicle in the scenario (see Fig. 2 for an example). This bird-eye view simplifies the perception task that the model has to conduct. Instead of having the immense entropy of a front-camera image, from which the model has to extract relevant features, the bird-eye view is a segmented image that includes only the pertinent classes for this particular problem (vehicles, pavement, road lanes...), reducing the complexity. Even though the perception task is simplified, the results and conclusions

presented here are general, and applicable for more complex perception setups (frontal camera vision) following the same ideas. They are simplified only for the sake of the focus on model optimization.

### B. Dataset and Training

Following the imitation learning approach, the dataset used for training the DL model is extracted from CARLA where an expert agent drives a car traveling through the scenario (Town 01) keeping the lane. At the same time, the control decisions and the bird-eye images are recorded. The raw dataset is unbalanced since the majority of cases involved in autonomous driving are usually driving straight forward and the amount of other cases, such as turn situations, remains small. To reduce this bias, some techniques are already common, such as DAgger [32]. In this case, we record turns more times to oversample the dataset with such cases. In addition, we also use common machine learning training techniques like shuffling, normalization, and data augmentation (modifications of brightness, contrast, gamma channel, hue saturation, PCA color augmentation, gaussian blur, and horizontal affine transformations). Horizontal affine transformation is really important for this particular problem. For this transformation, the input visual data is displaced some points horizontally, generating more examples that could be found online by the vehicle. Considering this transformation, the output is also altered accordingly.

### C. Deep Learning Optimization Techniques

Several optimizations are applied to this PilotNet* baseline model, which was implemented and trained in two of the most common DL frameworks, PyTorch and Tensorflow (TF). The optimizations used in this work are implemented using the framework support in which the model architecture is created. In this case, PyTorch and TensorFlow model optimization toolkits (TF Lite). In addition, the TensorRT framework for model optimization provided by NVIDIA was also used, since the available GPU for the experimental validation was from this company and those optimizations apply only to these architectures.

The optimization techniques used are (for a detailed diagram of some of them see Fig. 3):

- *Quantization:* Approximation of neural network inner values that use floating-point numbers to low bit width numbers. This optimization usually changes the floating-point numbers to float16 or even int8 precision.
- *Pruning:* Technique based on removing unnecessary connections or parameters of the DL architecture for reducing the size of a neural network.
- *Fine-tuning (retraining):* Some techniques also involve a few fine-tuning steps for generating the optimized model. This process of fine-tuning is combined with the rest of the optimizations to generate more precise final results.
- *Clustering:* Group similar weights in a neural network. Similar to pruning but in this case the weights are combined and represented with a single centroid value.

All these techniques improve the model efficiency, accelerating computations and reducing the model size. A reduction in model size can be critical in computation systems with sharp memory constraints. The acceleration of computations is also critical for systems with low computational capacity. From these base techniques, each DL framework has support for some of them and their combinations, as detailed in Table I. Applying each optimization requires tuning for the particular problem and model.

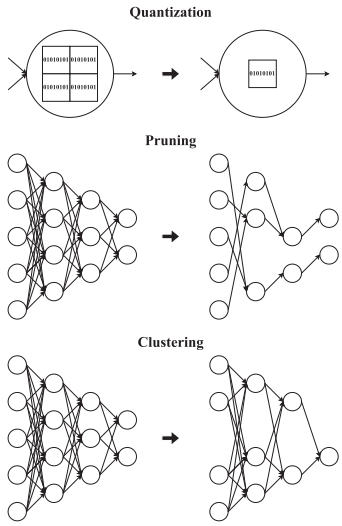


Fig. 3. Optimization techniques diagrams.

## III. Assessment of Optimized End-to-End Robot Control Models

In this section, we present the assessment software tool, named Behavior Metrics, and the framework used for experimental evaluation of the optimized deep learning models, including CARLA simulator and the custom evaluation metrics created and used. In a DL research project, there are some common offline metrics for evaluating the performance of the generated model, such as mean squared error (MSE) and the inference time. These standard metrics are usually not enough when evaluating the performance of a robot controller since there are more variables involved and needed to understand the model's execution. A model with good MSE metrics may perform poorly in a robot control task such as autonomous driving for visual lane-follow. An end-to-end model that generates satisfactory control decisions but at a low pace could lead to dangerous situations in a real-world scenario, where the vehicle may be at high speeds, making it useless for real application.

