AURORA: A Natural Language–Driven Agentic Framework for Understanding, Reasoning, and Orchestrating Reliable Air–Ground Co-Simulation

Keshu Wu[1], Hao Zhang[1], Rui Gan[2], Xiangbo Gao[3], Xiaopeng Li[2], Zhengzhong Tu[3], and Yang Zhou[1,†]

***Abstract*— Air–ground transportation research increasingly relies on co-simulation, yet constructing scenarios remains labor-intensive and difficult to validate. More importantly, a generated scenario may execute successfully while failing to realize the spatial, temporal, communication, or behavioral relationships requested by the user. This paper presents AURORA, a natural-language-driven agentic framework that treats air–ground scenario generation as a process of compilation with verification. Central to AURORA is the Air–Ground Scenario Graph (AGSG), a typed intermediate representation that explicitly connects agents, aerial missions, events, communication links, success conditions, and their cross-domain dependencies. This shared representation enables simulator-grounded parsing, joint road–airspace grounding, temporal planning, pre-execution feasibility checking, trace-based runtime verification, failure localization, and bounded repair within a unified workflow. We further introduce AURORA-Bench to evaluate not only whether generated scenarios execute, but whether they faithfully realize the requested interactions. Experiments across multiple language models show that structured execution substantially improves reliability, while runtime verification exposes silent failures that completion-based evaluation overlooks. Localized repair further resolves many violations without regenerating the entire scenario. The results show that reliable scenario generation requires verifying realized behavior, not merely executable code, and demonstrate the value of explicit intermediate representations for verifiable and repairable language-driven co-simulation.**

I. Introduction

Autonomous driving, intelligent transportation systems, and the emerging low-altitude economy are creating mobility systems in which unmanned aerial vehicles (UAVs) [1], roadside infrastructure [2], connected vehicles [3], [4], and other traffic participants [5] must perceive, communicate, and coordinate within a shared environment. Air–ground simulation [6], [7], [8] provides a practical testbed for studying such interactions before deployment, particularly for applications such as cooperative perception, emergency response, and connected transportation. Yet constructing such experiments remains labor-intensive. Researchers must manually coordinate map selection, valid actor placement, UAV routing, simulator synchronization, event triggers, communication links, and success criteria, and these steps must be repeated for each new scenario. As a result, the scale and diversity of air–ground experiments are often limited not by simulator capability, but by the effort required to author and validate scenarios.

Natural-language and generative-AI interfaces offer a promising way to reduce this burden and improve interaction with connected and automated transportation systems [9], [10], [11]. Recent work has explored language-driven traffic scenario generation and editing [12], [13], [14], [15] and natural-language UAV control in simulation [16], [17], [18]. However, air–ground scenario generation requires more than translating a prompt into executable code. A scenario can run while violating requested spatial, temporal, perceptual, or communication relations—for example, through invalid placement, premature mission transitions, missed messages, or interactions that never occur. Such failures are difficult to detect when generated scripts lack an explicit specification of what should happen. Specification-based simulation analysis and safety evaluation provide foundations for detecting such failures [19], [20], but air–ground scenarios additionally require joint feasibility across road geometry, three-dimensional airspace, sensing, communication, and event timing. The challenge is therefore not only to generate executable scenarios, but to verify that the requested interactions are realized before and during execution.

We present **AURORA**[1] (*A Natural Language–Driven **A**gentic Framework for **U**nderstanding, **R**easoning, and **O**rchestrating **R**eliable **A**ir–Ground Co-Simulation*), which treats language-driven scenario generation as *compilation with verification*. Natural-language requests are converted into typed, provenance-aware specifications and an Air–Ground Scenario Graph (AGSG) grounded in simulator measurements. Deterministic modules then resolve spatial and temporal feasibility, while a managed executor synchronizes the ground and aerial simulators and evaluates trace-level requirements. When a violation occurs, the AGSG localizes the implicated components and constrains the repair scope. The language model is therefore used for semantic interpretation and guarded edit proposals, while simulator-bound quantities are grounded and validated through deterministic procedures.

[1]Keshu Wu, Hao Zhang, and Yang Zhou are with the Zachry Department of Civil and Environmental Engineering, Texas A&M University, College Station, TX, USA. {keshuw, zhanghao230, yangzhou295}@tamu.edu

[2]Rui Gan and Xiaopeng Li are with the Department of Civil and Environmental Engineering, University of Wisconsin–Madison, Madison, WI, USA. {rgan6, xli2485}@wisc.edu

[3]Xiangbo Gao and Zhengzhong Tu are with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA. {xiangbog, tzz}@tamu.edu

[†]Corresponding author: Yang Zhou, yangzhou295@tamu.edu

[1]Project page: https://keshuw95.github.io/AURORA/

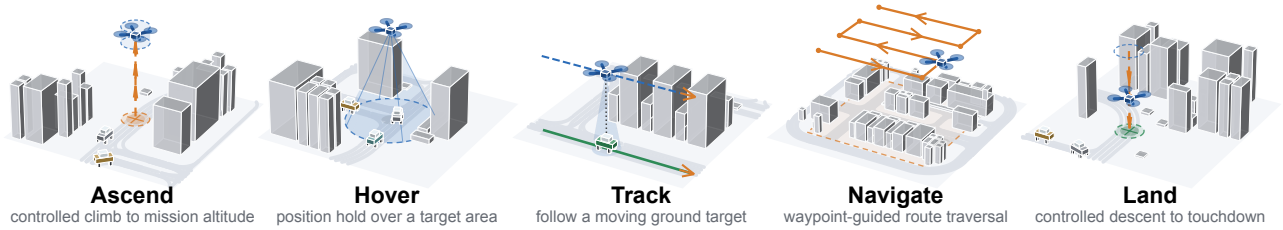


Fig. 1. UAV mission primitives in AURORA: ascend, hover, track, navigate, and land.

AURORA supports three core UAV mission modes—hover, track, and waypoint navigation—each executed between controlled ascent and landing phases (Fig. 1). Together, these primitives cover stationary aerial observation, dynamic air–ground following, and route-based missions while sharing a common execution and verification structure.

The key distinction is between *execution* and *realization*. AURORA does not treat simulator completion as success; it verifies whether the requested relations and outcomes hold in the execution trace. The AGSG links each requirement to its dependencies, verification conditions, and adjustable parameters, enabling unmet requirements to be localized and repaired without regenerating the full scenario. Verification thus guides both scenario generation and refinement rather than serving as a post hoc check.

The contributions are three-fold:

1) **AURORA for verified air–ground scenario generation.** We formulate natural-language-driven air–ground scenario generation as a verified compilation process and develop an end-to-end agentic workflow that integrates simulator-grounded understanding, joint spatial and temporal reasoning, synchronized co-simulation, runtime verification, and bounded repair.
2) **Air–Ground Scenario Graph for verification and repair.** We introduce the AGSG, a typed intermediate representation that links requested relations to executable verification conditions, execution evidence, and the parameters that can affect them. This shared representation supports pre-execution feasibility checking, trace-based monitoring, failure localization, and localized repair.
3) **AURORA-Bench for evaluating scenario realization.** We construct a dedicated benchmark for language-generated air–ground co-simulation, comprising 200 prompts, 904 annotated requirements, and boundary-feasible and infeasible cases. A six-configuration, five-LLM evaluation separates execution reliability, prompt fidelity, and runtime realization, revealing failures that completion-based evaluation would otherwise overlook.

