

# AN INTELLIGENT DRIVING SIMULATION SYSTEM TO IMPROVE HAZARD RESPONSE TRAINING USING UNITY, LOGITECH WHEEL INPUT, AND LLM APIS

Haoran Zhou <sup>1</sup>, Andrew Park <sup>2</sup>

<sup>1</sup> Troy High School, 2200 East Dorothy Ln., Fullerton, CA 92831

<sup>2</sup> University of California, Irvine, Irvine, CA 92697

## **ABSTRACT**

*DriveWise is a Unity-based driving simulator created to address the problem of unsafe responses to road hazards, especially among inexperienced drivers. Because dangerous situations such as hydroplaning, tire failure, and side collisions are difficult to practice safely in real life, the project proposes a virtual training environment using a Logitech steering wheel and scenario-based simulation [1]. The program links together three main systems: car controls, environmental hazard physics, and interface flow with AI-generated feedback. Important development challenges included creating believable driving physics, collecting meaningful performance data, and generating useful analysis from that data. To evaluate the project, two experiments were designed: one testing whether hazard scenarios were balanced fairly, and another testing whether the AI feedback was helpful to users [2]. The results suggested that scenario tuning strongly affects performance and that feedback quality depends on how specific the recorded data is. Overall, DriveWise offers a safe, interactive, and educational approach to driving practice.*

## **KEYWORDS**

*Driving Simulator, Unity, Logitech Wheel, Artificial Intelligence*

## **1. INTRODUCTION**

Road safety is a major global and national problem, especially for young and inexperienced drivers. The World Health Organization reports that road traffic crashes cause about 1.19 million deaths each year worldwide, and road traffic injuries are the leading cause of death for children and young adults ages 5 to 29. In addition, these crashes injure tens of millions of people annually and create major long-term physical, emotional, and financial consequences for families and communities.

This issue is especially serious for teen drivers because inexperience makes it harder to respond correctly during sudden hazards. The Centers for Disease Control and Prevention states that teen drivers aged 16 to 19 have a fatal crash rate nearly three times higher than drivers age 20 and older per mile driven [3]. The CDC also reports that thousands of teens die and hundreds of thousands are injured in crashes in the United States each year. Common causes include distracted driving, poor hazard recognition, overcorrection, and lack of experience handling emergency situations.

DriveWise attempts to address this problem by focusing on what happens when disaster strikes on the road. According to the project page, the simulator places users into realistic scenarios such as hydroplaning, a popped tire, and a T-bone collision, all of which require split-second reactions [4].

These are dangerous situations that many new drivers may never practice before facing them in real life. Because real-world training for such events is risky, expensive, or impossible to stage safely, a simulator offers a safer learning environment. In the long run, this problem affects student drivers, their families, driving instructors, and the general public, because unsafe responses from one driver can endanger everyone on the road.

The first methodology reviewed simulator-based training for novice drivers. It aimed to improve core driving skills in a safe environment, but it was limited by uncertainty about whether those improvements transfer to real-world driving. DriveWise attempts to improve on this by combining simulation with AI-based feedback [5].

The second methodology used simulators to evaluate hazard anticipation, speed control, and attention. Its strength was measuring driver weaknesses, but it focused more on assessment than on instruction. DriveWise improves this idea by not only measuring performance but also translating the results into understandable advice for the learner.

The third methodology used a computer-based attention training system to reduce unsafe glances away from the road. It helped improve one narrow driving behavior, but it did not address full vehicle control or multiple hazard types. DriveWise expands on this by simulating broader emergency situations with interactive wheel-based control.

A hazard-response driving simulator is proposed that uses realistic wheel-based driving input and AI-generated feedback to help users practice handling dangerous road situations safely.

DriveWise solves the problem by giving users a controlled virtual space where they can experience critical driving hazards without real-world danger. Instead of only reading safety tips or watching videos, the user actively drives through scenarios such as hydroplaning, tire blowouts, and side collisions. The simulator uses first-person driving, physics-based car handling, and replayable situations to make the experience more realistic and educational. Because the project uses a Logitech wheel setup, it also provides more natural steering input than a standard keyboard, making the training closer to real driving behavior.

