Scenario Generation for Interactive Urban Environments

Paritosh Sharma, Hui-Yin Wu

September 2025

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 start by reviewing prior work on pedestrian-in-the-loop simulations. Then, we review recent works on generative models for urban scenario generation as well as diffusion-based

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

4D generation. Finally, we discuss common validation methods 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 Generative Models for Urban Scenario Generation

3.2.1 Image-based Generation

Diffusion models have shown impressive results in generating high-quality images from textual descriptions. Models like DALL-E 2 [?], Imagen [?], and Stable Diffusion [?] have demonstrated the ability to create diverse and realistic images based on text prompts. Recent works such as GameNGen [?], Genie [?], and DIAMOND [?] have shown the potential of diffusion models to generate game environments in real-time. However, these models suffer from latency issues since they are not truly 3D and generate image-by-image.

3.2.2 Prompts

Prompts are textual cues provided to the generative model to guide the scenario creation process. Since the popularity of LLMs like GPT-3, 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.

3.2.3 Layout-guided Generation

Layouts are structured representations of scenarios that capture the spatial relationships between different elements, such as buildings, roads, and pedestrians. These may include bird's-eye views,

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
3D-SceneDreamer [?]	Prompt to point-of-view image followed by image to 3D	3D Scene
GraphCanvas3D [?]	Uses LLM to create a scenegraph and then a 3D scene	3D Scene
Scenethesis [?]	Uses LLM to create a scenegraph and then a 3D scene	3D Scene
SceneX [?]	LLM to plan PCG (Procedural Controllable Generation)	3D Scene
Surreal Drivers [?]	Chain-of-thought prompts	3D Scene
Text2nerf [?]	Text prompts	3D Scene
X-Scene [?]	Text prompts	3D Scene

Table 1: Prompt-based Models for Urban Scenario Generation

top-down maps, or other forms of spatial representations that provide a high-level overview of the scenario. Table ?? lists some of the recent layout-based models for scenario generation.

Model	Technique	Output
CC3D [?]	2D Layout-based 3D Scene Generation	3D Scene
CityDreamer4D [?]	Uses a Layout Generator and a traffic scenario generator	3D Scenario
Infinicube [?]	Text prompts, HD Maps and Bounding Boxes	3D Scenario
Work by Zhang et al. [?]	BEV map	3D Scene
UniScene [?]	BEV map	Multi-view video
Savkin et al. [?]	Scenegraph	Scenario Image

Table 2: Layout-based Models for Urban Scenario Generation

Additionally, 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 scenario generation.

Model	Input Type	Output
CityX [?]	Prompt, OSM file or Semantic Map	3D Scenario
CityCraft [?]	Layout data and text prompts	3D Scene
Work by Liu et al. [?]	Scenegraph assisted using text prompts	3D Scene
GAUDI [?]	Conditioning using prompts or point-of-view images	3D Scene
MagicDrive3D [?]	Text prompts, Bird Eye View (BEV) map and 3D Bounding Boxes	Reconstructed 3D video
Scene123 [?]	Text prompt, point-of-view image or Text Description	3D Scene
StreetScapes [?]	BEV and height map with support for prompts	Video
Urban Architect [?]	3D Layout and Text Prompts	3D Scene
Urban World [?]	Layout (generation) and prompts (refinement)	3D Scenario
Wonderplay [?]	point-of-view image and action (physics)	3D Video

Table 3: Multimodal Models for Urban Scenario Generation

3.3 Validation

Table ?? lists common qualitative and quantitative validation methods used to evaluate the performance of generative models for urban road-crossing scenarios. These methods assess the quality, realism, and diversity of generated scenarios.

Table 4: Qualitative and Quantitative Evaluation Methods for Road-Crossing Scenarios

Method Type	**Evaluation Approach**	**Description**	**Use Case**
Qualitative	Human Review	Human experts assess realism, scenario diversity, and layout plausibility.	Validates human-perceived quality and applicability of the scene.
	Scenario Visualization	3D visual inspection or rendered videos showing pedestrian, vehicle, and environment interactions.	Helps detect unrealistic or unnatural behavior/configurations.
	Surveys	Collects subjective feedback on realism, perceived difficulty, or stress levels from participants.	Measures human-centric realism or emotional response.
	Comparison	Compare scenario features (traffic density, gap opportunities, layout) to real-world statistics or distributions.	Validates realism by matching key scenario characteristics to empirical data.
	Failure Cases	Identification and analysis of implausible or unsafe scenarios generated by the model.	Guides iterative model improvement.
Quantitative	Scenario Feature Statistics	Statistical analysis of scenario properties (traffic speed, number/duration of gaps, crosswalk presence, etc.).	Ensures generated scenarios span realistic distributions.
	Coverage and Diversity Metrics	Measures distributional entropy or spread of key attributes across all generated scenarios.	Assesses generalizability, scenario variety, and model's exploration of edge cases.
	Criticality and Opportunity Metrics	Quantifies the frequency and severity of challenging situations (number of safe gaps, minimum feasible gap size, "no-go" cases).	Evaluates risk/challenge spectrum in the scenario catalog.
	Sim2Real Gap (Domain Distance)	Computes metrics like KL-divergence, Earth Mover's Distance, t-SNE or FID/KID between generated and real scenario feature distributions.	Evaluates how closely synthetic scenarios match reality.
	Controllability Metrics	Measures how well the model can generate scenarios based on prompts (CLIP, BLIP, VQA, etc.)	Assesses controllability and flexibility of the generative model.

3.4 Code

Currently tested code includes:

3.4.1 Scenic

Easy to setup and run on colab since the repository is well maintained.

3.4.2 MagicDrive

Had issues running due to model size and GPU memory limitations. Also realised only 2D video generation works for now since the code is not uploaded yet for 3D generation.