We introduce a software tool, named Behavior Metrics [33], for the experimental assessment of end-to-end model performance of robotic applications (see Fig. 1). It is built on top of an autonomous driving simulator, CARLA, for running experiments comparing different robot controllers that drive the vehicles, inspired by other research projects [34]. It uses ROS [35] for communicating with the simulator. It supports all the characteristics provided by CARLA and supports other virtual sensors, like the bird-eye view. The robot controller is the master brain of the robot or vehicle, which receives the

TABLE I
SUMMARY OF OPTIMIZATION CONFIGURATIONS AND THEIR SUPPORTED TECHNIQUES, INCLUDING THE DEVELOPMENT FRAMEWORK

| Optimization type | Framework | Quantization | Pruning | Fine-tuning | Clustering |
|---|---|---|---|---|---|
| **Tensorflow** | | | | | |
| **Baseline (No optimizations)** | Tensorflow | × | × | × | × |
| **TensorFlow Model Optimization Toolkit (TF Lite)** | | | | | |
| **Baseline in TF Lite (No optimizations)** | TF Lite | × | × | × | × |
| **Dynamic Range Quantization** | TF Lite | Int8+Float16 | × | × | × |
| **Integer Quantization** | TF Lite | Int8 | × | × | × |
| **Integer (float fallback) Quantization** | TF Lite | Int8+Float16 | × | × | × |
| **Float16 Quantization** | TF Lite | Float16 | × | × | × |
| **Quantization Aware Training** | TF Lite | Int8 | × | ✔ | × |
| **(Random sparse) Weight pruning** | TF Lite | × | ✔ | × | × |
| **(Random sparse) Weight pruning quantization** | TF Lite | Int8 | ✔ | × | × |
| **Cluster preserving quantization aware** | TF Lite | Int8 | × | ✔ | ✔ |
| **Pruning preserving quantization aware** | TF Lite | Int8 | ✔ | ✔ | × |
| **Sparsity and cluster preserving quantization aware training (PCQAT)** | TF Lite | Int8 | ✔ | ✔ | ✔ |
| **TensorFlow TensorRT** | | | | | |
| **(TensorRT) Float32 Quantization** | TF TensorRT | Float32 | × | ✔ | × |
| **(TensorRT) Float16 Quantization** | TF TensorRT | Float16 | × | ✔ | × |
| **(TensorRT) Int8 Quantization** | TF TensorRT | Int8 | × | ✔ | × |
| **PyTorch** | | | | | |
| **Baseline (No optimizations)** | PyTorch | × | × | × | × |
| **PyTorch Model Optimization Toolkit** | | | | | |
| **Dynamic Range Quantization** | PyTorch | Int8+Float16 | × | × | × |
| **Static Quantization** | PyTorch | Int8 | × | × | × |
| **Quantization Aware Training** | PyTorch | Int8 | × | ✔ | × |
| **Local Prune** | PyTorch | × | ✔ | ✔ | × |
| **Global Prune** | PyTorch | × | ✔ | ✔ | × |
| **Prune+Quantization** | PyTorch | Int8 | ✔ | ✔ | × |
| **PyTorch TensorRT** | | | | | |
| **(TensorRT) Float32 Quantization** | PyTorch TensorRT | Float32 | × | ✔ | × |
| **(TensorRT) Float16 Quantization** | PyTorch TensorRT | Float16 | × | ✔ | × |
| **(TensorRT) Int8 Quantization** | PyTorch TensorRT | Int8 | × | ✔ | × |

✔: Supported. ×: Unsupported

input data, performs some computations, and generates the final control commands that are sent to the vehicle actuators (throttle, steer, and brake) on an end-to-end basis. The robot controller can house a DL model, reinforcement learning policy [12], or explicitly programmed approach. Behavior Metrics conducts batch experiments, facilitating easy performance measurement across various models in different settings.

The CARLA Autonomous Driving Leaderboard [1] is a common evaluation framework used by the community for research purposes. Our work is inspired by CARLA's evaluation metrics and procedures, but as they do not fit our objectives completely, we have extended Behavior Metrics to generate additional evaluation metrics for a full understanding of the performance. They both provide an online and deeper evaluation of model performance when driving a car in the visual lane-follow application.

