# Drive Like a Human: Rethinking Autonomous Driving with Large Language Models

2025

Sajad Ahmadi, Rahul Chamarthi and Nandhini Ramachandran

Instructor: Dr. Bing Li

Final presentation of Agentic AI course

AuE6930 Milestone #2

Department of AUTOMOTIVE ENGINEERING

CLEMSON MECHANICAL ENGINEERING

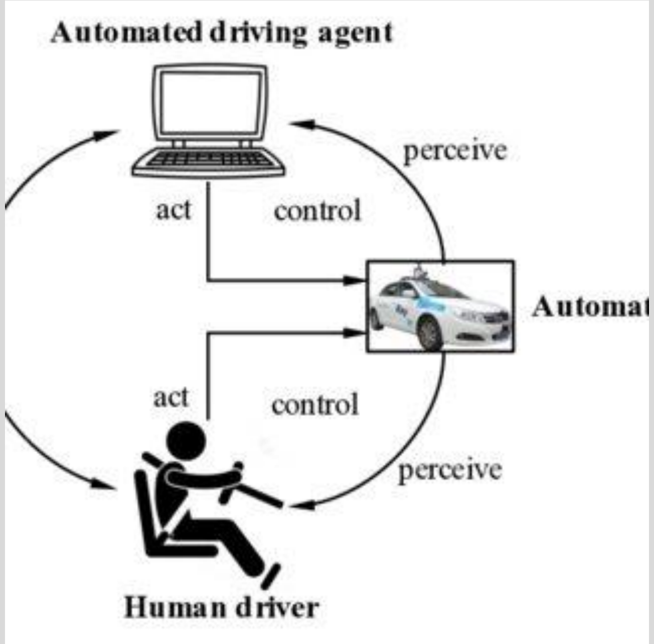
Department of AUTOMOTIVE ENGINEERING

# Table of Contents

# Introduction

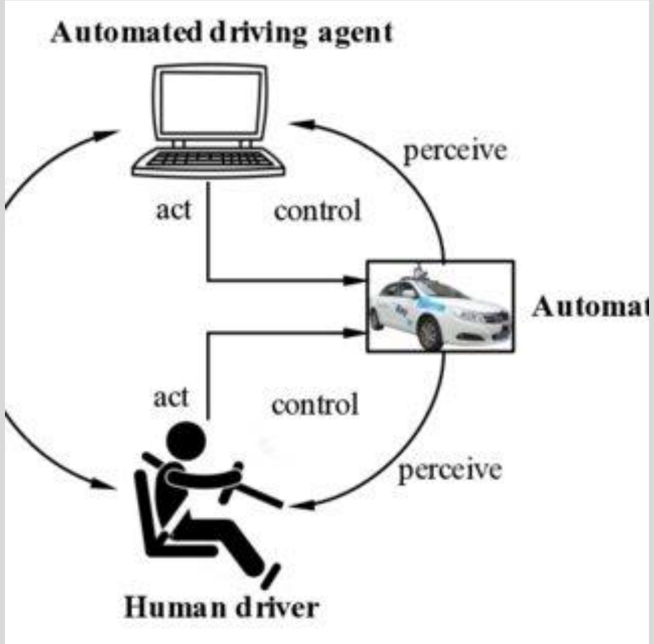
# Introduction

- **Autonomous driving today**: Modern autonomous driving (AD) systems excel in **frequent, well-covered scenarios** but often fail in rare or unusual "long-tail" cases such as erratic drivers, unusual merges, or partially occluded vehicles.
- **Data and generalization gap**: Classical end-to-end policies need **massive labeled datasets** and still struggle to robustly generalize to corner cases not seen during training.
- **Human drivers as inspiration**: In contrast, human drivers leverage **common sense, prior experiences, and causal reasoning** to quickly adapt to new situations without seeing millions of exact replicas.
- **Research question**: We ask whether **large language models**, with their strong reasoning and abstraction capabilities, can be integrated into a driving stack to make the system **behave more like an experienced human driver** rather than a brittle pattern recognizer.



