CARLA-AIR: Fly Drones Inside a CARLA World
A Unified Infrastructure for Air-Ground Embodied Intelligence

Tianle Zeng[1] Yanci Wen[1] Hong Zhang[1,†]

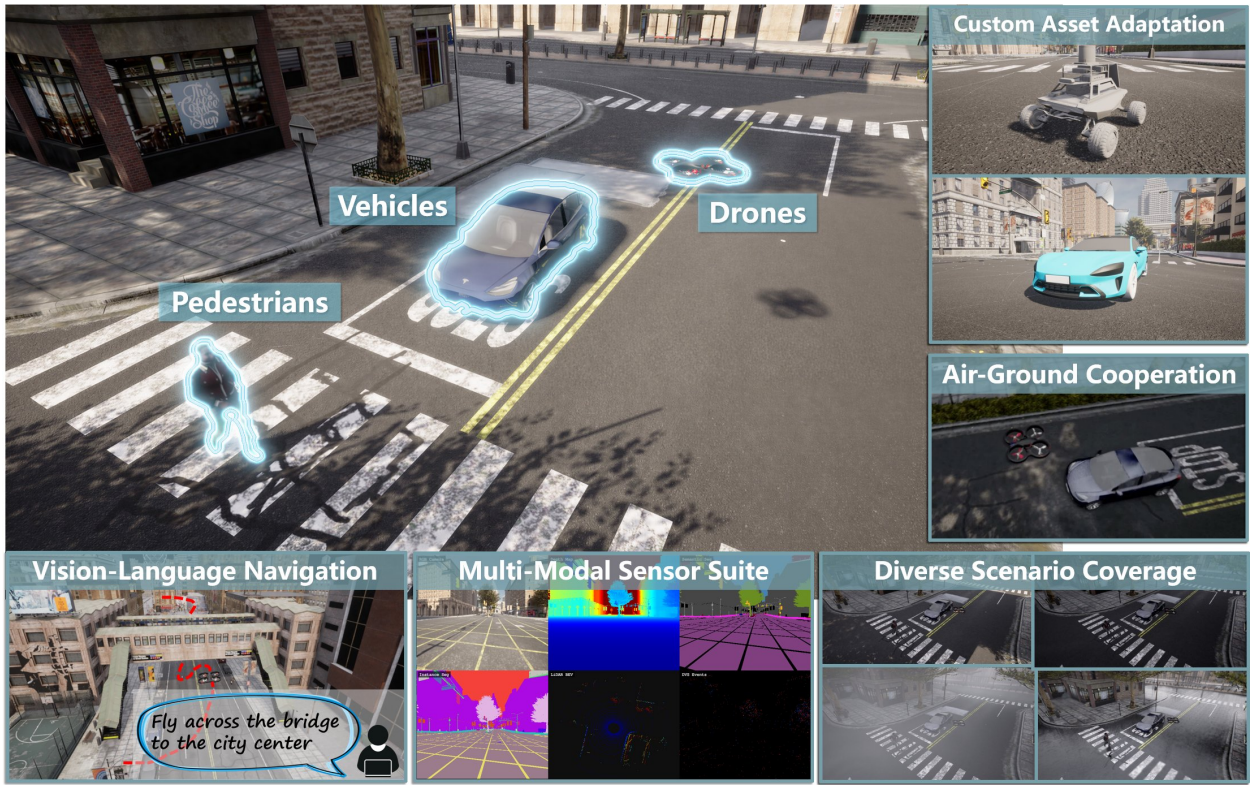

Figure 1: Overview of **CARLA-Air**, a unified simulation infrastructure for air-ground embodied intelligence. The examples shown here illustrate representative capabilities of the platform, including unified air-ground simulation, multi-modal sensing, embodied navigation, asset adaptation, and diverse urban scenarios within a single physically coherent environment.

Abstract

The convergence of low-altitude economies, embodied intelligence, and air-ground cooperative systems creates a growing need for simulation infrastructure capable of jointly modeling aerial and ground agents within a single physically coherent environment. Existing open-source platforms remain domain-segregated: urban driving simulators provide rich traffic populations but no aerial dynamics, while multirotor simulators offer physics-accurate flight but lack realistic ground scenes. Bridge-based co-simulation can connect heterogeneous backends, yet introduces synchronization overhead and cannot guarantee the strict spatial-temporal consistency required by modern perception and learning pipelines.

We present **CARLA-Air**, an open-source infrastructure that unifies high-fidelity urban driving and physics-accurate multirotor flight within a single Unreal Engine process, providing a practical simulation foundation for air-ground embodied intelligence research. The platform preserves both CARLA and AirSim native Python APIs and ROS 2 interfaces, enabling zero-modification reuse of exist-

†Corresponding author (hzhang@sustech.edu.cn).
[1]Shenzhen Key Laboratory of Robotics and Computer Vision, Southern University of Science and Technology.

For questions and technical inquiries, please contact louiszeng16@163.com.

ing codebases. Within a shared physics tick and rendering pipeline, CARLA-Air delivers photorealistic urban and natural environments populated with rule-compliant traffic flow, socially-aware pedestrians, and aerodynamically consistent UAV dynamics, while synchronously capturing up to 18 sensor modalities—including RGB, depth, semantic segmentation, LiDAR, radar, IMU, GNSS, and barometry—across all aerial and ground platforms at each simulation tick. Building on this foundation, the platform provides out-of-the-box support for representative air-ground embodied intelligence workloads, spanning air-ground cooperation, embodied navigation and vision-language action, multi-modal perception and dataset construction, and reinforcement-learning-based policy training. An extensible asset pipeline further allows researchers to integrate custom robot platforms, UAV configurations, and environment maps into the shared simulation world. By inheriting and extending the aerial simulation capabilities of AirSim—whose upstream development has been archived—CARLA-Air also ensures that this widely adopted flight stack continues to evolve within a modern, actively maintained infrastructure.

CARLA-Air is released with both prebuilt binaries and full source code to support immediate adoption: https://github.com/louiszengCN/CarlaAir

1 Introduction

Three converging frontiers are reshaping autonomous systems research. The *low-altitude economy* demands scalable infrastructure for urban air mobility, drone logistics, and aerial inspection. *Embodied intelligence* requires agents that perceive and act in shared physical environments through vision, language, and control. *Air-ground cooperative systems* bring these threads together, calling for heterogeneous robots that operate jointly across aerial and ground domains. Simulation is essential for advancing all three frontiers, as real-world deployment is costly, safety-critical, and difficult to scale. Yet no widely adopted open-source platform provides a unified infrastructure capable of jointly modeling aerial and ground agents within a single physically coherent environment. CARLA-Air is designed to fill this gap.

The simulation landscape and its gap. Existing open-source simulators address complementary domains without overlap. CARLA [2], built on Unreal Engine 4 [3], has become the de facto standard for urban autonomous driving research, offering photorealistic environments, rich traffic populations, and a mature Python API. AirSim [15], also built on UE4, provides physics-accurate multirotor simulation with high-frequency dynamics and a comprehensive aerial sensor suite. The limitation of each platform is precisely the strength of the other: CARLA lacks aerial agents, while AirSim lacks realistic ground traffic and pedestrian interactions. Meanwhile, AirSim's upstream development has been archived by its original maintainers, leaving a widely adopted flight simulation stack without an active evolution path. Other simulators across driving, UAV, and embodied AI domains similarly remain confined to a single agent modality (see Section 2 for a comprehensive survey). As a result, emerging workloads that span both air and ground domains—air-ground cooperation, cross-domain embodied navigation, joint multi-modal data collection, and cooperative reinforcement learning—lack a shared simulation foundation.

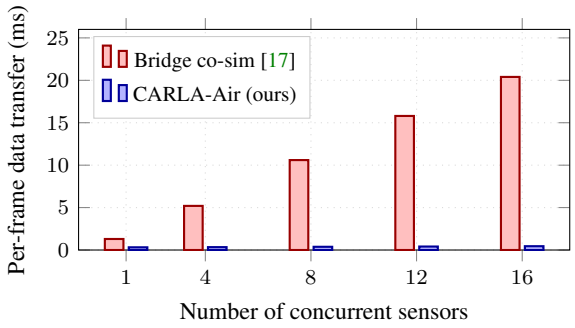

Figure 2: Per-frame inter-process data transfer time as a function of concurrent sensor count. Bridge-based co-simulation [17] exhibits near-linear growth with sensor count due to cross-process serialization, while CARLA-Air remains effectively constant (< 0.5 ms) owing to its single-process architecture.

Why not bridge-based co-simulation? A common workaround connects heterogeneous simulators through bridge-based co-simulation, typically via ROS 2 [9] or custom message-passing interfaces. While functionally viable, such approaches introduce inter-process synchronization complexity, communication overhead, and duplicated rendering pipelines. More critically, independent simulation processes cannot guarantee strict spatial-temporal consistency across sensor streams—a requirement for perception, learning, and evaluation in embodied intelligence systems. Fig. 2 quantifies the per-frame inter-process overhead contrast between bridge-based co-simulation and the single-process design adopted by CARLA-Air.