CARLA's metrics like collisions and lane invasions are taken into account. In addition, these four additional metrics, specifically developed for this project and added to Behavior Metrics, are also considered. They are the following, and are contributions of this letter:

- *Controller frequency:* Iterations completed by the controller in one second. Measured in Hz.
- *GPU inferences frequency:* Inference iterations completed by the GPU in generating control commands from the input visual data. Measured in Hz.
- *Position deviation mean per km:* Positional deviation from the middle of the lane during experimental validation per kilometer.
- *Successful runs:* Combination of some already included metrics and the most informative metric. We consider a run as successful when it ends without collisions and lane invasions while the average speed is close (±5 km/h) to the expert agent (25 km/h) and the position deviation is close to 0 (<1.5 m).

[1] https://leaderboard.carla.org/

## IV. EXPERIMENTS

This section presents the experiments conducted for the evaluation of the differences between non-optimized and optimized DL models. The baseline model, PilotNet*, has been implemented in two of the most popular deep learning frameworks, Tensorflow and PyTorch, to also understand their differences. With these two models, we applied a series of framework-level optimizations and their combinations, tuning each of them appropriately to gain insight into each optimization's advantages. In addition, hardware-level optimizations were also explored using the NVIDIA TensorRT framework that enhances model performance for NVIDIA GPUs. We do not include a comparison with prior baseline methods since to the best of our knowledge this is the first work that compares different optimization techniques for end-to-end control in autonomous driving based on visual perception.

The hardware used for this experimental validation includes 2 NVIDIA GeForce RTX 3090 GPUs. All the presented experiments are easily reproducible, with all the components released open-source [36], including the models' weights, architectures and dataset, software comparison tool, and simulator. Since the optimizations are state-of-the-art and build the enhancements on some precise components of the models, the results may vary depending on how modern and powerful the test hardware is, but the ideas described here are general and applicable to any type of hardware and software setup.

Experiments were conducted to gauge the impact of optimized models in robot control and assess how control decision quality and frequency influence vehicle behavior. Various optimizations were compared individually to determine their significance in performance. The experiments focused on visual lane-following in urban scenarios without traffic or obstacles, emphasizing model optimization importance. For clarity, only one CARLA scenario with random starting positions was used although the results are applicable to similar DL end-to-end control tasks.

### A. Model Performance Offline Evaluation Table

Offline evaluation of the models gives a general idea of how the model performs. In Table II, we present the offline evaluation results for each of the optimized models. This evaluation is conducted using batches of 64 images. During online testing, for each inference, only one image is used as input. Considering the use of a GPU for inference, the maximum gain in inference

TABLE II
OFFLINE EVALUATION OF BASELINE MODELS AND THEIR OPTIMIZED VERSIONS

| Optimization type | Model size (MB) | MSE test | GPU inferences frequency (Hz) |
|---|---|---|---|
| **Tensorflow** | | | |
| **Baseline (No optimizations)** | 18.35 | 0.015 | 22.38 |
| **TensorFlow Model Optimization Toolkit (TF Lite)** | | | |
| **Baseline in TF Lite (No optimizations)** | 6.09 | 0.01590 | 610.91 |
| **Dynamic Range Quantization** | 1.54 | 0.01593 | 764.11 |
| **Integer Quantization** | 1.54 | 0.01591 | 1104.86 |
| **Integer (float fallback) Quantization** | 1.54 | 0.01588 | 1216.82 |
| **Float16 Quantization** | 3.05 | 0.015902 | 614.68 |
| **Quantization Aware Training** | 1.54 | 0.01317 | 1181.89 |
| **(Random sparse) Weight pruning** | 6.09 | 0.00919 | 609.92 |
| **(Random sparse) Weight pruning quantization** | 1.53 | 0.00911 | 755.86 |
| **Cluster preserving quantization aware** | 1.54 | 0.01202 | 1195.73 |
| **Pruning preserving quantization aware** | 1.54 | 0.00932 | 1179.22 |
| **Sparsity and cluster preserving quantization aware training (PCQAT)** | 1.54 | 0.00859 | 1182.76 |
| **TensorFlow TensorRT** | | | |
| **(TensorRT) Float32 Quantization** | 6.29 | 0.01079 | 2579.90 |
| **(TensorRT) Float16 Quantization** | 6.29 | 0.01079 | 2368.63 |
| **(TensorRT) Int8 Quantization** | 6.35 | 0.04791 | 2954.75 |
| **PyTorch** | | | |
| **Baseline (No optimizations)** | 6.09 | 0.00435 | 229.83 |
| **PyTorch Model Optimization Toolkit** | | | |
| **Dynamic Range Quantization** | 1.94 | 0.01206 | 675.54 |
| **Static Quantization** | 1.61 | 0.01207 | 1367.17 |
| **Quantization Aware Training** | 1.61 | 0.01109 | 853.94 |
| **Local Prune** | 6.12 | 0.01085 | 695.05 |
| **Global Prune** | 6.12 | 0.01096 | 705.23 |
| **Prune+Quantization** | 1.61 | 0.01094 | 852.60 |
| **PyTorch TensorRT** | | | |
| **(TensorRT) Float32 Quantization** | 6.12 | 0.00957 | 4377.41 |
| **(TensorRT) Float16 Quantization** | 6.12 | 0.00957 | 3986.85 |
| **(TensorRT) Int8 Quantization** | 6.18 | 0.00969 | 4058.54 |

