MizSIM: A Headless Open-Source Simulation Framework for Training and Evaluating Artificial Intelligence

Andrii Soloviov, Derek T. Anderson, Andrew R. Buck, and Brendan Alvey

Electrical Engineering and Computer Science Department, University of Missouri, Columbia, MO 65211, USA

ABSTRACT

Advancements in deep learning have revolutionized the artificial intelligence (AI) landscape. However, despite considerable performance enhancements, their reliance on data and the intrinsic opacity of these models remains a challenge, hindering our ability to understand the reasons behind their failures. This paper introduces a headless open-source framework, coined MizSIM, built on the Unreal Engine (UE) to generate high volume and variety synthetic datasets for AI training and evaluation. Through the manipulation of agent and environment parameters, MizSIM can provide detailed performance analysis and failure diagnosis. Leveraging UE's open-source distribution, cost-effective assets, and high-quality graphics, along with tools like AirSim and the Robotic Operating System (ROS), MizSIM ensures user-friendly design and seamless data extraction. In this article, we demonstrate two MizSIM workflows: one for a single-life computer vision task and the other to evaluate an object detector across hundreds of simulated lives. The overarching aim is to establish a closed-loop environment to enhance AI effectiveness and transparency.

1. INTRODUCTION

Simulation has long served as an important tool for exploration, understanding, and testing across a spectrum of disciplines, ranging from robotics to biology, chemistry, physics, and beyond. Recent developments have seen a convergence of key factors such as enhanced computational resources, readily available assets (e.g., high-resolution photometrically scanned models and materials), and the proliferation of open-source software. The focus of the current article is a new headless open source library that leverages tools traditionally associated with gaming or film to generate a diverse array of high quality imagery accompanied by rich metadata and ground truth. These resources are critical for training, evaluating, and advancing our understanding of artificial intelligence (AI).

Despite notable advancements, two persistent bottlenecks impede our progress towards achieving a closed-loop AI framework with minimal human involvement. Firstly, there is the challenge of efficiently generating large, controlled datasets. Secondly, there is the issue of the considerable human intervention and oversight necessary. This article presents MizSIM, an open-source framework built on the Unreal Engine (UE) for Unix systems. MizSIM is headless and features a modular design, allowing for quick and easy customization to control various environments dynamically (e.g., objects, weather, time of day) and produce formatted sensor data, metadata, and ground truth for AI training and evaluation. Subsequent sections will delve into MizSIM's architecture, implementation, and provide examples to highlight its adaptability. The reader can find both our github 1 and github 2 repositories and example videos on the web.

In our first example, we showcase MizSIM simulating a single life cycle for an agent engaged in offline 3D reconstruction. This demonstration serves to illustrate how MizSIM can replicate and support a typical real-world workflow, encompassing pre-scripted flight, data collection, and post-flight processing. The process involves deploying a moving drone equipped with a single visual spectrum camera and positional sensors (GPS and IMU). Beginning with the remote initialization of an environment on the server, the agent is then deployed to systematically gather data along a pre-programmed flight path. Following the flight, collected data undergoes analysis through another algorithm, facilitating post-flight scoring and assessment. This workflow can seamlessly be transitioned to support real-time processing if the user desires. The end result is a reconstructed 3D scene, which is compared against the ground truth that is also collected during the flight.

The second example showcases MizSIM's capability to conduct multiple sequential or parallel simulations, manipulating environmental parameters to evaluate the performance of a fixed deep learning model across diverse

operational scenarios. By analyzing and consolidating results from these simulations, MizSIM can provide users with a comprehensive overview, illuminating the reliability and resilience of the model under examination. Although this example stands independently, it serves as a foundation for closed-loop simulation in AI support. Within MizSIM, a component is dedicated to determining the conditions for each simulation. While this example is set up to conduct a grid search over a limited number of variables, it can be expanded to analyze statistics across multiple runs, dynamically dictating the parameters for subsequent simulations. Thus, the core of closed-loop simulation support lies in the logic governing the selection of what to sample or simulate next. While this area warrants further exploration, MizSIM is equipped to facilitate it.

2. RELATED WORK

The present article covers a range of topics, from simulation (SIM) to environments, AI, SIM for AI, and more. Given this breadth, we cannot review everything. The following section is a succinct overview of the most pertinent works essential for comprehending the proposed work and the most closely related bodies of research.

In the realm of SIM environments, several notable platforms have emerged, each tailored to specific applications and requirements. One such platform, Gazebo,[1] has established itself as a classical SIM environment, particularly within the robotics domain. While Gazebo may not prioritize photorealism, its integration with the robotic operating system (ROS)[2] and support for a diverse range of physics and sensors, including RGB, RGBD, GPS, IMU, LiDAR, and sonar, render it a formidable tool for robotics research and development.

In contrast, Apple's HYPERSIM, Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding,[3] distinguishes itself by its emphasis on photorealistic imagery. Although lacking in physics SIM, HYPERSIM produces highly realistic visualizations, complete with geometry, material properties, lighting details, and per-pixel instance segmentations, effectively catering to applications requiring immersive visual fidelity.

NVIDIA's Isaac Gym,[4] on the other hand, offers an end-to-end platform optimized for large-scale, physically accurate multi-sensor SIM. Primarily targeted at applications like autonomous vehicles and virtual environments, Isaac Gym leverages GPU acceleration to deliver high-performance simulations, facilitating advanced research and development endeavors.

