Scenario Generation for Interactive Urban Environments

Paritosh Sharma, Hui-Yin Wu

February 2026

Project members

- Scientific team: Paritosh Sharma, Hui-Yin Wu

1 Context and objectives

The document highlights the work plan for the the WP4 of the ANR Creative 3D [1] project. The expected outcome of this project is to create a generative model that is capable of creating personalized training scenarios in urban environments.

2 Introduction

Virtual Reality (VR) and Augmented Reality (AR) technologies have advanced significantly in recent years, enabling the creation of immersive and interactive environments for various applications. These can be further used to provide personalized training and rehabilitation scenarios. In the context of low vision rehabilitation, these models can be particularly useful to study pedestrian behaviours under normal and simulated vision. However, most simulated environments suffer from perceptual gaps between the designer, the user, and the system. The existing GUsT-3D framework [**?**], developed during the Creative3D project, provides a foundation to address this. However, the current framework is still limited by its reliance on a fixed set of urban environments and interactions innhibiting its ability to scale.

In parallel, recent works on 3D generative models for urban scenario generation has shown promising results in generating diverse and realistic urban environments. Additionally, these models have been able to capture the diverse nature and the complexities of an urban setting, including the interactions between pedestrians, vehicles, and the environment. Driven by these extraordinary capabilities, exploring the potential of generative models will enable us to create more diverse and realistic urban scenarios.

3 Related Work

In this section, we first review prior work on pedestrian-in-the-loop simulation. We then survey recent generative approaches for urban scenario generation, covering symbolic, layout-guided, and

[1] `https://project.inria.fr/creattive3d/`

neural rendering–based methods, as well as diffusion-based models for dynamic scene synthesis. Finally, we discuss common validation protocols used to evaluate the performance of these models.

3.1 Pedestrian-in-the-Loop

VR Simulations have been widely used to study pedestrian behavior and interactions in urban scenarios [**?**, **?**, **?**, **?**]. This human-in-the-loop approach (also referred as pedestrian-in-the-loop[**?**]) allows researchers to collect data on how pedestrians interact with their environment, including their decision-making processes, movement patterns, and responses to various stimuli. These simulations can be used to study a wide range of scenarios, from simple road crossings to complex urban environments with multiple interacting agents. More recently, JaywalkerVR [**?**], a VR human-in-the-loop simulator, used CARLA [**?**] to create four different scenarios: jaywalking, parked cars, four-way stops, and parking lot entrances. Despite the obvious advantages, developing such simulators is still challenging because of the perceptual gaps between the designer, the user and the system as identified by Dourish [**?**].

The GUsT-3D framework [**?**] addresses this by first defining the scene using a scenegraph, which captures the relationships between different elements in the scenario (ontology) and then defining the task to be carried out during the course of the scenario using a GUTasks (intersubjectivity). Lastly, it uses a query component for logging and post-scenario analysis of the experience (intentionality). This framework was also applied by creating a dataset of 6 road-crossing scenarios to study pedestrian behavior under normal and low vision [**?**]. Even though GusT-3D framework addresses the perceptual gaps which were identified by Dourish, it still relies on a fixed set of scenarios and interactions.

3.2 Scenario Generation

3.2.1 Structured Scenario Generation

Structured scenario generation refers to methods that produce scenarios in formal, machine-readable representations suitable for execution in simulators and systematic evaluation.

The most widely used formats in autonomous driving research include:

- **Scenic** [**?**]: a probabilistic domain-specific language for defining constraints, distributions, and relationships among agents. Scenarios specified in Scenic can be sampled into concrete instances.
- **OpenSCENARIO DSL** [**?**]: an open standard for describing the dynamic content of scenarios used in driving and traffic simulators.
- **GeoScenario DSL** [**?**]: similar to OpenSCENARIO, but provides concrete, executable scenario specifications with explicit positions, trajectories, and temporal details, making scenarios directly runnable and reusable across different simulation platforms.
- **ScenarioNet** [**?**]: built on real-world data for structured, executable traffic scenarios.