time is expected to be achieved with batches of several images instead of only using one image for each timestamp because of the parallelism of the GPUs.

In general, the model size is reduced when optimizing (compressed). The explanation for this fact is that the optimizations reduce the complexity of the model, hence reducing the space that it needs for storage. The optimization that causes the most reduction is found for the models using int8 quantization (see Table I for details), due to the lower precision of the numbers in the network, they need less memory space. For each optimization, we fine-tune its parameters to their utmost limits, carefully balancing them on the threshold just before a noticeable decline in quality occurs. For example, adjusting aggressively clustering and pruning parameters to their maximum settings. The precise parameter values are contingent upon each specific optimization and combination (comprehensively documented in the accompanying open-source code).

Looking at the GPU inferences frequency, optimized models generate much better results than models without optimizations with a maximum gain of 135 times of improvement (Baseline TF compared to (TensorRT) int8 quantization, which is the best result for Tensorflow framework). If we do not consider TensorRT framework, models with int8 quantization (see Table I for details) generate the best results in terms of GPU inferences frequency with 50 times faster results at best. Other combinations of optimizations do not improve the frequency further. Since the operations are conducted with a lower precision because it uses int8, the calculations are much faster and require less memory. If we consider TensorRT, the frequency increases to the highest point (135 times faster at best). It is due to the hardware-level optimizations of TensorRT, which are specific for certain hardware combinations and include the already presented optimization techniques with fine-grained adjustments for the hardware. Similarly to considering model size, the introduction of clustering or pruning is ineffective in increasing further the GPU inference frequency when quantization is involved.

With the MSE, we can check whether the model generated results are close to the supervised examples or not. Looking at the MSE evaluation results, the values remain close to the baseline model, improving the results for certain optimizations. This is important since we can generate models of a similar quality in terms of MSE with lower size and more GPU inference frequency. In this case, the best results appear when combining several optimizations or all available options, e.g., PCQAT (see Table II for detailed values). The only outlier appears in the TensoRT Tensorflow int8 quantization, which generates worse MSE values. An explanation for this is that the model optimization limit has been surpassed, causing the final performance to drop significantly.

To summarize, the offline evaluation results show that optimizations provide more efficient models and the combination of all the optimizations generates the best results when considering model size, MSE, and GPU inference frequency. These results are repeated for both PyTorch and Tensorflow DL frameworks and their combinations with TensorRT. It is also important to point out that by comparing the TensorRT framework with DL framework-level optimizations, TensorRT generates the best results, although it is hardware-specific and more complicated to apply for different hardware combinations.

### B. Robot Control Online Evaluation Table

In this experiment, we evaluate the baseline and each optimized model while driving a car on an urban test scenario inside the CARLA simulator in regular conditions and generate evaluation metrics with Behavior Metrics. Each model runs inside a vehicle controller, driving for two minutes, five times, and starting from a random position in the map, which allows the agent to drive that amount of time without encountering junctions or other unconsidered scenarios in that run. An illustrative video with the baseline model driving is available at [37]. In Table III, we can see a summary of the results for each model. We select only the most informative evaluation metrics retrieved from Behavior Metrics for this particular experiment.