II. Related Work

A. Air–ground simulation and cooperative perception

Aerial sensing complements ground-based perception by reducing occlusion and extending coverage. Drone-assisted datasets demonstrate the value of vehicle–UAV collaboration across diverse environments [6], [21], [22], [23], extending prior vehicle–vehicle and vehicle–infrastructure cooperative perception and reasoning [24], [25], [26]. CARLA and AirSim provide widely used ground and aerial simulation capabilities [27], [28], while newer platforms integrate multirotor and vehicle dynamics [7]; OpenCDA supports cooperative driving automation [29]. Despite these capabilities, scenario construction and cross-domain coordination remain largely manual—valid placement, UAV mission design, synchronization, event coordination—and AURORA targets this authoring and verification layer rather than introducing another simulator.

B. Scenario specification and language-driven generation

Scenic [30] and OpenSCENARIO [31] provide structured traffic-scenario representations, but require specialized languages or predefined schemas. Learning-based methods further automate traffic generation and planning, including trajectory-generation approaches [32], [33], [34]. Recent language-driven methods, including LLMScenario [14], TTSG [12], LCTGen [35], ChatScene [13], and Talk2Traffic [15], generate or interactively edit traffic scenarios from high-level descriptions, while ScenarioNet [36] provides a data-driven framework for large-scale scenario modeling and generation. These approaches reduce manual specification but primarily target ground traffic and scene or trajectory generation. Air–ground scenarios additionally couple road geometry, three-dimensional flight, sensing, communication, and event timing, requiring a representation that captures cross-domain dependencies and joint realizability.

C. Language-model execution and scenario verification

Language models have also been explored for executable code, embodied control, and transportation reasoning [37], [38], [39], [10], [34], and have recently been applied to natural-language and agentic UAV control in simulated environments [16], [17], [18]. Closed-loop UAV systems have further used execution feedback to evaluate and refine LLM-generated operation code [40]. However, successful execution does not guarantee that a generated scenario satisfies the request, and simulator scripts typically lack an explicit dependency structure for detecting and localizing semantic failures. VerifAI [19] demonstrates specification-based simulation analysis, while SafeBench [20] provides systematic benchmarking of autonomous-driving safety. These approaches, however, do not address natural-language compilation and verification of coupled air–ground scenarios.

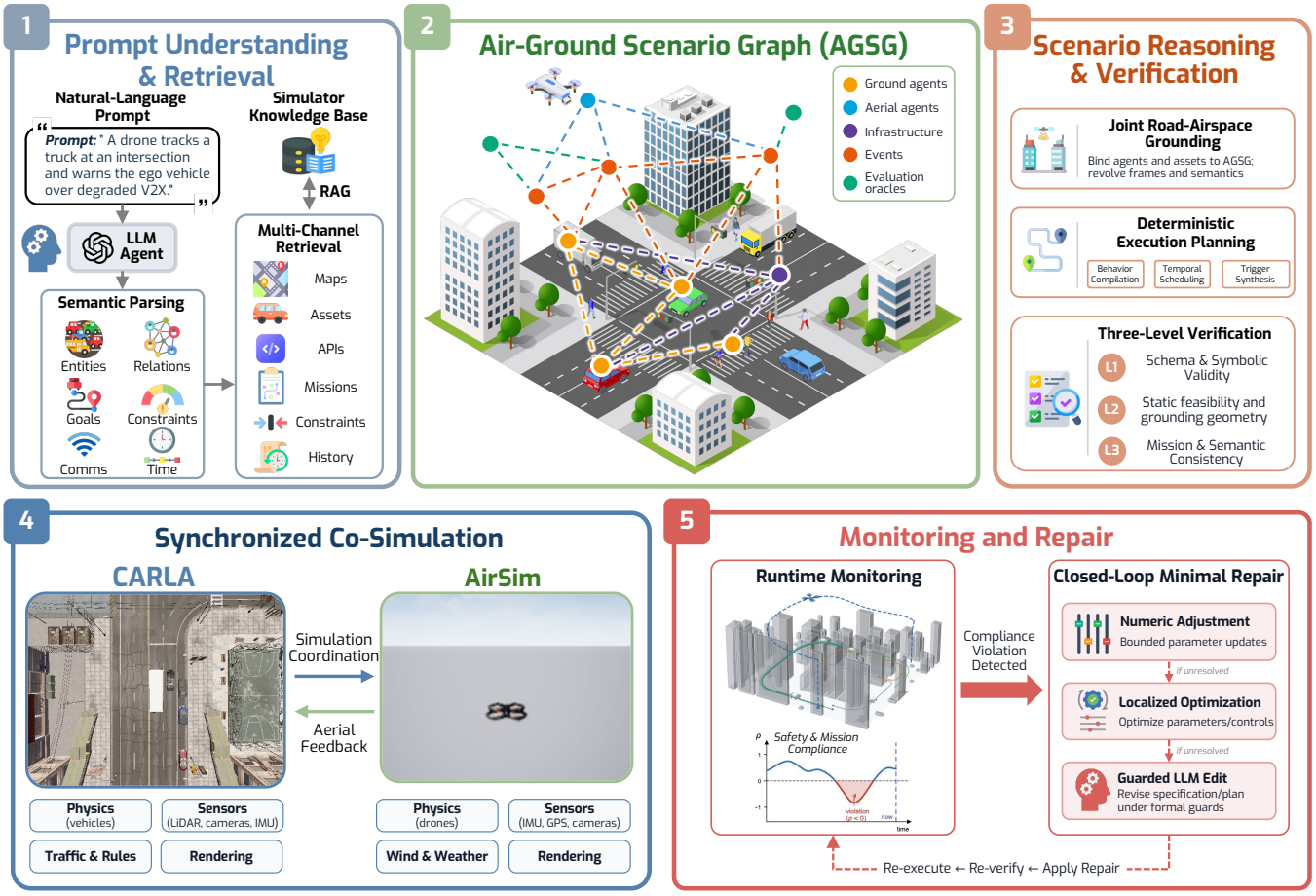


Fig. 2. Overview of AURORA. Retrieval-grounded parsing constructs the AGSG for joint road–airspace grounding, temporal planning, and feasibility checks. Synchronized CARLA–AirSim execution provides requirement-level measurements for graph-based failure localization and bounded repair.

AURORA bridges these directions through an Air–Ground Scenario Graph that supports feasibility checking, runtime verification, failure tracing, and localized repair.

III. Methods

A. Overview and Problem Formulation

AURORA converts a natural-language request into an executable and verified air–ground co-simulation. The framework receives a prompt p, a simulation environment $\mathcal{E}$ specifying the simulator backends and map, and a knowledge base $\mathcal{K}$ constructed offline from direct measurements of $\mathcal{E}$. The objective is to generate a scenario whose execution realizes the spatial, temporal, behavioral, and communication requirements expressed in the prompt. Executability and requirement satisfaction are treated separately: a scenario may complete without runtime errors while failing to realize the requested interaction.

AURORA represents the request through a typed scenario specification:

$$\mathcal{S} = (\mathcal{W}, \mathcal{A}, u, \mathcal{M}, \mathcal{V}_E, \mathcal{O}), \tag{1}$$

comprising world settings $\mathcal{W}$, ground agents and behaviors $\mathcal{A}$, UAV mission u, communication links $\mathcal{M}$, trigger–action events $\mathcal{V}_E$, and executable success conditions $\mathcal{O}$. The specification separates discrete scenario structure from the quantitative parameters that instantiate it, such as positions, altitudes, distances, timing thresholds, and communication settings. Each parameter also retains provenance indicating whether it is *explicit*, *inferred*, or *default*. This distinction allows later repair to preserve user-specified values more strongly while giving greater flexibility to values introduced by the parser or schema.

The generation and verification process is formulated as

$$(\mathcal{S}^*, \mathcal{H}^*, R^*) = \mathrm{AURORA}(p; \mathcal{E}, \mathcal{K}), \tag{2}$$

where $\mathcal{S}^*$ is the returned specification, $\mathcal{H}^*$ its execution trace, and R^* the verification and repair report. A successful output must pass pre-execution feasibility checks and satisfy all encoded success conditions during execution. Rather than returning only a completed simulation, AURORA retains the evidence needed to interpret the outcome, including violated conditions, their satisfaction margins, and any repairs applied.