CARLA-Air: a unified infrastructure for air-ground embodied intelligence. We present **CARLA-Air**, an open-source platform that integrates CARLA and AirSim within a single Unreal Engine process, purpose-built as a practical simulation foundation for air-ground embodied intelligence research. By inheriting and extending AirSim's aerial simulation capabilities within a modern,

actively maintained infrastructure, CARLA-Air also provides a sustainable evolution path for the large body of existing AirSim-based research. Key capabilities of the platform include:

(i) **Single-process air-ground integration.** CARLA-Air resolves a fundamental engine-level conflict—UE4 permits only one active game mode per world—through a composition-based design that inherits all ground simulation subsystems from CARLA while spawning AirSim's aerial flight actor as a regular world entity. This yields a shared physics tick, a shared rendering pipeline, and strict spatial-temporal consistency across all sensor viewpoints.

(ii) **Full API compatibility and zero-modification code migration.** Both CARLA and AirSim native Python APIs and ROS 2 interfaces are fully preserved, allowing existing research codebases to run on CARLA-Air without modification.

(iii) **Photorealistic, physically coherent simulation world.** The platform delivers rich urban and natural environments populated with rule-compliant traffic flow, socially-aware pedestrians, and aerodynamically consistent multirotor dynamics, with synchronized capture of up to 18 sensor modalities across all aerial and ground platforms at each simulation tick.

(iv) **Extensible asset pipeline.** Researchers can import custom robot platforms, UAV configurations, vehicles, and environment maps into the shared simulation world, enabling flexible construction of diverse air-ground interaction scenarios.

Building on these capabilities, CARLA-Air provides out-of-the-box support for representative air-ground embodied intelligence workloads across four research directions:

(a) **Air-ground cooperation**—heterogeneous aerial and ground agents coordinate within a shared environment for tasks such as cooperative surveillance, escort, and search-and-rescue.

(b) **Embodied navigation and vision-language action**—agents navigate and act grounded in visual and linguistic input, leveraging both aerial overview and ground-level detail.

(c) **Multi-modal perception and dataset construction**—synchronized aerial-ground sensor streams are collected at scale to build paired datasets for cross-view perception, 3D reconstruction, and scene understanding.

(d) **Reinforcement-learning-based policy training**—agents learn cooperative or individual policies through closed-loop interaction in physically consistent air-ground environments.

As a lightweight and practical infrastructure, CARLA-Air lowers the barrier for developing and evaluating air-ground embodied intelligence systems, and provides a unified simulation foundation for emerging applications in low-altitude robotics, cross-domain autonomy, and large-scale embodied AI research.

2 Related Work

Simulation platforms relevant to autonomous systems span autonomous driving, aerial robotics, joint co-simulation, and embodied AI. From the perspective of air-ground embodied intelligence, the central question is not whether a platform supports driving or flight in isolation, but whether aerial and ground agents can be jointly simulated within a unified, physically coherent, and practically usable environment. As illustrated in Fig. 3 and Table 1, existing open-source platforms largely remain separated by domain focus, and none simultaneously provides realistic urban traffic, socially-aware pedestrians, physics-based multirotor flight, preserved native APIs, and single-process execution in one shared simulation world.

2.1 Autonomous Driving Simulators

Autonomous driving simulators provide strong support for realistic urban scenes, traffic agents, and ground-vehicle perception. CARLA [2], built on Unreal Engine [3], has become the de facto open-source platform for urban driving research due to its photorealistic environments, rich actor library, and mature Python API. LGSVL [13] offers full-stack integration with Autoware and Apollo on the Unity engine. SUMO [8] provides lightweight microscopic traffic flow modeling. MetaDrive [7] enables procedural environment generation for generalizable RL, and VISTA [1] supports data-driven sensor-view synthesis for autonomous vehicles. These platforms collectively cover a broad range of ground-autonomy research needs, but none natively supports physics-based UAV flight, leaving air-ground cooperative workloads outside their scope.

2.2 Aerial Vehicle Simulators

Aerial simulators provide the complementary capability: accurate multirotor dynamics, onboard aerial sensing, and UAV-oriented control interfaces. AirSim [15] remains one of the most widely adopted open-source UAV simulators, offering physics-accurate multirotor flight and a comprehensive sensor suite on Unreal Engine, though its

upstream development has since been archived. Flightmare [16] combines Unity-based photorealistic rendering with highly parallel dynamics for fast RL training. FlightGoggles [5] provides photogrammetry-based environments for perception-driven aerial robotics. Gazebo [6], together with MAV-specific packages such as RotorS [4], offers a mature ROS-integrated simulation stack for multi-rotor control and state estimation. OmniDrones [19] and gym-pybullet-drones [12] target scalable, GPU-accelerated or lightweight RL-oriented multi-agent UAV training. While these systems are well suited to aerial autonomy in isolation, they generally lack realistic urban traffic populations, pedestrian interactions, and richly populated ground environments, limiting their use as infrastructure for air-ground cooperation or cross-domain data collection.

2.3 Joint and Co-Simulation Platforms

The most relevant prior efforts attempt to combine aerial and ground simulation through co-simulation. TranSimHub [17] connects CARLA with SUMO and aerial agents via a multi-process architecture supporting synchronized multi-view rendering. Other representative approaches include ROS-based pairings of AirSim with Gazebo [15, 6, 9]. These systems demonstrate that heterogeneous simulation backends can be functionally connected, but their integration typically depends on bridges, RPC layers, or message-passing middleware across independent processes. As summarized in Table 2, such designs do not preserve a single rendering pipeline, do not provide strict shared-tick execution, and often require adapting existing code to new interfaces. By contrast, CARLA-Air integrates both simulation backends within a single Unreal Engine process, preserving both native APIs while maintaining a shared world state, shared renderer, and synchronized sensing—a system-level distinction detailed in Section 3.

2.4 Embodied AI and Robot Learning Platforms

Embodied AI platforms prioritize a different design objective: scalable policy training rather than realistic urban air-ground infrastructure. Isaac Lab [11] and Isaac Gym [10] emphasize massively parallel GPU-accelerated reinforcement learning for locomotion and manipulation. Habitat [14] and SAPIEN [18] target indoor navigation and articulated object interaction, while RoboSuite [20] focuses on tabletop manipulation benchmarks. These platforms are valuable for embodied intelligence research, but they do not provide the urban traffic realism, socially-aware pedestrian populations, or integrated aerial-ground simulation required by low-altitude cooperative robotics. In this sense, they address complementary research needs and are not direct substitutes for CARLA-Air.

2.5 Summary

Fig. 3 positions CARLA-Air and representative platforms along two principal design axes: simulation fidelity and agent domain breadth. Driving simulators provide realistic urban ground environments without aerial dynamics; UAV simulators provide flight realism without populated ground worlds; joint simulators generally rely on multi-process bridging that sacrifices interface compatibility or synchronization fidelity; and embodied AI platforms focus on scalable learning rather than air-ground infrastructure. CARLA-Air is designed to sit at the intersection of these domains, combining realistic urban traffic, socially-aware pedestrians, physics-based multirotor flight, preserved native APIs, and single-process execution within one unified simulation environment. Table 1 provides a detailed feature-level comparison across all platforms discussed above.

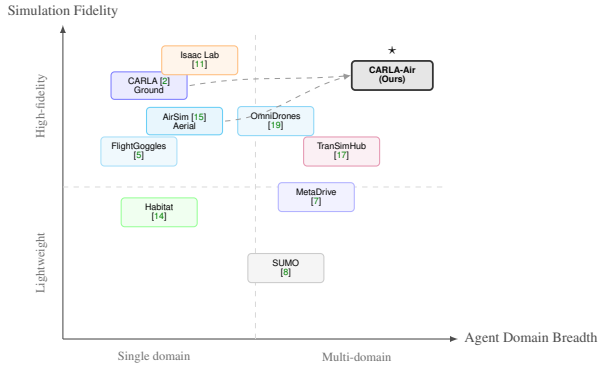

Figure 3: Platform positioning along simulation fidelity and agent domain breadth. CARLA-Air (⋆) occupies the high-fidelity, multi-domain quadrant without inter-process bridging. Dashed arrows indicate subsumption of upstream capabilities.

3 System Architecture

CARLA-Air integrates CARLA [2] and AirSim [15] within a single Unreal Engine [3] process through a minimal bridging layer that resolves a fundamental engine-level initialization conflict while preserving both platforms' native APIs, physics engines, and rendering pipelines intact. Fig. 4 presents the high-level runtime structure; the following subsections elaborate each design decision.

Table 1: Consolidated comparison of representative open-source simulation platforms for air-ground embodied intelligence research. ✓ = supported; ∼ = partial or constrained; × = not supported; — = not applicable.