Epic Games' Unreal Engine (UE)[5] emerges as another noteworthy contender in the SIM landscape. Originally conceived as a game engine, UE's expansive license options, ranging from free to profit percentage, have spurred its adoption across diverse domains including architecture, film, and computer graphics. Moreover, UE's flexibility and extensibility, exemplified by plugins such as AirSim for UE4,[6] underscore its versatility in accommodating various simulation needs, from external Python support to the simulation of low-altitude drones.

Unity,[7] a more recent entrant in the SIM arena, has garnered attention for its comprehensive suite of development tools and cross-platform compatibility, challenging established incumbents like UE.

Beyond these specific platforms, SIM encompasses a broader spectrum of methodologies and applications. Finite element modeling, wave simulation, and ray casting are among the techniques employed to model complex phenomena, ranging from electromagnetics to physics simulations. Platforms like COMSOL and MEEP serve as examples of specialized simulation tools catering to specific domains such as multiphysics simulation and electromagnetics, respectively.

While SIM tools offer immense potential, it is imperative to acknowledge the importance of rigorous verification and validation (V&V) processes. Despite the myriad possibilities presented by SIM, ensuring the reliability and accuracy of SIM results remains a critical endeavor, warranting further exploration and scrutiny within the research community.

Lastly, the aforementioned works primarily emphasize the creation of general environments and datasets. For more specialized investigations, readers may explore focused studies such as AVOIDDS, which addresses aircraft vision-based intruder detection;[8] SynCity, a platform tailored for urban planning and autonomous driving;[9] CARLA, for autonomous driving simulation;[10] and UnrealCV, for computer vision research.[11]

3. PROPOSED FRAMEWORK

Our research conducted at the University of Missouri extensively leverages the UE across diverse areas of study. Notably, UE has played a pivotal role in our exploration of explainable AI,[12–15] procedural simulation for AI,[16–18] workflows enhancing computer vision,[17,19–23] and multi-criteria decision making.[24,25] Additionally, UE has been instrumental in specific applications such as explosive hazard detection[14,18,26,27] and passive ranging.[28,29] While UE has served as a cornerstone in investigating these topics, the solutions developed are often tailored to specific applications and pose scalability challenges due to their reliance on local desktop computing. To facilitate greater experimentation, enhance training and evaluation efficiency, and accommodate the integration of new functionality, as well as support remote execution, the development of a new framework is imperative. Ultimately, this approach should exhibit the flexibility to support single experiments to multiple iterations, ultimately enabling closed-loop AI training and evaluation.

MizSIM was designed to support the following:

- *Headless execution*: Running UE headless enhances scalability and efficiency by enabling deployment on one or more remote high performance computing servers. This setup also helps ensure optimal resource allocation to the game engine.
- *Structured data extraction*: This feature facilitates organized handling of SIM output for downstream analysis, which includes sensor data (imagery), metadata (GPS and IMU data), and ground truth. MizSIM stores data in rosbags and community standard image and JSON file formats.
- *Real-time interaction*: Support for feedback and user input, if desired, throughout the SIM lifecycle, which caters to user preferences and helps ensure real-time insights into SIM progress.
- *Orchestrator*: Entity responsible for scheduling and monitoring SIM across processes. This module not only coordinates various aspects of the SIM but also governs its overall logic, enabling diverse scenarios like single-life runs, parameter grid searches, or dynamic course adjustments to support closed-loop operations.
- *Agents*: Foundational set of codes to support pre-determined, learned, or online behaviors. This functionality also supports the ability to run after, or in real-time, models like deep learning for computer vision. This support primarily comes not from MizSIM, but from ROS.
- *Statistics*. Easy to add nodes for tasks like performance evaluation and summarizing.

To this end, our library builds on the following existing functionality:

- *UE*: While we use UE, our general setup can be extended to other environments like Unity.
- *AirSim*:[6] an open-source plugin for UE that manages the kinematics of our agent(s). This library includes pre-integrated classes designed to SIM realistic movements of various vehicles, such as drones and cars. AirSim has been a preferred choice for our team's experiments, particularly given our focus on drones. While currently maintained primarily by the community, there is no indication that AirSim will lose relevance in its field anytime soon. The versatility of UE's plugin system means AirSim can be swapped out if needed, allowing users to integrate alternative solutions.
- *ROS*:[2] a flexible message passing system that provides the foundation for all our various processes to co-exist in a simple and coordinated fashion. Like UE, ROS has a long-established history, giving us confidence in the longevity of the overall system. It has proven to be a reliable solution for connecting different software components and, most importantly, for applying communication concepts to real-world scenarios where it needs to be deployed on hardware like drones or robots.

The following sections discussed how we achieved the above, followed by working examples.

4. FRAMEWORK: TECHNICAL DESCRIPTION

The subsequent sub-sections delineate our implementation of various components within MizSIM. Although the most current documentation and demonstrations are available in the repository and accompanying videos, the insights provided in this article aim to elucidate the rationale behind their existence and general functionality. It's important to note that the details presented here represent a snapshot of their current state, subject to evolution and enhancement. MizSIM has been intentionally designed to be highly adaptable and expandable. Thus, should the reader wish to customize these components to suit different requirements or specifications, theoretically, they can make the necessary modifications with achievable goals in mind.

4.1 Parallelization

MizSIM presents a level of intricacy stemming from its orchestration of multiple concurrent processes that need to execute tasks simultaneously. For this purpose, MizSIM utilizes TMUX, a powerful terminal multiplexer tool. TMUX efficiently organizes the parallel processes of a SIM session into distinct windows, facilitating seamless management. One notable advantage of this setup is its capability to enable users, if desired, to engage with these diverse processes through the command line interface while also monitoring the real-time status of each individual component (refer to Figure 1). While some users may opt to simply initiate all processes and allow them to run autonomously, others may wish to actively inspect for errors or status updates from ROS, dynamically interact with UE, or engage with the orchestrator for monitoring or interaction in scenarios involving human intervention.