3.4.3 MetaUrban/ScenarioNET

Easy to setup and run since docker container is available and working.

3.4.4 Threestudio

Threestudio [?] is a collection of 3D generative techniques. Easy to setup and run since docker container is provided. It is well maintained by the community and has a good documentation. Thus, good to test / refer different generative techniques for 3D.

3.4.5 City Dreamer

CityDreamer4D [?] tried testing the 3D branch with static scenes but encountered issues with the docker container.

3.4.6 Urban Architect

Urban Architect [?] - Working. Can generate semantic maps and depth maps from a layout input. Which then can be used to generate the 3D scene. However, the code has to be adapted to generate using multiple GPUs due VRAM limitations.

3.4.7 CLIP

CLIP [?] is a model that can be used to evaluate the quality of generated scenarios. Tested on colab with the pretrained ViT model.

3.4.8 GraphDreamer

GraphDreamer [?] works but generation takes over 30 hours for 1 scenegraph. However, cannot generate complex scenes.

4 Problem and Open Question

While recent generative models have demonstrated strong capabilities in synthesizing urban scenes from static layouts, they remain limited in their ability to model dynamic urban scenarios. In particular, they struggle to provide precise spatial and temporal control over objects and their interactions, which is essential for simulating a realistic traffic scenario.

This leads to the following open research question:

How can we design a generative model that can create dynamic urban scenarios with precise spatial and temporal control over objects and their interactions?

5 Work Plan

5.1 Model Architecture

Figure ?? illustrates the proposed model architecture for generating dynamic urban scenarios. The model takes a scenario prompt as input, which is processed by a multimodal large language model

(MLLM) to extract two key components: layout and tasks. Each component is then handled by specialized modules to generate the final 3D dynamic scene.

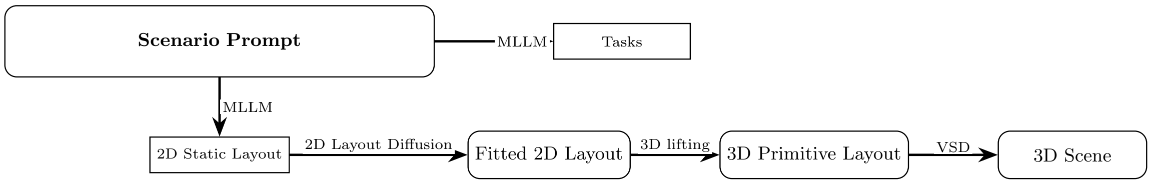


Figure 1: Model Architecture

5.2 Input and Splitting

Since no existing datasets provide paired text prompts with corresponding urban layouts and tasks, we propose leveraging the capabilities of MLLMs to parse and structure input prompts. Similar approaches have been explored in prior works, such as ChatDyn [?] for action planning, LayoutGPT [?] for layout generation, and LLM-grounded Diffusion [?] for LLM-conditioned image synthesis.

We first use a multimodal large language model (MLLM) (eg. Gemini) to process the input prompt and split it into the layout and tasks. The classes in the layout are based on the KITTI-360 [?] dataset. 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 goes here]

Generate a JSON object containing the following fields:

- **static_layout:**
  A 500×500 2D layout map represented as an array of objects. Each object must include:
  - `id`: unique identifier
  - `type`: one of {`pole`, `traffic sign`, `smallpole`, `lamp`, `trash bin`, `ground`, `road`, `sidewalk`, `parking`, `building`, `garage`, `fence`, `gate`, `vegetation`, `terrain`, `rail track`, `wall`, `box`, `vending machine`, `traffic light`, `rider`, `bicycle`, `motorcycle`, `motorbike`, `car`, `truck`, `bus`, `van`, `trailer`, `caravan`, `person`}
  - `position`: (x, y) coordinates in meters within the 2D plane
  - `orientation`: angle in degrees (0–360)
  - `size`: width and height in meters
  - `layer`: Layer classification for interactive scenarios. Choose from the default layers:
    - **Interactive layers:**
      - `"Container"`: Objects that can contain or hold other items (e.g., trash bins, boxes, bags)

        - "Support": Surfaces that can support objects being placed on them (e.g., tables, shelves, building roofs)
        - "Movable": Objects that can be moved or picked up (e.g., cars, people, bicycles, small objects)
        - "Interactable": Objects users can interact with or activate (e.g., traffic lights, buttons, switches, doors)
      - **Navigation layers:**
        - "Ground": Walkable surfaces (e.g., road, sidewalk, parking, ground)
        - "Wall": Vertical barriers (e.g., building walls, fences, barriers)
        - "Entryway": Passages and access points (e.g., doors, gates, entrances)
      - **Environment layers:**
        - "Light": Light sources (e.g., lamps, street lights)
        - "Camera": Viewpoints and surveillance (e.g., security cameras)
        - "State_object": Objects with changing states (e.g., animated elements)
        - "Animated": Objects with built-in movement (e.g., fountains, flags)

    Use default layers unless scenario requirements need custom combinations.

- **dynamic_layout:**
  - trajectories: Array of dynamic objects representing movement over time. Each object must include:
    - id: Unique identifier
    - type: One of {rider, bicycle, motorcycle, motorbike, car, truck, bus, van, trailer, caravan, person}
    - initial_position: (x, y) at time = 0
    - trajectory_description: Array of states; each state is of the form {"time": t, "position": [x, y]} (with t normalized between 0 and 1)
    - layer: Layer classification (usually "Movable" for dynamic objects)
- **tasks:**
  Before creating tasks, review the static_layout objects above.
  - 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"