3.2.2 Prompt-based Procedural Specification

Prompts are textual cues provided to the generative model to guide the scenario creation process. Since the popularity of LLMs like GPT [**?**], prompts have become a common way to interact with

generative models. They can be used to specify the desired characteristics of the scenario, such as the type of environment, objects, and their relationships. Table **??** lists some of the recent prompt-based models for scenario generation.

Model	Technique	Output
ScenicNL [**?**]	Converts LLM Prompts to Scenic Constraints	Scenic Scenario
ChatScene [**?**]	Conversational Agent for Scenario Definition using Scenic	Scenic Scenario
LayoutGPT [**?**]	Prompts converted to CSS-like Layout Formatting by LLMs	Layout Representation
ChatDyn [**?**]	LLM-based planning and low-level trajectory generation for Pedestrian and Vehicle	3D Scenario
Work by Feng et al. [**?**]	JSON to describe layout and 3D models	3D Scene
TTSG [**?**]	LLM-based planning and retrieval	3D Scenario
Scenethesis [**?**]	Uses LLM to create a scenegraph and then a 3D scene	3D Scene
SceneX [**?**]	LLM to plan PCG (Procedural Controllable Generation)	3D Scene

Table 1: Prompt-based Models for Urban Scenario Generation

3.2.3 Multimodal Procedural Specification

Some works have also combined multimodal inputs with different types of data, prompts, layouts and other structured representations, to provide a richer context for scenario generation. Table **??** lists some of the recent multimodal models for urban scenario generation.

Model	Input Type	Output
CityX [**?**]	Prompt, OSM file or Semantic Map	3D Scenario
CityCraft [**?**]	Layout data and text prompts	3D Scene

Table 2: Multimodal Models for Urban Scenario Generation

4 Problem and Open Question

While recent generative models have shown strong capabilities in synthesizing urban scenarios, they remain limited in their ability to model pedestrian dynamics within these environments. In particular, they struggle to establish a coherent link between spatio-temporal control over objects and the range of possible pedestrian interactions—an aspect that is essential for simulating realistic traffic scenarios.

This leads to the following open research question:

How can natural language–conditioned generative models be leveraged to produce urban scenarios for pedestrian agents, with explicit control over objects, their interactions, and task-level guidance?

5 Method

Figure **??** provides an overview of the method.

5.1 Input and Splitting

Since no existing datasets provide paired text prompts with corresponding urban layouts and tasks, we propose leveraging the capabilities of LLMs to parse and structure input prompts. Similar approaches have been explored in prior works, as shown in section **??**

We first process the input OSM and prompt and split it into the dynamics and task description using COT (Chain-of-Thought) reasoning. Here is an example of how the input prompt can be split:

Scene Specification Format

Prompt:
You are an urban scenario planning assistant. For the following urban scene prompt:

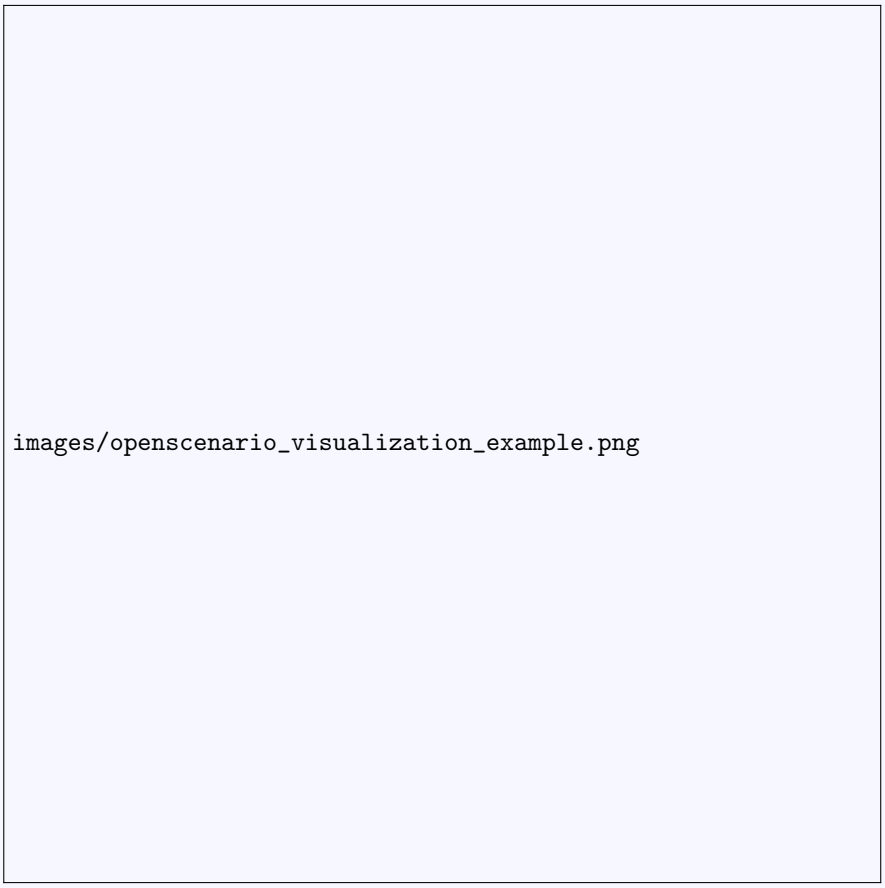
[Urban scene description]