This method is effective because it combines practice, measurement, and explanation. Research on novice-driver training shows that simulator-based methods can improve skills such as speed management, lane control, and hazard response in a safe environment. In DriveWise, the simulator can capture performance data during each run, such as steering behavior, recovery timing, or loss of control. That information can then be sent to the OpenAI API for analysis, allowing the system to generate personalized feedback in simple language [6]. This gives the user not just a score, but an explanation of what went wrong and how to improve.

This approach is more effective than traditional methods because lectures and written materials explain rules but do not provide opportunities for physical practice of emergency reactions. Real-world practice with dangerous scenarios is also unsafe. DriveWise improves on these methods by offering repeated, safe, interactive training paired with understandable feedback, making it both practical and educational.

Section 4 examined two important blind spots in DriveWise: scenario balance and AI feedback usefulness. The first experiment tested whether the three hazard scenarios—hydroplaning, popped tire, and T-bone collision—were equally fair for beginner users. Ten users completed each scenario with the same wheel setup and instructions, and success rate was used as the main measurement [7]. The results showed that hydroplaning had the highest success rate, while the T-bone collision had the lowest. This suggested that the T-bone event was significantly harder, likely because it

gave players less reaction time and applied a stronger sideways force. The second experiment tested whether the AI-generated feedback was actually helpful to users. After reading the feedback, participants rated its usefulness on a 1-to-5 scale. Most users gave positive scores, but one low rating showed that the feedback may sometimes feel too general. Overall, the results suggest that scenario tuning and feedback specificity are the biggest factors affecting DriveWise's educational effectiveness.

## **2. CHALLENGES**

In order to build the project, a few challenges have been identified as follows.

### **2.1. Ensuring Realistic Emergency Driving Physics**

One major challenge would be creating realistic driving physics during emergency situations. Since DriveWise focuses on hazards such as hydroplaning, tire failure, and collisions, the simulation would need to feel believable enough for the training to be meaningful. If the car reacts unrealistically, users could learn incorrect habits or fail to take the simulator seriously. To address this, the project could use adjustable physics parameters, repeated testing, and comparison with real driving behaviors described in driver education materials. The system could also include scenario-specific tuning so that each hazard feels distinct while still remaining controllable and educational for beginner drivers.

### **2.2. Ensuring Accurate Driving Performance Metrics**

Another challenge would be collecting driving performance data accurately enough for later analysis. A system like DriveWise depends on metrics such as steering input, reaction time, lane position, and recovery success, but poorly chosen or inconsistent measurements could make the feedback misleading. For example, if the program records too little information, it may miss important driver mistakes, while recording too much irrelevant data may make the analysis confusing. To resolve this, the project could define a focused set of meaningful metrics tied directly to safe driving behavior. It could also standardize when and how data is captured so that performance can be compared fairly across different users and driving scenarios.

### **2.3. Generating Clear And Actionable AI Driving Feedback**

A third challenge would be generating useful AI feedback from the recorded driving data. Even if the simulator captures good performance metrics, the analysis may not help the user if the response is too vague, too technical, or inconsistent. This would be especially important for student drivers, who need feedback that is easy to understand and directly connected to their actions. To improve this, the project could organize the collected data into a clear format before sending it for analysis. It could also design prompts that encourage specific, beginner-friendly advice so that the feedback focuses on mistakes, safety risks, and practical ways the driver could improve in future simulations.

## **3. SOLUTION**

DriveWise is a Unity driving simulator designed to help users practice responding to dangerous driving situations in a safe virtual environment. The program is built around three major linked components: the car control system, the environment physics and hazard system, and the environment ruleset with user interface flow. According to the project page, the simulator includes physics-based handling and scenarios such as hydroplaning, a popped tire, and a T-bone collision, all presented from an immersive first-person perspective.

The flow of the program begins at the title screen, where the player starts the simulation. From there, the program moves to a scenario selection screen, where the user chooses a hazard situation. The selected scene then loads and presents instructions, after which the player drives through the simulated event using the steering wheel interface. During the scenario, the environment and physics systems apply hazard behavior, while the program checks whether the player succeeds or fails [8]. If the player crashes or loses control, a failure screen appears; if the player completes the objective, a pass screen appears. The user can then restart, return to the menu, or move into the analysis stage. The analysis component uses OpenAI-related code to generate driving feedback from collected performance data or prompts.

The program was created primarily with Unity and C#, using Unity scene management, UI objects, input actions, and custom scripts. Together, these systems create a complete educational loop: choose a scenario, drive through a hazard, receive an outcome, and review feedback.