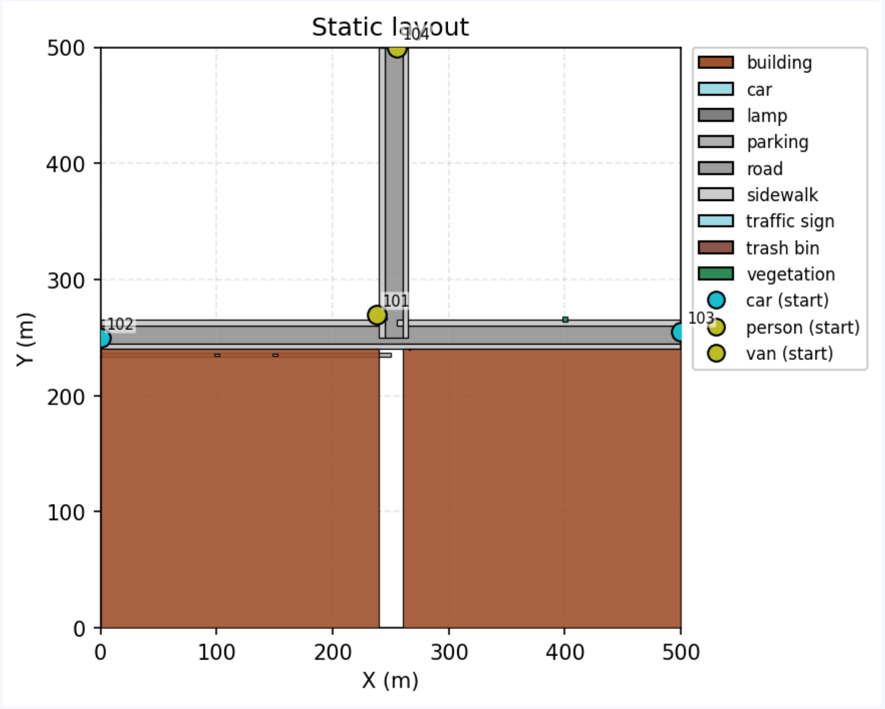
- `"place_on"`: Target must be `layer="Support"`
- `target`: Must match the exact `"type"` of an existing object in static_layout with the correct layer.
- `key_item`: Optional item needed (null if not needed)

* `constraints`: {`precedence`, `evaluation`}
* `help`: {`baseline`, `target`, `failure`}
* `failure_condition`: {`type`, `item`}
* `instructions`: {`text_en`, `text_fr`}
* `trigger`: {`object_id`, `function`}

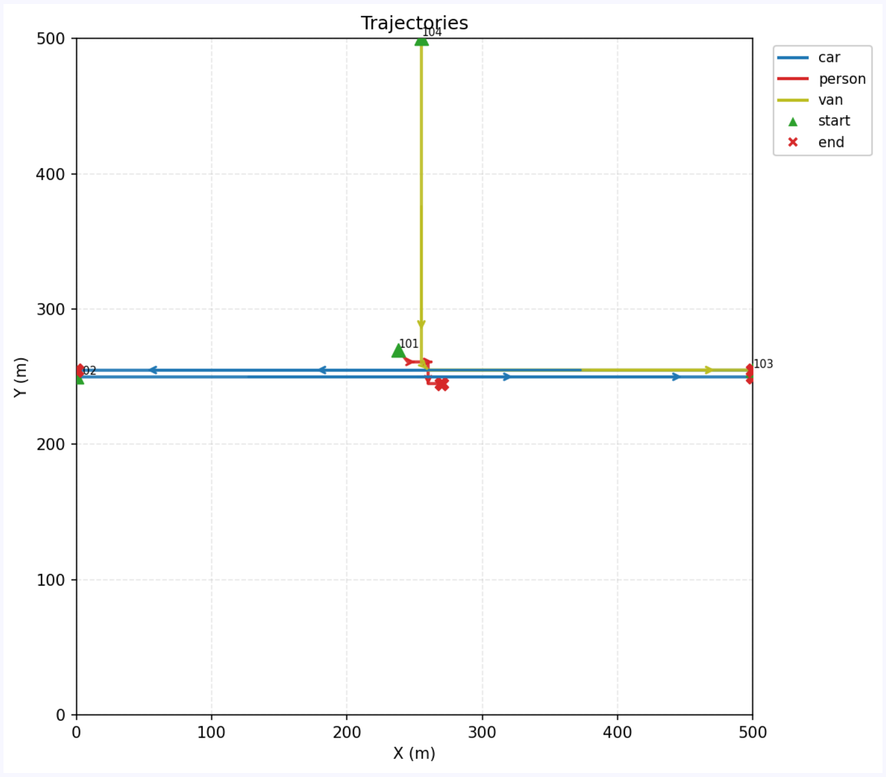
Input Example:

Afternoon at a three-way intersection with cars parked on the side. Cars arriving from all sides. Agent must yield and check for cars before crossing.

Output Example:
Static layout:



Trajectories:



Tasks:

- Walk from your starting position at [275, 275] to the edge of the crosswalk at [275, 260].
- Wait for the pedestrian signal to allow crossing the horizontal road.
- Cross the street to the opposite sidewalk, arriving at [275, 240].

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

5.3 Multinomial Diffusion for Semantic Map Generation

Input example:

Output example:

The next step is to generate a static 3D scene from the bouding boxes predicted by the MLLM.

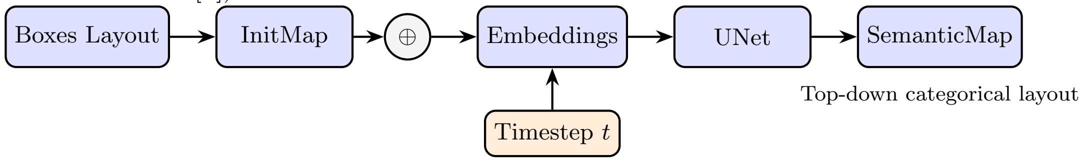
1. Preparation of Conditional Inputs Given the predicted bounding box layout predicted by the MLLM, we build a coarse instance-wise semantic segmentation map. We additionally define a *spatial prior mask* encoding object classes (traffic light, car, etc.) as a 2D array. Both the instance layout and the room mask are embedded via trainable embedding layers, yielding conditioning vectors that guide the subsequent diffusion process. The desired room type, if applicable, is also embedded and added to the timestep embedding.