The controller frequency is always lower than the GPU inference frequency. The controller is the core computational system that drives the vehicle and performs several operations at each iteration. It receives the bird-eye image and transforms it before giving it to the model for inference. Since the DL model is inside the controller, the GPU inference frequency (model inferences alone) will always be higher than the controller frequency. We can see that the number for GPU inference frequency is similar to the ones obtained in the offline evaluation but slightly lower. This

TABLE III
COMPARISON OF MODELS AND THEIR OPTIMIZED VERSIONS IN A TEST ENVIRONMENT CONSIDERING SOME MEASURED METRICS PROVIDED BY BEHAVIOR METRICS

| Model | Controller frequency (Hz) | GPU inferences frequency (Hz) | Position deviation mean per km (m) | Average speed (km/h) | Successful runs |
|---|---|---|---|---|---|
| **Tensorflow** | | | | | |
| **Baseline (No optimizations)** | 15.94 | 20.25 | 0.33 | 20.5 | 100% |
| **TensorFlow Model Optimization Toolkit (TF Lite)** | | | | | |
| **Baseline TF Lite** | 70.36 | 438.30 | 0.25 | 23.5 | 100% |
| **Dynamic Range Quantization** | 74.65 | 568.24 | 0.24 | 24.3 | 100% |
| **Integer Quantization** | 75.43 | 771.83 | 0.24 | 23.1 | 100% |
| **Integer (float fallback) Quantization** | 76.49 | 868.64 | 0.26 | 23.1 | 100% |
| **Float16 Quantization** | 69.63 | 432.42 | 0.26 | 22.8 | 100% |
| **Quantization Aware Training** | 77.17 | 823.85 | 0.25 | - | 100% |
| **(Random sparse) Weight pruning** | 70.03 | 437.44 | 0.25 | 23.0 | 100% |
| **(Random sparse) Weight pruning quantization** | 74.27 | 575.97 | 0.26 | 27.1 | 100% |
| **Cluster preserving quantization aware** | 75.46 | 850.83 | 0.24 | 24.2 | 100% |
| **Pruning preserving quantization aware** | 76.27 | 837.16 | 0.24 | 26.5 | 100% |
| **Sparsity and cluster preserving quantization aware training (PCQAT)** | 79.50 | 867.15 | 0.24 | 24.2 | 100% |
| **TensorFlow TensorRT** | | | | | |
| **(TensorRT) Float32 Quantization** | 79.10 | 1067.79 | 0.44 | 22.6 | 100% |
| **(TensorRT) Float16 Quantization** | 62.03 | 967.09 | 0.49 | 22.9 | 100% |
| **(TensorRT) Int8 Quantization** | 39.89 | 874.78 | - | 22.0 | 0% |
| **PyTorch** | | | | | |
| **Baseline (No optimizations)** | 69.92 | 437.93 | 1.41 | 20.9 | 100% |
| **PyTorch Model Optimization Toolkit** | | | | | |
| **Dynamic Range Quantization** | 72.28 | 579.97 | 0.24 | 23.1 | 100% |
| **Static Quantization** | 77.74 | 1020.53 | 0.26 | 21.6 | 100% |
| **Quantization Aware Training** | 76.37 | 1022.46 | 0.29 | 21.4 | 100% |
| **Local Prune** | 70.43 | 474.88 | 0.28 | 24.1 | 100% |
| **Global Prune** | 70.80 | 475.53 | 0.26 | 23.2 | 100% |
| **Prune+Quantization** | 78.07 | 1014.47 | 0.26 | 25.8 | 100% |
| **PyTorch TensorRT** | | | | | |
| **(TensorRT) Float32 Quantization** | 67.78 | 1030.00 | 0.45 | 20.1 | 100% |
| **(TensorRT) Float16 Quantization** | 68.94 | 1036.99 | 0.43 | 21.2 | 100% |
| **(TensorRT) Int8 Quantization** | 76.11 | 1496.99 | 4.76 | 20.1 | 20% |

difference comes from the introduction of other computational loads like the simulator and the number of images used as input batch.

The controller frequency is always better for optimized models. Having more GPU inference frequency is an indicator of possible higher controller frequencies. We can observe that the numbers are better but considering the GPU inference frequencies, we could expect even better controller frequency results. This situation occurs due to the initialization times of the DL model. When using optimized models, in the first iteration the DL model is loaded into the GPU so it is much slower than all other iterations. Considering the big difference in GPU inference frequency, we can ignore this difference, since it is expected to be negligible in the long term, for instance in longer experiments.

Both baseline models generate excellent results for the number of successful runs, completing all of them. The important point here is that the optimized models retrieve similar results, finishing the experimental evaluations with efficient results too, but with a higher model inference frequency. Looking at the number of GPU inference frequencies, similarly, as we have observed in the previous experiment, int8 quantized models (see Table I for details) are faster for both frameworks (see Fig. 4), generating 47 times faster inferences for Tensorflow and 2.3 times faster at best for PyTorch. Looking at the results of the position deviation and average speed metrics, they are similar to those obtained using the baseline models. The rest of the insights for the optimization's selection described in the previous experiment are also applicable in the online setting.