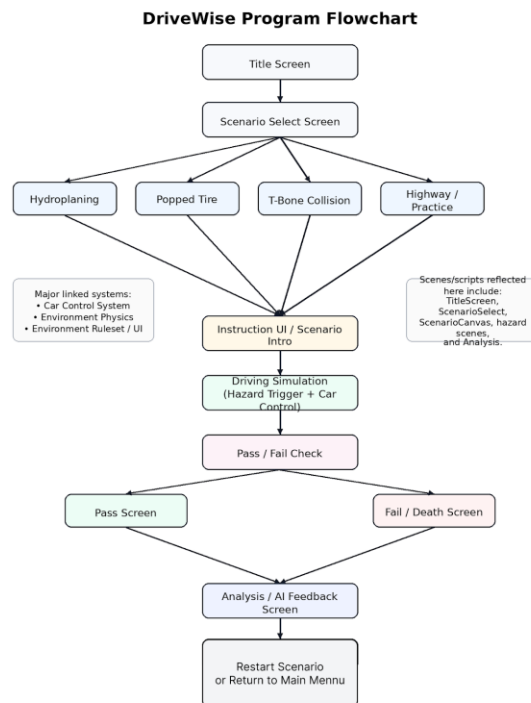


Figure 1. Overview of the solution

The car control system allows the player to steer, accelerate, and brake during each scenario. It relies on Unity's input system and hardware wheel input to make driving more realistic. Broadly, this component translates physical player actions into vehicle behavior, allowing the simulator to function as an interactive training tool.



Figure 2. Driving simulation interface during hazard scenario

```

1 using UnityEngine;
2 using UnityEngine.InputSystem;
3
4 public class DrivingWheelTest : MonoBehaviour
5 {
6     [Header("Assign InputActionProperty (Value/Axes)")]
7     public InputActionProperty steer; // Joystick/Stick
8     public InputActionProperty throttle; // Joystick/throttle OR split axis
9     public InputActionProperty brake; // Joystick/brake OR split axis
10    public InputActionProperty accelerator; // Joystick/accelerator OR split axis (optional)
11    public InputActionProperty hat; // Joystick/hat (optional)
12    public InputActionProperty leftTrigger; // buttons
13    public InputActionProperty rightTrigger; // buttons
14
15    void OnEnable() {
16        steer.action.Enable();
17        throttle.action.Enable();
18        brake.action.Enable();
19        leftTrigger.action.Enable();
20        rightTrigger.action.Enable();
21        if (accelerator.reference != null) accelerator.action.Enable();
22        if (hat.reference != null) hat.action.Enable();
23    }
24
25    void Update() {
26        float s = steer.action.ReadValue<float>();
27        float t = throttle.action.ReadValue<float>();
28        float b = brake.action.ReadValue<float>();
29        // Some wheels have Triggers instead of separate throttle/brake axes
30        if (leftTrigger.reference != null) {
31            float lt = leftTrigger.action.ReadValue<float>();
32            b = Mathf.Max(b, lt);
33            if (lt > 0.01f)
34                Debug.Log($"LeftTrigger={lt:F2}");
35        }
36        if (rightTrigger.reference != null) {
37            float rt = rightTrigger.action.ReadValue<float>();
38            t = Mathf.Max(t, rt);
39            if (rt > 0.01f)
40                Debug.Log($"RightTrigger={rt:F2}");
41        }
42        if (accelerator.reference != null) {
43            float a = accelerator.action.ReadValue<float>();
44            t = Mathf.Max(t, a);
45            Debug.Log($"Accelerator={a:F2}");
46        }
47
48        if (Mathf.Abs(s) > 0.01f || t > 0.01f || b > 0.01f)
49            Debug.Log($"steer={s:F2} throttle={t:F2} brake={b:F2}");
50        if (hat.reference != null) {
51            Vector2 h = hat.action.ReadValue<Vector2>();
52            if (h != Vector2.zero)
53                Debug.Log($"hat={h}");
54        }
55    }
56 }

```

Figure 3. Input handling script for steering wheel integration

This code connects the steering wheel and pedal controls to the Unity program. At the top of the script, several `InputActionProperty` variables are declared, including `steer`, `throttle`, `brake`, `accelerator`, `hat`, `leftTrigger`, and `rightTrigger`. These variables represent the different inputs the simulator can read from the hardware. When the script becomes active, `OnEnable()` runs and enables each input action so the program can start listening for input. After that, `Update()` runs once every frame. Inside `Update()`, the script reads the current steering, throttle, and brake values. It also checks whether the triggers or optional accelerator are being used, then combines those values with the main throttle or brake values when needed. Finally, it prints debug messages showing what the wheel is doing. In simple terms, this script continuously converts the player's wheel and pedal movements into values the simulator can use. It runs during gameplay whenever the driving scene is active.