2. Multinomial Diffusion for Layout Synthesis The core of our system is a multinomial (discrete) diffusion model adapted for semantic segmentation. At each denoising timestep t, a UNet backbone receives:

- the current noisy semantic map (as a categorical one-hot tensor),
- a sinusoidal positional embedding of the diffusion timestep.

The UNet predicts logits for categorical class distributions per pixel. During training, we minimize the variational bound (KL divergence) between the true and predicted posteriors as in [**?**]. During inference, the layout and mask embeddings provide strong spatial and semantic control, ensuring the generated layout adheres to both the LLM-predicted spatial arrangement and architectural constraints.

3. Output The final output is a top-down semantic map representing the static elements of the 3D scene to be generated. This map can then be used with score distillation sampling (as shown in Urban Architect [**?**]) to create the full 3D scene.



5.4 Tasks

The tasks predicted by the MLLM are a sequence of actions to be performed by the player.

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 MLLM (Gemini 2.5 pro for our case), the multinomial diffusion model 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 *Gemini 2.5 Pro* MLLM to process requests and manage data flow between the model and the Unity client. It exposes four primary `POST` endpoints:

- **/get_json** – Generates the initial layout or configuration in JSON format based on user input or scene specifications.

- **/refine_layout** – Refines the generated layout using the multinomial diffusion model.
- **/get_3d** – Produces 3D representations or assets produuced by Urban Architect [?].
- **/get_tasks** – Provides task-level metadata or scenario instructions to GUsT-3D

6.2 Frontend (Unity) Implementation

The frontend is developed as a part of 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.

