Muhammad Umer Khan 22i-0780
Mohammad Hamza Imran 22i-0865
Inam Ullah Shaikh 22i-0857
CS-H

AI – Project Report

1. Introduction & Objectives

The goal of this project was to build a neural-network-based controller for the TORCS racing simulator that can drive—steering, throttle, brake, clutch, gear—and also reliably recover from "stuck" situations. To maximize real-time performance, we combine a high-capacity yet efficient Keras. This report describes:

1. Dataset & preprocessing
2. Model architecture & training
3. Testing and Validation

2. Data Collection & Preprocessing

- **Source**: Logged 100 000+ time-step snapshots of TORCS sensor readings when driving various tracks (road shapes, speeds, off-track events).
- **Features (X)**:
    - Continuous:
        - Car orientation & motion: {Angle, SpeedX, SpeedY, SpeedZ, RPM, TrackPosition}
        - 19 range sensors: Track_1…Track_18 (distance to track edges at angles −90°…+90°)
- **Targets (Y)**: Continuous regression of five outputs:

1. **Acceleration** ([0…1])
2. **Braking** ([0…1])
3. **Clutch** ([0…1])
4. **Gear** (real-valued prediction, later rounded)
5. **Steering** (−1…1)

- **Scaling**:
    - Fitted StandardScaler on training features to zero-mean, unit-variance.
    - Saved mean/scale via pickle for driver-time normalization.
- **Train/Val Split**: 80% train, 20% validation (fixed random_state=42) to monitor overfitting.

3. Model Architecture

3.1. Mixed-Precision & Efficiency

- Enabled TensorFlow mixed precision (mixed_float16 globally) for faster matrix ops while forcing final outputs to float32 for numerical stability.

3.2. Layer Layout (Functional API)

Input(shape=[N_features])

→ Dense(512, kernel_initializer="he_normal")

→ GELU activation → BatchNorm → Dropout(0.3)

→ Dense(256)

→ GELU → BatchNorm → Dropout(0.3)

→ Dense(128) → GELU

→ Dense(64) → GELU

→ Dense(32) → GELU

→ Dense(16) → GELU

→ Dense(8) → GELU

→ Dense(5, dtype="float32") // outputs: accel, brake, clutch, gear_raw, steering

- **GELU**: smoother than ReLU, helps continuous control output.
- **BatchNorm + Dropout**: regularization to reduce overfitting on time-series transitions.

4. Training Procedure

4.1. Loss & Optimization

- **Custom Weighted MSE**:

  $$L=B1i=1∑B[1.0(ai−a^i)2+1.0(bi−b^i)2+0.5(ci−c^i)2+2.0(gi−g^i)2+2.0(si−s^i)2]$$

  where weights emphasize gear & steering accuracy (≥100% penalty).

- **Optimizer**: Adam (LR=1e-3).

4.2. Callbacks & Schedules

- **EarlyStopping**: patience=10 (stop when val_loss plateaus).
- **ReduceLROnPlateau**: factor=0.5, patience=5 (further fine-tune after plateau).

- **ModelCheckpoint**: save best-val_loss weights to model/torcs_nn_model.keras.
- **Epochs & Batch**: up to 200 epochs, batch_size=64.

5. Inference-Time Controller Logic

5.1. Normalization & TF Function

- Load saved scaler; wrap inference in a @tf.function predict_fn(x) that normalizes input: (x−μ)/σ

6. Stuck Detection & Recovery

- **Detect stuck**: speedX < 1.0 m/s for >20 consecutive ticks.
- **Reverse mode**: set gear= −1, accel=1.0, steer=0, run for 30 ticks.
- **Forward recovery**: freeze stuck logic for 30 ticks, then resume normal control.

7. Results & Next Steps

7.1. Performance

- **Validation MAE**: ~0.02 for accel/brake, ~0.15 gear RMS error, ~0.1 steer MAE.
- **On-track**: good cornering, smooth speed profiles, reliable un-stuck behavior.

7.2. Future Enhancements

1. **Fine-tuning** the network with on-policy data (behavior cloning + corrective demos).
2. **Reinforcement learning** (PPO/SAC) to learn optimal shift points and recovery strategies end-to-end.
3. **Adaptive gain scheduling** for PD-steer based on speed.
4. **Model pruning/quantization** for faster inference on CPU/GPU-constrained hardware.

Group Members & Contributions

Member	Contribution Summary
All members	Data Collection
Umer	Model Architecture Design
Umer and Hamza	Model Implementation
Umer	Driver Code Implementation
Hamza	Model Optimization and Enhancement
Inamullah	Testing and Validation
Inamullah	Report

Conclusion

This hybrid controller leverages a deep Keras model for rich, learned throttle and steering behaviors, complemented by light, deterministic rules for gear and recovery. The pipeline—from data preprocessing through mixed-precision training, custom loss weighting, to inference-time heuristics—yields a robust, smooth-driving agent in TORCS.