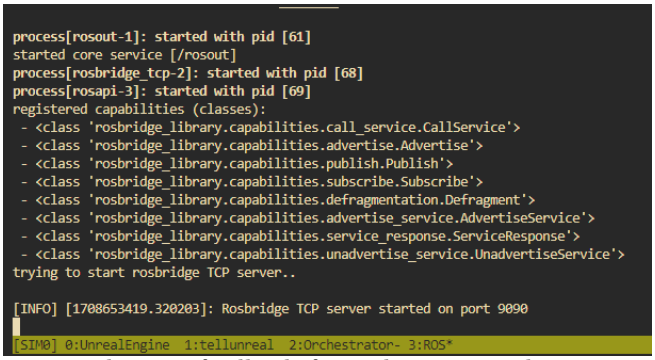

Figure 1: Example TMUX session showing feedback from the process that manages ROS. On the bottom, in yellow, is a listing of the different avilable terminal sessions; UE, tellunreal, orchestrator, and ROS.

4.2 TMUX Window 1: Unreal Engine

One of our processes oversees the functioning of UE. Consequently, this TMUX window serves as a repository for logging pertinent information. This setup proves convenient for promptly identifying compile errors, runtime glitches, or crashes that may occur during the simulation. While there are undoubtedly numerous benefits associated with this window, its primary utility thus far has been in expediting the development and debugging process by providing immediate access to critical information.

4.3 TMUX Window 2: tellunreal

This TMUX window serves as a conduit between the terminal window and UE. Within this terminal window, we establish a terminal alias named 'tellunreal'. This alias enables the transmission of user input directly to Unreal's command line interface. Facilitating communication between the user terminal and the engine is pivotal to our framework, particularly as our ultimate objective is to deploy this setup on a dedicated server where direct interaction with the engine's graphical user interface is not feasible. The implementation of the "listener" on the engine side is achieved through a custom plugin seamlessly linked upon the project startup. This integration ensures smooth and efficient communication between the user terminal and UE.

4.4 TMUX Window 3: Orchestrator

The Orchestrator window serves as the central hub for decision-making within our system. This entity is specified via an easy to modify "run.sh" script. A significant aspect of its functionality involves monitoring the log file of the current UE session. Specific phrases detected within these logs act as triggers for the framework to initiate scripts. Users have the flexibility to associate any script with phrases identified in the Unreal logs. Each association results in the creation of a new pane within the Orchestrator window dedicated to monitoring the specified trigger. This feature constitutes the primary tool utilized for constructing the backend logic of experiments. For instance, upon detecting a message such as "Editor startup took ...", indicating that all engine modules are ready, we can bind a script to this phrase for further experiment processing. This script might entail adjustments to the environment or the initiation of a Play In Editor session, thereby streamlining experimentation procedures. In summary, the orchestrator is a simple to edit script that dictates simple behavior like a single life of an agent in a specified environment to multi-life simulation and statistic gathering.

4.5 TMUX Window 4: ROS

This window hosts all components related to ROS (Robot Operating System). To commence the message listening process, a parent process called "roscore" must be running on a designated port, and this is precisely what this window initializes. This service facilitates the flow of messages from UE through ROS, culminating in the storage of these messages in files of the "rosbag" format. Rosbags offer a means to compress all ROS data into a single file, enabling subsequent extraction and processing.

It is essential to highlight the pivotal role ROS plays within MizSIM. Firstly, ROS serves as a crucial mediator, facilitating seamless communication between AirSim and UE. Secondly, ROS forms the backbone of AI in MizSIM, allowing for the integration of custom standalone components as ROS nodes without the need for fundamental alterations to the core MizSIM structure. The workflows demonstrated later in this article vividly illustrate how users can effortlessly incorporate diverse components, such as deep learning-based computer vision models for localization and detection, as ROS nodes. Furthermore, MizSIM enables more complicated interactions through a network of ROS nodes. For instance, users can deploy separate nodes for intelligent frame selection, structure from motion (SfM), and persistent maintenance of a global 3D world model via a process like UFOMap. In essence, ROS, widely used in settings like ground robotics and drones, empowers MizSIM to accomplish a myriad of AI functionality spanning sensing, decision-making, and control.

Another significant advantage of ROS in MizSIM is its role in broadcasting custom higher-quality Movie Render Queue (MRQ) imagery directly from UE. Acknowledging the visual fidelity limitations of AirSim-generated imagery, particularly when compared to outputs in UE5 utilizing nanite and lumen technologies, we developed a custom UE plugin that extracts imagery directly from UE via the MRQ. This approach ensures that imagery, along with associated layers like depth, stencil ID, and platform metadata, circumvents AirSim and is transmitted directly to ROS. Figure 2 shows the image layers we generate. Readers seeking further details on the specific layers, their formats, and documentation can refer to Ref. 17. While seemingly meticulous, these details are crucial, influencing factors such as file format, resolution of RGB, stencil, and depth imagery, and JSON ground truth information.

4.6 TMUX Window 5: AirSim

The AirSim window becomes active immediately upon the Orchestrator detecting the start of the game. Within this window, a dedicated Docker container launches an AirSim agent, tailored by the user through a Python script utilizing AirSim's Python API. This window serves multiple purposes for validation and verification. For instance, it confirms whether the agent defined in the Python script has successfully connected to the ongoing UE session. Furthermore, it enables us to verify if an agent has reached its designated goal point as intended.