AURORA follows three phases (Fig. 2). During *understanding*, retrieval-grounded parsing converts the prompt into a typed specification and its Air–Ground Scenario Graph (AGSG). During *reasoning*, deterministic modules determine where and when the requested relations can be realized

and whether the grounded configuration is feasible before execution. During *orchestration*, a managed executor synchronizes the simulators, records the resulting trace, and evaluates trace-level requirements. Violations are mapped through the AGSG to a constrained set of candidate edits, allowing verification feedback to directly guide subsequent repair. The LLM therefore handles semantic interpretation and guarded symbolic proposals, while simulator-bound grounding, validation, execution, and numerical evaluation remain deterministic.

B. Prompt Understanding and Scenario Representation

The LLM agent interprets the prompt using the scenario schema and retrieved simulator knowledge, producing a typed specification rather than simulator code. The schema constrains admissible agent types, mission modes, event structures, parameters, units, and references. Invalid outputs are returned to the agent with explicit feedback from schema and static checks, allowing the interpretation to be revised through bounded retries. Accepted fields retain their provenance, supporting both prompt-fidelity evaluation and later control over which fields may be modified during repair.

Retrieval draws on a knowledge base $\mathcal{K}$ built offline from direct simulator measurements and past executions. It contains verified spawn locations and map attributes, measured asset geometry, UAV mission templates and operating limits, feasibility constraints, simulator interfaces, and execution history. Core registries remain available throughout generation, while retrieval selects prompt-relevant assets, spawn examples from the active map, and prior episodes with similar requirements. This measurement-grounded knowledge reduces dependence on potentially inaccurate model knowledge of simulator APIs or geometry.

Importantly, the parser and verifier share the same simulator-grounded records. The knowledge base therefore acts as a common source of constraints: quantities presented to the LLM during interpretation are drawn from the same environment knowledge later used to evaluate feasibility. This alignment reduces inconsistencies in which a value appears plausible during generation but violates the actual simulator configuration during execution.

The validated specification is converted deterministically into an *Air–Ground Scenario Graph* (AGSG):

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}_G, \tau_{\mathcal{V}}, \tau_{\mathcal{E}}, \Theta, \omega)\,, \tag{3}$$

where $\mathcal{V}$ contains environment, ground-agent, UAV, communication, event, and success-condition nodes; $\mathcal{E}_G$ encodes their relations and dependencies; $\tau_{\mathcal{V}}$ and $\tau_{\mathcal{E}}$ assign node and edge types; Θ contains quantitative parameters; and ω maps each parameter to its owning node. The resulting graph explicitly represents dependencies among scenario components—for example, placement affects sensing geometry, sensing may trigger communication, and communication may enable a subsequent vehicle action.

Rather than serving only as a scenario description, the AGSG links requested relations to their verification conditions and influencing parameters. Typed dependencies across

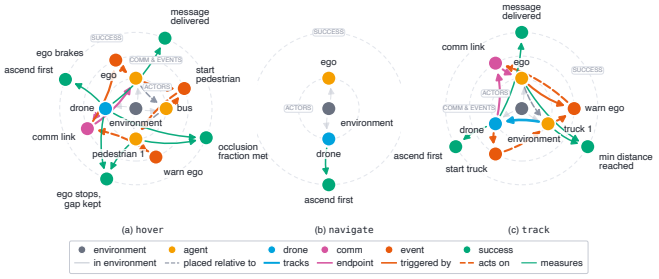


Fig. 3. Representative grounded AGSGs: (a) cooperative occlusion warning; (b) survey patrol; (c) tracking under degraded communication.

spatial, temporal, perceptual, communication, and behavioral requirements allow the same graph to support feasibility checking, trace-based monitoring, failure localization, and repair. When a condition fails, its dependencies can be traced to the components and parameters that may have caused the violation. The AGSG therefore connects what the user requests, what the simulator executes, and what can be changed during repair. Fig. 3 shows representative grounded AGSGs across the supported mission modes.

C. Joint Grounding, Planning, and Verification

The agentic workflow delegates spatial grounding, temporal planning, and feasibility checking to deterministic modules. These modules determine where and when the scenario can unfold and return constraint violations to guide refinement. Together, they translate the relational specification into a configuration that is grounded in the actual road network, airspace, and execution constraints.

a) Joint road–airspace grounding: A feasible location must simultaneously support the requested traffic event, UAV mission, sensing geometry, and communication geometry. Candidate road anchors are therefore ranked jointly as

$$s_{\text{joint}}(r_i) = \alpha s_{\text{road}}(r_i) + \beta s_{\text{air}}(r_i) + \gamma s_{\text{sense}}(r_i) + \eta s_{\text{comm}}(r_i), \tag{4}$$

where the four terms respectively evaluate road suitability, aerial clearance, air-to-ground visibility, and communication geometry. Equal weights are used in the present implementation. The road term characterizes whether the local roadway can support the requested ground interaction, while the remaining terms assess whether the same location provides suitable airspace and air–ground relationships.

Once an anchor is selected, ground agents are placed on valid road or pedestrian elements using relative-placement rules. Requested placements that are not directly realizable are projected onto feasible lanes or pedestrian elements while preserving their intended spatial relations as closely as possible. UAV routes are then checked against an obstacle-height lattice, and infeasible waypoints or route segments are corrected before execution. This division keeps structural geometric correction within grounding rather than deferring such defects to the later runtime repair loop.

b) Temporal planning: Each UAV mission is executed as a predicate-guarded state machine consisting of ascent, one of three supported mission modes (i.e., hover, track,

or waypoint navigation), and landing (Fig. 1). Tasks are scheduled to respect physical feasibility and execution dependencies; sequential actions are completed in the required order before subsequent actions begin. For instance, navigation begins only after the UAV reaches its target altitude. Using realized predicates rather than fixed delays prevents downstream actions from beginning before their physical prerequisites are satisfied.

Inter-event timing is modeled using a Simple Temporal Network [41], where event orderings and deadlines are expressed as interval constraints. For events e_i and e_j, temporal relations can be represented as $l_{ij} \leq t_{e_j} - t_{e_i} \leq u_{ij}$, allowing requirements such as message transmission preceding reception or a warning arriving before a subsequent vehicle action to be checked before execution.

After grounding, geometry-dependent triggers are adjusted to ensure that the requested conditions remain reachable from the realized initial configuration. This step is important because a trigger defined against an imagined configuration may become immediately true or unreachable once agents are projected onto valid simulator geometry. Temporal planning therefore operates on the grounded scenario rather than independently of it.

c) Three-level verification: Verification proceeds from inexpensive structural checks to trajectory-level requirements. First, *schema verification* checks agent and task types, ranges, units, references, and temporal consistency. These checks identify malformed or internally inconsistent specifications before simulator execution. Second, *static feasibility* checks whether placements, routes, relational constraints, and communication requirements can be realized in the grounded environment. These checks use the instantiated geometry and simulator-grounded limits rather than language-level plausibility alone. Violations are associated with the relevant fields or relations so that infeasible configurations can be revised before execution. Third, *semantic verification* evaluates trace-dependent requirements such as visibility, message delivery, event ordering, and mission completion. These properties depend on interactions over time and therefore cannot be established from the initial configuration alone. They are monitored using quantitative temporal-logic semantics [42], [43], [44], producing a signed margin ρ_i: $\rho_i \geq 0$ indicates satisfaction, whereas $\rho_i < 0$ quantifies the degree of violation. Unlike a binary pass/fail result, the signed margin also indicates how far a condition lies from satisfaction and can therefore guide subsequent correction. The same representation thus supports both verification and repair.

