National University
Of Computer and Emerging Sciences

Project Deliverable #2:

TORCS AI Race-Car Controller

Presented to

Sir, Ali Zeeshan

In partial fulfillment

of the requirements for the course of

Artificial Intelligence

By:

Hafsa Imtiaz (22i - 0959)

Abeer Jawad (22i - 1041)

Areen Zainab (22i - 1115)

Section H

TORCS - AI Controller

This report outlines the methodology for collecting telemetry data, developing a model, and training an autonomous driving agent in TORCS (The Open Racing Car Simulator). The aim is to create an Artificial Neural Network (ANN) capable of controlling a race car using real-time telemetry inputs.

The following sections detail the process of dataset generation, model architecture, training strategy, hyperparameters, and performance evaluation of the ANN within the TORCS environment.

Deliverable #1 - Telemetry Data Collection:

Telemetry data was collected from TORCS using a custom Python script ***(telemetry.py***) that logged data every 20 milliseconds. This high-frequency logging captured both sensor inputs and control outputs, storing each frame as a row in a CSV file for supervised training.

Each entry includes:

- **13 core vehicle values** (e.g., lap time, speed, gear, position, angle.
- **4-wheel spin values**
- **19 track sensors** (distances to track edges)
- **36 opponent sensors** (proximity to other cars)
- **5 control outputs** (steering, acceleration, gear, brake, clutch)

This results in 72 input features and 5 output values per frame. A dataset of approximately 350,000 samples was generated and used to train the model, capturing a wide range of driving behaviors and scenarios.

Deliverable #2 - Objective:

The objective is to train an ANN to predict three continuous control outputs—**steering**, **acceleration**, and **braking**—from sensor inputs using a supervised regression approach.

Gear was handled through rule-based logic, and the clutch was excluded as all vehicles were automatic, resulting in constant zero values. The model focuses on smooth, real-time control within the TORCS environment.

1. Data Preprocessing:

The preprocessing begins with reading the CSV file containing the telemetry logs. The model uses 30 inputs and time-series windows of 5 consecutive frames to predict three control outputs: steering, acceleration, and braking.

Input Features Used:

- **Base Inputs (25 total)**:
  - RPM, SpeedX, SpeedY, SpeedZ, TrackPosition, Z
  - Track Sensors (19 total)

- **Engineered Features (5 total):**

  1. **Speed** = $\sqrt{(SpeedX^2 + SpeedY^2 + SpeedZ^2)}$: Total vehicle speed.
  2. **TrackWidth** = Track_1 + Track_2: Width of track straight ahead.
  3. **Upcoming Curvature** = Mean of Track_3, Track_4, Track_5: Captures upcoming turn sharpness.
  4. **DistanceFromCenter** = |TrackPosition|: Measures how far the car is from the track center.
  5. **TrackPositionSquared**: Penalizes extreme lateral deviations more strongly.

Scaling:

- A RobustScaler is applied to key numerical inputs:
  - Scaled: *RPM, SpeedX, Speed, TrackWidth, UpcomingCurvature, DistanceFromCenter, TrackPositionSquared*
  - This handles outliers better than standard scaling.
  - The scaler is saved as ***racing_scaler.pkl*** for later use during prediction.

Data Preparation

- Input features (***X***) are constructed using a sliding window of 5 time steps.
- Targets (***y***) are the control outputs at the 5th frame.
- The dataset is split into 80% training and 20% validation sets.

2. Model Training:

The model used is a multi-output regression model. It uses a hybrid architecture by combining **LSTMs** (Long Short-Term Memory) with **dense layers** to extract both sequential and high-level features.

Model Selection

This hybrid model was selected for TORCS due to its ability to effectively handle both the temporal dependencies and complex feature interactions involved in driving tasks. Driving behavior is highly sequence-dependent; current control decisions are influenced by a series of preceding events along with current input. Long Short-Term Memory (LSTM) models are well-suited for this type of temporal modeling, as they are capable of learning patterns across multiple time steps and retaining relevant context over time.

Alongside the LSTM layers, the Dense layers enhance the model's ability to learn complex and nonlinear relationships among features such as speed, track curvature, and positional data. This architecture hence, enables the model to generate smooth and accurate predictions for continuous control outputs—steering, acceleration, and braking—making it an effective choice for real-time autonomous control in TORCS.

We also experimented with various other models, such as purely LSTM-based, purely dense, and K-fold cross-validation architectures. But none of them performed as well as the current one.

Model Architecture Summary

- Input Shape: (5, 30)
- Output: [steering, acceleration, brake]
- Epochs: 500

Layer	**Details**
Input	(5 timesteps, 30 features)
LSTM #1	128 units, returns sequences
LSTM #2	64 units, returns the last output only
Dense #1	64 neurons, ReLU activation
Dropout	20% rate for regularization
Dense #2	32 neurons, ReLU activation
Output	3 neurons (steering, accel, brake), linear activation

The model was trained using Mean Squared Error (MSE) as the loss function, which helps it learn (aka in training) by penalizing larger mistakes more heavily. Meanwhile, Mean Absolute Error (MAE) was used as a performance metric (aka evaluation) because it's easier to

interpret—MAE directly reflects the average difference between the model's predictions and the actual values.

