SUMO User Conference 2026

TIBconferencesession

TIB-OP will set DOI with \TIBdoi

Submitted: 2026-03-01

EfaSim - A Prototype for a SUMO-driven Emergency Vehicle Driving Simulator

A SUMO-Centered Approach

Rebecca Ahmed [1], Fabian Schuhmann[1], Cristian Cubides-Herrera[1], Markus Lienkamp[1]

[1]Technische Universität München, München, DE

*Correspondence: Cristian Cubides-Herrera, sebastian.cubides@tum.de

Abstract: EfaSim (German: ”Einsatzfahrt-Simulator”, English: ”Emergency Driving Simulator”) is a prototype for an emergency vehicle driving simulator which uses the microscopic traffic simulation SUMO (Simulation of Urban Mobility) to simulate the behavior of other traffic participants. Unreal Engine provides rendering power to visualize the environment, traffic, and the user interface. In the game engine, the user steers an emergency vehicle equipped with a toggleable blue light. This device influences the reaction of the other vehicles to the emergency vehicle within the designated radius. Communication between Unreal and Sumo is established using a dynamic linked library and the C++ implementation, Libsumo. Our goal is to create an efficient prototype that can simulate and render the game environment with a large number of simultaneously active vehicles while achieving an optimal frame rate of 60 frames per second (fps). In addition, we use a completely free-to-use pipeline for creating the environment and running the simulation. We benchmark our implementation to test its efficiency and further compare our implementation with an additional TraCI implementation. The results show that both simulations operate on similar fps; however, the DLL achieves a measurable increase in information flow efficiency. We conclude that our architecture performs satisfactorily for our requirements, being able to simulate more than 220 cars at 60 fps.

Keywords: Driving Simulator, Unreal Engine, Emergency Vehicle

1 Introduction

Emergency vehicles in action are at a high risk of traffic accidents, despite having sirens or blue light activated [1]. Major incidents often happen in difficult scenarios, such as angular junctions or curved roads [2], [3]. Thus it is important that the drivers of those vehicles are capable of driving safely at high speeds and able to navigate complex traffic situations, all while under the pressure of an imminent emgergency.
Training for these high-risk scenarios in live traffic is both dangerous and impractical. To bridge this gap, driving simulators have become the industry standard, offering a

safe, repeatable, and controllable environment that can be scaled far more easily than traditional field exercises [4]. Current data suggests these programs are highly effective; trainees report high satisfaction levels, and more importantly, show measurable improvements in real-world driving behavior [5], [6].
Building on this, we create a prototype which uses the Game Engine Unreal [7] for visual representation and as the interaction point for the player, and use the microscopic traffic simulation SUMO [8] as the foundation of the traffic behavior. To achieve flexibility we use Open-Street-Map [9] data to model the environment and gain an real-life road network.
The goal is to implement an efficient prototype that is able to visualize a large number of cars simultaneously while maintaining an optimal frame rate of 60 frames per second, or at least a minimum frame rate of 30 frames.

2 Related Work

SUMO has been integrated as part of co-simulations into game engines such as Unity [10] or the Unreal Engine [7]. The engines are used to represent the simulation visually and enhance realism by leveraging graphic capabilities and physics simulations. This architecture can be used to create digital twins of urban environments to recreate specific traffic scenarios [11]and simulate vehicle behavior [12].
Michele Roccotelli et al. [12] use SUMO and Unity to tackle route planning for Autonomous Vehicles within a digital twin of the city Bari. The environment is created using paid software. The simulation is entirely based on SUMO, whereas Unity is used purely for visualization.
A similar approach is taken by Xishun Liao [13], who focus their studies on connected automated vehicles to simulate ramp-merging scenarios. Unity is used to better analyse and evaluate their algorithm, but it only represents the state in the SUMO simulation and does not actively influence it.
The project Sumonity [14] crosses the threshold of only visualizing in the game engine by allowing for control within Unity itself. This opens up opportunities for refined vehicle control which could be expanded into a first person driving simulator.
Youssef et al. [11] integrate SUMO into Unreal Engine to simulate behavior in narrow passageways. This is then realized as a complete simulator in which hardware such as pedals and steering wheels mimic the environment of a car, with screens displaying the game state. The game environment itself, apart from the road network which is imported from SUMO to Unreal using a plugin, is manually configured.
To establish communication between the traffic simulation and the engine, a popular approach is to use the python library TraCI which is provided by SUMO and uses a TCP-based client/server architecture [11] [12] [14].
TraCI can be seen as the standard when working with SUMO and has a wide variety of commands for value retrieval and simulation control. For performance-focused tasks, however, the SUMO documentation refers to Libsumo, which is a C++ library that circumvents the overhead of socket communication by enabling low-level access [15].