D. Synchronized Execution, Monitoring, and Repair

a) Managed execution and synchronization: A specification that passes pre-execution checks configures a fixed *managed executor* for initialization, spawning, control, event scheduling, communication, logging, and cleanup. The LLM defines the scenario specification, while a validated execution path translates that specification into simulator operations. This separation avoids generating new simulator code for each scenario.

CARLA and AirSim advance under a shared logical clock. Coordinate transformation and a proxy UAV maintain a common ground–air representation, while online clock correction limits temporal drift. Communication is evaluated from realized sender–receiver geometry together with configured loss, latency, occlusion, and retransmission constraints. Agent states, message delivery, and event outcomes are recorded in the execution trace and evaluated against the encoded success conditions.

b) Runtime monitoring: Pre-execution feasibility does not guarantee that the requested interaction will occur. Moving agents may fail to establish the intended relation, tracking may not converge, or an event may miss its timing or communication requirement. Runtime monitoring therefore evaluates success conditions on the realized trace and returns their signed robustness margins. This separates verified realization from nominal completion and quantifies how far each violated condition lies from satisfaction.

c) Bounded local repair: Because trace-level verification requires co-simulation, AURORA uses bounded local repair:

$$\begin{aligned} \mathcal{S}_0 &= \mathrm{Ground}\big(\mathrm{Parse}_{\mathcal{K}}(p)\big), \\ \mathcal{S}_{k+1} &= \mathrm{Repair}(\mathcal{S}_k, \mathcal{VS}_k) \in \mathcal{N}(\mathcal{S}_k), \quad k < B, \end{aligned} \tag{5}$$

where $\mathcal{VS}_k = \{m : \rho_m(\mathcal{H}_k) < 0\}$ identifies conditions violated by trace $\mathcal{H}_k$, and B denotes the maximum number of repair iterations. The AGSG traces each violation to implicated parameters and fields, defining the admissible edit neighborhood $\mathcal{N}(\mathcal{S}_k)$. This localization prevents unrelated parts of the scenario from changing. Each numerical repair adjusts at most five parameters, while each symbolic repair revises at most six fields and must pass schema and static verification.

For coupled numerical failures, a kinematic replay surrogate evaluates candidate edits through a provenance-regularized objective:

$$\begin{aligned} \Delta\Theta^* = \underset{\Delta\Theta \in \Omega}{\mathrm{argmin}} \; & \max\Big(0,\, \varepsilon^\star - \min_m \hat{\rho}_m(\Theta + \Delta\Theta)\Big) \\ & + \lambda_{\mathrm{sem}} \sum_j w(c_j) \frac{|\Delta\theta_j|}{\mathrm{width}(\mathcal{D}_j)}. \end{aligned} \tag{6}$$

Here, Ω restricts edits to the localized parameter set and schema domains $\mathcal{D}_j$; $\hat{\rho}_m$ is the surrogate-estimated satisfaction margin; c_j denotes the provenance class of parameter θ_j; and $\varepsilon^\star > 0$ specifies the desired robustness headroom. The first term drives the worst monitored condition toward a positive margin, while the second penalizes deviation from the parsed request after normalizing edits by their admissible ranges. The coefficient λ_{sem} balances satisfaction against semantic deviation, and $w(c_j)$ penalizes changes to explicit values more heavily than changes to inferred values or defaults.

The surrogate screens candidate edits before another full co-simulation run. Repair proceeds from single-parameter adjustment to localized multivariate search, followed by guarded LLM revision when numerical search cannot remove

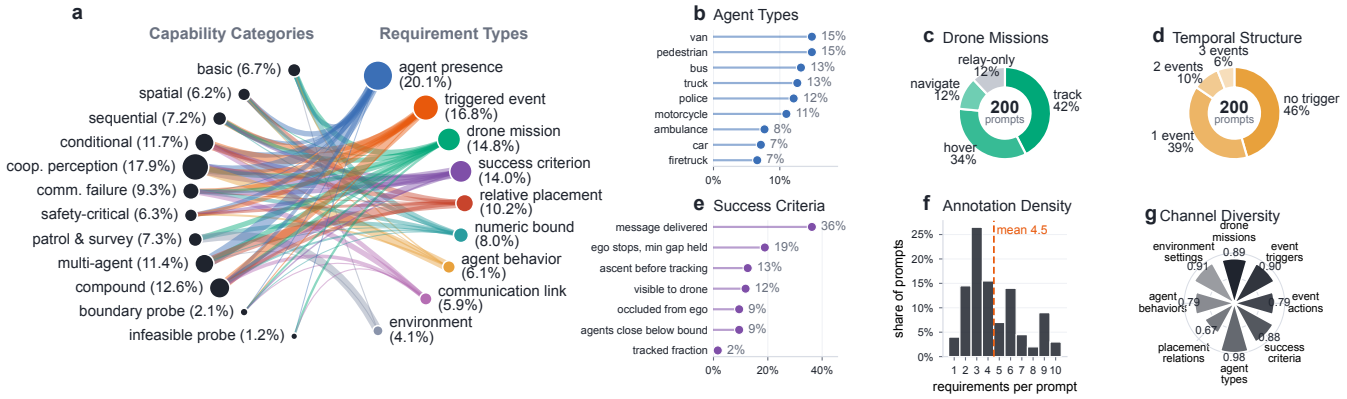


Fig. 4. AURORA-Bench composition, requirement annotations, and scenario diversity.

the violation. The LLM receives localized diagnostic feedback and may edit only within the admissible scope; each proposal must pass deterministic checks before re-execution. Structural geometric failures requiring broader changes, such as an infeasible route, are returned to the grounding stage rather than forced through the minimal-edit repair loop.

The loop terminates on success or budget exhaustion. The final report retains verification status, repair history, and unresolved margins. Because provenance weighting discourages but does not prohibit changes to explicit values, the report also distinguishes improvements in realized behavior from changes to the monitored conditions themselves.

IV. Experimental Design

A. Benchmark and Evaluation Protocol

AURORA-Bench comprises 200 prompts spanning 12 capability categories and 904 annotated requirements (Fig. 4). Annotations include reference specifications, feasible realizations where applicable, variation ranges, and paraphrase groups. For prompts without a flight-task requirement, any feasible UAV mode is accepted. End-to-end evaluation uses a 50-prompt subset: 40 behavioral scenarios across nine categories, five boundary-feasible probes, and five infeasible probes.

We evaluate five LLMs—GPT-4o, GPT-5.5, GPT-5.4-mini, Gemini 3.1 Pro, and Gemini 3.8 Flash—using provider-default sampling with bridged CARLA 0.9.15 and AirSim 1.8.1. All configurations use the same prompts and simulation seed. Within each campaign, one model handles parsing, baseline code generation, and repair proposals. Retrieval history is frozen before each campaign to prevent information transfer from evaluation runs. Specifications, execution traces, verification reports, and model transcripts are archived. GPT-4o serves as the reference model for the illustrative example and aggregate analysis.

B. Metrics and Baselines

We distinguish three properties of generated scenarios: *executability*, *prompt fidelity*, and *runtime realization*. Executability asks whether a scenario completes without setup errors, runtime errors, or timeouts. Prompt fidelity measures whether the generated specification preserves the annotated request. Runtime realization asks whether the monitored conditions actually hold during execution. Accordingly, *completion rate* (C.) measures successful execution, while *runtime-failure rate* (RF) captures execution failures. *Verified-pass rate* (P) measures executions in which all monitored conditions are satisfied, whereas *silent-failure rate* (SF) captures completed runs with unmet requirements. *Prompt fidelity* (Fid.) measures correspondence between the generated specification and annotated requirements.

