Real-to-Sim Generation: Synthesizing Scenario Programs from Real-World Data via Constraint Solving

PEISHAN HUANG*, National University of Defense Technology, China
WENMENG ZHANG*, National University of Defense Technology, China
YUSEN CHEN*, National University of Defense Technology, China
ZHENBANG CHEN*†, National University of Defense Technology, China

The demand for synthetic training data is hindered by the sim-to-real gap, as current data-driven and LLM-based generators often produce physically implausible scenarios. To address this, we propose R2SGEN, a Real-to-Sim framework that synthesizes structured scenario programs from real-world data. To overcome the combinatorial explosion and intractability of monolithic Satisfiability Modulo Theories (SMT) encoding, we introduce a decoupled synthesis strategy. This approach separates the discrete structural program search from continuous geometric resolution using lightweight, atomic SMT constraints. Furthermore, we significantly accelerate the search process by integrating two tailored pruning mechanisms: Common Prefix Abstraction-based pruning for Breadth-First Search and Branch-and-Bound for Depth-First Search. We evaluate R2SGEN on 20 real-world scenes of varying complexity from the nuScenes dataset. Experimental results show that our method guarantees consistency with the input scene and produces substantially lower-cost programs than the LLM-based baselines under the evaluated inputs. Both proposed search paradigms exhibit complementary advantages, proving highly efficient and scalable for high-complexity synthetic data generation.

CCS Concepts: • **Software and its engineering → Automatic programming**.

Additional Key Words and Phrases: Program Synthesis, Scenario Generation, Scenic, SMT

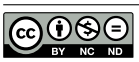
ACM Reference Format:
Peishan Huang, Wenmeng Zhang, Yusen Chen, and Zhenbang Chen. 2026. Real-to-Sim Generation: Synthesizing Scenario Programs from Real-World Data via Constraint Solving. *Proc. ACM Program. Lang.* 10, OOPSLA2, Article 395 (October 2026), 31 pages. https://doi.org/10.1145/3839527

1 Introduction

Modern autonomous-driving systems require large volumes of visual data for training, yet collecting such data in the real world is costly and difficult to scale [Caesar et al. 2020; Sun et al. 2020]. Synthetic data therefore provides a practical alternative for expanding training datasets [Dosovitskiy et al. 2017; Rong et al. 2020]. Currently, image data generation primarily relies on neural network-based methods. However, these approaches exhibit significant limitations: their "black-box" nature lacks interpretability during the generation process, making it difficult to achieve fine-grained,

*Also with the affiliation: State Key Laboratory of Complex & Critical Software Environment, National University of Defense Technology, Changsha, China.
†Zhenbang Chen is the corresponding author.

Authors' Contact Information: Peishan Huang, College of Computer Science and Technology, National University of Defense Technology, Changsha, Hunan, China, huang_ps@nudt.edu.cn; Wenmeng Zhang, College of Computer Science and Technology, National University of Defense Technology, Changsha, Hunan, China, wenmengzhang@nudt.edu.cn; Yusen Chen, College of Computer Science and Technology, National University of Defense Technology, Changsha, Hunan, China, chenyushen21@nudt.edu.cn; Zhenbang Chen, College of Computer Science and Technology, National University of Defense Technology, Changsha, Hunan, China, zbchen@nudt.edu.cn.



ACM 2475-1421/2026/10-ART395
https://doi.org/10.1145/3839527

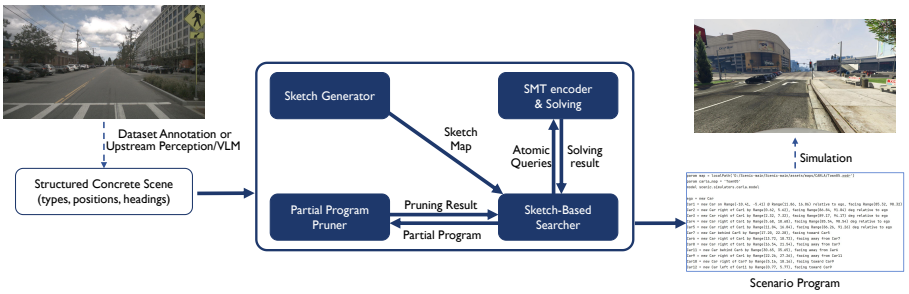


Fig. 1. Overview of R2SGen. Scene images are from the nuScenes dataset [Caesar et al. 2020], © Motional AD Inc., licensed under CC BY-NC-SA 4.0.

customized control over specific semantic elements within a scene. In contrast, Scenic [Fremont et al. 2019], a domain-specific language (DSL) tailored for scenario generation, demonstrates remarkable advantages in constructing complex driving scenarios due to its programmability and determinism.

Despite its powerful capabilities, Scenic's nature as a DSL requires developers to possess deep domain expertise and manually write code, a process that is not only inefficient but also incurs high labor costs. Recently, some studies [Elmaaroufi et al. 2024; Zhang et al. 2024] have attempted to leverage Large Language Models (LLMs) to automatically synthesize Scenic programs via natural language prompts. Nevertheless, constrained by the inherent hallucination issues of LLMs and the ambiguity of natural language descriptions, existing methods struggle to guarantee the correctness of the generated programs, let alone formally verify whether the synthesized Scenic programs strictly conform to the task specifications. Consequently, there is an urgent need to develop a novel technical framework. This framework should automatically synthesize scenario programs that are consistent with real-world scene data. Furthermore, it must provide rigorous formal verification capabilities to ensure that the semantics of the finally generated code conform to the task specifications.