In MizSIM, support for agents—whether driven by reinforcement learning or pre-scripted control logic—is facilitated through AirSim. AirSim acts as the conduit through which our agent receives sensor data (such as imagery, GPS, and IMU readings) and executes actions in UE, such as navigating to specific locations and adjusting pose. While alternative methods, such as using entities in Unreal or employing bluescripts, could achieve similar outcomes, MizSIM opts for AirSim primarily due to its comprehensive support for drone-based agents, including extended kinematic capabilities specifically tailored for UE.

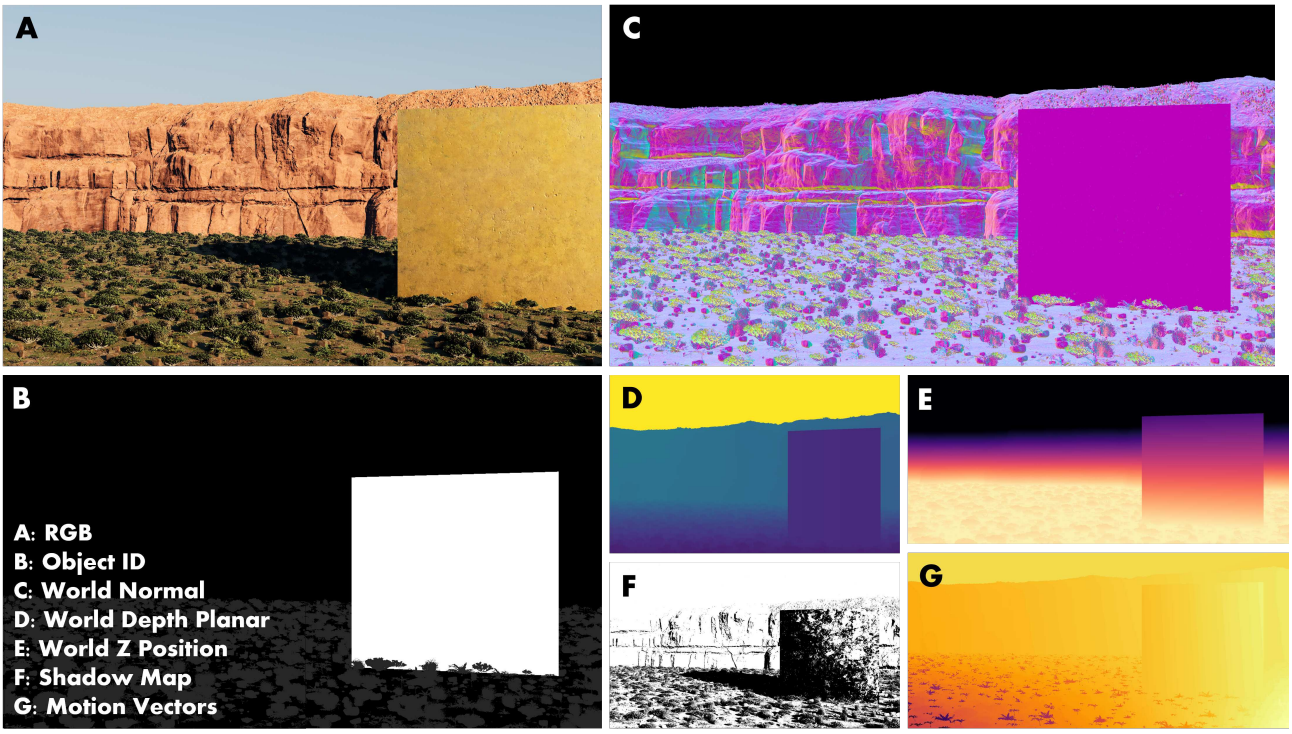

Figure 2: Example data and ground truth layers exported.

5. EXAMPLE WORKFLOWS

5.1 Workflow 1: Single Life

This workflow exemplifies the seamless integration of an existing computer vision agent, sourced from our UEUAVSim GitHub repository, into MizSIM. We used the UEUAVSim repository so the reader can, if desired, learn from but also reproduce the current article. This experiment also mirrors many real-world scenarios where users engage in drone flights, data collection, and subsequent offline processing.

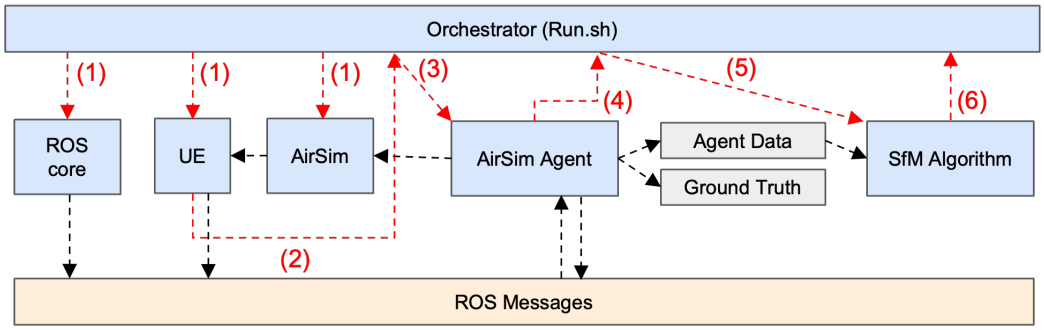

Figure 3: Workflow 1. The orchestrator starts up ROS, UE, and AirSim. Once UE is ready, the orchestrator spawns up the AirSim agent. When the agent finishes a final algorithm is executed to make the 3D reconstruction.