Platform	Urban Traffic	Pedes-trians	UAV Flight	Single Process	Shared Renderer	Native APIs	Joint Sensors	Prebuilt Binary	Test Suite	Custom Assets	Open Source
Autonomous Driving											
CARLA [2]	✓	✓	×	✓	✓	✓	×	✓	×	✓	✓
LGSVL [13]	✓	✓	×	✓	✓	✓	×	✓	×	∼	✓
SUMO [8]	✓	✓	×	✓	×	✓	×	✓	×	×	✓
MetaDrive [7]	✓	∼	×	✓	∼	✓	×	×	×	×	✓
VISTA [1]	∼	×	×	✓	∼	✓	×	×	×	×	✓
Aerial / UAV											
AirSim [15]	×	×	✓	✓	✓	✓	×	✓	×	✓	✓
Flightmare [16]	×	×	✓	✓	✓	✓	×	×	×	∼	✓
FlightGoggles [5]	×	×	✓	✓	✓	✓	×	×	×	∼	✓
Gazebo / RotorS [6, 4]	∼	∼	✓	✓	∼	✓	×	✓	×	✓	✓
OmniDrones [19]	×	×	✓	✓	✓	✓	×	×	×	✓	✓
gym-pybullet-drones [12]	×	×	✓	✓	∼	✓	×	×	×	∼	✓
Joint / Co-Simulation											
TranSimHub [17]	✓	✓	✓	×	×	×	∼	—	×	×	✓
CARLA+SUMO [2, 8]	✓	✓	×	×	×	∼	×	—	×	×	✓
AirSim+Gazebo [15, 6, 9]	∼	∼	✓	×	×	∼	∼	—	×	∼	✓
Embodied AI & RL											
Isaac Lab [11]	×	×	∼	✓	✓	✓	×	×	✓	✓	✓
Isaac Gym [10]	×	×	∼	✓	✓	✓	×	×	×	×	✓
Habitat [14]	×	∼	×	✓	✓	✓	×	×	×	×	✓
SAPIEN [18]	×	×	×	✓	✓	✓	×	×	×	∼	✓
RoboSuite [20]	×	×	×	✓	✓	✓	×	×	×	∼	✓
CARLA-Air (Ours)	✓	✓†	✓	✓	✓	✓	✓	✓	✓	✓	✓

†Pedestrian AI is inherited from CARLA and fully functional; behavior under high actor density in joint scenarios is an active engineering target (Section 6).

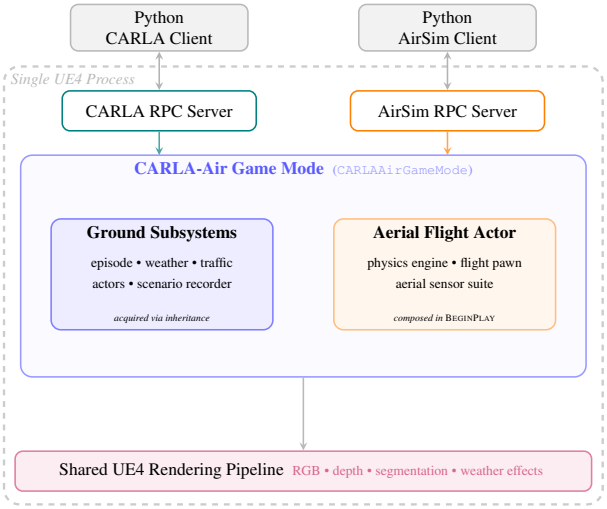

Figure 4: Runtime architecture of CARLA-Air. A single engine process hosts both simulation backends, each communicating with its respective Python client via an independent RPC server. `CARLAAirGameMode` acquires ground simulation functionality through class inheritance and integrates the aerial flight actor through composition. All world actors share a single rendering pipeline.

Table 2: Architectural comparison of representative joint and co-simulation platforms.[a]

Platform	Single Proc.	API Kept	Shared Render	Sync Mode	Open Source
TranSimHub [17]	×	×	×	Msg.	✓
CARLA+SUMO [2, 8]	×	∼	×	RPC	✓
AirSim+Gazebo [15, 6, 9]	×	∼	×	Decpl.	✓
AirSim+CARLA[b]	×	×	×	Decpl.	✓
CARLA-Air (Ours)	✓	✓	✓	Shared	✓

[a]Single Proc. = single-process execution; API Kept = preservation of native simulator APIs; Shared Render = shared rendering pipeline; Sync Mode: Msg. = message passing, Decpl. = decoupled execution, Shared = shared-tick within one process.
[b]Refers to community bridge solutions that run AirSim and CARLA as independent processes connected via ROS 2 or custom middleware; not an official product of either project.

3.1 Plugin and Dependency Structure

The system comprises two plugin modules that load sequentially during engine startup. The ground simulation plugin initializes first, establishing its world management subsystems before any game logic executes. The aerial simulation plugin declares a compile-time dependency on the ground plugin, enabling `CARLAAirGameMode` to access the ground platform's initialization interfaces during its own startup phase. This dependency is strictly one-directional: no ground-platform source file references any

aerial component, preserving the upstream CARLA codebase's update path without modification. Two independent RPC servers run concurrently within the single process—one per simulator—allowing the native Python clients of each platform to connect without modification.

Version compatibility across the two upstream codebases, network configuration, and port assignments are documented in Appendix A.1.

3.2 The GameMode Conflict and Its Resolution

UE4 enforces a strict invariant: each world may have exactly one active game mode. CARLA's game mode orchestrates episode management, weather control, traffic simulation, the actor lifecycle, and the RPC interface through a deep inheritance chain. AirSim's game mode performs a separate startup sequence—reading configuration files, adjusting rendering settings, and spawning its flight actor. Because the two game modes are unrelated by inheritance, assigning either to the world map silently skips the other's initialization, rendering a large portion of its API surface inoperative.

Architectural asymmetry. A structural difference between the two systems makes resolution tractable and constitutes the central design insight of CARLA-Air. CARLA's subsystems are tightly coupled to its game mode through inheritance and privileged class relationships; they cannot be relocated outside the game mode slot without invasive upstream refactoring. AirSim's flight logic, by contrast, resides in a class derived from the generic *actor* base—not the game mode base—and can therefore be spawned as a regular world actor at any point after world initialization.

Solution: single inheritance plus composition. We introduce `CARLAAirGameMode`, which inherits from CARLA's game mode base and occupies the single available slot. All ground simulation subsystems are thereby acquired through the standard UE4 lifecycle. The aerial flight actor is then *composed* into the world during the engine's BEGINPLAY phase, after ground initialization is complete, and never competes for the game mode slot. Fig. 5 contrasts the naive conflict with this adopted solution.

Source footprint. The integration modifies exactly two files in the upstream CARLA source tree: two previously private members are promoted to protected visibility and one privileged class declaration is added. All remaining integration code resides within the aerial plugin as purely additive content. The complete modification summary is provided in Appendix A.2.

3.3 Coordinate System Mapping

CARLA and AirSim employ incompatible spatial reference frames that must be reconciled to co-register aerial and ground sensor data. CARLA inherits UE4's left-handed system with X forward, Y right, and Z up, in centimeters. AirSim adopts a right-handed North-East-Down (NED) frame with X north, Y east, and Z down, in meters. Fig. 6 illustrates both frames and their geometric relationship.

Let $\mathbf{p} \in \mathbb{R}^3$ denote a point in the UE4 world frame and $\mathbf{o}$ the shared world origin established during initialization. The equivalent NED position is

$$\mathbf{p}_{\text{NED}} = \frac{1}{100} \begin{pmatrix} p_x - o_x \\ p_y - o_y \\ -(p_z - o_z) \end{pmatrix}, \tag{1}$$

where the scale factor converts centimeters to meters and the sign reversal on the third component reflects the Z-axis inversion. Because the X and Y axes are directionally aligned, no axis permutation is required.

For orientation, let $q = (w, q_x, q_y, q_z)$ denote a unit quaternion in the UE4 frame. The equivalent NED quaternion is

$$q_{\text{NED}} = \big(w,\ q_x,\ q_y,\ -q_z\big), \tag{2}$$

where negating q_z accounts for the Z-axis reversal and the associated change of frame handedness. Eqs. (1) and (2) together fully specify the pose transform, enabling consistent fusion of drone attitude from the aerial API with vehicle heading from the ground API across all joint simulation workflows.

3.4 Asset Import Pipeline

CARLA-Air provides an extensible asset import pipeline that allows researchers to bring custom robot platforms, UAV models, vehicles, and environment assets into the shared simulation world. Imported assets are fully integrated into the joint simulation environment: they participate in the same physics tick and rendering pass as all built-in actors, respond to both ground and aerial API calls, and are visible to all sensor modalities across both simulation backends. This capability enables evaluation of custom hardware designs—such as novel multirotor configurations or application-specific ground robots—within realistic air-ground scenarios without modifying the core CARLA-Air codebase. Fig. 14 shows two examples of user-imported assets operating within the platform.