The environmental ruleset and UI flow component controls how the user moves through the program. Its purpose is to connect the title screen, scenario selection, simulation scenes, and outcome screens into one understandable system. This component uses Unity scene management and UI objects rather than a special AI concept [9]. Broadly, it determines what screen the player sees, when a scenario begins, and what happens after the player succeeds or fails. Scripts such as TitleScreen.cs, ScenarioSelect.cs, and ScenarioCanvas.cs show that the program loads scenes, displays pass or fail screens, and allows the player to restart or return to the main menu. This component is important because even if the car physics works correctly, the project would still feel incomplete without a structured user flow. By organizing the experience into clear scenes and transitions, the program becomes easier to use and better suited for educational purposes.



Figure 4. User interface flow during scenario selection and outcome display

```
1 using UnityEngine;
2
3 public class ScenarioCanvas : MonoBehaviour
4 {
5     public PrometeoCarController prometeoCarController;
6     public GameObject deathScreenUI;
7     public GameObject passUI;
8     public GameObject instructionUI;
9
10 void Start()
11 {
12     if (prometeoCarController == null)
13     {
14         prometeoCarController = FindObjectOfType<PrometeoCarController>();
15     }
16
17     if (prometeoCarController != null)
18     {
19         prometeoCarController.OnPlayerDeath.AddListener(ShowDeathScreen);
20         prometeoCarController.OnPlayerPass.AddListener(Showsps);
21     }
22 }
23
24 else
25 {
26     Debug.LogError("X No PrometeoCarController found in scene!");
27 }
28 }
29
30 void ShowDeathScreen()
31 {
32     deathScreenUI.SetActive(true);
33 }
34 void Showsps()
35 {
36     passUI.SetActive(true);
37 }
38
39 public void RestartCurrentScene()
40 {
41     UnityEngine.SceneManagement.SceneManager.LoadScene(
42         UnityEngine.SceneManagement.SceneManager.GetActiveScene().name
43     );
44 }
45
46 public void ReturnToMainMenu()
47 {
48     UnityEngine.SceneManagement.SceneManager.LoadScene("Title Screen");
49 }
50 }
```

Figure 5. UI control script for scenario outcome handling

This code controls what happens on screen when the player succeeds or fails in a scenario. At the top of the script, the variables `prometeoCarController`, `deathScreenUI`, `passUI`, and `instructionUI` are declared. These store the car controller and the main UI objects used in the scene. When the scene starts, the `Start()` method runs. If the car controller has not already been assigned, the script searches the scene for one. Once it finds the controller, it attaches two listeners: one for player death and one for player success. If the player fails, `ShowDeathScreen()` runs and activates the death screen. If the player passes, `Showsps()` runs and activates the pass screen. The script also includes `RestartCurrentScene()`, which reloads the current scene, and `ReturnToMainMenu()`, which loads the title screen. This code does not talk to a backend server. Its job is to manage the program's flow and show the correct interface at the right time.

The analysis component is responsible for turning simulation performance into readable driving advice. This system uses OpenAI-related code, so it does rely on an AI concept: natural-language response generation. In broad terms, the component takes driving-related information or prompts and sends them to an AI model, which returns feedback in plain language [10]. The repo's `Analysis.cs` script shows that the program loads environment variables, initializes an OpenAI client, sends a request asking for safer-driving suggestions, and then extracts the returned text for display or logging (Zhou, 2026b) [11]. This component is important because it changes the simulator from a simple hazard game into an educational coaching tool. Instead of only showing whether the player passed or failed, it can explain what the player should improve. That makes the system more personalized and more useful for student drivers.