- **dynamics:** Generate an OpenScenario XML for the above prompt, the OpenDrive road network and the Open Street Maps layout.
- **tasks:**
  Before creating tasks, review the prompt, the OpenDrive road network and the OpenScenario dynamics.
  - Task targets must reference actual object `"type"` values in static_layout.
  - Ordered list of guided actions for the pedestrian; each task must include:
    - `id`
    - `goal`: {`type, target, key_item`}
      - `type`: One of [`"get"`, `"interact"`, `"interactWith"`, `"go"`, `"place_in"`, `"place_on"`] with exact definitions and required object layers:
      - `"get"`: Target must be `layer="Movable"`
      - `"interact"/"interactWith"`: Target must be `layer="Interactable"`
      - `"go"`: Target must be `layer="Ground"`
      - `"place_in"`: Target must be `layer="Container"`
      - `"place_on"`: Target must be `layer="Support"`
      - `target`: Must match the exact `"type"` of an existing object in static_layout with the correct layer.
      - `key_item`: Optional item needed (null if not needed)
    - `constraints`: {`precedence`, `evaluation`}
    - `help`: {`baseline`, `target`, `failure`}
    - `failure_condition`: {`type`, `item`}
    - `instructions`: {`text_en`, `text_fr`}
    - `trigger`: {`object_id`, `function`}

Input Example:

OSM layout:

Output Example:
OpenScenario Dynamics Visualization:



Tasks:

- Walk from

JSON for the above available in appendix section **??**

5.2 Tasks

The tasks predicted by the LLM are a sequence of guided actions to be performed by the player or agent in the environment. Each task is defined as a structured object, similar to the following JSON format:

- **id:** a unique identifier for the task.
- **goal:** specifies the type of action, the target object, and optionally a key item needed to perform the action.
  - `type` – one of the following:
    - `get`: locate and pick up an object (target must have layer "Movable").

    - `interact`: interact with an object (target must have layer "Interactable").
    - `interactWith`: interact with a specific object using a key item (target has layer "Interactable").
    - `go`: navigate to a location (target must have layer "Ground").
    - `place_in`: place an object inside a container (target has layer "Container", requires `key_item`).
    - `place_on`: place an object on a support surface (target has layer "Support", requires `key_item`).
  - `target` – the identifier of the target object, using the GUST annotation format:
    - Ground objects (layer "Ground"): `objecttype_x.0_y.0`, e.g., `sidewalk_50.0_62.0`, `road_50.0_50.0`.
    - Other objects: `objecttype_id`, e.g., `traffic_light_4`, `table_6`, `flower_pot_8`.
  - `key_item` – optional object required to perform the task (null for simple tasks, required for `place_in`, `place_on`, and `interactWith` tasks).