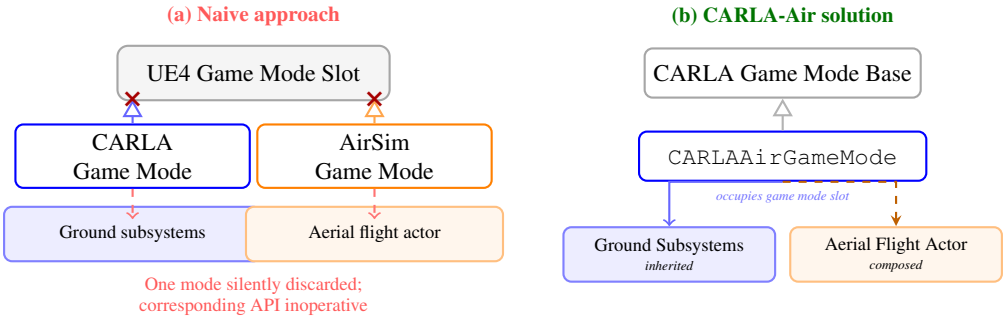

Figure 5: Resolving the UE4 single-game-mode constraint. (a) Both backends provide independent game mode classes; assigning either silently discards the other. (b) `CARLAAirGameMode` inherits all ground functionality from CARLA's game mode base while composing the aerial flight actor as a spawned world actor.

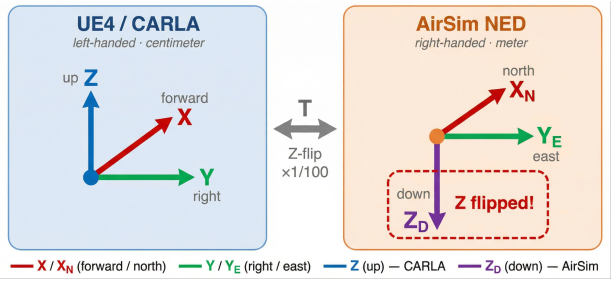

Figure 6: Coordinate frames of the two simulation backends. The transform T requires only a Z-axis sign flip and a centimeter-to-meter scale factor; the forward (X/X_N) and rightward (Y/Y_E) axes are aligned across both conventions.

4 Performance Evaluation

This section evaluates CARLA-Air under representative joint air-ground workloads across three experiments: frame-rate and resource scaling (Section 4.2), memory stability under sustained operation (Section 4.3), and communication latency (Section 4.4). Full configuration parameters and raw data are deferred to Appendix A.1.

Reference hardware. All measurements are collected on a workstation equipped with an NVIDIA RTX A4000 (16 GB GDDR6), AMD Ryzen 7 5800X (8-core, 4.7 GHz), and 32 GB DDR4-3200, running Ubuntu 20.04 LTS. The simulator runs in Epic quality mode with Town10HD loaded unless stated otherwise. All aerial experiments use the built-in SimpleFlight controller with default PID gains. CPU affinity and GPU power limits are left at system defaults to reflect realistic research deployment conditions.

4.1 Benchmark Methodology

Reliable performance measurement requires eliminating startup transients—map-loading jitter, first-frame shader compilation, and actor lifecycle initialization must be discarded before steady-state sampling begins. Algorithm 1 formalizes the benchmark harness used throughout this section.

Algorithm 1 Simulation Performance Benchmark Harness

Require: Workload $\mathcal{W}$, warm-up ticks T_w, measurement ticks T_m
Ensure: Frame-time $\mathbf{f}$, VRAM $\mathbf{v}$, latency $\boldsymbol{\ell}$

```
Connect to simulation; load map; wait for shader compilation
Spawn actors/sensors per W; enable synchronous mode
for t = 1 to T_w do                          ▷ Warm-up—discarded
    Advance one tick
end for
f, v, ℓ ← ⟨⟩
for t = 1 to T_m do
    t_0 ← NOW(); advance one tick
    f.APPEND(1/(NOW() − t_0))
    if VRAM sample interval reached then
        Record GPU memory → v; measure API round-trip → ℓ
    end if
end for
Destroy all actors; disable synchronous mode
return f, v, ℓ
```

Each profile uses $T_w = 200$ warm-up ticks and $T_m = 2{,}000$ measurement ticks. VRAM is sampled every 60 s. All reported frame rates are the harmonic mean of $\mathbf{f}$—the appropriate central tendency for rate quantities—with standard deviations alongside. The latency benchmark issues 500 warm-up calls followed by 5 000 measurement calls; actor spawn calls are each paired with an immediate destroy to prevent scene-state accumulation from contaminating subsequent measurements.

4.2 Experiment 1: Frame Rate and Resource Scaling

Workload design. Under synchronous-mode operation, per-tick wall time is bounded by the slowest of three concurrent contributors: the rendering thread (GPU-bound), the ground-actor dispatch loop (CPU-bound), and the aerial physics engine (CPU-bound, asynchronous at ≈1,000 Hz on a dedicated thread). Sensor rendering dominates at high resolution due to GPU memory bandwidth saturation; actor population contributes via per-mesh draw calls. These relationships motivate the workload stratification in Table 3.

Analysis. The moderate joint configuration—combining ground traffic, an aerial agent, and a full sensor suite—sustains 19.8 ± 1.1 FPS, sufficient for closed-loop policy evaluation at standard RL episode lengths. Comparing the standalone ground baseline (28.4 FPS) against the moderate joint profile (19.8 FPS) quantifies total integration overhead at 8.6 FPS (30.3%), of which 2.1 FPS is attributable to ground co-hosting and the remaining 6.5 FPS to the aerial physics engine. Crucially, this aerial overhead manifests entirely in CPU utilization (54% vs. 38%) rather than GPU memory: VRAM differs by only 39 MiB between ground-only and moderate-joint profiles. The traffic surveillance profile retains comparable throughput (20.1 FPS) despite doubling the vehicle count, confirming that sensor rendering—not actor population—is the dominant cost driver at 1920×1080. The 3-hour endurance profile (19.7 ± 1.3 FPS) confirms that throughput does not degrade under sustained operation.

4.3 Experiment 2: Memory Stability

Steady-state VRAM profile. The base map asset load accounts for approximately 3 702 MiB at idle (Town10HD, Epic quality); each additional ground sensor contributes 4–10 MiB at 1280×720. A transient peak of approximately 5 000 MiB occurs during map loading and resolves within 30 s. All steady-state profiles remain within 3 878 MiB, retaining approximately 12 506 MiB (76%) of the 16 GB device budget for co-located workloads such as GPU-based policy training.

Leak verification. Table 4 summarises the 3-hour endurance run across 357 actor spawn-and-destroy cycles. A linear regression of VRAM against cycle index yields a slope of 0.49 MiB/cycle with $R^2 = 0.11$, indicating no statistically significant accumulation trend. Fig. 7 visualises the VRAM trace: the negligible early-to-late drift (3,868 → 3,878 MiB, approximately 10 MiB over 327 cycles) is attributable to residual render-target caching rather than lifecycle leakage.

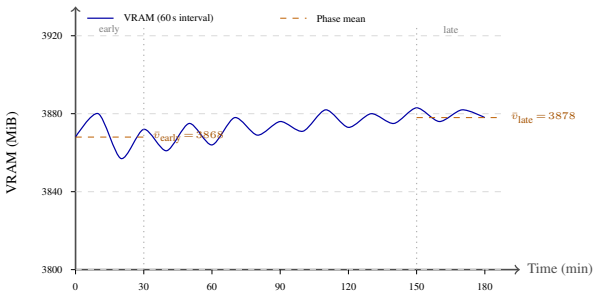

Figure 7: VRAM trace over a 3-hour stability run (357 spawn/destroy cycles, moderate joint configuration, RTX A4000). Early-to-late drift is ≈10 MiB; linear regression yields $R^2 = 0.11$, confirming no significant memory accumulation.

Zero API errors and zero simulation crashes across 357 cycles validate the robustness of the joint environment under the repeated agent-reset patterns typical of reinforcement-learning training.

4.4 Experiment 3: Communication Latency

Both simulation APIs operate within the same process on the loopback interface, eliminating inter-process serialization overhead. The two RPC servers use distinct wire protocols and port assignments (detailed in Appendix A.1). Table 5 reports round-trip latency for representative API calls.

Analysis. Lightweight state queries complete in 280–490 μs, well below the per-tick budget at 20 FPS (50 ms), confirming that API overhead does not contribute meaningfully to frame-time variance. Actor spawn latency (1 850 μs) reflects a one-time GPU synchronization point for render-asset registration at episode reset. Image capture latency (3 200 μs) covers the full sensor pipeline—rendering, buffer readback, and serialization—and is overlapped with the rendering thread at synchronous tick rates. All measured values fall below the lower bound of bridge-based cross-process synchronization costs reported in [17] (1 000–5 000 μs per frame).