Verification applies to the encoded monitored conditions, while fidelity and repair logs assess whether parsing and refinement preserve the original request, particularly when thresholds are modified. Supporting metrics characterize grounding accuracy, synchronization, event timing, communication, mission performance, and computational cost.

Six configurations progressively introduce framework components: direct code generation (**B1**), simulator documentation (**B2**), simulator-grounded retrieval (**B3**), the typed AGSG and managed executor without online verification (**B4**), static verification (**B5**), and runtime monitoring with localized repair (**AURORA**). Successful B1–B3 runs are classified as *completed-unverifiable*, as they lack executable specifications for the semantic evaluation used here. Completion is compared across all six configurations, while verified outcomes are compared across B4, B5, and AURORA.

V. Results

A. Illustrative Example

Fig. 5 illustrates how AURORA translates a tracking request into coordinated air–ground behavior. The UAV completes ascent before tracking the bus, while the police–bus interaction triggers a warning to the ego vehicle. The AGSG connects the participating agents, mission, communication link, and event dependencies to the grounded trajectories and execution schedule. These complementary views show not only how the requested interaction develops over time, but also whether its prerequisites and dependent events are realized in the intended order.

The warning reaches the ego vehicle before the UAV establishes close tracking. This sequence highlights why

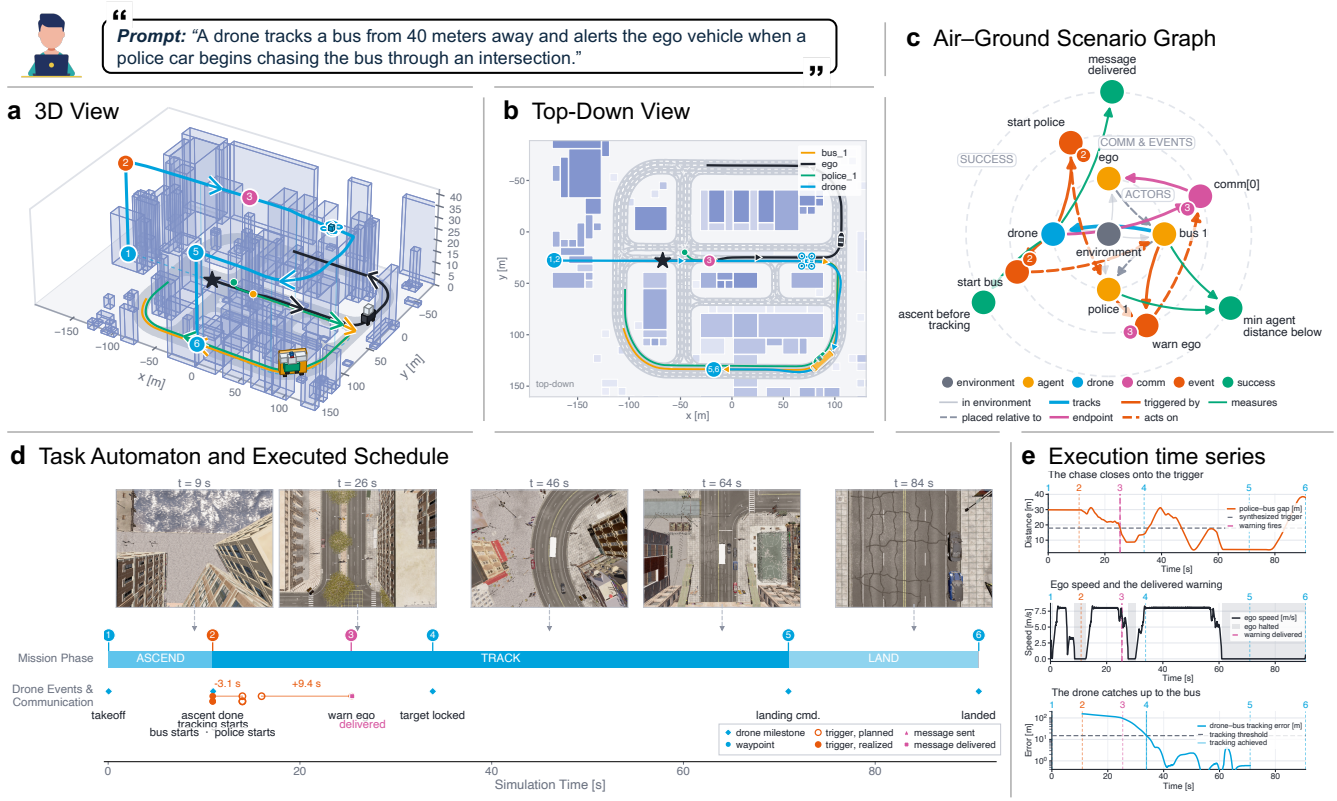


Fig. 5. Tracking scenario linking the language request, AGSG, grounded trajectories, and monitored outcomes.

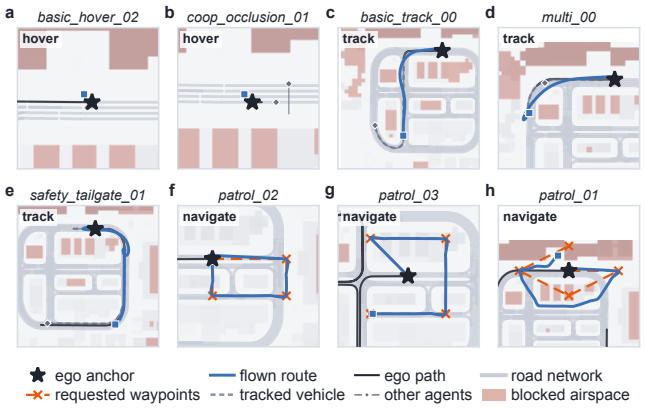


Fig. 6. Representative grounded hover, track, and navigation scenarios with UAV and ground-agent trajectories and obstacle-aware route correction.

communication and mission performance require separate verification: successful message delivery does not establish sustained tracking. The event records confirm the warning outcome, while the UAV–bus distance trace reveals tracking convergence and subsequent variation. A single completion label would collapse these outcomes, whereas requirement-level monitoring preserves their distinction. By linking each condition to execution evidence, the AGSG also provides a basis for targeted diagnosis when part of the requested interaction remains unmet.

Fig. 6 broadens the example across the three supported mission modes. Hover scenarios maintain aerial coverage around a grounded region, tracking scenarios couple UAV motion to a moving ground target, and navigation scenarios realize waypoint-based routes within the available airspace. Most requested routes are preserved directly, while geometrically infeasible waypoints are corrected during grounding before execution. Together with Fig. 5, these examples illustrate how AURORA translates different language-level mission structures into grounded and executable air–ground scenarios.

Fig. 7 shows the executed scenarios from the UAV camera, with one strip per request. The navigation example follows the requested survey route, while the tracking examples keep the motorcycle, bus, and tailgating van in view as the interactions unfold. These views provide qualitative evidence that the requested spatial and tracking interactions are realized in simulation, complementing the trace-based verification of communication and event outcomes.

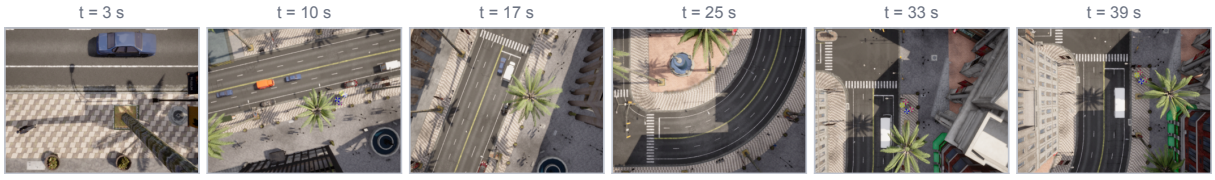
B. End-to-End Performance

The six configurations across five LLMs isolate the effects of simulator knowledge, structured execution, and verification-driven repair (Table I and Fig. 8). The progression also separates improvements in executability from improvements in realized behavior, which do not necessarily arise from the same framework components.

a) Execution reliability: Simulator knowledge improves code execution but does not eliminate runtime failures. Documentation raises completion across all five LLMs,

(a) “In clear weather at midday, a drone flies a rectangular survey route at 30 meters altitude over the streets around the ego car.”