2.1 Contribution

Our prototype seeks to enable communication between Unreal Engine and SUMO by exploring a different method than the commonly used server-socket approach. The architecture integrates three core technologies:

- Unreal engine [7]: The engine provides the rendering power to visualize the different traffic participants.
- SUMO [8]: The backbone, which provides the behavioral logic of the implemented actors to ensure a realistic traffic flow.
- Open Street Map [9]: OSM provides the real-world geospatial data, allowing us to replicate actual road networks with high precision.

The primary objective is to develop an efficient prototype capable of simulating and visualizing dense traffic environments. Our focus lies on maintaining an optimal frame rate (FPS) while managing a high volume of simultaneous vehicle actors to ensure a seamless, immersive training experience.

3 Implementation

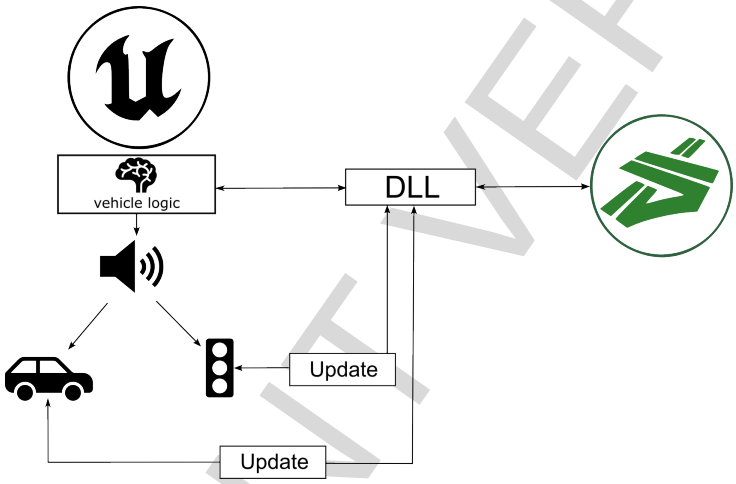

Figure 1. *Scematic implementation of the prototype*

To simulate traffic participants and their interactions with the ego-vehicle, the backend simulation is powered by Libsumo and visualized in Unreal Engine 5.5.4 [7]. While the Traffic Control Interface (TraCI) is the standard method for coupling SUMO with external applications, it relies on TCP socket communication, which often introduces significant latency [15]. As our goal is to create an efficient prototype and thus to have a fast update rate, Libsumo, the C++ implementation of TraCI, has been chosen. While Libsumo offers a slightly reduced feature set compared to the full TraCI API, it is sufficient for this project as SUMO is primarily used to retrieve simulation states, with Unreal providing input data sparsely.
The communication between both environments is established via a Dynamic Link Library (DLL). A DLL is an executable file that consists of multiple library functions and can be used in Unreal as a module. It is important to note that the DLL is not standalone; the user machine must have the SUMO repository installed, the necessary SUMO libraries linked, and the appropriate system environment paths configured.