Appendix

7 Examples

7.1 Layout JSON

```
{
  "static_layout": [
    {
      "id": 1,
      "type": "road",
      "position": [
        250,
        250
      ],
      "size": [
        500,
        20
      ],
      "orientation": 0,
      "layer": "Ground"
    },
    {
      "id": 2,
      "type": "road",
      "position": [
        250,
        250
      ],
      "size": [
        20,
        500
      ],
      "orientation": 90,
  "layer": "Ground"
},
{
  "id": 3,
  "type": "sidewalk",
  "position": [
    250,
    230
  ],
  "size": [
    500,
    10
  ],
  "orientation": 0,
  "layer": "Ground"
},
{
  "id": 4,
  "type": "sidewalk",
  "position": [
    250,
    270
  ],
  "size": [
    500,
    10
  ],
  "orientation": 0,
  "layer": "Ground"
},
{
  "id": 5,
  "type": "sidewalk",
  "position": [
    230,
    250
  ],
  "size": [
    10,
    500
  ],
  "orientation": 90,
  "layer": "Ground"
},
{
  "id": 6,
    "type": "sidewalk",
    "position": [
      270,
      250
    ],
    "size": [
      10,
      500
    ],
    "orientation": 90,
    "layer": "Ground"
  },
  {
    "id": 7,
    "type": "building",
    "position": [
      50,
      50
    ],
    "size": [
      100,
      100
    ],
    "orientation": 0,
    "layer": "Wall"
  },
  {
    "id": 8,
    "type": "building",
    "position": [
      450,
      50
    ],
    "size": [
      100,
      100
    ],
    "orientation": 0,
    "layer": "Wall"
  },
  {
    "id": 9,
    "type": "building",
    "position": [
      50,
      450
    ],
    "size": [
      100,
      100
    ],
    "orientation": 0,
    "layer": "Wall"
  },
  {
    "id": 10,
    "type": "building",
    "position": [
      450,
      450
    ],
    "size": [
      100,
      100
    ],
    "orientation": 0,
    "layer": "Wall"
  },
  {
    "id": 11,
    "type": "traffic light",
    "position": [
      240,
      220
    ],
    "size": [
      2,
      5
    ],
    "orientation": 0,
    "layer": "Interactable"
  },
  {
    "id": 12,
    "type": "traffic light",
    "position": [
      260,
      280
    ],
    "size": [
      2,
      5
    ],
    "orientation": 180,
    "layer": "Interactable"
  },
  {
    "id": 13,
    "type": "traffic light",
    "position": [
      220,
      260
    ],
    "size": [
      2,
      5
    ],
    "orientation": 270,
    "layer": "Interactable"
  },
  {
    "id": 14,
    "type": "traffic light",
    "position": [
      280,
      240
    ],
    "size": [
      2,
      5
    ],
    "orientation": 90,
    "layer": "Interactable"
  },
  {
    "id": 15,
    "type": "vegetation",
    "position": [
      150,
      150
    ],
    "size": [
      5,
      5
    ],
    "orientation": 0,
    "layer": "Environment"
  },
{
  "id": 16,
  "type": "vegetation",
  "position": [
    350,
    150
  ],
  "size": [
    5,
    5
  ],
  "orientation": 0,
  "layer": "Environment"
},
{
  "id": 17,
  "type": "vegetation",
  "position": [
    150,
    350
  ],
  "size": [
    5,
    5
  ],
  "orientation": 0,
  "layer": "Environment"
},
{
  "id": 18,
  "type": "vegetation",
  "position": [
    350,
    350
  ],
  "size": [
    5,
    5
  ],
  "orientation": 0,
  "layer": "Environment"
},
{
  "id": 19,
  "type": "ground",
  "position": [
      250,
      250
    ],
    "size": [
      20,
      20
    ],
    "orientation": 0,
    "layer": "Ground"
  }
],
"dynamic_layout": {
  "trajectories": [
    {
      "id": 101,
      "type": "car",
      "initial_position": [
        100,
        250
      ],
      "layer": "Movable",
      "trajectory_description": [
        {
          "time": 0.0,
          "position": [
            100,
            250
          ]
        },
        {
          "time": 0.5,
          "position": [
            250,
            250
          ]
        },
        {
          "time": 0.7,
          "position": [
            250,
            235
          ]
        },
        {
          "time": 0.9,
          "position": [
        265,
        250
      ]
    }
  ]
},
{
  "id": 102,
  "type": "car",
  "initial_position": [
    250,
    400
  ],
  "layer": "Movable",
  "trajectory_description": [
    {
      "time": 0.0,
      "position": [
        250,
        400
      ]
    },
    {
      "time": 0.5,
      "position": [
        250,
        250
      ]
    },
    {
      "time": 0.7,
      "position": [
        235,
        250
      ]
    },
    {
      "time": 0.9,
      "position": [
        250,
        265
      ]
    }
  ]
},
{
    "id": 103,
    "type": "bicycle",
    "initial_position": [
      300,
      230
    ],
    "layer": "Movable",
    "trajectory_description": [
      {
        "time": 0.0,
        "position": [
          300,
          230
        ]
      },
      {
        "time": 0.3,
        "position": [
          300,
          250
        ]
      },
      {
        "time": 0.6,
        "position": [
          250,
          250
        ]
      },
      {
        "time": 0.8,
        "position": [
          235,
          250
        ]
      }
    ]
  },
  {
    "id": 104,
    "type": "person",
    "initial_position": [
      300,
      215
    ],
    "layer": "Movable",
        "trajectory_description": [
          {
            "time": 0.0,
            "position": [
              300,
              215
            ]
          },
          {
            "time": 0.3,
            "position": [
              300,
              230
            ]
          },
          {
            "time": 0.6,
            "position": [
              300,
              250
            ]
          },
          {
            "time": 0.8,
            "position": [
              300,
              270
            ]
          }
        ]
      }
    ]
  },
  "tasks": [
    {
      "id": 1,
      "goal": {
        "type": "go",
        "target": "sidewalk_300.0_230.0",
        "key_item": null
      },
      "constraints": {
        "precedence": [],
        "evaluation": "arrive_at_target"
      },
      "help": {
```

```
      "baseline": "Head towards the sidewalk to wait for the cyclist.",
      "target": "Reach the designated waiting area.",
      "failure": "Do not enter the road until safe."
    },
    "failure_condition": {
      "type": "collision",
      "item": "car"
    },
    "instructions": {
      "text_en": "Go to the sidewalk at [300, 230] and wait.",
      "text_fr": "Allez sur le trottoir \u00e0 [300, 230] et attendez."
    },
    "trigger": {
      "object_id": 3,
      "function": "on_reach"
    }
  },
  {
    "id": 2,
    "goal": {
      "type": "interact",
      "target": "traffic_light_11",
      "key_item": null
    },
    "constraints": {
      "precedence": [
        1
      ],
      "evaluation": "light_green"
    },
    "help": {
      "baseline": "Observe the traffic light and wait for it to turn green.",
      "target": "Wait for the pedestrian signal to be green.",
      "failure": "Crossing on red is dangerous."
    },
    "failure_condition": {
      "type": "violation",
      "item": "traffic_light"
    },
    "instructions": {
      "text_en": "Wait for the traffic light at [240, 220] to turn green before proceeding.",
      "text_fr": "Attendez que le feu de circulation \u00e0 [240, 220] passe au vert."
    },
    "trigger": {
      "object_id": 11,
      "function": "on_light_green"
    }
  },
  {
    "id": 3,
    "goal": {
      "type": "go",
      "target": "road_250.0_250.0",
      "key_item": null
    },
    "constraints": {
      "precedence": [
        2
      ],
      "evaluation": "arrive_at_target"
    },
    "help": {
      "baseline": "Cross the intersection carefully.",
      "target": "Cross the road safely.",
      "failure": "Avoid vehicles."
    },
    "failure_condition": {
      "type": "collision",
      "item": "car"
    },
    "instructions": {
      "text_en": "Cross the intersection at [250, 250] when the light is green.",
      "text_fr": "Traversez l’intersection \u00e0 [250, 250] lorsque le feu est vert."
    },
    "trigger": {
      "object_id": 19,
      "function": "on_reach_road"
    }
  },
  {
    "id": 4,
    "goal": {
      "type": "go",
      "target": "sidewalk_270.0_250.0",
      "key_item": null
    },
    "constraints": {
      "precedence": [
        3
      ],
      "evaluation": "arrive_at_target"
    },
      "help": {
        "baseline": "Continue across the intersection to the other side.",
        "target": "Reach the other sidewalk.",
        "failure": "Stay on the pedestrian path."
      },
      "failure_condition": {
        "type": "collision",
        "item": "car"
      },
      "instructions": {
        "text_en": "Continue to the sidewalk at [270, 250].",
        "text_fr": "Continuez vers le trottoir \u00e0 [270, 250]."
      },
      "trigger": {
        "object_id": 6,
        "function": "on_reach"
      }
    }
  ]
}
```

8 Misc

8.1 Meeting Notes

8.1.1 01/08/2025

- **Clarify Image Usage:** Specify what kinds of images are being referred to in the input for the models (e.g., diagrams, real-world photos, layouts).
- **Add details on scenegraph:** Provide more information about the scenegraphs (e.g., format, usage, etc.) in the context of the models.
- **Avoid Overly High-Level Descriptions:** Some explanations are too abstract.
- **Improve Focus on Vocabulary:** Review and refine terminology. Ensure technical or domain-specific terms are clearly defined and used consistently.
- **Identify and Specify Missing Interaction Types:** Clearly outline which user/system interactions are missing or underexplored.
- **Sharpen Research Question (RQ):** Make the RQ more concrete.
- **Better Categorization of Literature:** Reorganize cited papers using clear categories such as research themes, methodologies, etc.