# Motivation, Why Drive like a human?

- **Long-tail challenges**: Rare events—unusual merges, sudden cut-ins, stalled cars at odd locations—are exactly where current AD stacks are **least reliable**, yet they matter the most for safety.
- **Common sense vs pattern matching**: Traditional policies mostly rely on **pattern recognition**; they are strong when data is abundant but fragile when confronted with novel configurations of vehicles and rules.
- **Human-like reasoning**: Humans constantly **reason about intent** ("that car is probably going to cut in"), **anticipate future risks**, and **adapt** using high-level knowledge of physics and road rules.
- **LLM opportunity**: Large language models encode rich semantic and commonsense knowledge; our motivation is to see whether that knowledge can be turned into a **high-level decision policy** that complements numerical planners.
- **Ultimate goal**: Move from "drives like a deterministic rule engine" to **"drives like a cautious, experienced human"**, especially in ambiguous or under-specified situations.



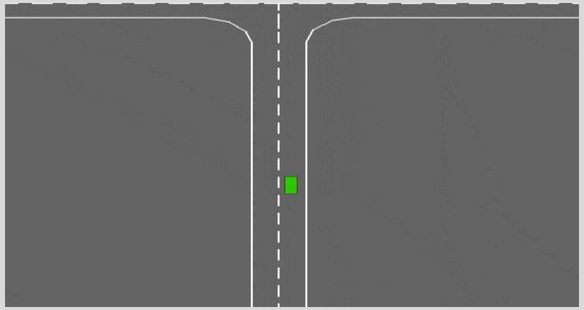
# Challenges

- **Poor generalization to corner cases:** Baseline policies often overfit to training distributions and make **unsafe or irrational choices** when scenes deviate from typical patterns.
- **Lack of interpretability:** Many AD policies operate as **black boxes**, making it difficult to understand *why* a particular action was taken or to debug failures.
- **No mechanism for learning from mistakes:** Conventional pipelines rarely have a **dedicated mechanism** that takes crashes or near-misses, explains them, and converts them into reusable lessons.
- **Safety without guarantees:** Prompting an LLM to “be safe” is not enough—there is **no numeric notion of risk**, headway, or time-to-collision, so obviously dangerous maneuvers may still be issued

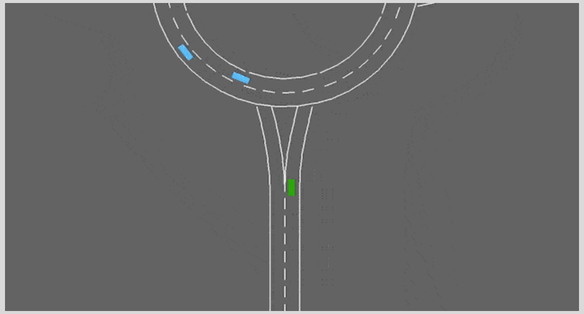
Department of AUTOMOTIVE ENGINEERING

# Problem Statement/ Goals(objectives)

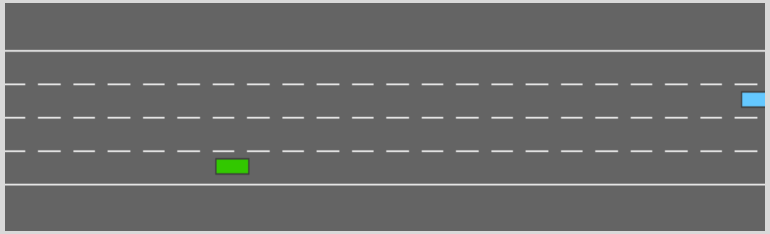
- **Limitations we target:**
  - Poor generalization to corner cases
  - Inability to explain decisions
  - No mechanism to incorporate past mistakes
  - Rigid rule-based planners
- **Our objective:** Build an **LLM-driven high level policy** that can:
  - Understand scenario semantics
  - Reason about intent, safety and traffic rules
  - Improve iteratively via a *failure memory bank*
  - Operate inside a real-time HighwayEnv loop across 4 in-built scenarios; **Highway, Merge, Roundabout, and Intersection**



**Video 1. Intersection Scenario** – Ego vehicle navigating a multi-lane intersection in HighwayEnv, interpreting right-of-way and cross traffic while choosing a safe path through the junction.



**Video 2. Roundabout Scenario** – Ego vehicle approaching and driving through a roundabout in HighwayEnv, selecting safe entry gaps, maintaining lane position, and exiting at the correct branch.