3.1 Unreal Execution and synchronization

Unreal serves as the central controller for the entire simulation. Instantiation, termination as well as every explicit simulation step in Sumo are triggered by Unreal. This approach ensures that every object's state is calculated and displayed before the next

timestep begins. To maintain visual clarity and enable the player to react to the simulated traffic, Unreal does not update the SUMO simulation every tick. Instead, it triggers the simulation every *x* ticks, where *x* is a configurable integer that defines the simulation delay.
At game initialization, Unreal performs a ”Setup” in which the DLL functions are loaded, and the Sumo simulation is initiated. In this stage, the ego-vehicle is added to the SUMO simulation, using the starting position.

3.2 Unreal Update Loop

An update loop in Unreal always follows the same pattern and is executed only when a simulation step in SUMO is taken.

- Vehicle Updates
- Ego Vehicle Update
- General Update Call

Vehicle Updates
Each update, Unreal retrieves information about the currently departed and arrived vehicles. Departed Vehicles are added, whereas arrived vehicles are removed. A pooling pattern is used to avoid constant memory allocation and deallocation when instantiating or deleting cars, as this can negatively impact performance [16]. Arrived cars are not written into a buffer and marked as inactive. They are not rendered, nor do they have collisions. When a new car departs, a vehicle is retrieved from the buffer, reactivated, and reassigned to the new profile. New instances are created only when the pool is empty.

Ego-Vehicle Update

The ego-vehicle is updated based on the player’s current position and angle. In SUMO, a vehicle position is typically restricted to a single edge. However, a human driver may cross the center line at any given time, and may change edges arbitrarily, or occupy two edges at the same time, as illustrated in figure 2. Vehicles on both edges, the current and the opposing, should be able to react to the ego-vehicle and, if active, the bluelight device. To model this behavior, the ego-vehicle is modeled as three separate proxy vehicles in SUMO, positioned at the front bumper, the center and the rear.

General Update Call
The general update call, represented by the loudspeaker symbol in figure 1, is an event dispatcher that, once triggered, causes all actors to execute a corresponding function independently. This is currently done for the vehicles and traffic light systems (TLS) but can be extended for other participants.
Vehicles refer to the DLL to retrieve their next position, angle, and car signal. As each vehicle updates its position independently, the position and angle values are interpolated based on the Unreal-SUMO update rate.
The TLS retrieves the current state of the traffic lights that are controlled by each junction. A string such as ”Ggrrgg” models a junction comprised of three streets. Each

update, this string is cut and forwarded to each lane. Traffic lights change according to the letter, except for ”G” and ”g”, which both model a green phase, once with and once without priority, respectively. While SUMO defaults to instantaneous Red-to-green transitions, the project introduces an intermediate red-yellow state within Unreal to provide a more realistic visual representation of European traffic signals.

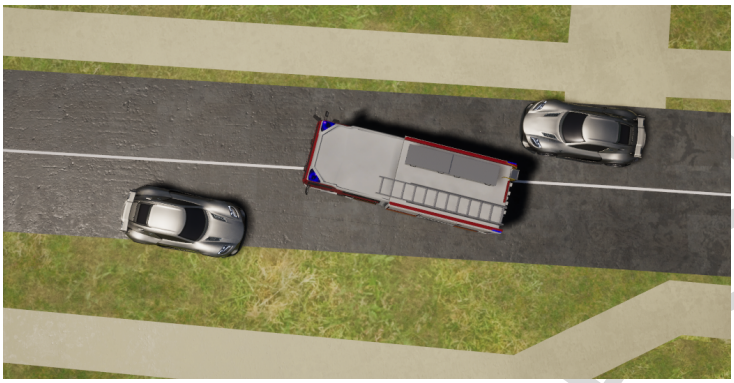

Figure 2. *Vehicles on both edges reacting to the ego vehicle with an active bluelight device*

3.3 Ego vehicle Interaction

The ego-vehicle is modeled as an emergency vehicle and controlled by the player. By default, other vehicles react to the vehicle. If the ego vehicle is blocking a path, other vehicles will wait behind it or at the start of the junction until the ego vehicle has cleared the path. To ensure a realistic variance, not all vehicles react perfectly, mimicking real-world driver unpredictability.
Interaction with traffic is further governed by a blue light device, modeled after the RescuePY framework [17] to simulate the formation of a ”rescue lane” (Rettungsgasse).
An active bluelight device will cause the vehicles to slow down and move to the lane margins; leftmost vehicles will move to the left, others to the right. Due to current Libsumo limitations regarding toggling states, we utilize the reaction distance to model the siren. A range of zero effectively switches off the bluelight device.
In reality, the blue light and the siren can be toggled individually. This is also modelled by the reaction distance. If both components are active, the range is larger than if only the blue light is activated.