Our key insight is to formulate scenario generation as a Programming by Example problem. Given a concrete scenario from real-world data, comprising precise positions, headings, and types of all entities, our goal is to automatically synthesize a scenario program. This synthesized program must satisfy two requirements: first, it should describe the input scenario, meaning the program's semantics must encompass the given concrete configuration; second, it should generalize beyond the single input example, enabling the generation of diverse scenario variants. We therefore propose R2SGen (Real-to-Sim Generation), an automated framework for scenario program synthesis from concrete scenes shown in Fig. 1. Our framework takes as input a structured concrete scene, *i.e.*, a set of entities annotated with their types, positions, and headings, as commonly provided by public datasets (*e.g.*, nuScenes) or by upstream perception systems or vision-language models (VLMs), and produces as output a structured scenario program in Scenic that abstracts the concrete coordinates into reusable spatial relations and parameterized ranges, which can then be rendered into diverse synthetic images for training autonomous driving systems. However, realizing this synthesis poses two key challenges.

The first challenge is verifying program correctness. The DSL describes scenarios using spatial relationships such as "ahead of", but the input is given as exact numeric coordinates. Determining whether a candidate program correctly captures the input requires automatically reasoning about whether these relational descriptions are satisfied by the concrete coordinates. For instance, given the traffic snapshot in Fig. 2a, one could trivially generate programs using absolute coordinates or numerical offsets (top and middle of Fig. 2b). However, these approaches simply encode the specific numerical values without capturing the underlying spatial relationships, such as "car3 is ahead of

Hypothetical semantic information:

ego: (0, 0, 0°) car2: (3, 5, 0°)
car1: (3, 8, 0°) car3: (0, 5, 0°)

(a) Real-world scene image from the nuScenes dataset [Caesar et al. 2020] (© Motional AD Inc., licensed under CC BY-NC-SA 4.0) and the assumed corresponding semantic information.

```
## Program with absolute coordinates
car1= new Car on 3@8, facing 0 deg
car2= new Car on 3@5, facing 0 deg
car3= new Car on 0@5, facing 0 deg

## Program with relative numerical offsets
car1= new Car on 3@8 relative to ego,
    facing 0 deg relative to ego
car2= new Car on 0@-3 relative to car1,
    facing 0 deg relative to car1
car3= new Car on -3@0 relative to car2,
    facing 0 deg relative to car2

## Program capturing the scene layout
car3= new Car ahead of ego by Range(3,8),
    facing away from ego
car2= new Car right of car3 by Range(1,6),
    facing Range(-5,5) deg relative to ego
car1= new Car ahead of car2 by Range(1,6),
    facing away from car2
```

(b) Valid scenario programs describing the given structured scene.

Fig. 2. Comparison of spatial representations for a given scene. (**a**) A real-world scene image from the ego vehicle's perspective (top) and the assumed corresponding semantic information (bottom). (**b**) Three variations of valid scenario programs describing the given scene: progressing from rigid absolute coordinates and relative numerical offsets, to the optimal target program that captures the inherent relational structure.

ego". Such rigid encodings are scene-specific and cannot generalize: a program encoding "car3 at (0, 5)" works only for that exact coordinate, whereas a program encoding "car3 ahead of ego by Range(3,8)" captures a reusable spatial pattern that can generate variations. Our goal is to synthesize programs like the bottom example in Fig. 2b, which abstracts the scene into relational primitives (*e.g.*, *ahead of* and *right of*) and parameterized ranges, enabling the program to generate structurally similar scenarios beyond the single observed instance. We draw inspiration from prior work [Kim et al. 2022], which encodes scenario programs into SMT constraints to validate their semantic alignment with real-world data. While their approach focuses on querying data instances that match a specific pre-defined program, we invert this paradigm to synthesize a scenario program directly from real-world data. We encode the geometric semantics of the DSL as satisfiability constraints and leverage SMT solvers to verify program consistency during the synthesis process. Since our synthesized programs are grounded in reality by design, they are inherently realistic, effectively mitigating the sim-to-real gap without requiring post-hoc validation.

The second challenge is the vast combinatorial search space. Synthesizing scenario programs that capture relational structures faces a severe combinatorial explosion. A fundamental constraint is the dependency order: the position and orientation of any entity must be defined relative to the global frame or an already declared entity. Therefore, beyond merely solving for specific geometric relationships (*e.g.*, *ahead of* or *facing toward*), we must also determine which entity serves as the reference object. Consequently, based solely on the permutations of entity declarations, the search space complexity scales factorially ($O(N!)$). Finding the optimal program in this vast space becomes computationally intractable as the number of entities increases. While one might intuitively encode the entire problem into a single, monolithic SMT formula, our empirical results show that the complexity of this joint discrete-continuous space causes solvers to fail even on simple three-entity scenes. To overcome this, we decouple the synthesis process, fundamentally separating its discrete and continuous dimensions. We delegate discrete structural decisions, such as declaration ordering

and relational operators, to an explicit search algorithm, while utilizing the SMT solver strictly as a geometric oracle for continuous constraints. By incrementally constructing the program via lightweight, atomic SMT queries, this decoupled approach makes the problem navigable, although the combinatorial space remains large. To mitigate this combinatorial explosion, we introduce two tailored pruning strategies. First, **Common Prefix Abstraction-based Pruning**, primarily for **Breadth-First Search (BFS)**, merges equivalent states from distinct declaration orders to eliminate redundant branches. Second, **Branch-and-Bound Pruning**, exclusively for **Depth-First Search (DFS)**, discards a partial branch when its accumulated cost is no lower than the cost of the best complete program found so far.