**Video 3. Highway Scenario** – Ego vehicle driving on a multi-lane highway in HighwayEnv, interacting with surrounding traffic while maintaining lane position and safe headway during lane-change

**Video 4. Merge Scenario** – Ego vehicle approaching an on-ramp merge in HighwayEnv, reasoning about gaps, adjusting speed, and selecting a safe lane to integrate smoothly into flowing traffic.

# Methods and Details:

# System Setup

- **Configuration:**

| Category | Specification |
|---|---|
| Simulator | **HighwayEnv** wrapped by the **DriveLikeAHuman** pipeline. |
| Environment/ Scenarios | highway-v0, merge-v0, roundabout-v0 and intersection-v1 |
| Episode | Each episode lasts up to **20 s.** If it doesn't stop with collision. |
| Observation space | HighwayEnv **Kinematics observation**: for ego + surrounding vehicles, the simulator provides positions, velocities, lane indices, and heading in each timestep. |
| Action space | HighwayEnv **DiscreteMetaAction** control: high-level maneuvers {LANE_LEFT, LANE_RIGHT, FASTER, SLOWER, IDLE}. |
| Collision reward | -1 |
| speed-based reward | Encouraging 20–30 m/s cruising speed while avoiding collisions. |
| API provider | OpenAI |
| Max tokens per response | **1024 tokens** maximum per call. |
| Request timeout | 60 s |

# Methods & Details: System Architecture Overview

- **Layered design**: Our system is organized into four main layers: **HighwayEnv (Environment)**, **Scenario Engine**, **LLMDriver (Cognitive)**, and **Safety Wrapper (Execution)**.
  - **Environment layer – HighwayEnv**: Provides the **kinematic state** (position, speed, lane index) of the ego vehicle and surrounding traffic at each timestep.
  - **Scenario Engine**: Converts raw numerical observations into a **structured JSON representation**, logs frames into **SQLite**, and maintains relationships such as "front vehicle," "left neighbor," and "right neighbor."
  - **LLMDriver**: Acts as the **reasoning core**, applying traffic rules, tool calls, memory retrieval, and multi-step predictions to output high-level actions.
  - **Safety Wrapper**: Filters the LLM's action through **numerical safety checks** (gaps, time-to-collision, lane-change viability) before sending commands back to HighwayEnv, creating a robust **perception → reasoning → action → logging** loop.

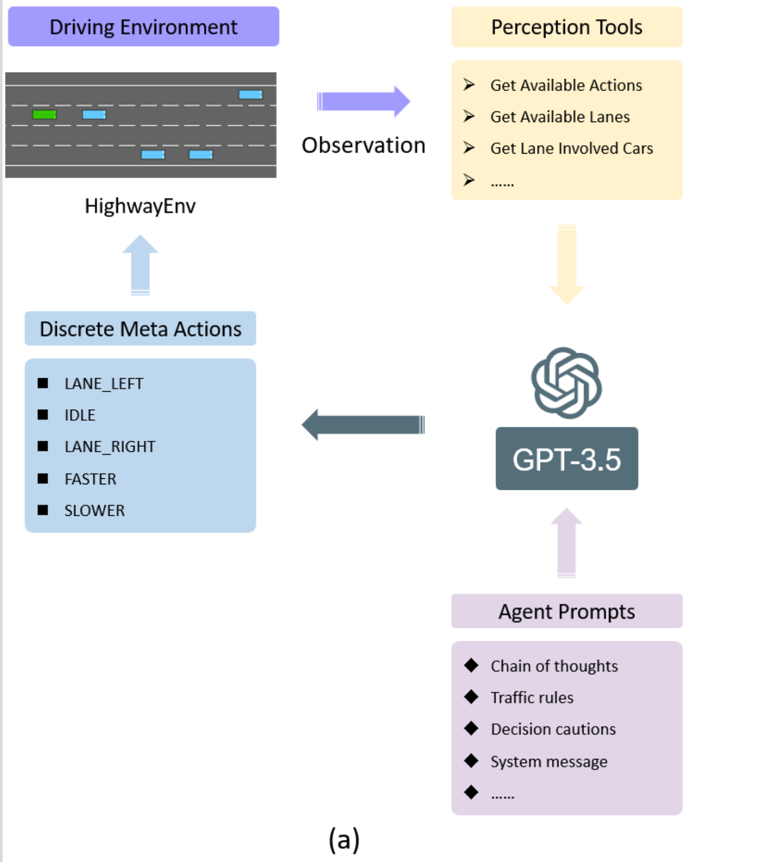



**Image 1.** End-to-end architecture of the LLM-driven highway agent. It shows how the layers are connected to each other in the system.