8.1.2 14/08/2025

- **RQ still too high level:** Provide more information about the scenegraphs (e.g., format, usage, etc.) in the context of the models.
- **Clarify the missing pieces in the SOTA:**
- **How to address:** diversity -¿ generative models, realism -¿ ontology, dynamic -¿ tasks
- **Start with WorkPlan:**
- **Remove relation between GUsT-3D and generation**

  #### 8.1.3 04/09/2025

  - **Semantic Information Missing:** Find a way to match the scene to Kitti's prior.
  - **Input unclear:** Scenegraph requires too much information. Need to simplify the input.
  - **Use Layout:** Graphdreamer based score distillation doesn't work, use layout instead.

8.1.4 29/09/2025

- **Organize docs and notes:** Clarify the layout preferably with images.
- **Add missing citations:** Add missing citations for stuff like scene background generation in the workplan.
- **Detail:** Detail the missing sections more.
- **Git:** Add the code to gitlab.

8.1.5 08/10/2025

- **Validate with more prompts:** Validate the system with additional prompts - realistic, illogical, variations of the same prompts, prompts that recreate Florent's scenarios, decomposing prompts which are "difficult" for a pedestrian (jaywalking, parking lot entrance), etc.
- **Add in the doc:** the prompts discussed in the meeting, diffusion architecture, more details on validation, add examples for failure conditions, SOTA on prompt engineering for multimodal content.

8.2 Urban Scenario Generation

This section discusses pre-existing models used for generating scenarios in urban environments. These models can be broadly classified into procedural and deep generative approaches.

8.2.1 Procedural Methods

Procedural methods leverage algorithms and rules to generate urban scenarios, often resulting in highly customizable environments. They can be further divided into two categories:

Classic Procedural Generation use rules or constraints to generate scenes. The most popular procedural approach used for urban scenarios is Scenic [?], which allows users to define scenarios using a probabilistic programming language. Similarly, MetaUrban [?] is a popular urban scenario generation framework to create urban micromobility scenarios using the Metadrive [?] simulator.

LLM-based Procedural Generation can be used to enhance procedural generation with natural language understanding. For example, ScenicNL [?] and ChatScene [?] use LLMs to generate scenic code from prompts and then define the scenarios in Scenic. TTSG[?] is another framework that uses an to plan a traffic scenario using an LLM in JSON and then render it using CARLA [?]. CityX [?] is multi-agent framework that uses LLM to assemble assets using the procedural content generation (PCG) library as well as plan actions for the agents in the scenario.

Even though procedural approaches can be highly customizable and allow for precise control over the generated scenarios, they typically suffer from a lack of realism, as they rely on predefined rules that may not capture the full complexity of real-world environments. However, they can be useful in creating diverse scenarios that adhere to specific constraints or requirements. For example, a scenario with a crossroad with a specific number of lanes, a certain type of road surface, and a defined layout of buildings. However, no real crossroad would exist in the world that conforms to these specifications.

8.2.2 Deep-Generative Methods

Deep-Generative Approaches have recently become popular for various sorts of 3D generation. To create a complete scenario, some techniques use generative models such as GANs, VAEs or Diffusion models in combination with procedural techniques. These can be broadly classified into the following categories:

Procedural Environments with Deep-Generative Dynamics generate the static aspects of the environment using procedural techniques, while the dynamic aspects such as vehicles and pedestrians are generated using generative models. For example, Chatdyn [?] uses CARLA [?] to construct the traffic environment, then populate it with pedestrians and vehicles, each equipped with an LLM agent to generate a high-level scenario plan. This high-level plan is then executed in a low-level PedExecutor using Text2Motion [?] for pedestrians and VehExectutor using a physics-based, history-aware reinforcement learning controller to produce vehicle trajectories.

Procedural Dynamics with Deep-Generative Environments generate the dynamic aspects such as vehicles and pedestrians using procedural techniques. UrbanWorld [?] converts 2.5D urban layout data into a structured 3D city with separated assets, applying depth-aware, multi-view diffusion-based texture rendering and UV inpainting to achieve high-fidelity visuals. It also uses an urban MLLM for designing the world and provides dynamic elements which are planned using a random tree path planning algorithm. CityDreamer4D [?] modularly integrates autoregressive token-based generation, neural rendering (e.g., NeRF-style volumetrics), and procedural traffic modeling to synthesize large-scale, time-varying 3D cities.

Complete Deep-Generative Approaches recreate the static as well as the dynamic aspects using the learnt representation. Infinicube [?] constructs a large, dynamic 3D voxel world from input HD maps, vehicle positions, and text prompts; then, it generates photorealistic driving videos and reconstructs the scene into a manipulable 3D environment by fusing voxel- and video-based representations. UniScene [?] UniScene generates driving scenes in three modalities—semantic occupancy, multi-view video, and LiDAR—by first producing a controllable, temporally consistent occupancy sequence from BEV layouts. This occupancy then serves as unified geometric–semantic guidance to synthesize realistic videos and point clouds, ensuring cross-modal consistency and editability.

8.3 Scenegraph-Controlled Diffusion

Despite the advancements in urban scenario generation, very few methods have explored the use of scenegraphs to control the generation process. In recent years, the achievements in text-to-image generation has enabled the advancement of text-to-3D generation using Score Distillation Sampling (SDS) [?] which optimizes a 3D model by aligning 2D images rendered at arbitrary view-points with the distribution derived from a text-conditioned diffusion model. Subsequent works including ProlificDreamer [?] introduced Variational Score Distillation (VSD) which addresses the over-regularization and mode collapse issues of SDS by introducing a variational formulation that jointly optimizes the 3D representation and a learnable Gaussian noise distribution, enabling more faithful geometry and richer texture generation. GraphDreamer [?] uses the the SDS process with scenegraphs to enable more structured and controllable scene generation. However, it uses a simplified static scenegraph representation as shown in Visual genome [?] in which, nodes represent the objects in the scene, edges represent the relationship between these nodes and attriubtes represent the properties of the node. Example, Elderly (attribute) man (Node) wearing (edge) a hat(node). This limits the ability to generate dynamic scenarios where the relationships between objects change over time.