We have implemented R2SGen in a prototype tool and evaluated it on 20 real-world scenes of varying complexity sourced from the nuScenes [Caesar et al. 2020] dataset. Under the evaluated inputs, R2SGen achieves complete syntax and semantic correctness and synthesizes substantially lower-cost programs under our cost function, whereas the semantically correct baseline outputs primarily encode absolute coordinates. Furthermore, the efficiency evaluation validates the absolute necessity of our decoupled architecture and pruning strategies. Monolithic SMT encodings and naive combinatorial searches are fundamentally intractable, failing even on minimal configurations. In contrast, our tailored BFS and DFS search paradigms efficiently navigate the combinatorial explosion, successfully scaling to highly complex traffic scenes with up to 24 entities. In summary, the contributions of this paper are as follows.

- We propose a scenario program synthesis technique that, given a structured concrete scene, automatically generates a structured scenario program. This program replaces rigid numerical coordinates with symbolic spatial relations, precisely capturing the underlying geometric layouts and relational constraints of the real world.
- We introduce a decoupled synthesis strategy to overcome intractability of monolithic SMT encoding. By separation, we resolve continuous geometric relations via lightweight, atomic SMT constraints, while significantly accelerating discrete program search using two tailored pruning mechanisms: **Common Prefix Abstraction** for BFS and **Branch-and-Bound** for DFS.
- We conduct extensive evaluations on 20 diverse real-world scenes sourced from the nuScenes dataset. The results demonstrate our approach's high computational efficiency and scalability, successfully generating valid scenario programs for highly complex environments with up to 24 entities where existing baselines fail.

2 System Overview

In this section, we present the overall architecture of R2SGen, as illustrated in Fig. 1, and detail the functionality of its key components.

Structured Concrete Scene. To bridge the sim-to-real gap, R2SGen starts from a structured concrete scene. As illustrated in Fig. 2a, the input describes a real-world traffic scene containing three entities, each annotated with its type, position, and heading. Such a structured scene can be obtained from annotated datasets or from upstream perception/VLM frontends. R2SGen synthesizes a generalized program from this input, which can subsequently be executed by a simulator to generate diverse scenario variations that preserve the core spatial relations of the original scene.

Expected Output. As an important scenario description language, Scenic is compatible with multiple simulators and can produce corresponding scene images through simulation. Therefore, R2SGen targets the synthesis of Scenic program that captures the spatial layouts between entities in the input scene, enabling the generation of additional images that preserve these spatial relations via these simulators. Our approach focuses on a core subset of Scenic. Rather than merely transcribing hard-coded numerical coordinates, our approach aims to discover the most natural relational dependencies (*e.g.*, ahead of, facing toward) to describe the scene layout. Given an input scene,

our approach synthesis a Scenic program such that the input scene lies within the scene space defined by that program. As illustrated in Fig. 2b, while the scene in Fig. 2a can be represented by various valid programs, R2SGen is designed to search for and synthesize the most structured representation. Specifically, R2SGen aims to output the bottom program as the optimal result, as it replaces fixed numerical values with symbolic spatial relations to capture the scene's core layout.

Sketch Generator. Because the synthesized target program is strictly scoped to the entities populating the scene, it structurally comprises exactly n entity-specific declarations. Given n distinct entities extracted from the input scene, we directly construct a structural program sketch containing n uninstantiated statements (*i.e.*, syntactic holes), as demonstrated below.

$$\square_{stmt} \ \square_{stmt} \ \cdots \ \square_{stmt} \tag{1}$$

Each entity-specific declaration $\square_{stmt}$ is defined as $\square_{entity}$ = new $\square_{type}$ $\square_{pos}$, $\square_{heading}$. Specifically, $\square_{entity}$ denotes the target entity to be declared, and $\square_{type}$ denotes the type of the entity. $\square_{pos}$ and $\square_{heading}$ represent the relative positional and orientational relationships, respectively.

Sketch-Based Searcher. The sketch-based search process fundamentally amounts to exploring the program space defined by the sketch, iteratively filling in the hole nodes within it. We primarily employ two distinct search strategies: breadth-first search (BFS) and depth-first search (DFS). Because positions and orientations are expressed relative to a reference frame, the search process is designed to address the following three parts:

- **Position attribute resolution**: The searcher prioritizes more direct and natural positional relation predicates (such as *ahead of* and *right of*) over explicit coordinate values. Concurrently, it infers feasible numerical ranges associated with these relations (*e.g.*, "5 to 10 meters ahead"). Explicit coordinate attributes (*i.e.*, concrete Cartesian positions) are assigned lower priority and incur higher synthesis cost compared to relational predicates.
- **Orientation attribute resolution**: Analogous to position, the searcher prioritizes relational orientations (such as *facing toward* or *facing away from*) over explicit angular offsets.
- **Selection of the relative reference frame**: When multiple entities are present, only the ego vehicle or declared entities may serve as valid reference frames, ensuring dependency chain validity. The resolution of the optimal positional and orientational attributes is intrinsically dependent on the choice of the reference entity. Specifically, we evaluate candidate references and select the one that yields the highest-priority spatial relation.

SMT Encoder & Solving. Building upon the exploration of the searcher, the exact instantiation of statement attributes is delegated to an SMT solver. Specifically, for a given target and reference entity, the synthesizer iterates through candidate spatial relations, systematically encoding their geometric semantics into SMT formulas. The solver acts strictly as a geometric oracle to verify the satisfiability of these hypothesized relations. Upon returning a `sat` determination, the solver extracts a valid model, directly yielding the concrete numerical assignments for the continuous parameters (*e.g.*, distance boundaries or angular ranges).