As shown in Figure 3, initiating this experiment involves launching UE alongside a terminal window with ROS active. Once the UE editor is operational, the user simply initiates a Docker container hosting the AirSim agent, establishing a connection with the UE instance. The provided AirSim-controlled drone, e.g., from UEUAVSim GitHub repository, executes a predefined lawnmower pattern, which captures both imagery and metadata (GPS and IMU data). In this example, our UEUAVSim agent stores its data in a custom file format. Following this step, the user can now close the terminal windows and the editor. The resultant data is then processed by

Figure 4: Example images of the Modular Neighborhood Pack.

Figure 5: Image showing our Workflow 1 in support of investigating structure from motion (SfM). This MizSIM example shows the simulation of a single agent in a single environment; see reference images in Figure 4. Specifically, it highlights how to collect a flight, store data, and process its results (a 3D point cloud) offline.

another set of code to reconstruct, using a structure from motion (SfM) algorithm, and view the 3D scene in Open3D (see Figures 4 and 5) relative to the SIM ground truth.

While the aforementioned tasks could have be staged manually, as originally performed in the UEUAVSim repository, MizSIM allows it to be fully automated and altered with ease. MizSIM allows us to bind actions to engine events. Using this, users can trigger the ROS startup to the start of the engine, start the AirSim container when the game session starts, and another set of post-processing code can be ran afterwards.

This initial example serves as a proof of concept, setting a minimum threshold for functionality. Its primary objective is to demonstrate the flexibility of our framework, showcasing users' ability to integrate an existing

AirSim-based agent, additional code modules (such as offline processing scripts), and control their execution via the orchestrator module. Should users opt to assess their algorithm's real-time performance, they can relatively easily configure the setup by providing an AirSim script to control the agent (e.g., fly a specified pattern) and another ROS node to perform real-time SfM. In a related work,[20] we (Buck et al.) did this to explore the performance trade offs of fixed vs data-driven frame selection for real-time SfM utilizing EpiDepth, which was in return employed to construct, in real-time, a probabilistic 3D world in UFOmap. Implementing this scenario requires setting up Docker containers for EpiDepth and UFOmap, with both operating as ROS nodes running in parallel and communicating with each other. Figure 6 illustrates this in a flow diagram. This example highlights the versatility and scalability of our framework, largely attributed to ROS. It demonstrates the framework's ability to adapt to various experiment setups with minimal adjustments, showcasing its flexibility and robustness.

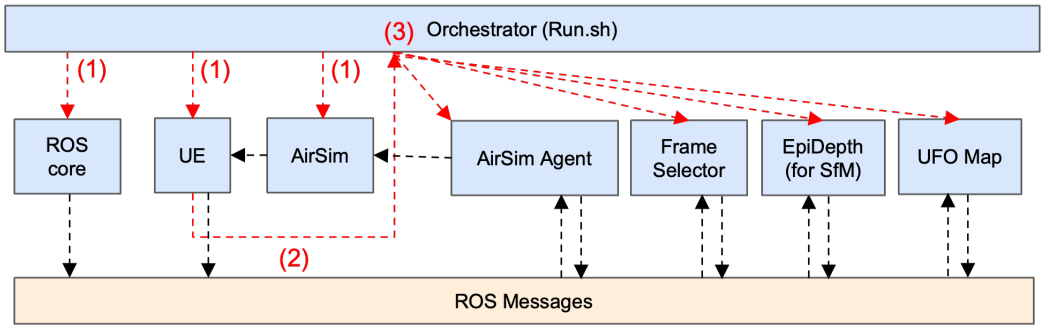

Figure 6: Illustration showing edits needed to extend Workflow 1 to online versus offline processing.

5.2 Workflow 2: Multiple Agent Lives

Workflow 2 focuses on assessing the performance of a deep learning-based object detection and localization model across diverse scenarios to evaluate its generalization capabilities amidst environmental variations. Specifically, and without loss of generality, we use the popular YOLO detector. To accomplish this objective, MizSIM is executed one hundred times, or "lives," where each life is independent and corresponds to a situation where either the agent and/or the environment is altered. In this example, we maintain consistency with the AI and detector while introducing variations in the environment.

Specifically, we chose to simulate a low altitutde drone flying around an environment sourced from the UE Marketplace, namely the "Real City SF - Downtown Environment Mega Pack." This expansive cityscape comprises towering buildings, with occasional passing cars serving as the objects for evaluation by YOLO. We arbitrarily opted to track the performance of a single vehicle in this environment to assess performance in a controlled setting, where the camera (drone) has a constant altitude and slant angle. The reader could just as easily measure performance of YOLO on all vehicles in this scene if desired. Furthermore, in a related study,[14] we (Alvey et al.) conducted a comprehensive analysis of performance concerning the relative viewing angles between the platform and the object. However, for the purposes of this article, delving into such granular detail was deemed unnecessary. Interested readers can further enhance the results reported herein using the methodologies outlined in Refs. 14 or 12 if desired.