- **baselineTime:** time in seconds that a baseline user should take to complete the task (typically 10–30 seconds for simple tasks, 30–60 seconds for complex tasks).
- **targetTime:** maximum time in seconds allowed for the user to complete the task (typically 1.5–2× `baselineTime`).
- **Help system:** provides progressive assistance to the user at different stages.
  - `baselineHelp`: type of help provided after `baselineTime` elapses ("text", "path", or "nothing").
  - `targetHelp`: type of help provided after `targetTime` elapses ("text", "path", or "nothing").
  - `failureHelp`: type of help provided after task failure ("text", "path", or "nothing").
  - `baselineText`: English help text shown after `baselineTime` (e.g., "Walk straight along the sidewalk").
  - `targetText`: English help text shown after `targetTime` (e.g., "Reach the sidewalk safely").
  - `failureText`: English help text shown after failure (e.g., "Stay on designated walking areas").
- **Failure conditions:** defines when and how the task fails.
  - `failure`: type of failure condition ("collision" or "none").
  - `failureItem`: object that causes failure when collided with (e.g., "vehicle", "person"; null if `failure` is "none").
- **constraints:** includes task ordering and evaluation criteria.
  - `precedence`: list of task IDs that must be completed before this task can begin.
  - `evaluation`: condition that defines successful task completion (e.g., "on_sidewalk", "item_picked_up").

- **instructions:** natural language instructions for the player in multiple languages.
  - `text_en`: English instructions (e.g., "Walk to the sidewalk at position [50, 62]").
  - `text_fr`: French instructions (e.g., "Marchez vers le trottoir à la position [50, 62]").
- **trigger:** specifies the object and function that triggers task completion.
  - `object_id`: ID of the object from `static_layout`.
  - `function`: trigger function name (e.g., "on_reach", "on_interact", "on_pickup").

The structure is based on the existing GUsT-3D framework [**?**] and is integrated with the existing system to provide guided task execution with real-time feedback and progressive assistance.

6 Implementation

The implementation of the system is divided into two components — the **Backend (Python)** and the **Frontend (Unity)** — which together enable interaction between the LLM and the Unity-based scenario generation environment.

6.1 Backend (Python) Implementation

The backend is implemented as an HTTP server in Python that interfaces with the LLM to process requests and manage data flow between the model and the Unity client. It exposes two primary `POST` endpoints:

- **/get_3d** – Receives the OSM bounding box coordinates and returns the 3D scene. For generating the OSM scene, we modify the existing blosm addon for Blender to add support for trash, crosswalks, bus stops and traffic lights.
- **/get_scenario** – Sends the prompt and receives the OpenScenario dynamics as well as the GUTasks from the Qwen3 LLM agent.

6.1.1 Blosm implementation

The blosm addon for Blender is extended to add support for trash, crosswalks, bus stops and traffic lights. However, the OSM semantics only provide positional information and no orientation information of the semantic classes. Thus, for elements dependent on the road direction, we use algorithm **??** to calculate the orientation of 3D assets.

Algorithm 1 Compute Orientation for Road-Aligned OSM Objects

Require: Semantic object O (e.g., crosswalk, bus stop) with position p_O
Require: OSM road network R composed of segments S_i with start and end coordinates
Ensure: Orientation vector $\vec{d_O}$ for the 3D asset

Find the nearest road segment S_n to p_O
Compute the road direction vector $\vec{v}_n$ of S_n:

$$\vec{v}_n = \frac{S_n.\text{end} - S_n.\text{start}}{\| S_n.\text{end} - S_n.\text{start}\|}$$

if O is a crosswalk **then**
    $\vec{d_O} \leftarrow$ vector perpendicular to $\vec{v}_{road}$
else if O is a bus stop or traffic light **then**
    $\vec{d_O} \leftarrow \vec{v}_{road}$ ▷ aligned parallel to the road
else
    $\vec{d_O} \leftarrow \vec{v}_{road}$ ▷ default: parallel to road
end if
Place 3D asset for object O at position p_O with orientation $\vec{d_O}$

6.2 Frontend (Unity) Implementation

The frontend is developed as a part of the existing GUsT-3D add-on. It communicates with the Python backend via HTTP, sending requests to the defined endpoints and parsing the responses to generate the scenarios within Unity.