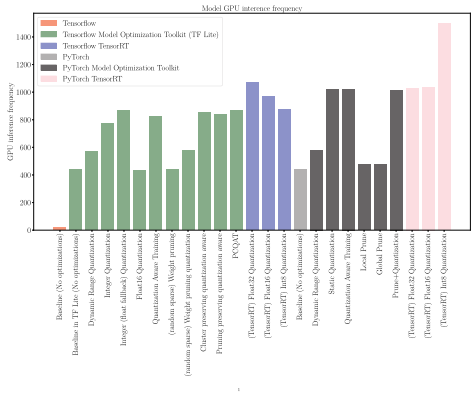


Fig. 4. Detail of each model's GPU inference frequency.

We have proved that the optimized models drive properly in an online test evaluation, without a reduction in the quality of the decisions and increasing their pace. The optimized models have the same overall behavior quality and are faster and smaller. The results are similar for PyToch and Tensorflow, which proves that the ideas are applicable to several configurations. If we consider TensorRT framework optimizations, they achieve the best number considering GPU inference frequency while maintaining the overall behavior quality, measured as successful runs. Considering int8 TensorRT optimizations, neither of them works correctly, which can again be explained because of trying to improve the model too much and reaching its optimization limit, which reduces the final online performance.

A summary of the results is shown in Table IV, including the improvement rate observed over the baseline model. We select the best-optimized models for each DL framework as the model that combines the majority of optimization techniques (quantization, pruning, fine-tuning, and/or clustering) since we have proved that t is the best-performing one. We observe improvement for each optimized option compared to the baseline, obtaining a gaining of a maximum of 47 times improvement in an online evaluation of inference frequency and 5 times of controller frequency. We have proved that optimizing the model is key for robot control efficiency.

### C. *Inference Frequency and Quality of Decisions in Robot Control Performance*

Most successful robots in real world have an important reactive part, which usually has one main control loop. This loop runs iterations at a given frequency, reading sensor measurements and commanding low-level decisions to the actuators on each iteration. For instance, in autonomous driving, there is usually a reactive local navigation controller. When following an end-to-end approach, on each iteration, the sensor data are provided as the input of the DL model, it is called for inference, and its outputs are commanded to the vehicle actuators. The overall quality of the robot's behavior depends both on the quality of the control decisions at each iteration and on the frequency of those iterations.

TABLE IV
COMPARISON SUMMARY OF BEST MODELS PERFORMANCE WITH THE IMPROVEMENT RATE OBSERVER

| Model | Optimization | Model size (MB) | MSE test | Offline GPU inferences frequency (Hz) (improvement) | Controller frequency (Hz) (improvement) | Online GPU inferences frequency (Hz) (improvement) | Successful runs |
|---|---|---|---|---|---|---|---|
| **Tensorflow** | | | | | | | |
| **Baseline** | - | 18.35 | 0.015 | 22.38 | 15.94 | 20.25 | 100% |
| **Best optimized TF Lite** | **Sparsity and cluster preserving quantization aware training (PCQAT)** | 1.54 | 0.00859 | 1182.76 (x52) | 79.50 (x5) | 867.15 (x42) | 100% |
| **Best optimized TF TensorRT** | **(TensorRT) Float16 Quantization** | 6.29 | 0.01079 | 2368.75 (x100) | 62.03 (x3) | 967.09 (x47) | 100% |
| **PyTorch** | | | | | | | |
| **Baseline** | - | 6.09 | 0.00435 | 229.83 | 69.92 | 437.93 | 100% |
| **Best optimized PyTorch** | **Prune+Quantization** | 1.61 | 0.01094 | 852.60 (x3) | 78.07 (x1.2) | 1014.47 (x2.3) | 100% |
| **Best optimized PyTorch TensorRT** | **(TensorRT) Float16 Quantization** | 6.12 | 0.00957 | 3986.85 (x17) | 68.94 (x1) | 1036.99 (x2.3) | 100% |

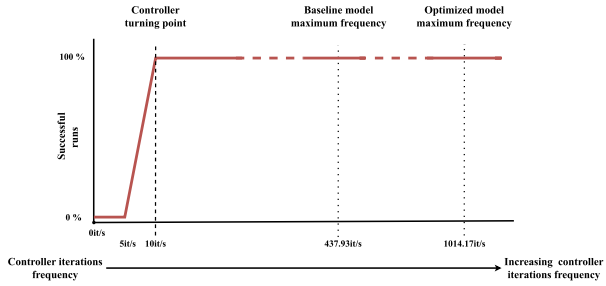


Fig. 5. Quality of the robot behavior (measured as % of successful runs) vs the frequency of the control decisions.