Department of
AUTOMOTIVE ENGINEERING

# Methods & Details: HELLM.py- Main Control Loop

- **Central orchestrator**: HELLM.py is the **top-level script** that coordinates all components of the driving stack for each simulation episode.
- **Initialization**: It configures **HighwayEnv**, initializes the chosen **scenario configuration**, and loads the **LLM model and associated tools** (safety checks, prediction tools, memory modules).
- **Per-timestep workflow**: On every timestep, it
  - extracts the latest observation from HighwayEnv,
  - updates the scenario JSON using the Scenario Engine,
  - calls LLMDriver.agentRun() to obtain a high-level action,
  - applies the safety wrapper to refine or veto that action, and
  - executes the final action in the environment.
- **Logging and termination**: Each step is **logged into SQLite**, and the loop terminates when a **collision** or **timeout** is detected, enabling detailed post-episode analysis.

```
# environment setting
config = {
    "observation": {
        "type": "Kinematics",
        "features": ["presence", "x", "y", "vx", "vy"],
        "absolute": True,
        "normalize": False,
        "vehicles_count": vehicleCount,
        "see_behind": True,
    },
    "action": {
        "type": "DiscreteMetaAction",
        "target_speeds": np.linspace(0, 32, 9),
    },
    "duration": 40,
    "vehicles_density": 2,
    "show_trajectories": True,
    "render_agent": True,
}
```

**Image 2.** After configing the LLM, Environment configuration for highway-v0, defining kinematic observations, discrete meta-action target speeds, and simulation settings such as duration, vehicle density, and visualization options.

Department of
AUTOMOTIVE ENGINEERING

# Methods & Details: Scenario Engine (scenario.py)

- **Lane graph construction**: Builds a **lane graph** tailored to each scenario (highway, merge, intersection, roundabout), capturing connectivity, allowed transitions, and lane adjacency.
- **Vehicle objects**: Creates **Vehicle objects** for both the **ego car** and **neighboring vehicles**, tracking their lane positions, velocities, and IDs.
- **JSON state representation**: Converts raw HighwayEnv observations into a **compact JSON tuple** containing lane index, longitudinal position, speed, neighbor relationships, and scenario-level metadata.
- **Database logging**: For each frame, the Scenario Engine saves the structured state into a **SQLite database**, providing a convenient backend for failure analysis and memory building.
- **Replay and visualization**: Supports **scenario replay** via scenarioReplay.py, allowing us to reconstruct and visualize episodes when studying failures and long-tail cases.


**Image 3.** Defining the Scenario class: it builds the 4-lane road graph and vehicle objects, initializes the SQLite database tables for vehicle states and LLM decisions, and starts the frame counter for logging the driving episode.

Department of AUTOMOTIVE ENGINEERING

# Methods & Details: LLMDriver (Reasoning + Tools)

- **Core responsibilities**: LLMDriver orchestrates **high-level reasoning** about the scene and decides the ego vehicle's next maneuver.
- **agentRun() pipeline**
    - Builds a **system prompt** that includes traffic rules, scenario descriptions, and decision-making cautions.
    - Injects the **current scenario JSON** with lane/topology information.
    - Performs **tool calls** (e.g., lane checks, collision checks, speed checks, multi-step prediction).
    - Integrates **memory retrieval** from the Failure Memory Bank (when enabled).
    - Outputs a final action in a strict format (e.g., Final Answer: <ACTION>).
- **exportThoughts()**: Extracts the LLM's **chain-of-thought** and saves it for later analysis, giving a window into *why* certain decisions were made.