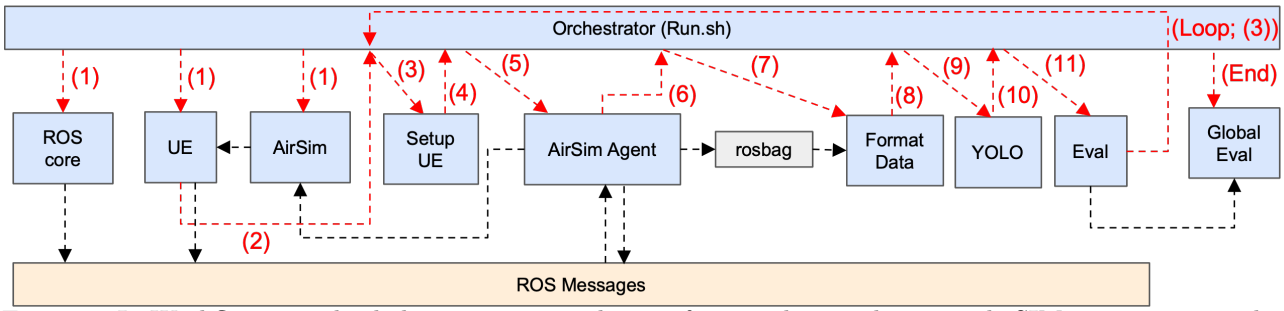

Figure 7: In Workflow 2, multiple lives are executed, one after another, within a single SIM environment. Their performance is evaluated during each life, and summarized across lifes.

Figure 7 illustrates the logic of the orchestrator. At the start of the session, a headless version of UE is launched. Once all internal initialization steps have taken place, the system will send a "Play in Editor" (PiE) command via tellunreal, start up the AirSim agent, and launch ROS recording. The life will continue until we reach the desired number of messages with RGB from the drone and ground truth.

At the conclusion of each life, we introduce modifications to our singular city environment. This process is initiated by associating a script with a specific phrase extracted from the editor's logs, signaling the conclusion of a Play in Editor (PIE) session. Leveraging Unreal's Python API within the executed Python script grants us the flexibility to execute a wide array of actions within the engine, including spawning or removing actors, adjusting their properties, manipulating time settings, and more. However, for the purposes of our experiment's design, the alterations are relatively straightforward. During this phase, we simply adjust the intensity values of our rain and time-of-day actors accordingly. The objective is to guide the drone through various combinations of time-of-day settings and rain intensities across the span of 100 lives, with each variable undergoing 10 changes throughout the process. Figure 8 is a picture of our MRQ improved imagery, Figure 9 shows the effect of altering the time of day, and Figure 10 shows changes in rain intensity.


Figure 8: Example visual spectrum image for our real-time UE-based MRQ ROS capture.

Figure 9: Our MizSIM UE plugin MRQ ouput imagery for four different times of day.

Figure 10: Output imagery for different rain intensity levels.

After each simulation, the recorded data in the rosbag format undergoes conversion into a format compatible with the object detector's requirements. Subsequently, the detector is executed, and its outcomes are archived in a JSON file, accompanied by corresponding ground truth data. The choice of JSON format was deliberate due to its simplicity, readability (user-friendly nature), ease of parsing, and its widespread adoption as a community standard. These JSON files serve as the foundation for generating performance evaluations.

In this context, we constructed receiver operating characteristic (ROC) curves, which plot the system's positive detection rate (PDR) against the false alarm rate (FAR). Figure 11 shows three contexts from our one hundred simulated lives.

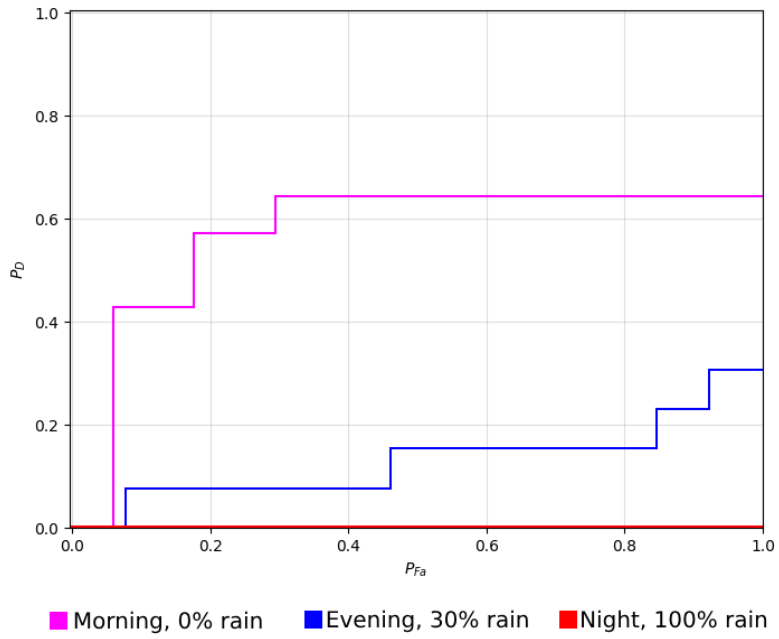

Figure 11: ROC curves for different times of day and rain percentages.

Although this example is a mere illustration to highlight a workflow, it distinctly showcases challenges encountered using a pre-trained YOLO model. In particular, performance drop as the rain level escalates to 25-30 percent. While this example is illustrative in nature, it effectively demonstrates the capacity to integrate a detector, define scene modifiers, and subsequently aggregate, score, and compile results for analysis.

The proposed example has the potential for seamless extension to encompass dynamic versus fixed agent behaviors facilitated by an AirSim agent, integration of multiple agents operating either concurrently or in parallel, deployment of diverse object detection algorithms, and beyond. MizSIM, leveraging UE and ROS, facilitates the specification and control of such functionalities at a high ROS level. This design was deliberately crafted to enable the rapid prototyping of functionalities in a highly flexible manner, primarily at the ROS node and message level, thus minimizing the need for core modifications at the underlying SIM level. Essentially, the user's task mainly involves specifying and scripting the desired changes to the SIM environment for each life, while MizSIM handles the orchestration of subsequent processes.