(b) “A drone ascends to 40 meters and then follows a motorcycle as it drives through the streets.”



(c) “A bus drives ahead of the ego car and an ambulance follows behind it while two background cars circulate; a drone tracks the bus from 35 meters.”

(d) “A van tailgates the ego car closely, closing from 45 meters; a tracking drone alerts the ego when the van comes within 20 meters.”

Fig. 7. Prompt-to-execution examples across four scenarios, showing six onboard UAV views sampled over each execution and the corresponding natural-language request.

TABLE I
END-TO-END PERFORMANCE ACROSS SIX CONFIGURATIONS AND FIVE LANGUAGE MODELS. BEST VALUE IN EACH COLUMN IS UNDERLINED.

	GPT-4o					GPT-5.5					GPT-5.4-mini					Gemini 3.1 Pro					Gemini 3.8 Flash				
Method	C.↑	RF↓	P↑	SF↓	Fid.↑	C.↑	RF↓	P↑	SF↓	Fid.↑	C.↑	RF↓	P↑	SF↓	Fid.↑	C.↑	RF↓	P↑	SF↓	Fid.↑	C.↑	RF↓	P↑	SF↓	Fid.↑
B1: Direct Code Generation	0.42	0.58	—	—	—	0.82	0.18	—	—	—	0.70	0.30	—	—	—	0.80	0.20	—	—	—	0.72	0.28	—	—	—
B2: + Documentation	0.68	0.32	—	—	—	0.92	0.08	—	—	—	0.86	0.14	—	—	—	0.94	0.06	—	—	—	0.92	0.08	—	—	—
B3: + Grounded Retrieval	0.66	0.34	—	—	—	0.92	0.08	—	—	—	0.78	0.22	—	—	—	0.98	0.02	—	—	—	0.98	0.02	—	—	—
B4: + AGSG & Executor	1.00	0	0.58	0.42	0.966	1.00	0	0.42	0.58	1.000	1.00	0	0.50	0.50	1.000	1.00	0	0.66	0.34	0.993	1.00	0	0.62	0.38	0.984
B5: + Static Verification	0.98	0	0.56	0.42	0.980	1.00	0	0.48	0.52	1.000	1.00	0	0.60	0.40	0.995	0.98	0	0.70	0.28	0.995	1.00	0	0.54	0.46	0.971
AURORA	0.98	0	0.76	0.22	0.980	1.00	0	0.64	0.36	1.000	1.00	0	0.70	0.30	0.995	0.98	0	0.84	0.14	0.995	1.00	0	0.74	0.26	0.971

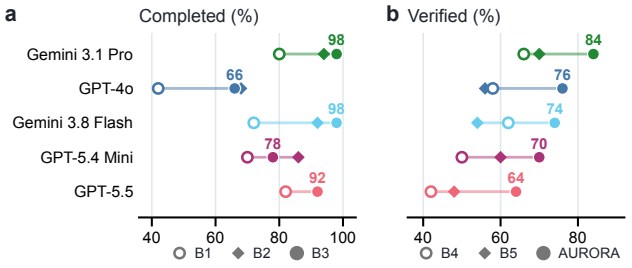


Fig. 8. Configuration comparison across models: (a) completion for code-generation baselines; (b) verified passes for AGSG-based configurations.

while retrieval provides mixed additional gains, indicating that additional context alone does not guarantee reliable execution. The larger change comes from structured execution: introducing the typed AGSG and managed executor (B4) raises completion to 100% across all models and eliminates observed runtime failures. Yet 34–58% of scenarios remain silent failures. Where later configurations have slightly lower completion despite zero runtime failures, the difference reflects pre-execution rejection rather than execution breakdown. Reliable execution therefore does not imply successful realization, making completion alone insufficient for evaluating generated scenarios.

b) Verification and repair: Runtime feedback provides the main gain in scenario realization. Static verification produces mixed changes in verified-pass rate, because identifying a feasible configuration before execution does not guarantee that its trace-dependent requirements will hold. Adding runtime monitoring and localized repair improves

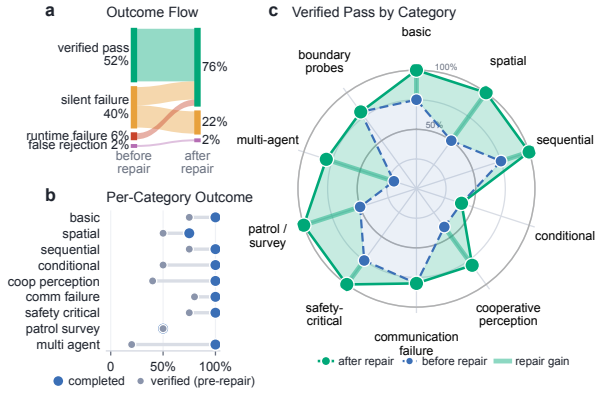


Fig. 9. GPT-4o evaluation: (a) repair outcomes; (b) completion and initial verification by category; (c) category-level repair gains.

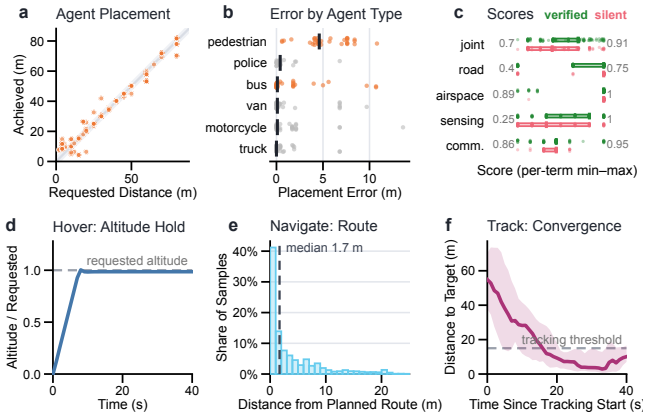


Fig. 10. Grounding validation: placement accuracy and per-type error (a, b), anchor scores by outcome (c), and mission realization (d–f).

verified-pass rates by 10–20 percentage points over B5 across the five models, reaching 64–84% under AURORA. The gains vary by scenario category (Fig. 9): spatial, sequential, and patrol scenarios improve substantially, while conditional scenarios remain more challenging. This pattern suggests that localized repair is most effective when a violation can be traced to a compact set of geometric, mission, or timing parameters; longer conditional chains remain harder to resolve. Remaining violations are reported explicitly rather than being absorbed into completion.

c) Prompt fidelity and realized behavior: Prompt fidelity and runtime realization capture different properties. GPT-5.5, for example, achieves perfect specification fidelity without achieving the highest verified-pass rate. Preserving the request during parsing is therefore necessary but not sufficient; the grounded geometry, parameters, event timing, and mission dynamics must also realize the required conditions during execution. Notably, reported fidelity remains unchanged from B5 to AURORA for all five models, while verified-pass rates increase. The repair gains therefore do not coincide with a broad reduction in specification fidelity, although individual edits still require inspection when thresholds or explicit conditions are modified. Fidelity and verification thus provide complementary evidence of requirement preservation and realized behavior. Because B1–B3 lack executable specifications for semantic verification, their completion rates alone cannot establish whether the generated scenarios fulfill the requests.