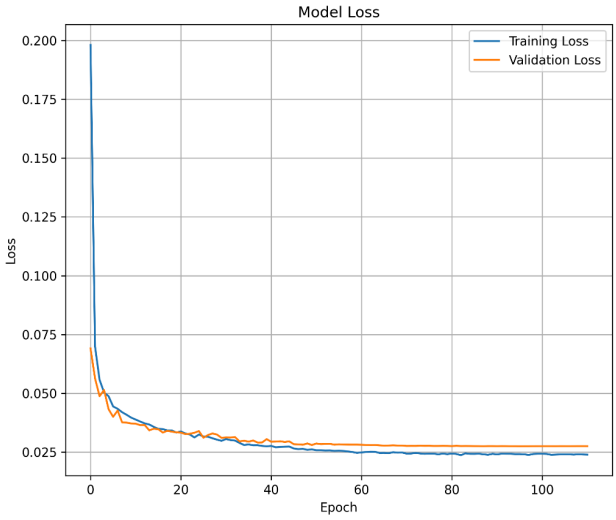
This setup allows the model to effectively learn driving behavior over time while giving a clear sense of how accurate the control outputs—steering, acceleration, and braking—are during evaluation.

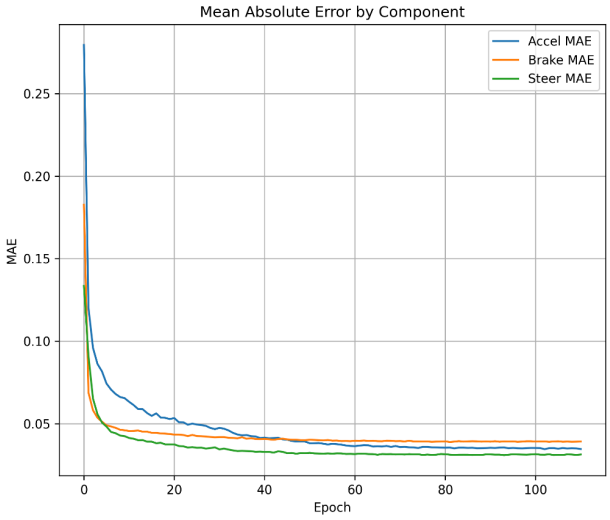
The trained model is saved as ***torcs_model.h5*** and also ***torcs_model.tflite***. These files are loaded by our Python class (***CarControllerANN***), which uses the model and the real-time sensor input to make predictions.

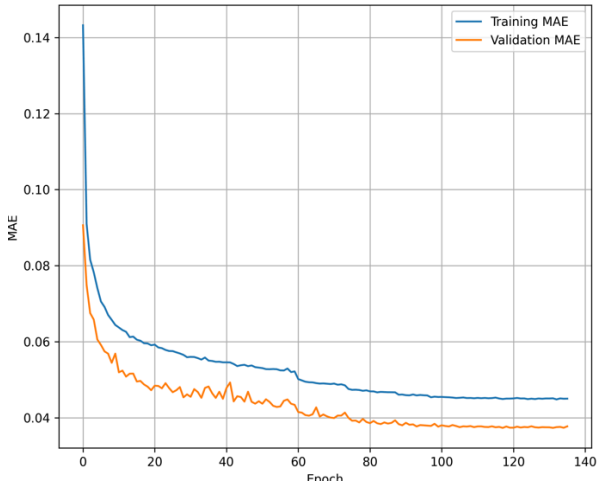
Model Insights

The following chart shows the model's performance on the training data and the testing (validation) data. The first chart shows the MSE (loss) of the model throughout training, over various epochs.

The second chart is the evaluation (the MAE) of each predicted component.







- **Regularization:** Used Dropout (20%) and EarlyStopping to prevent overfitting during training.

- **Performance:** No significant overfitting — validation MAE ≈ training MAE.

- **Optimizer:** Adam updates each weight (of each perceptron) individually by adapting the weights based on past gradients

(derivatives of the loss function w.r.t each weight) to ensure efficient learning.

Contributions

Deliverable #1 - Data collected by all members
Deliverable #2 -
Preprocessing: Abeer Jawad
Model Training: Hafsa Imtiaz
Report - Areen Zainab

Conclusion

This project demonstrates the development of an LSTM-based autonomous driving controller in TORCS. The model was trained on a diverse telemetry dataset and successfully learns to predict continuous control signals using sequential sensor input. The combination of using engineered inputs, time-series modeling via LSTM, and robust (preprocessing &) training strategies resulted in a capable and efficient driving agent.

Although the model is far from perfect and fails in scenarios like awful collisions and rapid turns, these issues can be blamed on the limited dataset we could generate and the inappropriate training facilities. With more data, a more complex model structure, and training time, the model's performance could improve significantly.