8.4 Scenegraph-guided Generation

Scenegraph-guided 3D generation is the process of creating 3D environments by leveraging scene graphs, which are structured representations of the objects and their relationships within a scene. A major advantage of scenegraph-guided generation is that it provides a clear relationship between the elements which allows for better generation than other techniques such as simple text prompts or bounding boxes. Scenethesis [?] uses a Vision-Language Model (VLM) to create a scenegraph with parent-child relationships and localizes objects using 3D bounding boxes. Work by Liu et al. [?] defines scenegraphs as graphs where instance nodes represent countable objects with semantic and positional features, a singleton road node encodes global scene structure, and edges capture both physical proximity among instances and connectivity to the road. X-Scene [?] is another work that uses LLMs (Large-Language Models) to create a scenegraph with nodes (objects) and edges (relationships) to facilitate the generation process. Graphdreamer [?] employs scenegraphs structured around the Visual Genome [?] format, where nodes represent objects with associated attributes, and edges encode the relationships between these objects to guide the generation process. Despite the advancements in scenegraph-guided generation, all of these works focus on static scenes and do not consider dynamic scenarios where the state of the objects changes over time.

8.5 Grounding: Input for Scene Generation

Generative models have already been applied to urban scenario generation, where models synthesize plausible urban environments, pedestrian layouts, or vehicle positions from various types of input. An input in the context of a generative model is any data—such as a prompt, image, or scenario graph—provided to influence the output. Grounding, on the other hand, is the process of linking elements of that input to specific, coherent representations in the generated scenario, such as ensuring the ”footpath” appears visually plausible, is correctly positioned on the ”road”, and respects spatial relationships or physical constraints. Grounding ensures the generated scenario isn't just randomly composed but meaningfully aligned with the intended semantics of the input. The common ways to do grounding are,

8.5.1 Rules and Constraints

Rules can be used to give a prescriptive logic that defines how to generate or modify a scenario. Rule-based systems are necessarily procedural, meaning they follow a set of predefined steps or algorithms to create scenarios. Table **??** lists some of the recent rule-based models for scenario generation. Although these systems allow users to define rules for generating scenarios—such as the placement of buildings, roads, and other elements—they often require an intermediate representation which captures the real-world environments. Creating such representations can be challenging, as it requires a deep understanding of the underlying rules and relationships between different elements. Additionally, rule-based systems can be limited in their ability to generate diverse and realistic scenarios, as they rely on predefined rules that may not capture the full complexity of real-world environments.traffic

Model	**Technique**	**Output**
CityEngine [**?**]	Procedural Modeling with Rules	3D scene
Infinigen [**?**]	Procedural Modeling with Rules	3D scene
MetaUrban [**?**]	Description Scripts for Scene Layout	3D Scenario

Table 5: Rule-based Models for Urban Scenario Generation

Constraints define conditions that must be satisfied to get a valid scenario. Scenic [**?**] is a probabilistic programming language that allows users to define constraints for generating scenarios. It uses a declarative approach to specify the properties of the scenario, such as the layout, objects, and their relationships. These can then be rendered using a frontend such as CARLA [**?**]. However, it suffers from the same limitations of rule-based systems, as it can only generate scenarios that fit within the defined constraints, potentially missing out on the richness and variability of real-world environments.

8.6 Diffusion-based 4D Generation

Diffusion models have emerged as a powerful approach for generating high-quality 3D content by iteratively refining a random noise input into a coherent output. DreamFusion [**?**] initially introduced text-to-3D synthesis by optimizing a neural radiance field so that rendered views, when noised and denoised by a pretrained text-conditioned diffusion model, produce identical gradients via Score Distillation Sampling (SDS). This framework was extended to dynamic scenes in MaV3D, which employs video-based SDS to animate a time-conditioned radiance field into a 4D scene [**?**]. However, MaV3D's reliance on a single diffusion prior leads to trade-offs between appearance fidelity, 3D consistency, and motion realism. 4D-fy [**?**] addresses this by hybridizing three SDS signals—3D-aware image SDS for geometry, standard image SDS for texture, and video SDS for motion—alternating updates to preserve all three qualities. Animate124 [?] further refines single-image animation into 4D using a coarse-to-fine 4D grid backbone optimized first with 2D and 3D image priors, then with video diffusion, and finally with a personalized ControlNet fine-tuning stage to prevent semantic drift. More recently, Trans4D [?] leverages a multimodal large language model to perform physics-aware 4D scene planning—generating object trajectories, rotations, and transition times—and introduces a Transition Network that predicts point-wise appearance or disappearance probabilities to realize complex geometry-aware transitions such as a missile transforming into an explosion cloud. Despite their impressive generative capabilities, these diffusion-based 4D synthesis methods still lack the explicit, structured control afforded by scene graphs, making it difficult to enforce complex object relationships or constraints at generation time.

8.7 Datasets

In this section, we review the datasets that are used to train generative models for road crossing scenarios. Table **??** lists some of the datasets commonly used for training and evaluating generative models in urban environments.

Table 6: Overview of selected datasets for foundation model-based scenario generation and analysis.