In this experiment, the visual lane-follow driving was the robotics application and the robot controller was the baseline DL model PilotNet*, as it is a successful one. All the control decisions were taken with that DL model and their quality is assumed to be good. The settings, worlds, starting points, etc. were exactly the same as in the previous experiment, but many different frequencies of the controller were enforced, and the corresponding overall quality of the lane-follow application was measured, in terms of successful runs.

To enforce different low controller frequencies, extra dummy computation loads were introduced on each iteration. To enforce high controller frequencies, the simulator itself was stopped and resumed so more controller iterations took place at each second of simulated time.

The results of this experiment are shown in Fig. 5. For that lane-follow application, we found a lower limit of 10 Hz for the controller frequency, a turning point in the robot's performance. Below that frequency threshold, the robot simply fails. Above it, the performance is pretty much the same. It is important to note that the only difference is the frequency of the decisions, their quality is always good as they come from the same DL model. The particular number of the frequency threshold for other applications and scenarios may depend on many factors, such as the car speed (faster cars will require a higher control frequency threshold), the complexity of the robot, and the dynamism of the environment, but this profile seems to appear in many robotics contexts. For this experiment, a simulated Tesla Model 3 with stable speed of 30 km/h driving in scenario Town02 is used.

The results in Fig. 5 also show the potential benefit of optimizing the DL models. For a given computing hardware, achieving faster models may increase the robot controller frequency, and so increase the robot application performance. In limited hardware, it may be even critical when regular models fall below the frequency threshold and optimized ones may be above it.

Finally, with the available hardware GPUs already used in all the previous experiments, we measured the performance at the maximum inference frequency of both the baseline PilotNet* and of the best-optimized model from it. The baseline model reached 437 Hz with PyTorch, far from the minimum 10 Hz. Best optimized model reaches 1014 Hz. They are shown as vertical dotted lines in Fig. 5. Similar increments were obtained with Tensorflow. The particular values are not relevant, but they illustrate the advantage of optimizing the DL model. In this particular visual lane-follow application, with the autonomous car moving slowly, are not critical, but in more demanding applications, higher car speeds, or even this one running on limited computing hardware, may make the difference.

## V. CONCLUSION

In this letter, we have presented and studied several optimization techniques for deep learning models and applied them for the end-to-end visual control of an autonomous vehicle based on imitation learning. The particular application is lane-follow driving in urban scenarios. We have used optimized models and proved experimentally that optimizations improve the final system performance thanks to the speed-up in controller iteration frequency without losing quality on the control decisions. We have applied these optimizations individually and combined, implementing all variants in two different DL frameworks (PyTorch and Tensorflow). Hardware-specific optimizations (TensorRT) were applied too.

These models have been tested and validated offline and online in a state-of-the-art simulator for autonomous driving, CARLA. The experimental results show the impact of each optimization on the final robot application performance. Combining all the optimization techniques and tuning them in consonance with the baseline model, the optimized DL models drive with a similar quality of the control decisions but much faster. The inference frequency of the best-optimized DL model is 47 times faster than the baseline model in the online evaluation. And the

robot controller with it runs 3 times faster, so the optimizations have proven its relevance. The optimized models can be used in a broader variety of hardware, especially low-resource and edge devices. This adaptability extends beyond the conventional scope of autonomous driving research, constituting a significant aspect of our innovation. We have proved these premises experimentally using a comparison software tool, Behavior Metrics, and the evaluation metrics that we have specifically developed, which are also a contribution of the present work. These metrics complement the common MSE on the supervised dataset and those directly provided by CARLA. Altogether they provide a more informative description of the performance of the system.

The dataset, models' weights and architectures, and comparison software tools are provided as open-source materials for the research community, which makes easy the replication of the presented results.

For future work, the ideas presented and proved here should be experimentally validated in a physical vehicle with low hardware resources.