Partial Program Pruner. As the number of entities increases, the synthesis search space expands factorially ($O(N!)$) due to the permutation of insertion orders. As the complexity of the target scene increases, the search space (despite being guided by a sketch) remains prohibitively large. To accelerate the search process, we introduce search pruning techniques. Specifically, we integrate the following two complementary pruning mechanisms:

- **Common Prefix Abstraction-based Pruning.** When declaring a new entity, its description must be anchored to either the ego vehicle or an already-declared entity as the reference. Given the same set of declared entities, selecting the same reference for a new entity incurs identical incremental cost. Consider the example in Fig. 2a, where the input scene contains three vehicles (denoted as car1, car2, and car3). Suppose we have two partial programs: P_a = {car1 car2 $\square_{stmt}$}

where car1 and car2 have been declared in that order , and P_b = {car2 car1 $\square_{stmt}$} which declares the same two entities but in a different order. Both P_a and P_b have declared the same set of entities (car1 and car2), so any subsequent entity (*e.g.*, car3) will be described relative to the same reference entity under our semantics, incurring identical incremental cost. If the total cost of P_a is lower than that of P_b, then for any valid completion of the hole, the full program derived from P_a will always have a lower (or equal) total cost than the corresponding completion of P_b. Therefore, P_b can be safely pruned from the search space without sacrificing optimality.

- **Branch-and-Bound Pruning.** Branch-and-bound pruning is an exact algorithmic strategy commonly used for solving combinatorial optimization problems. Its core idea is to systematically enumerate candidate solutions while leveraging upper- and lower-bound information to eliminate subspaces that cannot contain an optimal solution, thereby avoiding exhaustive enumeration. In our setting, we maintain the best complete candidate program P found so far (*i.e.*, the one with the lowest total cost). During search, if a partial program's accumulated cost meets or exceeds the cost of P, then any full program derived by extending this partial program will necessarily have a cost no less than that of P. Consequently, such a partial program can be safely pruned from the search space without risking the loss of an optimal solution.

3 Preliminaries

This chapter establishes the formal foundations of our scenario synthesis framework. We first introduce the syntax and semantics of our DSL. Building upon these definitions, we provide a rigorous formulation of the scenario synthesis problem.

3.1 DSL Syntax

We formally define the syntax of our DSL in Figure 3. Derived from the Scenic, the DSL is designed to model the spatial layout of entities by defining the relative relationships between them. Our DSL deliberately focuses on static spatial layouts, retaining Scenic primitives for describing relative positions and headings among entities. For numerical variation, it uses **Range** as its sole probabilistic operator, representing uniform distributions over distance and heading intervals. Dynamic behaviors, temporal interactions, and the richer probabilistic and procedural constructs of full Scenic are outside the current scope. A program $\langle P \rangle$ consists of a sequence of entity statements $\langle S \rangle$. Each statement $\langle S \rangle$ instantiates an entity and defines its state via three core components:

- **Type** ($\langle Type \rangle$): Specifies the category of the instantiated object (*e.g.*, **Car**, **Pedestrian**, ...).
- **Position** ($\langle Pos \rangle$): Specifies the spatial location of an entity using three distinct modes of representation. First, we favor high-level directional primitives (*e.g.*, **ahead of**, **left of**) that establish semantic spatial relations relative to a reference entity. These semantic primitives are assigned higher priority and incur lower synthesis cost. Second, the syntax supports explicit local Cartesian coordinates defined directly within a reference entity's local frame (**on** ... **relative to**). Third, it allows for absolute global Cartesian coordinates (**on** ... @ ...) to place entities independently in the global space.
- **Heading** ($\langle Heading \rangle$): Specifies an entity's orientation via three distinct modes. First, we favor high-level orientational primitives (*e.g.*, **facing toward**, **facing away from**) establishing semantic alignment relative to a reference entity. These primitives are assigned higher priority and incur lower synthesis cost. Second, the syntax supports explicit angular ranges defined relative to a reference entity's heading (**facing** ... **deg relative to**). Third, it allows absolute global angular ranges (**facing** ... **deg**) to orient entities independently in the global coordinate system.

To enable generalization over the structured concrete scene, we employ the parameter $\langle Range \rangle$ to construct interval-based uniform distributions within the real-valued domain $\mathcal{R}$, thereby describing a continuous space of plausible scenes. Although the syntax permits referencing arbitrary identifiers,

$$
\begin{aligned}
\langle P \rangle &::= \langle S \rangle^* \\
\langle S \rangle &::= \langle Entity \rangle = \textbf{new } \langle Type \rangle\ \langle Pos \rangle,\ \langle Heading \rangle \\
\langle Type \rangle &::= \textbf{Car} \mid \textbf{Pedestrian} \mid \ldots \\
\langle Pos \rangle &::= (\textbf{ahead of} \mid \textbf{behind} \mid \textbf{left of} \mid \textbf{right of})\ \langle Ref \rangle\ \textbf{by } \langle Range \rangle \\
&\quad \mid \textbf{on } \langle Range \rangle @ \langle Range \rangle \textbf{ relative to } \langle Ref \rangle \mid \textbf{on } \langle Range \rangle @ \langle Range \rangle \\
\langle Heading \rangle &::= \textbf{facing } (\textbf{toward} \mid \textbf{away from})\ \langle Ref \rangle \mid \textbf{facing } \langle Range \rangle \textbf{ deg relative to } \langle Ref \rangle \\
&\quad \mid \textbf{facing } \langle Range \rangle \textbf{ deg} \\
\langle Range \rangle &::= \textbf{Range}(a, b), \quad a, b \in \mathcal{R} \\
\langle Ref \rangle &::= \langle Entity \rangle \mid \textbf{ego} \\
\langle Entity \rangle &::= id_1 \mid id_2 \mid \ldots
\end{aligned}
$$

Fig. 3. The syntax of the Scenic DSL, where $\mathcal{R}$ denotes the set of real numbers. A program consists of n entities, each specified by a tuple of three core attributes: its type, spatial position, and orientation.

we impose a strict semantic constraint: an entity may only use the **ego** (*i.e.*, the self-entity) or previously declared entities as its reference frame.