Dataset	**Year**	**View**	**Source**
SIND [?]	2022	BEV	Real
Waymo Open [?]	2020	FPV	Real
Argoverse [?][?]	2023	BEV/FPV	Real
nuScenes [?]	2022	FPV	Real
KITTI [?]	2012	FPV	Real
Cityscapes [?]	2016	FPV	..
HoliCity [?]	2020	FPV	..
OmniCity [?]	2023	FPV	..
GoogleEarth [?]	2024	BEV	Real
OSM [?]	2024	BEV	Real
CarlaSC [?]	2022	BEV/FPV	Synthetic
CityTopia [?]	2025	BEV/FPV	..

8.8 Input Scenegraph

We represent a temporal (dynamic) scenegraph as a tuple

$$S = (V, E, \mathcal{T}, \tau) \tag{1}$$

where

- $V = \{v_i\}_{i=1}^{N}$ is a finite set of nodes (objects). Each node v_i is itself a tuple

$$v_i = \big(\mathrm{id}_i, \mathcal{A}_i^{\text{static}}, \{(I_{i,k}, \mathcal{A}_i^{\text{dynamic}}|_{I_{i,k}})\}_{k=1}^{K_i}\big), \tag{2}$$

where

- id_i is a unique identifier (string),
- $\mathcal{A}_i^{\text{static}}$ is the attribute map for invariant attributes:

$$\mathcal{A}_i^{\text{static}} : K_{\text{static}} \rightharpoonup V,$$

  e.g., type, material,
- $\{(I_{i,k}, \mathcal{A}_i^{\text{dynamic}}|_{I_{i,k}})\}$ is an ordered set of temporal states where each

$$I_{i,k} = [t_{i,k}(0), t_{i,k}(1)] \subseteq \mathcal{T}$$

  is a closed time interval, and $\mathcal{A}_i^{\text{dynamic}}|_{I_{i,k}}$ are the dynamic attribute values valid for that interval (e.g., position, lid state, traffic-light color).

- $E \subseteq V \times V$ is a set of directed edges. Each edge $e = (v_i, v_j)$ is augmented by

$$e = (v_i, v_j, r_{ij}, C_{ij}), \tag{3}$$

  where

  - v_i is the source node,
  - v_j is the target node,
  - r_{ij} is the relation type (e.g., "next_to", "controls"),
  - C_{ij} is an optional constraint function mapping dynamic attributes of v_i to changes in v_j (e.g., if traffic light changes to red, car stops).

- $\mathcal{T}$ is the global time domain.
- $\tau : \bigcup_i \mathcal{S}_i \cup \bigcup_{i,j} \mathcal{I}_{ij} \to \mathcal{P}$ is an optional grounding map to represent the dynamic aspects w.r.t. time.

Semantics A dynamic scenegraph S encodes both

1. the *static layout* through static attributes $\mathcal{A}_i^{\text{static}}$ and relations r_{ij} that are continuously active (or active on $\mathcal{T}$),
2. the *dynamic behaviour* through temporal attributes $\mathcal{A}_i^{\text{dynamic}}$ and constraint relations C_{ij} between nodes.

Example

Figure **??** illustrates the example dynamic scenegraph.

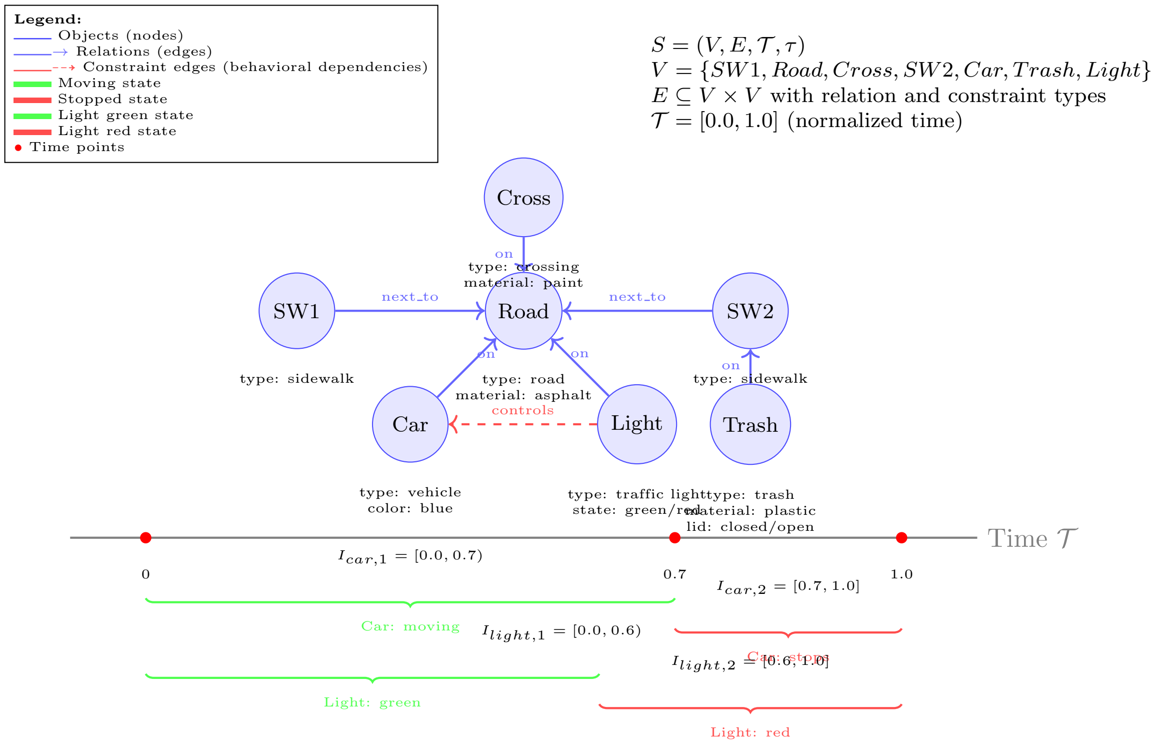


Figure 2: Example dynamic scenegraph with constraints (nodes, relations with active intervals, temporal states, and dependency edges).