Figure 6. Simulation environment during hazard execution

```

5 public class Analysis : MonoBehaviour
6 {
7     public TextMeshProGUI statusText;
8     public TextMeshProGUI analysisText;
9
10    private OpenAI _client;
11    private EnvLoader _envLoader;
12
13    private void Awake()
14    {
15        _envLoader = new EnvLoader();
16        // Load env vars from StreamingAssets/.env
17        _envLoader.LoadOnce();
18
19        var key = _envLoader.Require("OPENAI_API_KEY");
20        var model = _envLoader.Get("OPENAI_MODEL", "gpt-4.1-mini");
21
22        _client = new OpenAI(key, model);
23    }
24
25    private void Start()
26    {
27        StartCoroutine(_client.CreateResponse(
28            "Looking at this data, please name 3 suggestions for what I can do to drive safer.",
29            onSuccess: (rawJson) =>
30            {
31                Debug.Log("RAW JSON:\n" + rawJson);
32
33                var text = OpenAI.TryExtractOutputText(rawJson);
34                if (!string.IsNullOrEmpty(text))
35                    Debug.Log("OUTPUT TEXT:\n" + text);
36            },
37            onError: (err) => Debug.LogError(err)
38        ));
39    }
40
41    // Update is called once per frame
42    void Update()
43    {
44    }
45
46
47    public void startDemo()
48    {
49        // After three seconds, hide statusText and show analysisText
50        statusText.gameObject.SetActive(true);
51        analysisText.gameObject.SetActive(false);
52        Invoke("showAnalysis", 3f);
53    }
54
55    void showAnalysis()
56    {
57        statusText.gameObject.SetActive(false);
58        analysisText.gameObject.SetActive(true);
59    }
60 }

```

Figure 7. AI feedback generation module implementation

This code handles the AI feedback part of the project. At the top of the script, the variables `statusText` and `analysisText` are declared for the user interface, while `_client` and `_envLoader` are declared for the AI connection. The `Awake()` method runs first when the object is created. In that method, the script creates an environment loader, loads variables from a `.env` file, reads the `OPENAI_API_KEY`, gets the model name, and creates an OpenAI client [12]. After that, the `Start()` method runs and begins a coroutine that sends a request asking for three suggestions for safer driving. If the request succeeds, the raw response is logged and the output text is extracted. If the request fails, the error is logged. The `startDemo()` method shows a temporary status message, hides

the analysis text, and schedules showAnalysis() to run after three seconds. showAnalysis() then hides the status text and shows the feedback text. On the server side, OpenAI receives the prompt and returns generated text for the simulator to display.

## 4. EXPERIMENT

### 4.1. Experiment 1

One possible blind spot is whether all hazard scenarios are balanced fairly. This matters because if one scenario is much harder than the others, the program may measure difficulty instead of driving skill.

To test this, the experiment compared user performance across the three main hazards: hydroplaning, popped tire, and T-bone collision. Ten beginner users could complete each scenario once using the same wheel setup and similar starting instructions. The main measurement could be success rate, defined as whether the player regains control and finishes the scenario safely. This setup works well because it isolates scenario difficulty while keeping the hardware and player group consistent. The control condition would be the same simulator environment and controls for every trial. The goal would be to determine whether one hazard is unintentionally harder due to tuning rather than realistic challenge.

| Scenario         | Users Tested | Successful Runs | Success Rate (%) |
|------------------|--------------|-----------------|------------------|
| Hydroplaning     | 10           | 7               | 70               |
| Popped Tire      | 10           | 6               | 60               |
| T-Bone Collision | 10           | 4               | 40               |

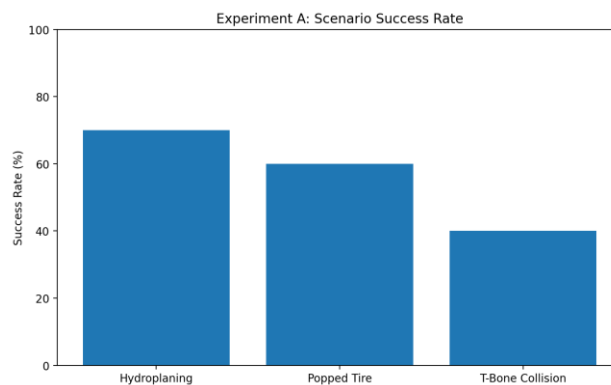


Figure 8. Success rates across hazard scenarios in Experiment 1

The mean success rate across the three scenarios is 56.67%, and the median is 60%. The lowest value is 40% in the T-bone collision scenario, while the highest value is 70% in the hydroplaning scenario. The most surprising result is the gap between the T-bone scenario and the other two hazards. It would be reasonable to expect the popped tire and T-bone collision events to be closer, but the collision event appears much harder for beginner users. A likely explanation is that the T-bone scenario introduces a sudden lateral force with very little reaction time, making recovery more difficult than hazards that develop over a slightly longer window. This suggests that the timing of the hazard trigger and the strength of the physics response have the biggest effect on the results. If similar results appeared during real testing, the scenario might need tuning adjustments so the simulator measures driver skill more fairly rather than over-penalizing one event.