For the OpenScenario interpretation, we modify the esmini addon and add support for pedestrians.

7 Experiments

This section presents a log of all experiments conducted to generate the dynamics and tasks with the *LLM*.

7.1 Experiment 1

8 Evaluation

8.1 Quantitative Evaluation

To assess the fidelity and diversity of the generated results, we use the following metrics.

8.1.1 Dynamics Interpretation Success Rate

The dynamics are considered successfully interpretable if the generated OpenScenario XML can be loaded and visualized in Unity.

8.1.2 Prompt Adherence for Tasks and Dynamics

Since no ground-truth scenarios are available for open-ended prompts, we evaluate the semantic quality of generated outputs in terms of *prompt adherence*. Prompt adherence measures how faithfully the generated tasks and dynamics satisfy the semantic requirements explicitly specified in the input prompt.

For each prompt, we extract a set of requested semantic classes, including task types, dynamic behaviors, road and environment attributes, agent roles, and quantitative constraints (e.g., speed ranges or traffic density). From the generated OpenSCENARIO XML and GUTasks representation, we then extract the corresponding set of realized semantic classes.

Prompt adherence is evaluated using precision and recall over these semantic classes. Precision measures the extent to which generated elements are relevant to the prompt and penalizes the introduction of hallucinated or unrequested behaviors. Recall measures the completeness of the generation with respect to the prompt and captures whether all requested tasks and dynamics are present in the output.

Formally, let R denote the set of semantic classes requested by the prompt and G the set of semantic classes extracted from the generated output. Prompt precision and recall are defined as:

$$\text{Precision} = \frac{|R \cap G|}{|G|}, \quad \text{Recall} = \frac{|R \cap G|}{|R|}$$

For quantitative attributes, such as speeds or timing constraints, a class is considered correctly generated if the corresponding value lies within a predefined tolerance range.

8.1.3 GUTasks Interpretation Success Rate

Generated GUTasks are considered successfully interpretable if GuST-3D and Unity can load and process them.

8.1.4 Task Feasibility Rate

Generated tasks are considered feasible if their required navigation and interaction components can be realized within the simulation environment. Specifically, navigation tasks are deemed feasible if a valid path exists for Unity's NavMesh Agent between the specified start and goal states. Interaction tasks are considered feasible if their preconditions, such as spatial distance, visibility, and agent states, can be satisfied during execution.

8.2 Qualitative Evaluation

For our experiments, we evaluate the generation qualitatively using a fixed set of predefined prompts, as well as through a user study in which participants freely author their own prompts and generate corresponding scenes.

8.2.1 Prompts

We assess the pipeline along two complementary axes: prompt specificity and scenario plausibility. This framework is informed by SOTA practices in which prompts are structured to test the model across a spectrum of detail, starting from vague to detailed, structured guidelines (rubric).

- **Prompt specificity:** Measures the model's robustness to variations in instruction detail [**?**, **?**].
  - *Vague prompts:* Underspecified instructions for generating urban scenarios, where the model must infer missing details. For example, "Generate an urban scenario with streets and buildings" leaves most design aspects open to the model.
  - *Criteria Specific prompts:* Prompts that specify the desired properties or evaluation criteria for the urban scenario, without detailing exact layouts or steps. For instance, "Generate an urban scenario that is realistic, has safe traffic flow, and includes diverse building types" guides the model toward satisfying these properties while leaving implementation and arrangement open.
  - *Detailed prompts (Rubric):* Fully specified instructions with explicit objectives, constraints, and optional step-by-step guidance for urban scene generation. This may include a rubric enumerating required features, such as "Generate an urban scenario with at least three traffic intersections with controlled signals, a sidewalk along the road and a crosswalk every 50 metres, and pedestrian pathways connecting all major areas. Agent arrives from the north, presses on the pedestrian light and then crosses the road when green."
- **Scenario plausibility:** Evaluates the realism, logical coherence, and potential risk of the model-generated tasks [**?**, **?**].
  - *Illogical scenarios:* Contain contradictory or impossible instructions to probe reasoning limitations . Example, "Create a scenario with no roads with busy traffic. Agent must cross using the crosswalk."
  - *Realistic scenarios:* Plausible, safe instructions representing typical interactions. Example, "Create a scenario with multiple cars and bicycles. Agent must wait for the pedestrian signal before crossing."
  - *Critical scenarios:* Edge-case or high-risk tasks revealing latent model limitations. Example, Jaywalking, Parking lot navigation, etc.