3.4 Visual presentation

The prototype leverages several open-source and free-to-use tools to translate the SUMO environment into Unreal Engine, the results of which can be observed in figure 3. To create a realistic test scenario, the OSM Web Wizard [18] has been used.

Vehicle Assets
The simulated cars are high-poly assets provided by the Unreal Vehicle Template. The player model, an HLF20, is custom-made, complete with materials and the ability to display the toggled blue light. Physical movement and wheel animations are handled by the Unreal Chaos-vehicle component.

Buildings
Buildings are generated using BLOSM [19], a Blender [20] add-on that uses data from

OpenStreetMap [9] and generates buildings accordingly. The results of the standard version are rectangular houses with different roof shapes. To obtain a more realistic approximation, an extension of this that utilizes Geometry nodes has been used [21]. This add-on is not only free to use, but the modular nature of it allows for more control in the generation, by manipulating the set of building modules, thus allowing for enhanced optimization and realism.
While BLOSM is able to extract a road network from OSM, network appears to differ slightly from the SUMO network in regard to spacing and orientation. To ensure a SUMO-accurate mapping, SumoNetVis [22] is used.

Object Spawning
Traffic light metadata (ID, positions, traffic lights per lane, light configuration, angles) and natural elements such as trees are exported via custom scripts to .txt files, which Unreal then parses to procedurally place actors in the world and connect them to other components if necessary.

Figure 3. *Impressions of the driving simulator*

4 Results

As the driving simulator is currently in the prototype stage, a preliminary evaluation has been conducted to assess its core simulation capabilities. The primary focus of this benchmark is to compare the performance of the Libsumo (DLL-based) implementation with that of a traditional TraCI (Server-based) implementation. Advanced features such as vehicle pooling and other features were deactivated in the Libsumo version to match the limitations of the TraCI setup. All benchmarks were performed on a high-end workstation to ensure that hardware bottlenecks did not influence the software comparison:

- CPU: Intel(R) Core(TM) i9-14900K
- GPU: NVIDIA GeForce RTX 4080
- RAM: 192 GB

TraCI implementation
In the TraCI implementation, a server is set up in Python, and a connection from Unreal to the server is established using Websockets. Every update, Unreal sends a message containing ego-vehicle information to the server, which uses it to update the simulation and then returns the changes to all other actors as a JSON string. The remaining steps in Unreal are performed analogously to the Libsumo implementation.

4.1 Frame Rate Analysis

Figure 4 shows the TraCI and Libsumo implementations, both running for 1000 simulation steps, with one simulation step being made every 20 frames in Unreal. Every ten seconds, the average frame rate has been recorded, and the average result of 5 simulations has been plotted on the graph. Both, the TraCI and the Libsumo implementation show very similar results regarding the frame rate progression.
The frame rate progression for both implementations remains remarkably consistent. As shown in the performance table 1, both methods maintain high stability even as the vehicle count increases. The data suggests that for the current scope of the