6. CONCLUSIONS AND FUTURE WORK

The present article introduces a novel open-source headless framework tailored for AI training and assessment. Crafted through a fusion of Unreal Engine, ROS, AirSim, and proprietary codes from the University of Missouri, this tool is detailed alongside its implementation. Furthermore, we elucidate two distinct workflows to offer readers a comprehensive understanding of its functionality. Workflow 1 delves into a single life simulation, emulating low-altitude drone data collection followed by offline processing. Conversely, Workflow 2 underscores the repetitive execution of a consistent agent performing localization and detection using a deep neural network for computer vision amidst varying environmental parameters across multiple life cycles. While our results were just to show off MizSIM, the reader could see how quickly the pre-trained YOLO network drops off in performance with respect to weather changes and time of day.

In our forthcoming endeavors, we aim to elevate MizSIM by delving into closed-loop training and evaluation scenarios across a spectrum of AI applications, spanning from autonomy to automatic target recognition (ATR). For instance, MizSIM offers the potential to delve into intricate tasks such as online structure from motion algorithm optimization or identifying the limitations of an ATR algorithm. While our current study primarily showcases MizSIM's capability in grid searching environment parameters, Figure 12 illuminates its seamless

adaptability through Run.sh (the orchestrator) modifications to intelligently reason about experiment selection and parameter sets. The pivotal challenge lies not within MizSIM itself, but rather in devising the underlying logic for intelligent experiment sampling—a quest pursued not only in AI but also in diverse fields like material and drug design. The point is, it is our belief that MizSIM is set up to perform this task.

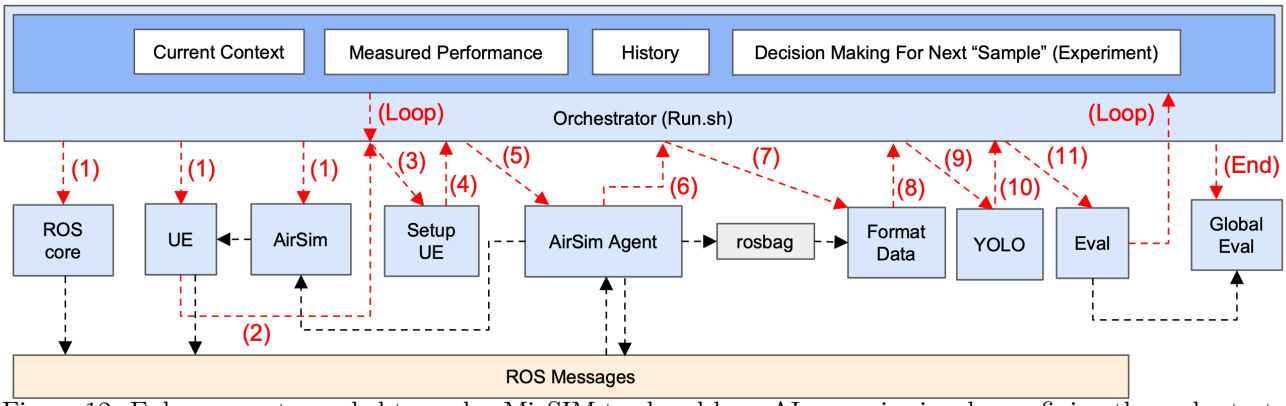

Figure 12: Enhancements needed to evolve MizSIM to closed-loop AI scenarios involves refining the orchestrator to incorporate aspects like the current experiment, its performance, historical data across experiments, and logic to determine a new set of object and environment parameters.

Lastly, we anticipate several minor enhancements in the pipeline for MizSIM. For instance, MizSIM currently boasts the capability to run across multiple servers, a feature particularly beneficial for tasks amenable to a divide-and-conquer approach. To augment this functionality, we aim to implement multi-server communication, facilitating seamless interaction between distributed instances. Moreover, in our future iterations, we intend to phase out AirSim, as its utility primarily caters to specific drone-based agent applications. Instead, we envision a more generalized abstraction layer that seamlessly interfaces with UE, potentially leveraging a proxy solution akin to AirSim. Additionally, while our framework inherently supports multi-agent scenarios, such as parallel execution of agents within ROS, we aspire to streamline this process by enriching MizSIM's functionality, thereby simplifying multi-agent setups for users.

REFERENCES

[1] "Gazebo," http://gazebosim.org/, (Accessed: 1 March 2021).

[2] "Robot Operating System (ROS)," https://ros.org, (Accessed: 8 March 2022).

[3] M. Roberts, J. Ramapuram, A. Ranjan, A. Kumar, M. A. Bautista, N. Paczan, R. Webb, and J. M. Susskind, "Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding," in *ICCV*, 2021. [Online]. Available: https://arxiv.org/pdf/2011.02523.pdf

[4] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "Gpu-accelerated robotic simulation for distributed reinforcement learning," 2018.

[5] "Unreal Engine," https://www.unrealengine.com/, (Accessed: 8 March 2022).

[6] "AirSim," https://github.com/microsoft/AirSim, (Accessed: 1 March 2021).

[7] "Unity," https://unity.com/, (Accessed: 1 March 2021).

[8] E. Q. Smyers, S. M. Katz, A. L. Corso, and M. J. Kochenderfer, "Avoidds: Aircraft vision-based intruder detection dataset and simulator," 2023.

[9] Synchrony, "SynCity: Synthetic data generation platform," https://synchrony.io/syncity, 2022.

[10] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," 2017.

[11] Y. Z. S. Q. Z. X. T. S. K. Y. W. A. Y. Weichao Qiu, Fangwei Zhong, "Unrealcv: Virtual worlds for computer vision," *ACM Multimedia Open Source Software Competition*, 2017.