## REFERENCES


[1] SAE International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2021. Accessed: Oct. 10, 2023. [Online]. Available: https://www.sae.org/standards/content/j3016

[2] T. Litman, "Autonomous vehicle implementation predictions implications for transport planning," 2022.

[3] Y. Cabon, N. Murray, and M. Humenberger, "Virtual KITTI 2," 2020, *arXiv:2001.10773*.

[4] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. 1st Annu. Conf. Robot Learn.*, 2017, pp. 1–16.

[5] E. Espié, C. Guionneau, B. Wymann, C. Dimitrakakis, R. Coulom, and A. Sumner, "TORCS, the open racing car simulator," 2005.

[6] B. Balaji et al., "DeepRacer: Educational autonomous racing platform for experimentation with Sim2Real reinforcement learning," 2019, *arXiv:1911.01562*.

[7] L. Paull et al., "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 1497–1504.

[8] M. Bojarski et al., "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*.

[9] M. Bojarski et al., "Explaining how a deep neural network trained with end-to-end learning steers a car," 2017, *arXiv:1704.07911*.

[10] F. Codevilla, M. Müller, A. Dosovitskiy, A. M. López, and V. Koltun, "End-to-end driving via conditional imitation learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 1–9.

[11] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 7151–7160.

[12] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 2070–2075.

[13] Y. Hu et al., "Planning-oriented autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 17853–17862.

[14] C. Gómez-Huélamo et al., "How to build and validate a safe and reliable Autonomous Driving stack? A ROS based software modular architecture baseline," in*Proc. IEEE Intell. Veh. Symp.*, 2022, pp. 1282–1289.

[15] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2019.

[16] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 9329–9338.

[17] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," 2023, *arXiv:2306.16927*.

[18] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–37, Mar. 2023.

[19] R. Gray and D. Neuhoff, "Quantization," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2325–2383, Oct. 1998.

[20] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021, *arXiv:2106.08295*.

[21] S. Vadera and S. Ameen, "Methods for pruning deep neural networks," *IEEE Access*, 2020, vol. 10, pp. 63280–63300, 2022.

[22] A. Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[23] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 265–283.

[24] Tensorflow Authors, "TensorFlow model optimization," 2023. Accessed: Oct. 10, 2023. [Online]. Available: https://www.tensorflow.org/model_optimization

[25] NVIDIA, "TensorRT NVIDIA," 2023, Accessed: Oct. 10, 2023. [Online]. Available: https://developer.nvidia.com/tensorrt

[26] H. Caesar et al., "nuScenes: A multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11618–11628.

[27] F. Yu et al., "BDD100 K: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2633–2642.

[28] H. Caesar et al., "NuPlan: A closed-loop ML-based planning benchmark for autonomous vehicles," 2022, *arXiv:2106.11810*.

[29] E. Santana and G. Hotz, "Learning a driving simulator," 2016, *arXiv:1608.01230*.

[30] M. Bansal, A. Krizhevsky, and A. Ogale, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," 2018, *arXiv:1812.03079*.

[31] Y. Bai, L. Li, Z. Wang, X. Wang, and J. Wang, "Performance optimization of autonomous driving control under end-to-end deadlines," *Real-Time Syst.*, vol. 58, pp. 509–547, Dec. 2022.

[32] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 627–635.

[33] Behavior Metrics contributors, "Behavior metrics," 2023, Accessed: Oct. 10, 2023. [Online]. Available: https://github.com/JdeRobot/BehaviorMetrics

[34] J. de la Peña, L. M. Bergasa, M. Antunes, F. Arango, C. Gómez-Huélamo, and E. López-Guillén, "AD PerDevKit: An autonomous driving perception development kit using CARLA simulator and ROS," in *Proc. IEEE 25th Int. Conf. Intell. Transp. Syst.*, 2022, pp. 4095–4100.

[35] M. Quigley et al., "ROS: An open-source robot operating system," vol. 3, 2009.

[36] Open-source paper resources contributors, "Open-source paper resources," 2023, Accessed: Oct. 10, 2023. [Online]. Available: https://roboticslaburjc.github.io/publications/2023/model_optimization_in_deep_learning_based_robot_control_for_autonomous_driving

[37] Open-source paper resources contributors, "Improved imitation learning with bird-eye view for follow-lane autonomous driving in CARLA simulator," 2023, Accessed: Oct. 10, 2023. [Online]. Available: https://www.youtube.com/watch?v=3KflagFjR8Q