AuE8200 Project Proposal

# Experiment and Results: Setup

Department of
AUTOMOTIVE ENGINEERING

# Setup Overview

**Perception**
- Kinematics (x, y, vx, vy, presence)
- Front Gap, Lane-Clearance, TTC

**Context Enhancement Layer**
- Failure Memory Bank

**LLM Decision Layer (SDG)**
- LLM outputs a Structured Decision Graph:
  - Hazard_assessment
  - Gap_change_prediction
  - Lane_feasibility
  - Collision_risk & TTC recommended action

**Action Execution**
- Final safe action → Environments
- Vehicle moves based on SDG + safety rules

**Metrics & Behavior Analysis**
- Ego Speed, Front Gap, Min TTC
- Lane Scores + Best Lane
- Raw vs Safe Actions, Collision Risk

Department of AUTOMOTIVE ENGINEERING

# Extension 1: Failure Memory Bank + Case-based reasoning

- **Purpose**: Enable the agent to **accumulate driving experience** over time by storing structured memories of **failures** (crashes or near-misses) and reusing them in future episodes.
- **On failure**: When a crash occurs, we log
    - scenario type and difficulty,
    - ego and nearby vehicle states,
    - the LLM's thought process leading up to the failure, and
    - reward and outcome statistics.
- **LLM reflection**: After the crash, we prompt the LLM to **diagnose its own mistake**—identify root causes and summarize a **lesson** that should be remembered.
- **Case abstraction**: Each failure is stored as a **case** with a compact description and a textual "lesson," forming the core of the **Failure Memory Bank.**

# Extension: Current Memory Implemented (Baseline Imitation Memory)

- **Baseline memory mechanism:** The current repository's "memory" is implemented as a **simple lookup table** mapping an observation to a **human driver action.**
- **Data source:** For states encountered during training, the human driver's ground-truth actions are **stored as reference entries.**
- **Decision improvement:** At runtime, when the agent sees a state similar to one in the memory, it **imitates the human action**, often correcting poor LLM decisions in those repeated states.
- **Limitations:** This approach relies heavily on **human-provided corrections** and does not generalize well beyond **exactly encountered states**, motivating our case-based Failure Memory Bank.

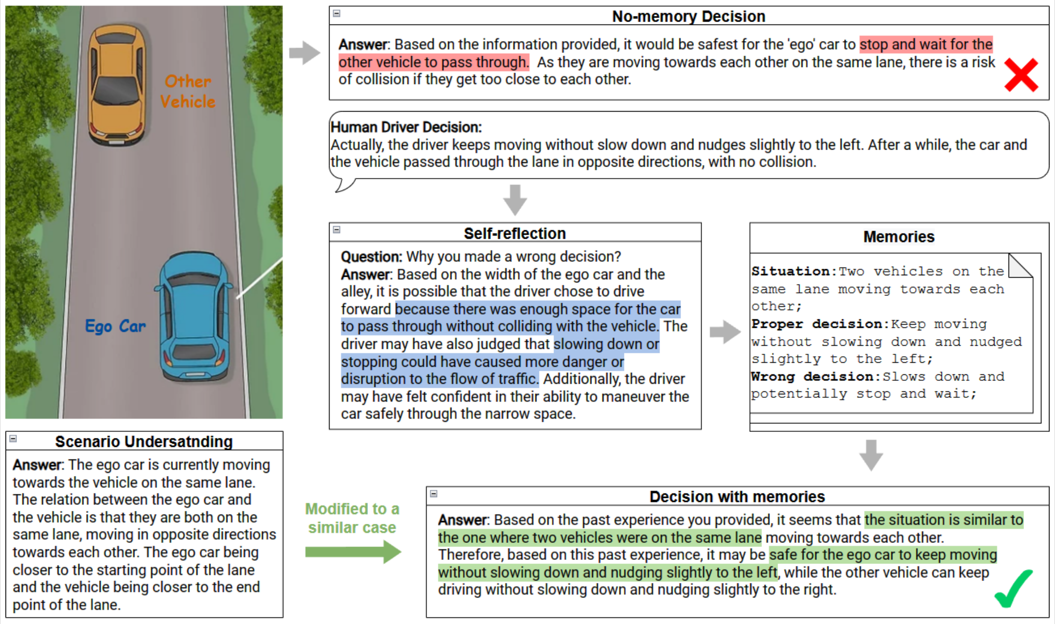



**Image 4.** Illustration of the baseline memory mechanism that compares the LLM's proposed action ('no-memory decision') with the human driver's ground-truth behavior. For each observed traffic scene, the human action is stored as a memory entry. When a similar situation recurs, the system retrieves the stored action and adjusts the LLM's output, yielding an 'improved decision with memory' that better matches human driving behavior."