Tick-rate reconciliation. The aerial physics engine advances at ≈1,000 Hz on its dedicated thread, while the rendering tick runs at ≈20 Hz under the moderate joint workload. Sensor callbacks read the aerial state at rendering tick boundaries, so each ground-aerial sensor frame pair reflects the drone's integrated physical state over ≈50 aerial physics steps. Ground actor states are also resolved at tick boundaries, ensuring temporal co-registration across all sensor modalities within a single tick. Applications requiring finer aerial state resolution should reduce the fixed

Table 3: Frame rate and resource consumption across representative joint workloads (RTX A4000, Town10HD, Epic quality, synchronous mode). Harmonic mean ± 1 SD over 2 000 ticks after 200 warm-up.

Profile	Configuration	FPS	VRAM (MiB)	CPU (%)
Standalone baselines				
Ground sim only[s]	3 vehicles + 2 pedestrians; 8 sensors @ 1280×720	28.4 ± 1.2	$3{,}821 \pm 10$	31 ± 3
Aerial sim only[s]	1 drone; 8 sensors @ 1280×720	44.7 ± 2.1	$2{,}941 \pm 8$	29 ± 3
Joint workloads				
Idle	Town10HD; no actors; no sensors	60.0 ± 0.4	$3{,}702 \pm 8$	12 ± 2
Ground only	3 vehicles + 2 pedestrians; 8 sensors @ 1280×720	26.3 ± 1.4	$3{,}831 \pm 11$	38 ± 4
Moderate joint	3 vehicles + 2 pedestrians + 1 drone; 8 sensors @ 1280×720	19.8 ± 1.1	$3{,}870 \pm 13$	54 ± 5
Traffic surveillance	8 autopilot vehicles + 1 drone; 1 aerial RGB @ 1920×1080	20.1 ± 1.8	$3{,}874 \pm 15$	61 ± 6
Stability endurance	Moderate joint; 357 spawn/destroy cycles; 3 hr continuous	19.7 ± 1.3	$3{,}878 \pm 17$	55 ± 5

[s]Standalone baseline: single simulator running without CARLA-Air integration.

Table 4: Stability endurance results over 3 hours and 357 actor lifecycle cycles (moderate joint configuration).

Metric	Early (cycles 1–30)	Late (cycles 328–357)
Frame rate (FPS)	19.9 ± 1.2	19.7 ± 1.3
VRAM (MiB)	$3{,}868 \pm 14$	$3{,}878 \pm 17$
CPU utilization (%)	53 ± 5	55 ± 5
API error count	0	0
Crash count	0	0
VRAM regression slope	0.49 MiB/cycle, $R^2 = 0.11$	

simulation delta-time accordingly; the throughput trade-off follows from Table 3.

5 Representative Applications

CARLA-Air is validated through five representative workflows that collectively exercise the platform's core capabilities across the four research directions identified in Section 1: air-ground cooperation, embodied navigation and vision-language action, multi-modal perception and dataset construction, and reinforcement-learning-based policy training. Table 6 provides a structured overview; detailed descriptions follow in Sections 5.1–5.5.

All workflows share a common dual-client architecture: both API clients execute within the same Python process and operate on the same world state without inter-process communication. The following minimal pattern is common to every workflow:

```
ground = carla.Client('localhost', 2000)
aerial = airsim.MultirotorClient()
world  = ground.get_world()  # shared
aerial.enableApiControl(True)
```

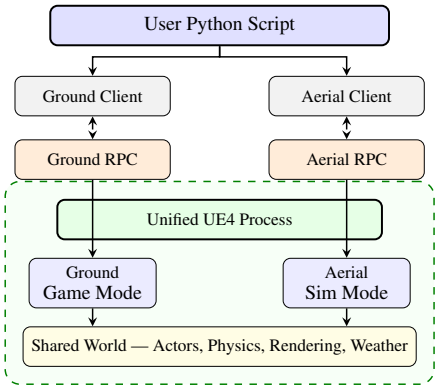

Figure 8: Dual-client architecture shared by all five workflows. A single Python script drives both clients, connected via TCP to independent RPC servers inside the unified engine process.

5.1 W1: Air-Ground Cooperative Precision Landing

Autonomous precision landing of a UAV on a moving ground vehicle is a representative and challenging scenario for air-ground cooperation, with direct applications in drone-assisted logistics, mobile recharging, and multi-agent coordination. This workflow demonstrates real-time cross-domain coordination within CARLA-Air's tick-synchronous control loop: the drone must continuously track the vehicle's position, plan a descent trajectory, and execute a smooth touchdown—all while the vehicle is in motion through urban traffic.

Setup. A ground vehicle follows an autopilot route through Town10HD at moderate speed. A drone starts at ≈12 m altitude above the vehicle and is tasked with landing on the vehicle's roof. At each synchronous tick, the vehicle's 3D pose is queried via the ground API, transformed into the aerial NED frame using the coordinate

Table 5: Round-trip API call latency on the loopback interface (median ± IQR; 5 000 calls after 500 warm-up; RTX A4000; idle scene).

API	Call	Median (μs)	IQR (μs)
Ground sim	World state snapshot	320	40
Ground sim	Actor transform query	280	35
Ground sim	Actor spawn (+ paired destroy)	1,850	210
Ground sim	Actor destroy	920	95
Aerial sim	Multirotor state query	410	55
Aerial sim	Image capture (1 RGB stream)	3,200	380
Aerial sim	Velocity command dispatch	490	60
Bridge IPC [17]	Cross-process state sync (ref.)	3,000	2,000

Table 6: Summary of five representative workflows validated on CARLA-Air, mapped to the four research directions from Section 1.

	Workflow	Research Direction	Platform Features Exercised	FPS	Key Outcome
W1	Precision landing	Air-ground cooperation	Tick-sync control, descent planning, cross-frame coordination	≈19	<0.5 m final landing error
W2	VLN/VLA data generation	Embodied navigation	Dual-view sensing, waypoint planning, semantic annotation	—	Cross-view VLN data pipeline
W3	Multi-modal dataset	Perception & dataset	12-stream sync capture, shared tick	≈17	≤1-tick alignment error
W4	Cross-view perception	Perception & dataset	Shared renderer, weather consistency	≈18	14/14 weather presets verified
W5	RL training env.	RL policy training	Sync stepping, stable resets, cross-domain reward	—	357 reset cycles, 0 crashes

mapping from Section 3.3, and used to compute a descent command issued to the aerial flight controller.

Control architecture. The landing controller operates in three phases: *approach* (horizontal alignment with the vehicle), *descent* (controlled altitude reduction while maintaining horizontal tracking), and *touchdown* (final landing). Let $\mathbf{q}_k \in \mathbb{R}^2$ denote the vehicle's horizontal position at tick k in the NED frame. The drone's commanded horizontal target tracks the vehicle continuously:

$$\hat{\mathbf{q}}_k = \mathbf{q}_k + \mathbf{d}, \qquad \hat{z}_k = z_k^{\mathrm{ref}}, \tag{3}$$

where $\mathbf{d}$ is the coordinate origin offset from Eq. (4) and z_k^{ref} decreases according to a smooth descent profile from the initial altitude to ground level.

Results. Fig. 9 shows the complete landing sequence. The drone descends from ≈12 m to touchdown over ≈20 s, with horizontal convergence error decreasing from ≈6 m to within the ±0.5 m tolerance band. The 3D trajectory overview confirms that the UAV trajectory converges smoothly onto the vehicle trajectory throughout the descent. Table 7 summarizes the landing performance.

Table 7: W1 cooperative precision landing results (Town10HD, RTX A4000).

Metric	Value	Notes
Mean FPS	19.3	Harmonic mean
Initial altitude	≈12 m	Start of descent
Landing duration	≈20 s	Approach to touchdown
Final horiz. error	<0.5 m	Within tolerance band
Initial horiz. error	≈6 m	At descent start
RPC errors	0	Both clients

5.2 W2: Embodied Navigation and VLN/VLA Data Generation

Vision-language navigation (VLN) and vision-language action (VLA) are among the most active research directions in embodied intelligence, requiring agents to navigate and act in realistic environments grounded in natural language instructions and visual observations. A key bottleneck is the availability of large-scale, diverse training data that pairs language instructions with rich visual observations from multiple viewpoints. CARLA-Air provides a natural

