AI-Based Car Racing Controller in TORCS

Muhammad Shayan (22i-0773)
Awais Khan (22i-0997)
Ali Haider (22i-1210)

1 Introduction

Welcome to our journey into building an AI-driven racing controller for the TORCS simulator! In this project, we leverage real-time telemetry, deep learning, and TFLite optimization to craft a neural-network-based bot that races competitively on diverse tracks. Our goal is to replace brittle rule-based drivers with a data-driven controller that learns from examples and generalizes to unseen scenarios.

We split into three teammates—Shayan handling data pipelines and model orchestration, Awais leading integration with TORCS and real-time inference, and Ali focusing on network design, training, and evaluation. Together, we navigate challenges from noisy data to sub-10 ms response constraints.

2 Project Objectives

1. **Telemetry Implementation** Parse and preprocess 60+ sensor channels (track edges, speed, RPM, gear, etc.).
2. **Neural Controller Design** Develop, train, and export a feed-forward neural network predicting five actuator commands (accel, brake, clutch, gear, steering).
3. **Real-Time Inference** Integrate a TFLite model within a Python client to respond to TORCS server packets within a 10 ms deadline.
4. **Evaluation & Benchmarking** Compare lap times against baseline drivers and analyze stability across multiple runs.
5. **Report & Documentation** Summarize architecture, implementation details, challenges, team contributions, results, and future directions.

3 Technical Implementation

3.1 Data Pipeline

- **Data Source**: Telemetry from 50+ practice races on three TORCS tracks.
- **Cleaning**: Removed infinite/NaN readings; resolved duplicate `Gear` columns.
- **Feature Selection**: Excluded opponent distances and lap times for solo-driving focus.
- **Normalization**:
  (a) Z-score using stored per-feature means & standard deviations.
  (b) MinMax scaling to $[0, 1]$ for inputs and outputs.
- **Train/Val Split**: 80% training, 20% validation (`random_state=42`).

3.2 Model Training & Export

- **Frameworks**: TensorFlow 2.x, scikit-learn, joblib.
- **Checkpointing**: Saved Keras `.h5` model; supported resume training.
- **TFLite Conversion**: Used `Optimize.DEFAULT` for quantization hints, producing a lightweight `.tflite` (200 KB).

3.3 TORCS Integration

- **Client-Server**: Patched TORCS with `scr_server`; Python UDP client.
- **Parsing**: `msgParser` decodes telemetry; `carState` stores structured state.
- **Inference Loop**:
  (1) Parse sensor message → feature vector.
  (2) Preprocess (Z-score + MinMax).
  (3) Run TFLite inference (< 3 ms).
  (4) Post-process (clamp, round gear).
  (5) Send control string back.

4 Neural Network Architecture

Layer	Units	Activation	Dropout
Input	—	—	—
Dense	128	ReLU	0.2
Dense	64	ReLU	0.2
Dense	32	ReLU	—
Output	5	Linear	—

Table 1: Feed-forward network for actuator prediction.

Dimensions: ~60 inputs (track[19], opponents[36], speed, RPM, gear_in, etc.). **Loss:** MSE. **Metric:** MAE. **Optimizer:** Adam (lr=0.001).

5 Racing Controller

1. `init()`: Specifies 19 range-finder angles from $-90°$ to $+90°$.
2. `drive(msg)`:
   - Update `CarState`.
   - Build feature dict in consistent order.
   - Apply Z-score → MinMax scaling.
   - TFLite inference → raw predictions.
   - Clip values: steer $\in [-1, 1]$, gear $\in \{-1, \ldots, 6\}$, accel/brake/clutch $\in [0, 1]$.
   - Return formatted control message.

6 Challenges & Solutions

Challenge	Solution
Noisy/missing sensor data	Dropped NaN/$\pm\infty$ rows; validated range-finder reliability.
Duplicate `Gear` columns	Auto-renamed inputs vs. outputs early in pipeline.
Real-time (10 ms) constraint	Switched to TFLite; kept model shallow and pre-allocated NumPy arrays.
Scaling consistency	Saved scalers & feature list to mirror training pre-processing in inference.
Lack of baseline	Implemented random-action driver for initial benchmarking (150 s lap).

7 Team Contributions

- **Muhammad Shayan**: Data pipeline design, normalization routines, training orchestration, TFLite export.
- **Awais Khan**: TFLite inference integration, UDP parsing, real-time optimization.
- **Ali Haider**: Network architecture design, training regimen, performance logging.

8 Results and Evaluation

8.1 Training Performance

- **Final Training Loss (MSE)**: 0.0032
- **Validation Loss (MSE)**: 0.0041
- **Validation MAE**: 0.045

9 Conclusion and Future Work

We successfully delivered a neural-network-powered racing controller that outperforms naive baselines, meets real-time constraints, and generalizes across multiple tracks.

Future Directions:

- Recurrent architectures (LSTM/GRU) to capture temporal context.
- Imitation learning from expert human drives.
- Reinforcement learning fine-tuning (e.g., DDPG).
- Sensor fusion (camera + telemetry).
- Further quantization and pruning for embedded deployment.