Prompt Grid Design To systematically explore model behavior, each task is tested using prompts that span the two axes.

Prompt Specificity	**Illogical**	**Realistic**	**Critical**
Vague	P1	P2	P3
Detailed	P4	P5	P6

8.2.2 User Studies

In addition to evaluation using predefined prompts, we conduct a user study to qualitatively assess the behavior of the system under open-ended, user-authored inputs.

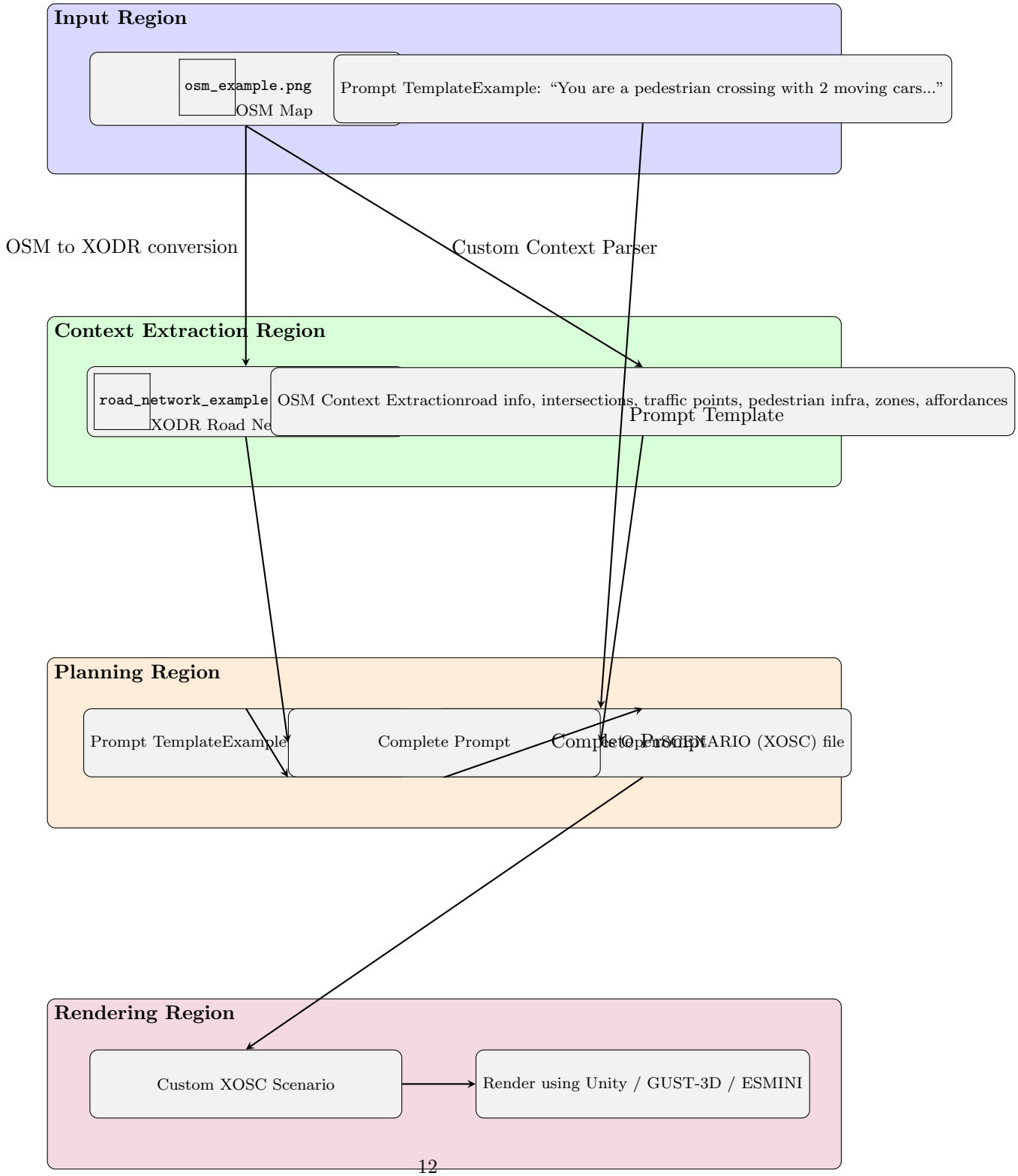


Figure 1: Overview of the guided user task scenario generation pipeline. **Input Region:** OSM map and Prompt Template examples. **Context Extraction Region:** XODR Road Network (with example image) and OSM Context Extraction. **Planning Region:** Prompt Template, Complete Prompt block, and LLM Planner combine inputs to generate OpenSCENARIO (XOSC) file. **Rendering Region:** Custom XOSC Scenario rendered with Unity / GUST-3D / ESMINI.

Appendix

9 Examples

9.1 Generated JSON Example

9.2 Meeting Notes

9.2.1 06/01/2026

- **OSM Ontology tree:** Look for the complete OSM ontology tree.
- **Architecture diagram:** Add the complete architecture diagram of the system from OSM procedural generation to Scenario generation.
- **Finalise the ontology for extending the XOSC:** Finalize the ontology needed to extend the OpenScenario format. Focus on 5 categories - Static objects for interaction, Dynamic Objects with no interaction, Dynamic objects with interactions, purely static objects and user-controlled.
- **Fix OSM generation:** Come up with a way to fix the orientations of the street furniture in OSM generations.
- **Use INRIA road network for reference:** Example, driving a car on the rue de lucioles.

9.2.2 14/01/2026

- **Schema:** Add the OpenScenario schema tree in the doc for reference.
- **Focus on Navigation:** Start with navigation tasks from A to B for validation.