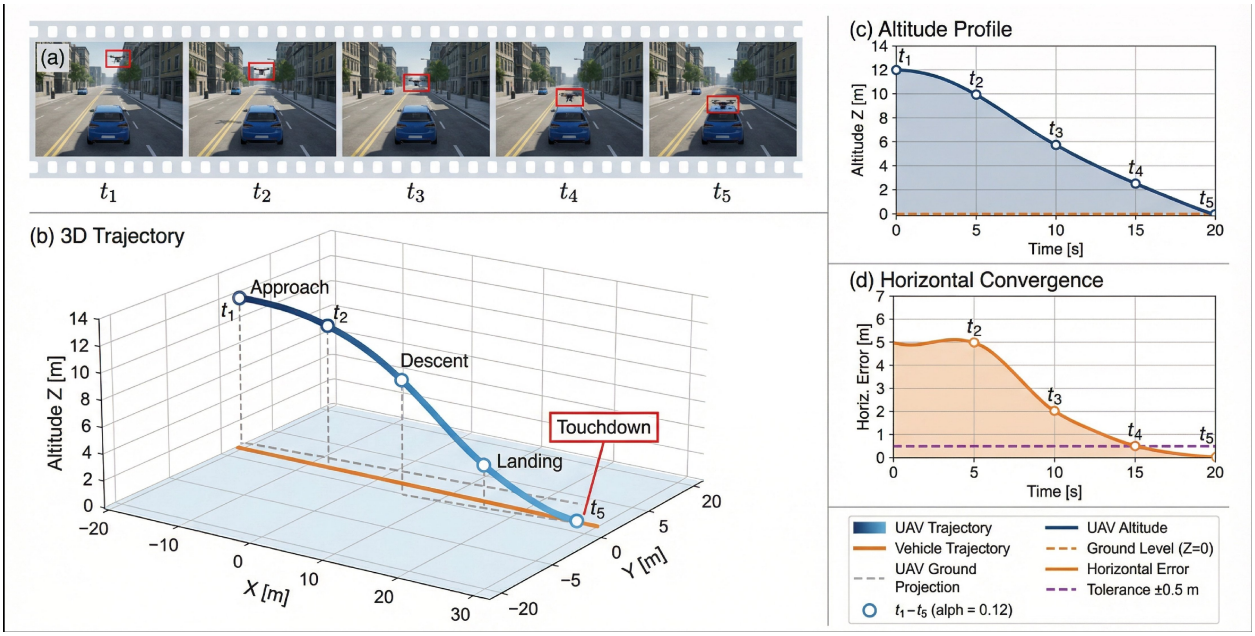

Figure 9: W1: Air-ground cooperative precision landing on a moving vehicle. **(a)** Time-lapse sequence (t_1–t_5) showing the drone (red box) descending toward and landing on a moving ground vehicle through approach, descent, and touchdown phases. **(b)** 3D trajectory overview: the UAV trajectory (blue) converges onto the vehicle trajectory (orange), with the ground projection (dashed) showing horizontal alignment. **(c)** Altitude profile over time, illustrating smooth descent from 12 m to ground level. **(d)** Horizontal convergence error, decreasing from ≈6 m to within the ±0.5 m tolerance band. All data are recorded within CARLA-Air's synchronous tick loop.

data generation infrastructure for this purpose: its photo-realistic urban environments, socially-aware pedestrians, dynamic traffic, and simultaneous aerial-ground sensing enable the construction of VLN/VLA datasets with cross-view visual grounding that single-domain platforms cannot provide.

Platform capabilities for VLN/VLA. CARLA-Air supports VLN/VLA data generation through several platform-level features: (i) both aerial and ground agents can be equipped with egocentric RGB, depth, and semantic segmentation cameras, providing paired bird's-eye and street-level visual observations along any navigation trajectory; (ii) the ground API exposes waypoint-based route planning with lane-level precision, which can serve as the basis for generating language-grounded navigation instructions; (iii) the shared rendering pipeline guarantees that all visual observations are captured under identical weather, lighting, and scene conditions at each tick; and (iv) the aerial overview provides a natural "oracle view" for generating spatial referring expressions and verifying navigation progress.

5.3 W3: Synchronized Multi-Modal Dataset Collection

Generating large-scale paired aerial-ground datasets is a bottleneck for training and evaluating cooperative perception models. Manual synchronization across separate simulator processes introduces alignment errors that corrupt correspondence annotations. CARLA-Air eliminates this bottleneck: because both sensor suites are driven by the same tick counter, the resulting dataset records carry guaranteed per-tick correspondence with zero interpolation overhead.

Setup. Eight ground sensors (RGB, semantic segmentation, depth, LiDAR, radar, GNSS, IMU, collision) and four aerial sensors (RGB, depth, IMU, GPS) are attached concurrently. The simulation runs in synchronous mode at a fixed timestep; all 12 sensor callbacks are registered before the first tick advance. Ground traffic is populated with 30 autopilot vehicles and 10 pedestrians.

Dataset structure. Each record $\mathcal{R}_k$ at tick k contains all 12 sensor observations sharing a common tick index, plus vehicle and drone pose in the unified world frame. Records are serialized to disk in a flat per-tick directory structure with metadata in JSON. No timestamp interpolation is

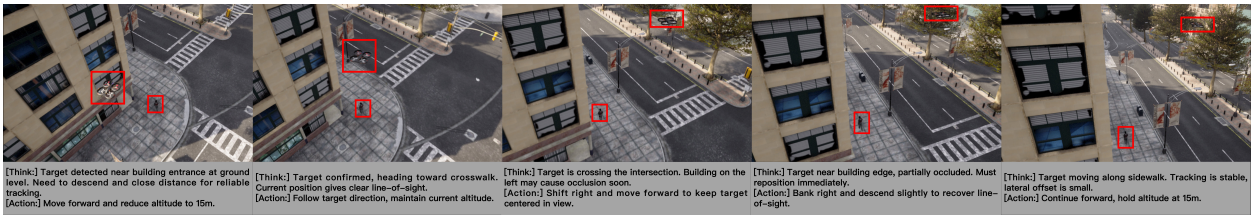

Figure 10: W2: Embodied navigation with aerial reasoning. A UAV autonomously tracks a pedestrian (red box, bottom) through an urban environment using bird's-eye visual observations. Each frame is annotated with the drone's chain-of-thought reasoning, illustrating how the agent interprets the scene, anticipates occlusions, and adjusts its flight path to maintain persistent visual contact with the target. All frames are rendered within CARLA-Air's shared simulation world under consistent lighting and physics.

required: the shared tick index serves as the alignment key.

Results. Over 1 000 ticks, the workflow produces 1 000 fully synchronized 12-stream records at a mean collection rate of ≈17 Hz. The maximum observed tick-alignment deviation across all 12 streams is one tick (occurring transiently under disk-write saturation). Table 8 summarizes collection performance.

Table 8: W3 multi-modal dataset collection results (1 000 ticks, 30 vehicles, 10 pedestrians, Town10HD, RTX A4000).

Metric	Value	Notes
Mean FPS	17.1	Harmonic mean
Concurrent streams	12	8 ground + 4 aerial
Records collected	1 000	One per tick
Max alignment dev.	$\leq$1 tick	Normal disk-write load
RPC errors	0	Both clients
Per-tick write latency	61 ± 9 ms	Incl. serialization

5.4 W4: Air-Ground Cross-View Perception

Cross-view perception—fusing aerial bird's-eye observations with ground-level sensing—is an emerging research direction for cooperative autonomous driving, urban scene understanding, and 3D reconstruction. This workflow demonstrates CARLA-Air's ability to provide spatially and temporally co-registered multi-modal sensor streams from both aerial and ground viewpoints within a single shared environment.

Setup. A drone equipped with a depth camera hovers above a road segment while a ground ego vehicle equipped with a semantic segmentation camera traverses the same segment. Both sensors are queried at each synchronous tick.

Sensor co-registration. Co-registration leverages the coordinate mapping from Section 3.3. The origin offset $\mathbf{d} \in \mathbb{R}^3$ between the aerial NED frame and the ground world frame is computed once at initialization:

$$\mathbf{d} = T\big(\mathbf{p}_{\text{spawn}}^{\text{world}}\big) - \mathbf{p}_{\text{spawn}}^{\text{NED}}, \tag{4}$$

where $T(\cdot)$ applies the scale conversion and axis remapping. When the drone spawns at the world origin, $\mathbf{d} = \mathbf{0}$.

Weather consistency. To verify rendering consistency across both sensor layers, the workflow iterates through all 14 official weather presets. For each preset involving significant illumination change, the mean pixel intensity of the aerial RGB frame shifts by more than 5% relative to the previous preset, confirming single-pass weather propagation. Because both sensors share the same rendering pipeline, temporal alignment is guaranteed:

$$\epsilon_k = \left| t_k^{\text{gnd}} - t_k^{\text{air}} \right| = 0, \tag{5}$$

a property that bridge-based architectures cannot provide.

Results. Over 500 ticks, the workflow produces 500 co-registered aerial-depth / ground-segmentation pairs at ≈18 Hz with zero RPC errors. All 14 weather presets pass the illumination consistency assertion. Table 9 summarizes the measured outcomes.

Table 9: W4 cross-view perception results (500 ticks, Town10HD, RTX A4000).