Department of
AUTOMOTIVE ENGINEERING

# Extension : Failure Memory Bank: Logging Layer

- **Lightweight logging**: Each episode writes a collection of **key fields** into a simple SQLite/JSON store, allowing us to systematically catalogue failures.
- **Stored attributes**: We record **scenario ID**, observation summaries, actions taken, rewards, whether a crash occurred, and additional episode context.
- **Failure-centric design**: The logging layer is optimized to make **failed or risky episodes** easy to query so that we can build and refine memory entries from them.
- **Scalability**: Because entries are stored as **compact records**, the Failure Memory Bank can scale to many episodes without becoming unwieldy.

```
        reflection_prompt = f"""We executed a highway-driving episode in scenario_type={scenario_type},
difficulty={difficulty}, and the episode ended with a crash.

Crash reason from the environment or evaluator:
{crash_reason}

Here is a brief log of the last steps before the crash:
{episode_summary}

You are an expert driving instructor for an autonomous agent.
In 1▯2 sentences, explain the root cause of this failure.
Then, in another 1▯2 sentences, state a concise LESSON the agent should
remember to avoid this in the future.

Respond in the following JSON format:
{{"root_cause": "...", "lesson": "..."}}
"""

        messages = [
            {"role": "system", "content": "You analyse failures and extract lessons."},
            {"role": "user", "content": reflection_prompt},
        ]
```

**Image 5.** Prompting template that asks the LLM to analyze its own mistake. After a crash, we feed a compact log of the last steps to the LLM and ask it to identify the root cause and articulate a concise lesson that should be remembered for similar future situations

```
    def __init__(self, path: str = "results-db/memory_bank.jsonl"):
        self.path = path
        os.makedirs(os.path.dirname(self.path), exist_ok=True)
        self._memories: List[EpisodeMemory] = []
        self._load()

    def _load(self) -> None:
        if not os.path.exists(self.path):
            return
        with open(self.path, "r", encoding="utf-8") as f:
            for line in f:
                line = line.strip()
                if not line:
                    continue
                data = json.loads(line)
                self._memories.append(EpisodeMemory(**data))

    def _save_append(self, memory: EpisodeMemory) -> None:
        with open(self.path, "a", encoding="utf-8") as f:
            f.write(json.dumps(asdict(memory)) + "\n")

    def add(self, memory: EpisodeMemory) -> None:
        self._memories.append(memory)
        self._save_append(memory)
```

**Image 6.** Lightweight logging layer for the Failure Memory Bank. Each episode writes key fields, scenario ID, observation summary, action, reward, crash flag, and additional context, into a simple SQLite / JSON store, making failures easy to query and analyze later.

# Extension: Using the memory (case retrieval)

- **Structured memory entries**: After our extensions, every failure is transformed into a **rich memory record** containing scenario type, difficulty, outcome statistics, and a textual lesson.
- **Similarity search**: Before choosing an action, the agent queries the Failure Memory Bank for **similar past episodes** based on scenario metadata and observation summaries.
- **Prompt augmentation**: Retrieved cases are injected into the LLM prompt as **“previous similar failures and their lessons”**, biasing the agent toward safer, more cautious actions in analogous situations.
- **From one-off mistakes to experience**: This effectively turns repeated exposure to long-tail scenarios into **incremental experience accumulation**, moving toward an agent that “drives like a human and learns from its own mistakes.”

```
=== Memory 299 ===
 Episode ID   : cones_high_density-1-seed1
 Scenario     : cones_high_density
 Difficulty   : 1
 Steps        : 1
 Total reward : 0.033
 Time         : 2025-11-29 13:14:10.627104
 Failure reason:
   The root cause of the failure was the ego vehicle's inability to react quickly enough to the high-densit
y cones scenario, resulting in a collision or offroad incident.
 Lesson:
   The agent should remember to anticipate and react promptly to changes in the environment, especially in
high-density scenarios, to avoid collisions or offroad incidents in the future.

=== Memory 300 ===
 Episode ID   : cones_high_density-1-seed2
 Scenario     : cones_high_density
 Difficulty   : 1
 Steps        : 4
 Total reward : 2.294
 Time         : 2025-11-29 13:14:13.989392
 Failure reason:
   The ego vehicle was driving too slowly, which may have caused other vehicles to approach too closely and
 resulted in a collision or offroad incident.
 Lesson:
   The agent should maintain a safe and appropriate speed for the given scenario to avoid potential collisi
ons or offroad incidents.
```