C. Grounding and Execution Validation

We next examine scenario grounding and coordinated execution across the coupled simulators.

a) Spatial grounding and mission realization: Fig. 10 shows close agreement between requested and realized scenario geometry. Ground-agent placements are generally accurate, with larger deviations mainly caused by projection onto valid sidewalks or lanes. UAV missions likewise remain close to their intended geometry, with navigation deviations concentrated around turns. Most geometric differences therefore reflect feasibility enforcement rather than grounding failure. Tracking adds a temporal dimension: valid initialization does not ensure immediate mission fulfillment, because the UAV must first converge to the moving target and maintain the required relation over time. Placement accuracy and mission realization consequently answer different questions—whether the scenario is grounded correctly at initialization and whether the requested relation persists once the agents begin to move.

b) Grounding quality and scenario outcomes: Verified runs have higher joint anchor scores than silent failures, with the largest separation in the sensing term (Fig. 10). Road, airspace, and communication scores differ less between the two groups. Successful grounding therefore depends not only on valid roads and aerial clearance, but also on whether the selected location supports the sensing relationship required by the interaction. Joint grounding captures these constraints before execution, while the resulting traces reveal whether the anticipated relationships persist as agents move. The anchor score is therefore useful as a pre-execution indicator of scenario quality, but it cannot replace trace-level verification of the realized interaction.

c) Coordination across models: Fig. 11 shows consistent simulator coordination across the five LLMs, with small clock offsets and limited event-timing error. The distributions nevertheless include delayed events and larger placement deviations, making variation beyond the median important. Mission performance varies more substantially, particularly in tracking convergence and time on target. Because all models use the same managed executor, this contrast separates low-level co-simulation reliability from model-dependent scenario realization. Stable synchronization provides the foundation for coordinated execution, but does not by itself guarantee the requested air–ground behavior.

D. Repair Analysis and Computational Cost

We further analyze repair effectiveness and its computational cost.

a) Repair effectiveness: Repair connects violated conditions to the components that can be adjusted. In a tracking example, two iterations move the tracked-fraction margin from negative to positive, while infeasible patrol routes are

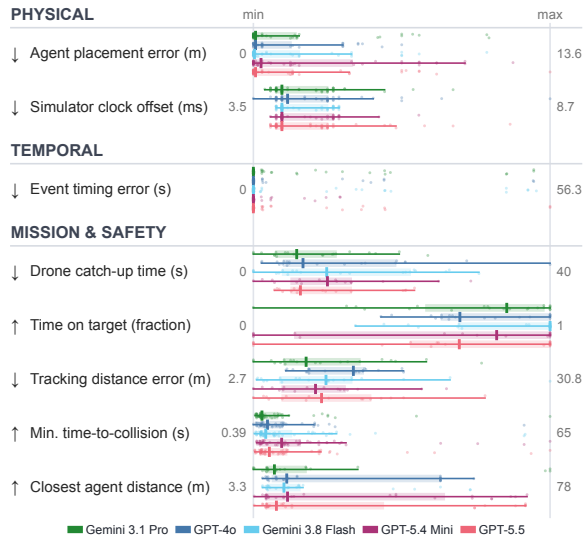


Fig. 11. Execution and mission metrics across language models.

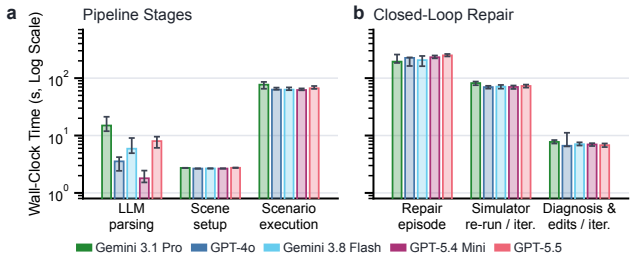


Fig. 12. Computational cost across models: (a) pipeline-stage latency; (b) repair-episode cost and per-iteration components.

corrected during grounding through waypoint snapping. The two cases illustrate a useful division of responsibility: localized quantitative violations are handled by the repair loop, whereas structural geometric defects are returned to grounding rather than forced through small parameter edits. Routing failures to the appropriate stage avoids regenerating the full scenario while keeping each correction within its intended scope. Improved margins must nevertheless be interpreted alongside the edits that produced them. Some repairs relax acceptance thresholds, and parsing may reformulate an infeasible request instead of rejecting it. Provenance weighting discourages such changes but does not prohibit them. Repair logs therefore distinguish corrections to realized behavior from changes to the conditions used to evaluate it. Verified-pass rate should consequently be interpreted together with prompt fidelity and edit history rather than as a standalone measure of preservation of the original request.

b) Computational cost: Fig. 12 shows that simulator execution dominates repair cost. Median repair episodes take 194–249 s across models, largely due to repeated co-simulation runs, while diagnosis and edit generation contribute comparatively little. Parsing latency varies across LLMs, whereas static checks and kinematic preview remain inexpensive. This asymmetry supports placing inexpensive verification and surrogate screening before full re-execution whenever possible. Localized edits constrain the search space, but the larger opportunity for reducing repair cost lies in avoiding unnecessary simulator reruns rather than accelerating edit generation alone.

VI. Conclusion

AURORA connects natural-language scenario generation with explicit feasibility checking, synchronized air–ground execution, and trace-based verification. Its Air–Ground Scenario Graph links requested relations to executable conditions, execution evidence, and repair variables, providing a shared representation for grounding, monitoring, failure localization, and refinement. Evaluation across five language models distinguishes three properties of generated scenarios: whether they execute, preserve the request, and realize the requested conditions. Structured execution improves reliability, but successful completion can still mask semantic failures; runtime verification exposes these failures, while localized repair resolves many without regenerating the full scenario. More broadly, the results support treating language-driven scenario generation as verified compilation rather than direct code generation. Explicit intermediate representations make cross-domain dependencies inspectable and provide a structured path from runtime violations to the parameters that can affect them, reducing reliance on language models for simulator-specific reasoning and improving failure diagnosis.

The current framework supports a defined scenario schema, three UAV mission templates, simplified sensing and communication models, and one simulation environment. Because parsing can reinterpret infeasible requests and repair can relax monitored thresholds, a verified pass should be interpreted as satisfaction of the encoded monitored conditions rather than unconditional satisfaction of the original prompt. Future work will strengthen preservation of explicit requirements and terminal handling of infeasible requests, while extending AURORA toward multi-UAV planning and control, richer cooperative air–ground interactions, vision–language–action models for closed-loop perception and decision making, and additional simulation environments and real-world testbeds.

References

[1] T. Feng, X. Wang, F. Han, L. Zhang, and W. Zhu, "U2udata: A large-scale cooperative perception dataset for swarm uavs autonomous flight," in *Proceedings of the 32nd ACM International Conference on Multimedia*, 2024, pp. 7600–7608.

[2] F. Yang, B. Yu, Y. Zhou, X. Luo, Z. Tu, and C. Liu, "Edge-based multimodal sensor data fusion with vision-language-action (vla) model for real-time autonomous vehicle accident avoidance," *Engineering Applications of Artificial Intelligence*, vol. 180, p. 115186, 2026.

[3] R. Xu, H. Xiang, Z. Tu, X. Xia, M.-H. Yang, and J. Ma, "V2x-vit: Vehicle-to-everything cooperative perception with vision transformer," in *European conference on computer vision*. Springer, 2022, pp. 107–124.

[4] K. Wu, P. Li, Y. Cheng, S. T. Parker, B. Ran, D. A. Noyce, and X. Ye, "A digital twin framework for physical-virtual integration in v2x-enabled connected vehicle corridors," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, no. 9, pp. 14 407–14 420, 2025.

[5] X. Gao, Y. Wu, R. Wang, C. Liu, Y. Zhou, and Z. Tu, "Langcoop: Collaborative driving with language," in *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2025, pp. 4226–4237.