[12] B. Alvey, D. Anderson, J. Keller, and A. Buck, "Linguistic explanations of black box deep learning detectors on simulated aerial drone imagery," *Sensors*, vol. 23, no. 15, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/15/6879

[13] B. Alvey, D. Anderson, and J. M. Keller, "Explainable ai via linguistic summarization of black box computer vision models," *2023 IEEE Conference on Artificial Intelligence (CAI)*, pp. 357–358, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:260386165

[14] B. J. Alvey, D. T. Anderson, C. Yang, A. Buck, J. M. Keller, K. E. Yasuda, and H. A. Ryan, "Characterization of deep learning-based aerial explosive hazard detection using simulated data," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021, pp. 1–8.

[15] B. Alvey, D. T. Anderson, and J. Keller, "Linguistic comparisons of black box models," in *FUZZ-IEEE*, 2024.

[16] J. Kerley, D. T. Anderson, B. Alvey, and A. Buck, "Generating simulated data with a large language model," in *SPIE*, 2024.

[17] ——, "How should simulated data be collected for AI/ML and unmanned aerial vehicles?" in *Synthetic Data for Artificial Intelligence and Machine Learning: Tools, Techniques, and Applications*, C. L. Howell, K. E. Manser, and R. M. Rao, Eds., vol. 12529, International Society for Optics and Photonics. SPIE, 2023, p. 125290J. [Online]. Available: https://doi.org/10.1117/12.2663717

[18] J. Kerley, A. Fuller, M. Kovaleski, P. Popescu, B. Alvey, D. T. Anderson, A. Buck, J. M. Keller, G. Scott, C. Yang, K. E. Yasuda, and H. A. Ryan, "Procedurally generated simulated datasets for aerial explosive hazard detection," in *Chemical, Biological, Radiological, Nuclear, and Explosives (CBRNE) Sensing XXIII*, J. A. Guicheteau and C. R. Howle, Eds., vol. 12116, International Society for Optics and Photonics. SPIE, 2022, p. 1211611. [Online]. Available: https://doi.org/10.1117/12.2618798

[19] B. Alvey, D. T. Anderson, A. Buck, M. Deardorff, G. Scott, and J. M. Keller, "Simulated photorealistic deep learning framework and workflows to accelerate computer vision and unmanned aerial vehicle research," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021, pp. 3882–3891.

[20] A. Buck, J. Akers, D. T. Anderson, J. Keller, R. Camaioni, M. Deardorff, and R. Luke, "Frame selection strategies for real-time structure-from-motion from an aerial platform," in *AIPR*, 2023.

[21] J. Akers, A. Buck, D. T. Anderson, J. Keller, R. Camaioni, M. Deardorff, and R. Luke, "Improving real-time aerial 3d reconstruction: Towards fusion of a hand-crafted sfm algorithm with a data-driven deep neural network," in *AIPR*, 2023.

[22] J. Akers, A. Buck, R. Camaioni, D. T. Anderson, R. H. Luke, J. M. Keller, M. Deardorff, and B. Alvey, "Simulated gold-standard for quantitative evaluation of monocular vision algorithms," in *SPIE Defense + Commercial Sensing*, 2023.

[23] T. Kucukpinar, D. Kavzak, J. Fraser, A. Buck, D. T. Anderson, and K. Palaniappan, "Deep learning-based shadow detection in aerial images using synthetically generated scenes," in *SPIE*, 2024.

[24] D. Buffum, A. Buck, J. Akers, and D. Anderson, "Autonomous drone behavior via mcdm of ufomap layers," in *SPIE*, 2024.

[25] A. Buck, R. Camaioni, B. Alvey, D. T. Anderson, J. M. Keller, R. H. L. III, and G. Scott, "Unreal engine-based photorealistic aerial data generation and unit testing of artificial intelligence algorithms," in *Geospatial Informatics XII*, K. Palaniappan, G. Seetharaman, and J. D. Harguess, Eds., vol. 12099, International Society for Optics and Photonics. SPIE, 2022, p. 1209908. [Online]. Available: https://doi.org/10.1117/12.2618812

[26] B. Alvey, J. Kerley, V. Escobar, D. T. Anderson, C. Yang, M. Donnelly, B. Libbey, S. Hockey, B. Burns, R. Lydic, J. Williams, M. Aeillo, and B. Barlow, "Advancing above ground explosive hazard detection: Simulated dataset generation for low altitude aerial drones," in *MSS*, 2024.

[27] M. Kovaleski, A. Fuller, J. Kerley, B. J. Alvey, P. Popescu, D. Anderson, A. Buck, J. Keller, G. Scott, C. Yang, K. E. Yasuda, and H. A. Ryan, "Explosive hazard pre-screener based on simulated data with perfect annotation and imprecisely labeled real data," in *Chemical, Biological, Radiological, Nuclear, and Explosives (CBRNE) Sensing XXIII*, J. A. Guicheteau and C. R. Howle, Eds., vol. 12116, International Society for Optics and Photonics. SPIE, 2022, p. 121160X. [Online]. Available: https://doi.org/10.1117/12.2618792

[28] A. Buck, P. Popescu, D. T. Anderson, J. Keller, M. Talbott, K. Manser, M. Jeiran, B. Bahhur, and B. Vogel, "Leveraging a digital twin to train, evaluate, and understand single image depth estimation for infrared imaging," in *MSS*, 2024.

[29] A. Buck, J. Kerley, D. T. Anderson, and J. Keller, "Simulated data to train and evaluate deep learning-based passive monocular vision algorithms at medium to long ranges," in *MSS*, 2023.