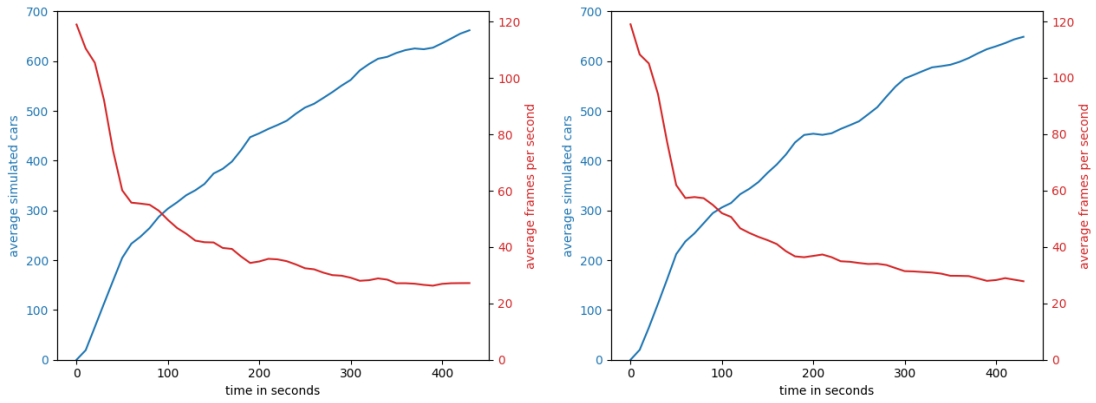

Figure 4. *Results of the fps benchmarking of the TraCI(left) and Libsumo(right) implementation both with the traffic scale set to 20.*

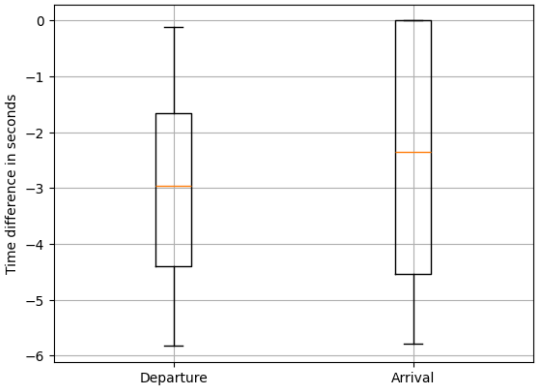

Figure 5. *Time difference comparisons of vehicle departure and arrival times, a negative number implies that the Libsumo implementation took less time*

Table 1. *Direct comparison of TraCI and Libsumo in regard to fps*

Performance Metric/Implementation	TraCI	Libsumo
Update Method	Asynchronous Websockets	Synchronos C++ API
Optimal fps (60 fps)	229 Vehicles	223 Vehicles
Minimum Requirement (30 fps)	588 Vehicles	592 Vehicles

prototype, a significant performance divergence in frame rate cannot be observed. Both implementations are capable of simulating over 220 vehicles while maintaining an average frame rate of 60 fps.

4.2 Latency Analysis

A time-series analysis (illustrated in Figure 5) has been conducted over 400 simulation steps with a traffic scale of 1 to evaluate synchronization efficiency.
The time difference is calculated by averaging the iterations of the Libsumo and TraCI implementation and subtracting the TraCI values from the Libsumo values.

$$\text{Difference} = \frac{\sum_r (x_r - y_r)}{|r|}$$

Here, r denotes the different iterations, $x \in \mathbf{R}^d$ are the timestamps of Libsumo and $y \in \mathbf{R}^d$ the timestamps of TraCI.
Vehicles in the Libsumo implementation consistently depart earlier than those in the TraCI implementation, with an average lead of 3 seconds. This temporal gap widens as the simulation progresses, indicating cumulative overhead in the socket-based TraCI communication.