[6] X. Gao, Y. Wu, F. Yang, X. Luo, K. Wu, X. Chen, Y. Wang, C. Liu, Y. Zhou, and Z. Tu, "Airv2x: Unified air-ground vehicle-to-everything collaboration," *arXiv preprint arXiv:2506.19283*, 2025.

[7] T. Zeng, Y. Wen, and H. Zhang, "Carla-air: Fly drones inside a carla world–a unified infrastructure for air-ground embodied intelligence," *arXiv preprint arXiv:2603.28032*, 2026.

[8] H. Zhang, X. Yue, K. Tian, S. Li, K. Wu, Z. Li, D. Lord, and Y. Zhou, "Virtual roads, smarter safety: A digital twin framework for mixed autonomous traffic safety analysis," *IEEE Internet of Things Journal*, 2025.

[9] Y. Wang, S. Xing, C. Cui, R. Li, H. Hua, K. Tian, Z. Mo, X. Gao, M. Pavone, Y. Zhou, *et al.*, "Generative ai for autonomous driving: Frontiers and opportunities," *ACM Computing Surveys*, 2025.

[10] K. Wu, P. Li, Y. Zhou, R. Gan, J. You, Y. Cheng, J. Zhu, S. T. Parker, B. Ran, D. A. Noyce, *et al.*, "V2x-llm: Enhancing v2x integration and understanding in connected vehicle corridors," *arXiv preprint arXiv:2503.02239*, 2025.

[11] X. Gao, K. Wu, H. Zhang, K. Tian, Y. Zhou, and Z. Tu, "Automated vehicles should be connected with natural language," *arXiv preprint arXiv:2507.01059*, 2025.

[12] B.-K. Ruan, H.-T. Tsui, Y.-H. Li, and H.-H. Shuai, "Traffic scene generation from natural language description for autonomous vehicles with large language model," *arXiv preprint arXiv:2409.09575*, 2024.

[13] J. Zhang, C. Xu, and B. Li, "Chatscene: Knowledge-enabled safety-critical scenario generation for autonomous vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 15 459–15 469.

[14] C. Chang, S. Wang, J. Zhang, J. Ge, and L. Li, "Llmscenario: Large language model driven scenario generation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 54, no. 11, pp. 6581–6594, 2024.

[15] Z. Sheng, Z. Huang, Y. Qu, Y. Leng, and S. Chen, "Talk2traffic: Interactive and editable traffic scenario generation for autonomous driving with multimodal large language model," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2025, pp. 3827–3836.

[16] A. Phadke, A. Hadimlioglu, T. Chu, and C. N. Sekharan, "Integrating large language models for uav control in simulated environments: A modular interaction approach," *arXiv preprint arXiv:2410.17602*, 2024.

[17] A. Shibu, M. Saleh, M. Al-Musleh, and N. Abdulaziz, "Skysim: A ros2-based simulation environment for natural language control of drone swarms using large language models," *arXiv preprint arXiv:2602.01226*, 2026.

[18] A. Koubaa and K. Gabr, "Agentic uavs: Llm-driven autonomy with integrated tool-calling and cognitive reasoning," *arXiv preprint arXiv:2509.13352*, 2025.

[19] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 432–442.

[20] C. Xu, W. Ding, W. Lyu, Z. Liu, S. Wang, Y. He, H. Hu, D. Zhao, and B. Li, "Safebench: A benchmarking platform for safety evaluation of autonomous vehicles," *Advances in Neural Information Processing Systems*, vol. 35, pp. 25 667–25 682, 2022.

[21] J. Wang, X. Cao, J. Zhong, Y. Zhang, Z. Han, H. Yu, C. Zhang, L. He, S. Xu, and J. Wang, "Griffin: Aerial-ground cooperative detection and tracking dataset and benchmark," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 40, no. 12, 2026, pp. 9867–9875.

[22] H. Ye, R. Sunderraman, and S. Ji, "Uav3d: A large-scale 3d perception benchmark for unmanned aerial vehicles," *Advances in Neural Information Processing Systems*, vol. 37, pp. 55 425–55 442, 2024.

[23] Y. Cui, X. Dong, B. Gao, J. Xiang, D. Li, and Z. Tu, "Airsimag: A high-fidelity simulation platform for air-ground collaborative robotics," *arXiv preprint arXiv:2603.23079*, 2026.

[24] R. Xu, H. Xiang, X. Xia, X. Han, J. Li, and J. Ma, "Opv2v: An open benchmark dataset and fusion pipeline for perception with vehicle-to-vehicle communication," *arXiv preprint arXiv:2109.07644*, 2021.

[25] H. Yu, Y. Luo, M. Shu, Y. Huo, Z. Yang, Y. Shi, Z. Guo, H. Li, X. Hu, J. Yuan, *et al.*, "Dair-v2x: A large-scale dataset for vehicle-infrastructure cooperative 3d object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 21 361–21 370.

[26] J. You, Z. Jiang, Z. Huang, H. Shi, R. Gan, K. Wu, X. Cheng, X. Li, and B. Ran, "V2x-vlm: End-to-end v2x cooperative autonomous driving through large vision-language models," *Transportation Research Part C: Emerging Technologies*, vol. 183, p. 105457, 2026.

[27] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[28] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics: Results of the 11th international conference*. Springer, 2017, pp. 621–635.

[29] R. Xu, Y. Guo, X. Han, X. Xia, H. Xiang, and J. Ma, "Opencda: an open cooperative driving automation framework integrated with co-simulation," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021, pp. 1155–1162.

[30] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, 2019, pp. 63–78.

[31] ASAM e.V., "ASAM OpenSCENARIO: Dynamic content in driving simulation, standard v1.2," https://www.asam.net/standards/detail/openscenario/, 2022.

[32] Z. Zhong, D. Rempe, D. Xu, Y. Chen, S. Veer, T. Che, B. Ray, and M. Pavone, "Guided conditional diffusion for controllable traffic simulation," *arXiv preprint arXiv:2210.17366*, 2022.

[33] K. Wu, Y. Zhou, H. Shi, D. Lord, B. Ran, and X. Ye, "Hypergraph-based motion generation with multi-modal interaction relational reasoning," *Transportation Research Part C: Emerging Technologies*, vol. 180, p. 105349, 2025.

[34] R. Gan, P. Li, K. Long, B. An, J. You, K. Wu, and B. Ran, "Planning safety trajectories with dual-phase, physics-informed, and transportation knowledge-driven large language models," *arXiv preprint arXiv:2504.04562*, 2025.

[35] S. Tan, B. Ivanovic, X. Weng, M. Pavone, and P. Kraehenbuehl, "Language conditioned traffic generation," *arXiv preprint arXiv:2307.07947*, 2023.

[36] Q. Li, Z. M. Peng, L. Feng, Z. Liu, C. Duan, W. Mo, and B. Zhou, "Scenarionet: Open-source platform for large-scale traffic scenario simulation and modeling," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 3894–3920. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/0c26a501df8fb919a0350e2df06b5d39-Paper-Datasets_and_Benchmarks.pdf

[37] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[38] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International conference on robotics and automation (ICRA)*. IEEE, 2023, pp. 9493–9500.

[39] S. Yao, J. Zhao, D. Yu, I. Shafran, K. R. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[40] W. Wang, Y. Li, L. Jiao, and J. Yuan, "Large language model-driven closed-loop uav operation with semantic observations," *IEEE Internet of Things Journal*, 2025.

[41] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial intelligence*, vol. 49, no. 1-3, pp. 61–95, 1991.

[42] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *International symposium on formal techniques in real-time and fault-tolerant systems*. Springer, 2004, pp. 152–166.

[43] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[44] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International conference on formal modeling and analysis of timed systems*. Springer, 2010, pp. 92–106.