3.2 DSL Semantics and SMT Encoding

3.2.1 Basic Definition. Before presenting the specific denotational semantics, we formalize the fundamental notations.

Definition 3.1 (Entity State Space). The state space of an entity is defined as $\mathcal{X} = \mathcal{T} \times \mathcal{R}^2 \times \Theta$, where $\mathcal{T}$ is a finite set of entity types ($\mathcal{T}$ contains types such as Car and Pedestrian), and $\Theta = [0, 360^\circ)$ denotes the heading angle. We establish a 2D Cartesian coordinate system where the heading angle θ is measured counter-clockwise starting from the positive Y-axis (North). A concrete state $x = (t, p, \theta) \in \mathcal{X}$ comprises a type $t \in \mathcal{T}$, a position $p \in \mathcal{R}^2$, and a heading $\theta \in \Theta$.

Definition 3.2 (Scene). Let $\mathcal{E}$ be a finite set of available entities. A concrete scene σ is a partial function assigning each involved entity a state, formally defined as $\sigma : \mathcal{E} \rightharpoonup \mathcal{X}$. We denote the universe of all such scenes as Σ. The scene containing no entities is denoted the empty scene $\sigma_\emptyset \in \Sigma$.

Definition 3.3 (Scenario). A scenario Ω is formalized as a set of concrete scenes, *i.e.*, $\Omega \subseteq \Sigma$. The semantic domain of scenarios is defined as the powerset 2^Σ. We specifically define the empty scenario as $\Omega_\emptyset = \{\sigma_\emptyset\} \in 2^\Sigma$.

3.2.2 Denotational Semantics. The denotational semantics maps the syntactic program into a scenario. The semantic functions are defined as follows:

$$
\begin{aligned}
&\mathbb{T} : \langle Type \rangle \to \mathcal{T} && \mathbb{H} : \langle Heading \rangle \times \mathcal{X} \times \mathcal{R}^2 \to 2^\Theta \\
&\mathbb{R} : \langle Range \rangle \to 2^\mathcal{R} && \mathbb{V} : \langle S \rangle \times 2^\Sigma \to 2^\Sigma \\
&\mathbb{P} : \langle Pos \rangle \times \mathcal{X} \to 2^{\mathcal{R}^2} && \mathbb{D} : \langle P \rangle \times 2^\Sigma \to 2^\Sigma
\end{aligned}
$$

The base valuations $\mathbb{T}$ and $\mathbb{R}$ deterministically map syntactic primitives to their fundamental mathematical domains: $\mathbb{T}$ maps a type identifier to its corresponding semantic element in $\mathcal{T}$, and $\mathbb{R}(\textbf{Range}(a, b)) = \{r \in \mathcal{R} \mid a \le r \le b\}$, where $a, b \in \mathcal{R}$ are real numbers.

Crucially, the statement semantics $\mathbb{V}$ incrementally constructs the scenario. Let S be an entity declaration statement of the form $e = \textbf{new } type\ pos, heading$. Given a current scenario Ω, $\mathbb{V}$ evaluates S by extending every existing scene $\sigma \in \Omega$ with the newly instantiated entity e:

$$
\mathbb{V}(S, \Omega) = \left\{ \sigma \cup \{e \mapsto (t, p, \theta)\} \,\middle|\, \sigma \in \Omega \wedge t = \mathbb{T}(type) \wedge p \in \mathbb{P}(pos, s_{pos}) \wedge \theta \in \mathbb{H}(heading, s_{heading}, p) \right\}
$$

where $s_{pos} = \sigma(ref_{pos})$ and $s_{heading} = \sigma(ref_{heading})$ are the states of the reference entities specified in pos and $heading$ respectively (or undefined for absolute coordinates).

The program semantics $\mathbb{D}$ is then the composition of individual statement valuations. For a program $P = S\ P'$, the scenario is progressively constructed:

$$\mathbb{D}(S\ P', \Omega) = \mathbb{D}(P', \mathbb{V}(S, \Omega))$$

Ultimately, the final scenario is derived by evaluating the program starting from the initial empty scenario: $\Omega_P = \mathbb{D}(P, \Omega_\emptyset)$.

Semantics of Position ($\mathbb{P}$) We formalize the positional primitives (spanning high-level directional relations, explicit relative Cartesian coordinates, and absolute global placements) by defining their denotational semantics. The semantic function $\mathbb{P}$ resolves these spatial specifications into concrete geometric regions conditionally evaluated given the state of the reference entity. For relative primitives, let $s_e = (t_e, (x_e, y_e), \theta_e) \in \mathcal{X}$ denote the state of the reference entity e. High-level directional relations permit a tolerance in angular alignment. We introduce a tolerance parameter δ_1 to define the acceptable angular deviation. In our implementation, we empirically set δ_1 to 10°. This bounds the valid spatial region to a ±10° fan-shaped sector strictly aligned with the reference entity's forward directional axis. For example, consider the semantic evaluation of the expression "**ahead of** e". This expression restricts the valid target coordinates to a narrow 20° cone directly in front of the reference entity e. As shown in (2), the semantic evaluation of a specific directional expression ($dir \in \{\textbf{ahead of}, \textbf{behind}, \textbf{left of}, \textbf{right of}\}$) denotes the subset of coordinates in $\mathbb{R}^2$ that satisfy the distance and angular constraints relative to the exact state of e.

$$\mathbb{P}(dir\ e\ \textbf{by Range}(a, b), s_e) = \left\{ (x, y) \;\middle|\; \begin{array}{l} \sqrt{(x - x_e)^2 + (y - y_e)^2} \in \mathbb{R}(\textbf{Range}(a, b)) \\ \wedge\ \mathcal{W}\left(\mathcal{N}(\beta((x, y), (x_e, y_e))),\ \mathcal{N}(\theta_e + \phi_{dir})\right) \leq \delta_1 \end{array} \right\} \tag{2}$$