**Image 7.** Example of a stored failure case. The record includes the episode ID, descriptive scenario label, performance metrics, a textual summary of the final states and actions, and LLM-generated ‘root cause’ and ‘lesson’ fields. This case can be retrieved later to remind the agent of what went wrong and how to avoid repeating the same mistake.

# Experiment and Results: Evaluation Metrics

Department of AUTOMOTIVE ENGINEERING

# Metrics Details

- Extension 1,3,4 – Failure Memory Bank + Case-Based Reasoning
  - **Collision Rate:** Share of episodes that end in a crash before/after enabling the Failure Memory Bank. Lower = fewer repeated mistakes.
  - **Average Episodic Reward:** Mean HighwayEnv reward per episode (speed, lane-keeping, penalties, collisions). Higher = safer yet efficient driving.
  - **Average Speed:** Mean ego-vehicle speed per episode. Checks that safety gains do not come from driving unrealistically slowly.

# Experiment and Results: Quantitative & Qualitative

AuE8200 Project Proposal

Department of AUTOMOTIVE ENGINEERING

# Quantitative Results – Extension 1,3,4

- We evaluate all driver variants on a **mixture of long-tail highway scenarios**, each designed to stress the decision-making of the LLM:
  - **High-speed cut-in** – a fast vehicle suddenly merges into the ego lane with a small gap.
  - **Partial occlusion** – the ego's view of a nearby vehicle is blocked by a larger car until late.
  - **Erratic braking** – the lead vehicle brakes unpredictably, forcing rapid reassessment of safe headway.
  - **Multi-vehicle merge** – several vehicles compete to join the same lane, creating complex interactions.
  - **Truck-debris scenario** – a slow truck and static obstacle/debris force the ego vehicle to make a timely lane change.
- Reward and metrics

We use the **HighwayEnv default reward function**, which jointly encourages **high average speed, staying on the road, lane-keeping discipline, and avoiding collisions.**

For each driver kind we report two aggregate metrics over many episodes:

  - **Crash rate** – fraction of episodes in which the ego vehicle collides with another vehicle or obstacle (lower is better).
  - **Total reward** – mean cumulative reward per episode under the default HighwayEnv reward (higher is better).

| Driver kind | Crash rate(100 episodes) | Total reward | Average speed |
|---|---|---|---|
| Base LLM | 65.7% | 10.327 | 28.8 |
| LLM with memory | 20.4% | 26.936 | 20.4 |

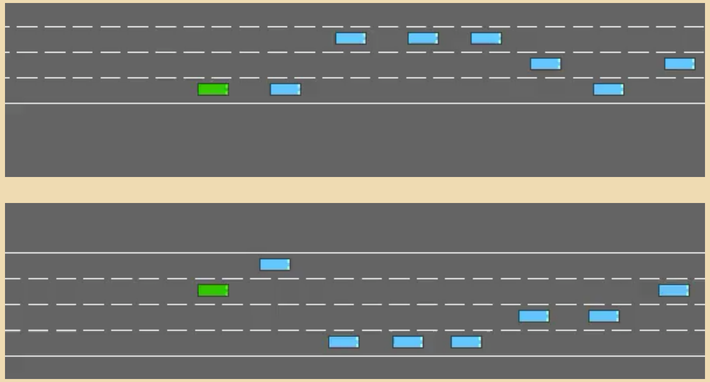
**Table 1.** Quantitative comparison of different driver configurations on a mixture of long-tail highway scenarios (high-speed cut-ins, partial occlusions, erratic braking, multi-vehicle merges, and truck-debris situations).