5 Discussion

The frame rate progression in both simulations is very similar, which can be explained by the architecture of the simulations. The Libsumo architecture calls dll functions per tick. The functions are sequential, yet the dll allows for direct manipulation of data. The TraCI implementation works with a server-socket communication using Websockets. Thus data is sent in the form of a Json and must be processed in Unreal. However, Unreal continues to perform update steps and sends updates even if the server has not responded at that point in time. Only once a message is received, Unreal updates the actors in the scene. Thus, the frame rate is independent of the server response. As there have been no further attempts to optimize the meshes for the rendered vehicles, the main reason for the drop in frames is most likely the car rendering.
The time difference in the arrival and departure times shows that the information flow between Sumo and Unreal is faster in the Libsumo implementation, as vehicles depart and arrive consistently earlier than in the TraCI implementation, even though the simulation conditions are equal. As the difference in time goes up consistently, this could lead to delays of several seconds in long simulations. Calculating the p-value over the averaged departure times in Libsumo and TraCI yields a value of practically zero. Thus, we can confirm that the Libsumo implementation is significantly faster in processing vehicle departures than the TraCI implementation.
Regarding architecture, both implementations perform similarly in terms of fps, yet the departure times in Libsumo suggest that the information flow is faster with a DLL. As this delay grows over time, it could potentially noticeably slow down the updates between Unreal and Sumo, causing the ego-vehicle to feel less responsive in practice.
In general, the benchmark shows that the simulation performs sufficiently well, rendering a large number of vehicles simultaneously at the desired optical frame rate.

6 Conclusion and Future Work

In this work, we introduced Efasim, a prototype for an interactive driving simulator that uses SUMO as its backbone to simulate the behavior of other traffic participants. As the prototype is still under development, a user study has not yet been made. Instead,

a preliminary benchmarking has been performed, comparing the libsumo implementation with a TraCI implementation, also developed by us. We show that the libsumo implementation can simulate more than 220 vehicles at the optimal framerate of 60 fps. TraCI achieves the same frame rate with a similar number of cars. The speed at which information is exchanged between Sumo and Unreal appears to be faster when using a DLL, potentially improving the interaction reaction speed of traffic participants to the ego vehicle.
Regarding future work, several limitations need to be addressed. The current network is car-only. This is in part for simplicity when building the prototype, and also because the OSM Web Wizard tool struggles to create not-car-only networks, which becomes evident when junctions are involved. Future iterations of this prototype will need to resolve this issue, as problems and difficult situations for emergency drivers are not only caused by cars but also by pedestrians, bikes, or others.
The different stages of importing, extracting, and spawning are fragmented processes that must be done manually. It would be beneficial to consolidate these into a single workflow that automates the steps in between, reducing workload and enabling people not involved in development to create their own scenarios.
Lastly, to determine whether this prototype is suitable for training emergency vehicle drivers, a comprehensive study including the target audience must be conducted.

Data availability statement

Data is available upon request. Please contact sebastian.cubides@tum.de for further information.

Author contributions

Rebecca Ahmed has been in charge of the software development, validation, visualization, and the original draft. Fabian Schuhmann has been responsible for the acquisition of funding and project administration, as well as review and editing of the draft. Cristian Cubides-Herrera supported the final development stages by overseeing technical implementation and refinements. Markus Lienkamp has taken on the supervision of the project.

Competing interests

The authors declare that they have no competing interests.

Funding

This work was supported by the German Federal Ministry for Research, Technology, and Space (BMFTR) under the grant number 13N17326 as part of the joint project ’Einsatzfahrt-Simulator für das Ehrenamt (EfaSim)’ – sub-project: ’Prozedurale Generierung und Integration von Umgebungsinformationen in die digitale Ausbildung’ at the Technical University of Munich.

References

[1] M. J. Prohn, D. Nowak, and B. Herbig, *Achtung Blaulicht - Evaluation des Trainings „Verkehrssicherheit bei Einsatzfahrten“ des DVR und der DGUV (FP-0366)*. München: Institut und Poliklinik für Arbeits-, Sozial- und Umweltmedizin, Klinikum der Universität München, 2018.

[2] A. Vrachnou, “An analysis of emergency vehicle crash characteristics,” Ph.D. dissertation, Virginia Tech, 2003.

[3] N. Abdelwanis, “Characteristics and contributing factors of emergency vehicle crashes,” Ph.D. dissertation, Clemson University, 2013.

[4] LandesFeuerwehrVerband Bayern e.V. “Einsatzfahrten-simulator.” (2025), [Online]. Available: https://www.lfv-bayern.de/angebote/ausbildungsangebote/einsatzfahrten-simulator/ (visited on 02/15/2026).