- **Extend OpenScenario DSL:** Extend the DSL Actions and entities to support navigation scenarios.
- **Pharmacy example:** Low vision patients tend to confuse pharmacies with green light, a scenario for that could be interesting.
- **Finalize the LLM part:** Finalize how LLM can take a functional description and create a valid open scenario from it.

References

[1] Asam openscenario® dsl, 2024. Accessed: February 1, 2026.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] J. Deng, W. Chai, J. Huang, Z. Zhao, Q. Huang, M. Gao, J. Guo, S. Hao, W. Hu, J.-N. Hwang, et al. Citycraft: A real crafter for 3d city generation. *arXiv preprint arXiv:2406.04983*, 2024.

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

[5] P. Dourish. *Where the action is: the foundations of embodied interaction*. MIT press, 2001.

[6] K. Elmaaroufi, D. Shanker, A. Cismaru, M. Vazquez-Chanlatte, A. Sangiovanni-Vincentelli, M. Zaharia, and S. A. Seshia. Scenicnl: generating probabilistic scenario programs from natural language. *arXiv preprint arXiv:2405.03709*, 2024.

[7] W. Feng, W. Zhu, T.-j. Fu, V. Jampani, A. Akula, X. He, S. Basu, X. E. Wang, and W. Y. Wang. Layoutgpt: Compositional visual planning and generation with large language models. *Advances in Neural Information Processing Systems*, 36:18225–18250, 2023.

[8] Y. Feng, J. Jiang, J. Ren, W. Li, R. Li, and X. Fan. Text-guided editable 3d city scene generation. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.

[9] 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*, pages 63–78, 2019.

[10] M. Hartmann, M. Viehweger, W. Desmet, M. Stolz, and D. Watzenig. "pedestrian in the loop": An approach using virtual reality. In *2017 XXVI International Conference on Information, Communication and Automation Technologies (ICAT)*, pages 1–8. IEEE, 2017.