Department of
AUTOMOTIVE ENGINEERING

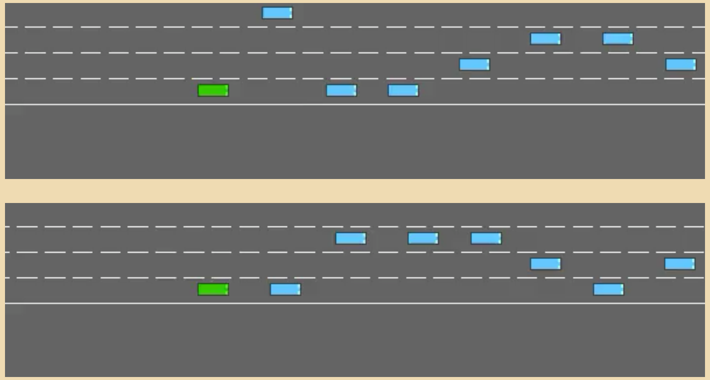
# Qualitative Results – Extension

- Here are some example of failure and success in each driver kind:

- Base LLM



- LLM with memory



# Discussion

Department of AUTOMOTIVE ENGINEERING

# Discussion and Next steps

1. Key Findings & conclusion

• Combining LLM reasoning with our four extensions – Failure Memory Bank helped to reduce the collision rate in long tail scenarios.

2. Limitations and drawbacks

• Simulator limits: HighwayEnv offers clean, fully observable scenes and limited control over truly adversarial long-tail setups, which caps how hard we can stress-test the stack and may overestimate real-world robustness.

• Prediction & memory brittleness: When rollouts are ambiguous, the LLM can overreact (e.g., unnecessary hard braking), and memory retrieval is still coarse, relying on simple similarity rather than rich semantic search.

# Future works

- More challenging scenarios:
  - Emergency vehicles (ambulance approaching, all lanes stopped)
  - Stalled cars and work zones would better probe the agent's ability to handle rare but safety-critical events.
  - Pedestrians gesturing at 4-way stops
  - Multi-agent negotiation (merging, cut-ins, aggressive drivers)
  - Limited Sensor / Partial Observability Parking with blind spots
- Adding more sensors
- Integrate with platforms like CARLA, MetaDrive, DriveArena, LimSim or DiLu-style frameworks.

Sample Approach

- Choose one vehicle as the "ambulance".
- Add new scenario with semantics, is_emergency = True.
- Spawn or position the ambulance behind the ego.
- Script background traffic to stop (all lanes) or slow down.
- Keep the ambulance moving faster than traffic.
- Inform RL agent about the ambulance.

Department of
AUTOMOTIVE ENGINEERING

# References

1. 1. Hu, Y., Yang, J., Chen, L., Li, K., Sima, C., Zhu, X., ... & Li, H. (2023). Planning-oriented autonomous driving. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 17853-17862).
2. 2. Chen, L., Wu, P., Chitta, K., Jaeger, B., Geiger, A., & Li, H. (2024). End-to-end autonomous driving: Challenges and frontiers. IEEE Transactions on Pattern Analysis and Machine Intelligence.
3. 3. Deng, H., Zhao, Y., Wang, Q., & Nguyen, A. T. (2023). Deep reinforcement learning based decision-making strategy of autonomous vehicle in highway uncertain driving environments. Automotive Innovation, 6(3), 438-452.
4. 4. Yang, Z., Jia, X., Li, H., & Yan, J. (2023). Llm4drive: A survey of large language models for autonomous driving. arXiv preprint arXiv:2311.01043.
5. 5. Fu, D., Li, X., Wen, L., Dou, M., Cai, P., Shi, B., & Qiao, Y. (2024, January). Drive like a human: Rethinking autonomous driving with large language models. In 2024 IEEE/CVF Winter Conference on Applications of Computer VisionWorkshops (WACVW) (pp. 910-919). IEEE
6. Fu, D., Li, X., Wen, L., Dou, M., Cai, P., Shi, B., & Qiao, Y. (2024, January). Drive like ahuman: Rethinking autonomous driving with large language models. In 2024IEEE/CVF Winter Conference on Applications of Computer Vision Workshops(WACVW) (pp. 910-919). IEEE
7. Wang, Y., Jiao, R., Lang, C., Huang, C., Wang, Z., Yang, Z., & Zhu, Q. (2023).Empowering autonomous driving with large language models: A safety perspective.arXiv. 2023. arXiv preprint arXiv:2312.00812.

# Thank you for your attention!

# Questions?