[5] J. T. Lindsey and A. E. Barron, “Effects of simulation on emergency vehicle drivers’ competency in training,” *Prehospital and Disaster Medicine*, vol. 23, no. 4, pp. 361–368, 2008. DOI: 10.1017/S1049023X00006014.

[6] A. Neukum, B. Lang, and H. Krueger, “A simulator-based training for emergency vehicle driving,” in *Proceedings of the Driver Simulation Conference*, University of Iowa Dearborn^ eMichiganIowa Michigan, 2003.

[7] Epic Games, *Unreal engine 5.5.4*. [Online]. Available: https://www.unrealengine.com.

[8] P. A. Lopez, M. Behrisch, L. Bieker-Walz, *et al.*, “Microscopic traffic simulation using sumo,” in *The 21st IEEE International Conference on Intelligent Transportation Systems*, IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/.

[9] OpenStreetMap contributors, *Planet dump retrieved from https://planet.osm.org*, https://www.openstreetmap.org, 2017.

[10] Unity, *Unity*. [Online]. Available: https://unity.com.

[11] P. Youssef, K. Plant, and B. Waterson, “Joining sumo and unreal engine to create a bespoke 360 degree narrow passage driving simulator,” *SUMO Conference Proceedings*, vol. 5, pp. 93–112, Jul. 2024. DOI: 10.52825/scp.v5i.1104. [Online]. Available: https://www.tib-op.org/ojs/index.php/scp/article/view/1104.

[12] M. Roccotelli, G. Volpe, M. P. Fanti, and A. M. Mangini, “A co-simulation framework for autonomous mobility in urban mixed traffic context*,” in *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, 2024, pp. 812–817. DOI: 10.1109/CASE59546.2024.10711495.

[13] X. Liao, X. Zhao, Z. Wang, *et al.*, “Game theory-based ramp merging for mixed traffic with unity-sumo co-simulation,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 9, pp. 5746–5757, 2022. DOI: 10.1109/TSMC.2021.3131431.

[14] M. Pechinger and J. Lindner, “Sumonity: Bridging sumo and unity for enhanced traffic simulation experiences,” *SUMO Conference Proceedings*, vol. 5, pp. 163–177, Jul. 2024. DOI: 10.52825/scp.v5i.1115. [Online]. Available: https://www.tib-op.org/ojs/index.php/scp/article/view/1115.

[15] SUMO Documentation. “Libsumo.” (2026), [Online]. Available: https://sumo.dlr.de/docs/Libsumo.html (visited on 02/15/2026).

[16] C. Coutinho, “Implementing efficient object pooling,” in *Unity Cookbook: Core Recipes for Game Developers*. Berkeley, CA: Apress, 2024, pp. 585–623, ISBN: 979-8-8688-0853-1. DOI: 10.1007/979-8-8688-0853-1_7. [Online]. Available: https://doi.org/10.1007/979-8-8688-0853-1_7.

[17] F. Schuhmann, M. Sievers, S. Schrott, I. Kapovich, L. Feng, and M. Lienkamp, "Rescuepy: Simulation-based rescue response impact assessment," *SUMO Conference Proceedings*, vol. 5, pp. 17–37, Jul. 2024. DOI: 10.52825/scp.v5i.1029. [Online]. Available: https://www.tib-op.org/ojs/index.php/scp/article/view/1029.

[18] SUMO Documentation. "Osmwebwizard." (2026), [Online]. Available: https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html (visited on 04/15/2026).

[19] vvoovv. "Blosm." (2026), [Online]. Available: https://github.com/vvoovv/blosm (visited on 02/15/2026).

[20] Blender Foundation, *Blender*. [Online]. Available: https://www.blender.org/.

[21] P. Oliva. "Buildify." (2026), [Online]. Available: https://paveloliva.gumroad.com/l/buildify (visited on 02/15/2026).

[22] P. Malcolm. "SumoNetVis." (2025), [Online]. Available: https://github.com/patmalcolm91/SumoNetVis (visited on 02/15/2026).