Where:

- $\beta((x, y), (x_e, y_e))$ computes the absolute angle from the reference point (x_e, y_e) to the target point (x, y). Since the standard $atan2$ function (assumed to output degrees) measures the angle relative to the positive X-axis, we apply a $-90°$ offset to align the result with our coordinate system, where 0° is defined as the positive Y-axis (North):

$$\beta((x, y), (x_e, y_e)) = atan2(y - y_e, x - x_e) - 90°$$

- $\mathcal{N}(\alpha)$ is the normalization function that maps any angle α to the range $[0, 360)$

$$\mathcal{N}(\alpha) = \begin{cases} \alpha - 360° & \text{if } \alpha \geq 360° \\ \alpha + 360° & \text{if } \alpha < 0° \\ \alpha & \text{otherwise} \end{cases}$$

- $\mathcal{W}(u, v)$ normalizes the absolute difference between two angles $u, v \in [0, 360)$ to the range $[0, 180°]$.

$$\mathcal{W}(u, v) = \begin{cases} 360° - |u - v| & \text{if } |u - v| > 180° \\ |u - v| & \text{otherwise} \end{cases}$$

- ϕ_{dir} specifies the base angular offset for the target direction. These values represent the relative rotation applied to the reference entity's current heading θ_e, strictly following our counter-clockwise coordinate convention:

$$\phi_{ahead} = 0°, \quad \phi_{left} = 90°, \quad \phi_{behind} = 180°, \quad \phi_{right} = 270°$$

The evaluation of the relative expression **on Range**(a, b)@**Range**(c, d) **relative to** e denotes the subset of coordinates in $\mathcal{R}^2$ that fall within the Cartesian product of longitudinal and lateral

intervals defined in the reference entity's local frame:

$$\mathbb{P}(\textbf{on Range}(a,b)\textbf{@Range}(c,d)\ \textbf{relative to}\ e, s_e) = \left\{(x,y) \;\middle|\; \begin{array}{l} ((x-x_e)\cos\theta_e + (y-y_e)\sin\theta_e) \in \mathbb{R}(\textbf{Range}(a,b)) \\ \wedge\ ((y-y_e)\cos\theta_e - (x-x_e)\sin\theta_e) \in \mathbb{R}(\textbf{Range}(c,d)) \end{array}\right\} \tag{3}$$

The absolute Cartesian primitive defines a spatial bounding box directly in the global coordinate system, operating independently of any reference entity:

$$\mathbb{P}(\textbf{on Range}(a,b)\textbf{@Range}(c,d)) = \{(x,y) \mid x \in \mathbb{R}(\textbf{Range}(a,b)) \wedge y \in \mathbb{R}(\textbf{Range}(c,d))\}$$

Semantics of Heading ($\mathbb{H}$) We formalize the heading primitives (spanning high-level interactive orientations, explicit relative angles, and absolute global angles) by defining their denotational semantics. The valuation function $\mathbb{H}$ denotes the set of valid absolute angles in $[0, 360°)$ that satisfy the given constraints.

To similarly capture the semantic looseness of orientational constraints like **facing toward** or **facing away from**, we introduce a heading tolerance parameter δ_2, which relaxes strict geometric alignment into a valid continuous interval. In our implementation, we empirically set δ_2 to 5°.

Crucially, computing these interactive orientations inherently depends on both the exact state of the reference entity and the spatial location of the current entity itself. Let $\sigma \in \Sigma$ be the concrete scene currently being evaluated, which fixes the reference entity's state as $\sigma(e) = (t_e, (x_e, y_e), \theta_e)$. Furthermore, let $p = (x, y) \in \mathcal{R}^2$ represent a candidate position drawn from the entity's positional feasible region (as bound in the statement semantics $\mathbb{V}$). The denotational semantics evaluates the valid angular intervals conditional on both the scene σ and the candidate position $p = (x, y)$:

$$\mathbb{H}(\mathit{heading}, s_e, (x,y)) = \begin{cases} \{\phi \mid \mathcal{W}(\phi, \mathcal{N}(\beta((x_e,y_e),(x,y)))) \le \delta_2\} \\ \quad \text{if } \mathit{heading} = \textbf{facing toward } e \\ \{\phi \mid \mathcal{W}(\phi, \mathcal{N}(\beta((x,y),(x_e,y_e)))) \le \delta_2\} \\ \quad \text{if } \mathit{heading} = \textbf{facing away from } e \\ \{\phi \mid \exists \alpha \in \mathbb{R}(\textbf{Range}(a,b)),\ \phi = \mathcal{N}(\theta_e + \alpha)\} \\ \quad \text{if } \mathit{heading} = \textbf{facing Range}(a,b)\ \textbf{deg relative to}\ e \\ \{\phi \mid \exists \alpha \in \mathbb{R}(\textbf{Range}(a,b)),\ \phi = \mathcal{N}(\alpha)\} \\ \quad \text{if } \mathit{heading} = \textbf{facing Range}(a,b)\ \textbf{deg} \end{cases} \tag{4}$$

SMT Encoding of Position ($\mathbb{P}$) and Heading ($\mathbb{H}$) Our SMT formulation computationally realizes these semantics, serving as a constraint-based oracle to verify the feasibility of candidate spatial and orientational configurations.

For position encoding, given a candidate coordinate (x, y) and reference state (x_e, y_e, θ_e), the solver evaluates the satisfiability of the geometric membership. For high-level directional relations (**ahead of**, **behind**, **left of**, **right of**), the solver performs conditional spatial checks to verify if geometric membership holds within fuzzy angular regions. In contrast, explicit relative and absolute coordinates are mathematically guaranteed to be SAT; since the physical states are fixed during evaluation, their corresponding coordinates can always be determined. Upon a SAT result, we extract valid boundaries to instantiate spatial ranges.