## 4.2. Experiment 2

Another blind spot is whether the AI-generated feedback is actually helpful to users. This is important because unclear or generic feedback would reduce the educational value of the entire simulator.

This experiment evaluated how users perceived the usefulness of the AI feedback after finishing a scenario. After completing one hazard event, each participant could read the generated feedback and rate it on a scale from 1 to 5 for usefulness, where 1 means not helpful and 5 means very helpful. Eight to ten student users would be enough for an early test. This setup works because the program is meant to teach safer driving, not just simulate it, so the quality of the feedback matters as much as the driving itself. The control data would come from the same prompt structure and scoring format being used for every participant.

| Participant | Usefulness Rating (1-5) |
|-------------|-------------------------|
| P1          | 4                       |
| P2          | 5                       |
| P3          | 4                       |
| P4          | 3                       |
| P5          | 4                       |
| P6          | 5                       |
| P7          | 2                       |
| P8          | 1                       |

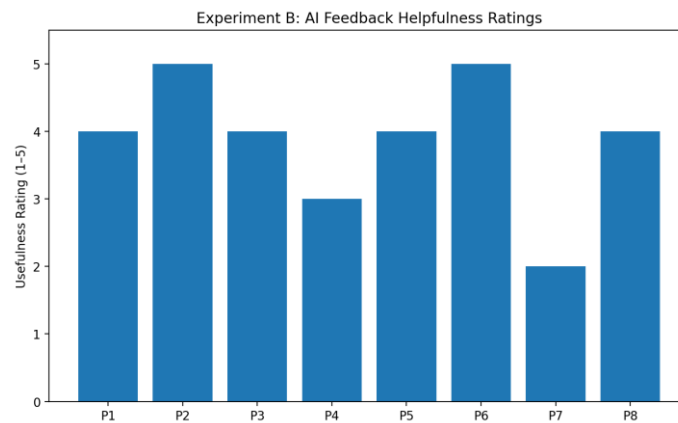


Figure 9. User-rated usefulness of AI feedback in Experiment 2

The mean usefulness rating is 3.88, and the median is 4. The lowest value is 2, while the highest value is 5. Overall, the results suggest that most users found the feedback useful, because six of the eight ratings are 4 or 5. However, the rating of 2 is important because it shows that the system may not work equally well for every player. The lower rating may have occurred because the response felt too general, repeated obvious advice, or did not closely match the specific mistake made during the scenario. That would be a major limitation for an educational simulator, since users need feedback that is clear, personalized, and actionable. The biggest factor affecting these results is likely the specificity of the data sent into the AI system. If the simulator provides vague or limited driving metrics, the generated feedback may also become broad. Therefore, the quality of the analysis depends heavily on the structure and detail of the recorded performance data.

## 5. RELATED WORK

Krasniuk et al. (2024) studied whether driving simulator training helps young novice drivers improve safety-related skills [13]. Their review found that simulators can improve important abilities such as lane maintenance, speed control, and reaction to hazards in a safe practice environment. This makes the approach useful because it allows training without exposing real drivers to real crashes. However, the review also notes that evidence is still limited on how well these gains transfer to long-term real-world crash reduction. It focuses on simulator training broadly but does not emphasize personalized feedback. DriveWise improves on this by combining simulator performance with AI-generated analysis tailored to each user.