Metric	Value	Notes
Mean FPS	18.2	Harmonic mean
Co-registered pairs	500	Aerial depth + ground seg.
Per-tick latency	52 ± 6 ms	Full collection loop
Sensor alignment	0 ticks	Sync mode guarantee
Weather presets passed	14/14	All official presets
RPC errors	0	Both clients

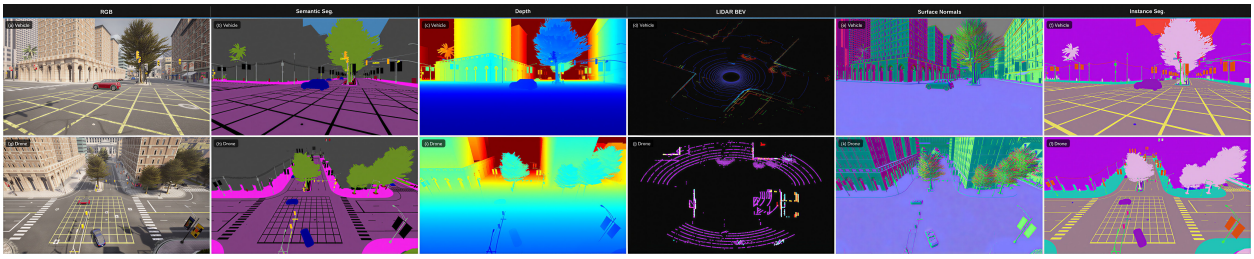

Figure 11: W3: Synchronized multi-modal dataset collection at a single simulation tick. **Top row** (vehicle perspective): RGB, semantic segmentation, depth, LiDAR bird's-eye view, surface normals, and instance segmentation. **Bottom row** (drone perspective): the same six modalities captured simultaneously from the aerial viewpoint. All 12 sensor streams share an identical tick index and are rendered under the same lighting and weather conditions through CARLA-Air's shared rendering pipeline, guaranteeing per-tick spatial-temporal correspondence without interpolation.

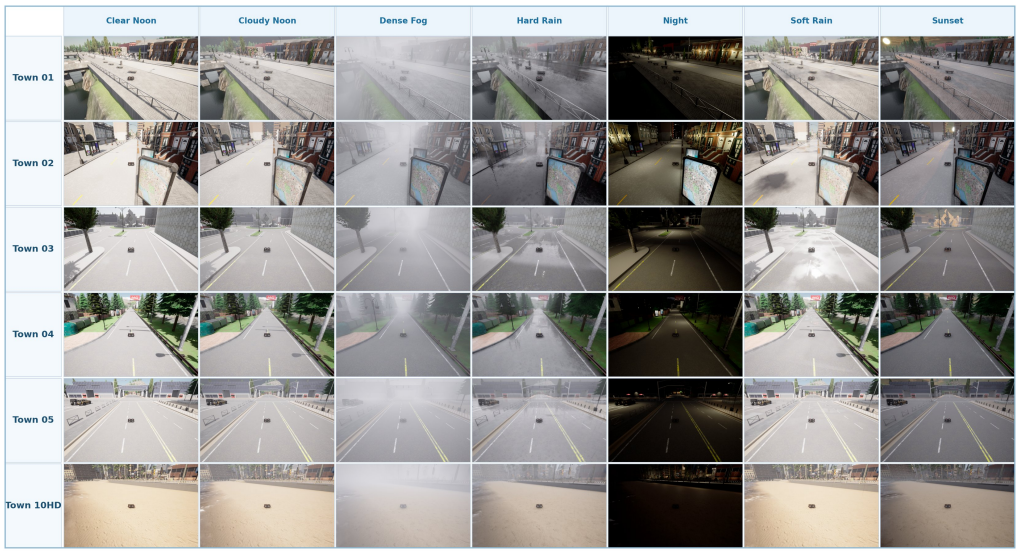

Figure 12: W4: Air-ground cross-view perception across diverse environments and weather conditions. Each row shows an aerial RGB view from the drone across six representative CARLA maps (Town 01–05 and Town 10HD); each column corresponds to a different weather preset (Clear Noon, Cloudy Noon, Dense Fog, Hard Rain, Night, Soft Rain, and Sunset).

5.5 W5: Reinforcement Learning Training Environment

Reinforcement learning in air-ground cooperative settings requires a simulation environment that provides closed-loop interaction, consistent state observations across heterogeneous agents, and stable long-horizon episode execution without memory leaks or synchronization drift. CARLA-Air's single-process architecture naturally satisfies these requirements, and the stability results from Section 4.3 (zero crashes and zero memory accumulation over 357 reset cycles) directly validate its suitability as an RL training environment.

Platform capabilities for RL. CARLA-Air supports RL training workflows through several platform-level features: (i) synchronous stepping mode provides deterministic state transitions compatible with standard Gym-style training loops; (ii) both aerial and ground agents expose programmatic control interfaces (velocity commands, waypoint targets, autopilot toggles) that can serve as action spaces; (iii) the full sensor suite (RGB, depth, segmentation, LiDAR, IMU, GPS) provides rich observation spaces for both state-based and vision-based policies; (iv) episode resets via actor spawn/destroy are validated for stability across hundreds of consecutive cycles (Section 4.3); and (v) the shared world state ensures that reward signals computed from cross-domain interactions (e.g., aerial-ground relative positioning) are physically consistent.

Example: cooperative positioning. As a representative RL scenario, consider a drone learning to maintain an optimal aerial observation position relative to a moving ground vehicle under varying traffic conditions. The observation space comprises the drone's pose, the vehicle's pose, and surrounding traffic state; the action space is a 3D velocity command; the reward encodes lateral tracking error and altitude maintenance. This scenario exercises the full air-ground control loop within CARLA-Air's synchronous tick, and can be implemented using standard RL libraries (e.g., Stable-Baselines3, RLlib) with minimal wrapper code.

6 Limitations and Future Work

The current release of CARLA-Air is validated for the workflows presented in Section 5: single- and dual-drone aerial operations over moderate-density urban traffic scenes.

- **Actor density.** Joint simulation performance is characterized at moderate traffic loads; high-density scenes with large simultaneous actor populations remain an active engineering target.
- **Environment resets.** Map switching requires a full process restart due to independent actor lifecycle management in each simulator backend; staged in-session resets are planned for a future release.
- **Multi-drone scale.** Configurations beyond two drones are functional but not yet formally validated across a wide range of scenarios; expanded multi-drone characterization will be documented once inter-agent behavior has been fully profiled.

None of these constraints affect the workflows in Section 5, all of which operate within the current boundaries.

Because CARLA-Air inherits and extends AirSim's aerial capabilities—whose upstream development has been archived—long-term maintenance of the aerial stack is managed within the CARLA-Air project itself. Bug fixes, compatibility updates, and feature extensions to the aerial subsystem are released as part of CARLA-Air's regular update cycle, ensuring that the flight simulation capabilities continue to evolve independently of the original upstream repository.

Looking ahead, near-term work will address physics-state synchronization between the two engines and a ROS 2 [9] bridge that republishes both simulator streams as standard topics for broader ecosystem integration. Longer-term, we aim to support GPU-parallel multi-environment execution in the spirit of Isaac Lab [11] and OmniDrones [19], bringing CARLA-Air closer to the episode throughput required for large-scale reinforcement learning.

7 Conclusion

Simulation platforms for autonomous systems have historically fragmented along domain boundaries, forcing researchers whose work spans ground and aerial domains to maintain inter-process bridge infrastructure or accept capability compromises. CARLA-Air resolves this fragmentation by integrating CARLA [2] and AirSim [15] within a single Unreal Engine process [3], exposing both native Python APIs concurrently over a shared physics tick and rendering pipeline.

Technical contributions. The central technical contribution is a principled resolution of the single-GameMode constraint through a composition-based design: `CARLAAirGameMode` inherits CARLA's ground simulation subsystems while composing AirSim's aerial flight actor as a standard world entity, with modifications to exactly two upstream source files. This design yields three properties unavailable in bridge-based approaches: a shared physics tick that eliminates inter-process clock drift, a shared rendering pipeline that guarantees consistent weather and lighting across all sensor viewpoints, and full preservation of both upstream Python APIs.

Platform capabilities. Building on this architecture, CARLA-Air provides a photorealistic, physically coherent simulation world with rule-compliant traffic, socially-aware pedestrians, and aerodynamically consistent multi-rotor dynamics. Up to 18 sensor modalities can be synchronously captured across aerial and ground platforms at each simulation tick. The platform directly supports four research directions in air-ground embodied intelligence: air-ground cooperation, embodied navigation and vision-language action, multi-modal perception and dataset construction, and reinforcement-learning-based policy training.

Validation. The platform is validated through performance benchmarks demonstrating stable operation at $\approx$20 FPS under joint workloads, a 3-hour continuous stability run with zero crashes across 357 reset cycles, and five representative workflows that exercise the platform's core capabilities. A consolidated feature comparison with representative platforms is provided in Table 1 (Section 2).