For heading encoding, high-level heading relations (**facing toward**/**facing away from**) are evaluated by checking the satisfiability of the orientation variable ϕ bounded by the tolerance δ_2. For numerical primitives (**facing Range**(a,b) **deg relative to** and **facing Range**(a,b) **deg**), the constraints explicitly bound the valid angular space either as an offset from the reference's heading or within the global coordinate system, from which the solver extracts the angular boundaries.

Although an individual relation over fixed coordinates can be checked using closed-form arithmetic, we use SMT to express relation feasibility and the synthesis of distance or angle interval parameters in a uniform nonlinear-real constraint system. This formulation also provides a direct extension path for jointly reasoning about additional continuous constraints, such as uncertain input regions or map/lane constraints.

To accelerate solving and ensure practical alignment, we employ unified fixed-length constraints: for positional ranges, $b = a + C_{dist}$ and $d = c + C_{dist}$ with $C_{dist} = 5\,m$; for heading ranges, $b = a + C_{head}$ with $C_{head} = 5^\circ$. These four constants (δ_1, δ_2, C_{dist}, and C_{head}) serve two different roles. The angular tolerances δ_1 and δ_2 define the acceptable angular deviations for high-level directional and heading relations. Increasing these tolerances allows such relations to cover larger angular regions, whereas decreasing them requires closer geometric alignment and may cause the synthesizer to use explicit relative coordinates or angles instead. In contrast, C_{dist} and C_{head} determine the widths of the synthesized **Range** expressions: larger values allow broader variations in position or heading, while smaller values keep sampled scenes closer to the input. We use the same values throughout our evaluation; other domains or entity scales may use different values to reflect their spatial characteristics and the desired amount of scenario variation.

3.3 Problem Definition

We cast the generation of scenario programs as a program synthesis task, specifically searching for the optimal scenario program whose semantic domain contains the given concrete scene.

Definition 3.4. **(Specification)** We utilize the input scene $\sigma_{in} \in \Sigma$ as a formal specification for program synthesis, thereby guiding the search process toward valid program configurations.

Definition 3.5. **(Consistency)** We define a consistency relation between a scenario program P and a specification σ_{in} as $P \models \sigma_{in}$. This relation holds if and only if σ_{in} resides within the semantic domain induced by P, *i.e.*, $\sigma_{in} \in \mathbb{D}(P, \Omega_\emptyset)$.

Definition 3.6. **(Cost Function C)** To rank semantically equivalent programs by how well they capture scene relational structures, we define a cost function over the n statements of P as

$$C(P) = \sum_{i=1}^{n} \left(penalty(pos_i) + penalty(heading_i)\right)$$

The penalty prioritizes high-level relational semantics:

$$penalty(pos_i) = \begin{cases} 2 & \text{if } pos_i = \textbf{on } \langle Range\rangle @ \langle Range\rangle \\ 1 & \text{if } pos_i = \textbf{on } \langle Range\rangle @ \langle Range\rangle \textbf{ relative to } \langle Ref\rangle \\ 0 & \text{if } pos_i = dir\ \langle Ref\rangle \textbf{ by } \langle Range\rangle \end{cases}$$

$$penalty(heading_i) = \begin{cases} 2 & \text{if } heading_i = \textbf{facing } \langle Range\rangle \\ 1 & \text{if } heading_i = \textbf{facing } \langle Range\rangle \textbf{ deg relative to } \langle Ref\rangle \\ 0 & \text{if } heading_i = \textbf{facing (toward } | \textbf{ away from)} \langle Ref\rangle \end{cases}$$

Definition 3.7. **(Problem)** The scenario program synthesis task is formalized as a combinatorial optimization problem. We seek to find an optimal valid program $\mathcal{P}^*$ that minimizes $C(P)$, under the strict constraint that the denotational semantics of $\mathcal{P}^*$ contains the input concrete scene σ_{in}.

$$\mathcal{P}^* = \underset{P}{arg\,min}\ C(P) \quad \text{s.t.} \quad \mathcal{P}^* \models \sigma_{in} \tag{5}$$

4 Synthesis Algorithm

This section presents the algorithm for the scenario synthesis problem. We first introduce the top-level algorithm, followed by its other components.

Algorithm 1 ConstraintSolvingBasedSynthesizer(σ_{in}, mode)

1: **input:** A concrete input scene σ_{in} containing a finite set of entities $\mathcal{E}$, and search strategy mode $\in$ {DFS, BFS}.
2: **output:** An optimal scenario program $\mathcal{P}^*$ such that $\mathcal{P}^* \models \sigma_{in}$.
3: $\mathcal{S} \leftarrow$ SketchGenerator(σ_{in})
4: **if** mode = BFS **then**
5: $\quad \mathcal{P}^* \leftarrow$ SketchBasedSearchBFS($\mathcal{S}, \sigma_{in}$)
6: **else**
7: $\quad \mathcal{P}^* \leftarrow$ SketchBasedSearchDFS($\mathcal{S}, \sigma_{in}$)
8: **return** $\mathcal{P}^*$

4.1 Top-Level Synthesis Algorithm

Alg. 1 outlines our top-level synthesis procedure, ConstraintSolvingBasedSynthesizer. It accepts two inputs: a concrete scene σ_{in} (extracted from a real image, containing state labels for a finite set of entities $\mathcal{E}$) and a specified search strategy mode $\in$ {DFS, BFS}. The objective is to output an optimal scenario program $\mathcal{P}^*$ rigorously satisfying given specifications (Definition 3.7). The algorithm begins by invoking SketchGenerator (Line 3) to construct a structural sketch $\mathcal{S}$ from σ_{in}. Specifically, using $\mathcal{E}$'s cardinality, this generator instantiates an equal number of statements with uninstantiated holes, as previously illustrated in Eq.(1). Following the sketch generation, the algorithm routes execution to either SketchBasedSearchBFS (Line 5) or SketchBasedSearchDFS (Line 7), depending on the specified mode, to systematically resolve the holes and discover $\mathcal{P}^*$. Crucially, assuming a perfect SMT oracle, our search framework guarantees soundness and completeness. Formal discussions and proofs of these properties are detailed in Section 4.4.