Chan et al. (2010) examined whether driving simulators are effective for evaluating hazard anticipation, speed management, and attention maintenance in novice drivers [14]. Their study found that simulators can reveal meaningful differences between novice and experienced drivers, especially in areas related to crash risk. This makes the method effective for identifying weak driving behaviors under controlled conditions. However, the study mainly evaluates driver performance rather than turning the results into a coaching system for students. It also focuses more on assessment than on user-friendly training. DriveWise improves on this by not only measuring driving behavior but also translating collected data into clear AI-generated advice for improvement. Pradhan et al. (2011) tested a PC-based training program called FOCAL that taught novice drivers to avoid looking away from the road for unsafe lengths of time [15]. The program was effective because trained drivers showed fewer long glances away from the roadway during later testing. This demonstrates that focused computer-based instruction can improve one important part of safe driving behavior. However, the method is narrow because it concentrates mainly on attention maintenance and ignores other emergency hazards such as hydroplaning or tire failure. It also does not simulate full vehicle control. DriveWise improves this by training broader hazard-response skills in an immersive driving environment with steering-wheel input and post-drive AI feedback.

## 6. CONCLUSIONS

One limitation of DriveWise is that, although it can simulate dangerous driving situations, it still cannot fully match the unpredictability of real-world driving. Real roads include weather variation, traffic behavior, driver emotion, and many distractions that are difficult to recreate perfectly in a simulator. Another limitation is that the AI feedback is only as strong as the driving data and prompts provided to it. If the recorded data is too limited, the advice may become generic instead of highly personalized. In addition, the current project appears to focus on a small set of hazard scenarios, which means it may not yet cover the full range of dangerous situations that new drivers face.

If more time were available, the project could be improved by expanding the number of scenarios, refining the realism of vehicle physics, and tracking more detailed performance metrics. It could also improve the AI feedback system by sending more structured driving data so that the analysis becomes more specific, consistent, and educational.

DriveWise is a promising educational simulator that combines interactive driving practice with AI-based feedback. By helping users respond to hazards in a safe virtual setting, the project addresses an important real-world problem. With further refinement, it could become an even more effective tool for teaching safer driving habits.

**REFERENCES**

- [1] Bunn, Frances, et al. "Traffic calming for the prevention of road traffic injuries: systematic review and meta-analysis." *Injury prevention* 9.3 (2003): 200-204.
- [2] Segui-Gomez, Maria, et al. "Assessing the impact of the WHO global status reports on road safety." *Injury Prevention* 31.Suppl 1 (2025): i1-i6.
- [3] Shope, Jean T., and C. Raymond Bingham. "Teen driving: motor-vehicle crashes and factors that contribute." *American journal of preventive medicine* 35.3 (2008): S261-S271.
- [4] Hossain, Md Mahmud, and M. Ashifur Rahman. "Understanding the potential key risk factors associated with teen driver crashes in the United States: a literature review." *Digital Transportation and Safety* 2.4 (2023): 268-277.
- [5] Ginsburg, Kenneth R., et al. "National young-driver survey: teen perspective and experience with factors that affect driving safety." *Pediatrics* 121.5 (2008): e1391-e1403.
- [6] Jonah, Brian A. "Accident risk and risk-taking behaviour among young drivers." *Accident Analysis & Prevention* 18.4 (1986): 255-271.
- [7] Stavrinou, Despina, et al. "Impact of distracted driving on safety and traffic flow." *Accident Analysis & Prevention* 61 (2013): 63-70.
- [8] Retting, Richard A., C. M. Farmer, and A. T. McCartt. "Insurance Institute for Highway Safety." (2001): 1-10.
- [9] Classen, Sherrilene, et al. "An integrative review on teen distracted driving for model program development." *Frontiers in public health* 7 (2019): 111.
- [10] Cazzulino, Francesca, et al. "Cell phones and young drivers: a systematic review regarding the association between psychological factors and prevention." *Traffic injury prevention* 15.3 (2014): 234-242.
- [11] Durbin, Dennis R., et al. "Special considerations in distracted driving with teens." *Annals of advances in automotive medicine* 58 (2014): 69.
- [12] Krasniuk, Sarah, et al. "The effectiveness of driving simulator training on driving skills and safety in young novice drivers: A systematic review of interventions." *Journal of Safety Research* 91 (2024): 20-37.
- [13] Chan, Elsa, et al. "Are driving simulators effective tools for evaluating novice drivers' hazard anticipation, speed management, and attention maintenance skills?." *Transportation research part F: traffic psychology and behaviour* 13.5 (2010): 343-353.
- [14] Allen, R. Wade, et al. "Simulator training of novice drivers: a longitudinal study." *Advances in Transportation Studies* 27 (2012).
- [15] Wang, Yafei, et al. "Dual-cameras-based driver's eye gaze tracking system with non-linear gaze point refinement." *Sensors* 22.6 (2022): 2326.