Broader impact. By providing a shared world state for aerial and ground agents, CARLA-Air enables research directions that are structurally inaccessible in single-domain

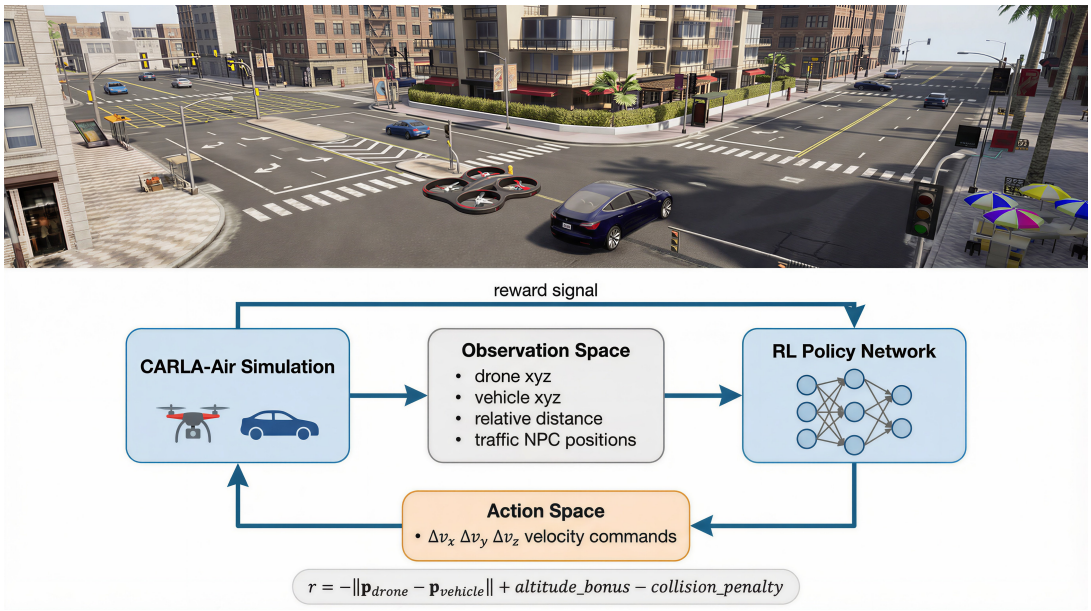

Figure 13: W5: Reinforcement learning training environment. **Top:** a drone learns to maintain an optimal aerial observation position above a moving ground vehicle within CARLA-Air's urban traffic environment. **Bottom:** the closed-loop RL pipeline. At each synchronous tick, CARLA-Air provides an observation space (drone and vehicle poses, relative distance, surrounding traffic state) to the policy network, which outputs 3D velocity commands as actions. The reward signal encodes tracking accuracy, altitude maintenance, and collision avoidance.

platforms: paired aerial-ground perception datasets from physically consistent viewpoints, coordination policies over joint multi-modal observation spaces, and embodied navigation grounded in cross-view visual and linguistic input. By also inheriting and extending the aerial simulation capabilities of AirSim—whose upstream development has been archived—CARLA-Air ensures that this widely adopted flight stack continues to evolve within a modern, actively maintained infrastructure.

References

[1] Alexander Amini, Tsun-Hsuan Wang, Igor Gilitschenski, Wilko Schwarting, Zhijian Liu, Song Han, Sertac Karaman, and Daniela Rus. VISTA 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4349–4356, 2022.

[2] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 1–16, 2017.

[3] Epic Games. Unreal Engine 4 documentation. https://docs.unrealengine.com/4.26/, 2021.

[4] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. RotorS—a modular gazebo MAV simulator framework. In *Robot Operating System (ROS): The Complete Reference*, volume 1, pages 595–625. Springer, 2016.

[5] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. FlightGoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6941–6948, 2019.

[6] Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2149–2154, 2004.

[7] Quanyi Li, Zhenghao Peng, Lan Feng, et al. MetaDrive: Composing diverse driving scenarios for generalizable reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3461–3475, 2023.

[8] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, et al. Microscopic traffic simulation using SUMO. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582, 2018.

[9] Steve Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[10] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, et al. Isaac Gym: High performance GPU-based physics simulation for robot learning. In *NeurIPS Datasets and Benchmarks Track*, 2021.

[11] NVIDIA. NVIDIA Isaac Lab: A unified and modular framework for robot learning. *arXiv preprint arXiv:2511.04831*, 2025.

[12] Jacopo Panerati, Hehui Zheng, SiQi Zhou, Amanda Prorok, and Angela P. Schoellig. Learning to fly—a gym environment with PyBullet physics for reinforcement learning of multi-agent quadcopter control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7512–7519, 2021.

[13] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, et al. LGSVL Simulator: A high fidelity simulator for autonomous driving. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2020.

[14] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, et al. Habitat: A platform for embodied AI research. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9339–9347, 2019.

[15] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, pages 621–635. Springer, 2018.

[16] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 1–16, 2021.

[17] Maonan Wang, Yirong Chen, Yuxin Cai, Aoyu Pang, Yuejiao Xie, Zian Ma, Chengcheng Xu, Kemou Jiang, Ding Wang, Laurent Roullet, Chung Shue Chen, Zhiyong Cui, Yuheng Kan, Michael Lepech, and Man-On Pun. TranSimHub: A unified air-ground simulation platform for multi-modal perception and decision-making. *arXiv preprint arXiv:2510.15365*, 2025.

[18] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, et al. SAPIEN: A simAted part-based interactive ENvironment. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11097–11107, 2020.

[19] Botian Xu, Feng Gao, et al. OmniDrones: An efficient and flexible platform for reinforcement learning in drone control. *arXiv preprint arXiv:2309.12825*, 2023.

[20] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, et al. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

A Appendix

A.1 System Configuration

Figure 14: Custom assets imported into CARLA-Air through the extensible asset pipeline. **Top:** a four-wheeled mobile robot with onboard LiDAR, imported from an external FBX model. **Bottom:** a custom electric sport car with user-defined vehicle dynamics. Both assets operate within the shared simulation world alongside all built-in CARLA traffic and AirSim aerial agents, and are visible to all sensor modalities.

Reference hardware. All experiments in this report were conducted on the following configuration: Ubuntu 20.04/22.04 LTS, NVIDIA RTX A4000 (16 GB GDDR6), AMD Ryzen 7 5800X (8-core, 4.7 GHz), 32 GB DDR4-3200.

Software stack. CARLA 0.9.16, AirSim 1.8.1 (final stable open-source release), Unreal Engine 4.26, Python 3.8+.

Network configuration. Table 10 lists the default port assignments. Both RPC servers bind to `localhost` by default; remote connections require explicit IP configuration.

Table 10: Default network port assignments.

Service	Protocol	Port
CARLA RPC Server	TCP	2000
CARLA Streaming	UDP	2001
AirSim RPC Server	TCP	41451

Distribution. The prebuilt binary package is approximately 19 GB and includes a one-command launcher (`CarlaAir.sh`). The source distribution is approximately 651 MB and is released under the MIT license.

A.1.1 API Compatibility Summary

Table 11 summarizes the API compatibility status and test coverage of the current CARLA-Air release. All 89 automated CARLA API tests pass without modification; the full AirSim flight control and sensor access API has been verified through manual and scripted testing. A total of 63 ROS 2 topics are published across both simulation backends.

Table 11: API compatibility and test coverage.

Component	Status
CARLA API	89/89 automated tests passing
AirSim API	Full flight control and sensor access verified
ROS 2 topics	63 total (43 CARLA + 14 AirSim + 6 generic)
Key upstream scripts confirmed working	
`manual_control.py`	CARLA manual driving interface
`automatic_control.py`	CARLA autopilot demonstration
`dynamic_weather.py`	CARLA weather preset cycling
`hello_drone.py`	AirSim basic flight demonstration

A.2 Upstream Source Modifications

CARLA-Air is designed to minimize modifications to the upstream CARLA codebase. The integration touches only two header files and one source file, totaling approximately 35 lines of changes. All other integration code is purely additive, residing within the aerial simulation plugin as the `CARLAAirGameMode` class ($\sim$1,405 lines of C++). Table 12 provides the complete modification summary.

Table 12: Upstream CARLA source modifications.

File	Modification
`CarlaGameModeBase.h`	`friend` declaration for `CARLAAirGameMode`
`CarlaEpisode.h`	Sensor tagging visibility: `private` $\rightarrow$ `protected`
`CarlaGameModeBase.cpp`	State flag assignment (1 line)
New integration layer (no upstream modification)	
`CARLAAirGameMode`	Unified game mode class; $\sim$1,405 lines C++

A.3 Custom Asset Import

CARLA-Air supports importing custom 3D assets (robot platforms, vehicles, UAV models, and environment objects) through an asset pipeline built on Unreal Engine's content framework. Imported assets are registered as spawnable actor classes and become accessible through the standard CARLA Python API (`world.spawn_actor()`). Once registered, custom assets share the same physics tick, rendering pass, and sensor visibility as all built-in actors, ensuring full consistency within the joint simulation environment.