[11] 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. *Advances in neural information processing systems*, 36:3894–3920, 2023.

[12] L. Ling, C.-H. Lin, T.-Y. Lin, Y. Ding, Y. Zeng, Y. Sheng, Y. Ge, M.-Y. Liu, A. Bera, and Z. Li. Scenethesis: A language and vision agentic framework for 3d scene generation. *arXiv preprint arXiv:2505.02836*, 2025.

[13] K. Mukoya, E. Weng, R. Choudhury, and K. Kitani. Jaywalkervr: A vr system for collecting safety-critical pedestrian-vehicle interactions. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9600–9607. IEEE, 2024.

[14] B. Murugadoss, C. Poelitz, I. Drosos, V. Le, N. McKenna, C. S. Negreanu, C. Parnin, and A. Sarkar. Evaluating the evaluator: Measuring llms' adherence to task evaluation instructions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 19589–19597, 2025.

[15] M. Parmar, N. Patel, N. Varshney, M. Nakamura, M. Luo, S. Mashetty, A. Mitra, and C. Baral. Logicbench: Towards systematic evaluation of logical reasoning ability of large language models. *arXiv preprint arXiv:2404.15522*, 2024.

[16] R. Queiroz, T. Berger, and K. Czarnecki. Geoscenario: An open dsl for autonomous driving scenario representation. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 287–294, 2019.

[17] 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.

[18] S. Schneider and K. Bengler. Virtually the same? analysing pedestrian behaviour by means of virtual reality. *Transportation research part F: traffic psychology and behaviour*, 68:231–256, 2020.

[19] T. T. M. Tran, C. Parker, and M. Tomitsch. A review of virtual reality studies on autonomous vehicle–pedestrian interaction. *IEEE Transactions on Human-Machine Systems*, 51(6):641–652, 2021.

[20] Y. Wei, J. Wang, Y. Du, D. Wang, L. Pan, C. Xu, Y. Feng, B. Dai, and S. Chen. Chatdyn: Language-driven multi-actor dynamics generation in street scenes. *arXiv preprint arXiv:2412.08685*, 2024.

[21] H. Wu, D. H. Ashmead, H. Adams, and B. Bodenheimer. Using virtual reality to assess the street crossing behavior of pedestrians with simulated macular degeneration at a roundabout. *Frontiers in ICT*, 5:27, 2018.

[22] H.-Y. Wu, F. Robert, T. Fafet, B. Graulier, B. Passin-Cauneau, L. Sassatelli, and M. Winckler. Designing guided user tasks in vr embodied experiences. *Proceedings of the ACM on Human-Computer Interaction*, 6(EICS):1–24, 2022.

[23] H.-Y. Wu, F. A. S. Robert, F. F. Gallo, K. Pirkovets, C. Quere, J. Delachambre, S. Ramanoël, A. Gros, M. Winckler, L. Sassatelli, et al. Exploring, walking, and interacting in virtual reality with simulated low vision: a living contextual dataset (2023), 2023.

[24] F. Xu, Q. Lin, J. Han, T. Zhao, J. Liu, and E. Cambria. Are large language models really good logical reasoners? a comprehensive evaluation and beyond. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[25] 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*, pages 15459–15469, 2024.

[26] S. Zhang, M. Zhou, Y. Wang, C. Luo, R. Wang, Y. Li, Z. Zhang, and J. Peng. Cityx: Controllable procedural content generation for unbounded 3d cities. *arXiv preprint arXiv:2407.17572*, 2024.

[27] M. Zhou, Y. Wang, J. Hou, S. Zhang, Y. Li, C. Luo, J. Peng, and Z. Zhang. Scenex: Procedural controllable large-scale scene generation. *arXiv preprint arXiv:2403.15698*, 2024.

[28] K. Zhu, J. Wang, J. Zhou, Z. Wang, H. Chen, Y. Wang, L. Yang, W. Ye, Y. Zhang, N. Zhenqiang Gong, et al. Promptbench: Towards evaluating the robustness of large language models on adversarial prompts. *arXiv e-prints*, pages arXiv–2306, 2023.