4.2 Sketch-Based Search with Pruner

In this subsection, we formulate the scenario program generation process as a combinatorial search problem over the permutation space of entity declarations. As formalized in Eq. 1, the scenario sketch $\mathcal{S}$ is conceptualized as an ordered sequence of uninstantiated statement holes ($\square_{stmt}$). The primary objective is to dynamically determine the optimal declaration order for the finite set of entities $\mathcal{E}$, while minimizing the cumulative semantic penalty C. To efficiently navigate this permutation space, we implement two variant search strategies: Breadth-First Search (BFS) and Depth-First Search (DFS).

Both search strategies operate on a unified state-space tree representing the permutation of different entities. In this tree, each node represents the instantiation of a single entity's statement. Consequently, the continuous path from the root to any given node constitutes a search state, which is formally defined by a partial sketch $\mathcal{S}_{curr}$ and its accumulated semantic penalty C_{curr}.

At each expansion step, the search controller determines the set of defined entities E_{curr} within $\mathcal{S}_{curr}$. If unassigned entities remain, the controller fetches the next available structural template $\square_{stmt}$. Crucially, the search strategy operates purely at the macro-level: it selects an unassigned entity $e_i \in \mathcal{E} \setminus E_{curr}$ and binds it to the entity identifier slot ($\square_{stmt}.\square_{entity} \leftarrow e_i$). The concrete geometric instantiation of the remaining sub-holes ($\square_{type}$, $\square_{pos}$, and $\square_{heading}$) is then delegated to the micro-level SMT solver via the SynthesizeStatement function (detailed later in Alg. 4). This strict decoupling of macro-level topological ordering from micro-level geometric constraint solving effectively bounds the search complexity, making an otherwise intractable problem solvable.

Breadth-First Search (BFS) and Common Prefix Abstraction-based Pruning. Alg. 2 details the BFS approach to solve the synthesis problem. The algorithm takes an initial sketch $\mathcal{S}$ and the concrete scene σ_{in} (containing a finite set of entities $\mathcal{E}$) as inputs, aiming to synthesize the optimal program $\mathcal{P}^*$ with the minimal semantic penalty. Initialization begins by setting the optimal program

Algorithm 2 SketchBasedSearchBFS($\mathcal{S}, \sigma_{in}$)

1: **input:** The input sketch $\mathcal{S}$ and the concrete scene σ_{in} containing a finite set of entities $\mathcal{E}$.
2: **output:** The optimal program $\mathcal{P}^*$ such that $\mathcal{P}^* \models \sigma_{in}$.
3: $\mathcal{P}^* \leftarrow \perp$, $\mathcal{M}_{smt} \leftarrow \{\}$ ▷ $\perp$ denotes null; $\mathcal{M}_{smt}$: SMT query cache
4: Let $e_{ego} \in \mathcal{E}$ be the designated ego entity
5: $Q \leftarrow \{(\mathcal{S}, 0)\}$
6: **while** $Q \neq \emptyset$ **do**
7: $\quad\mathcal{T}_{layer} \leftarrow \{\}$ ▷ Hash table: $E_{next} \mapsto (\mathcal{S}, C)$ for current layer
8: $\quad$**for each** $(\mathcal{S}_{curr}, C_{curr}) \in Q$ **do**
9: $\quad\quad E_{curr} \leftarrow$ Entities($\mathcal{S}_{curr}$) ▷ Extract set of entities already declared in $\mathcal{S}_{curr}$
10: $\quad\quad$**if** $\mathcal{E} \setminus E_{curr} = \emptyset$ **then**
11: ▷ Optimal complete program found; $\mathcal{P}^* \models \sigma_{in}$ by construction
12: $\quad\quad\quad\mathcal{P}^* \leftarrow \mathcal{S}_{curr}$
13: $\quad\quad$**for each** $e_i \in \mathcal{E} \setminus E_{curr}$ **do**
14: $\quad\quad\quad\square_{stmt} \leftarrow$ GetNextStatement($\mathcal{S}_{curr}$) ▷ Fetch next uninstantiated statement hole
15: $\quad\quad\quad\square_{stmt}.\square_{entity} \leftarrow e_i$ ▷ Dot notation accesses sub-holes of the statement template
16: $\quad\quad\quad E_{ref} \leftarrow \{e_{ego}\} \cup E_{curr}$
17: $\quad\quad\quad\mathcal{S}_{new}, \Delta c \leftarrow$ SynthesizeStatement($\square_{stmt}, e_i, E_{ref}, \mathcal{S}_{curr}, \sigma_{in}, \mathcal{M}_{smt}$)
18: $\quad\quad\quad C_{new} \leftarrow C_{curr} + \Delta c$
19: $\quad\quad\quad E_{next} \leftarrow E_{curr} \cup \{e_i\}$
20: $\quad\quad\quad$**if** $E_{next} \notin \mathcal{T}_{layer}$ **or** $C_{new} < \mathcal{T}_{layer}[E_{next}].C$ **then**
21: ▷ **Common Prefix Abstraction-based Pruning**
22: $\quad\quad\quad\quad\mathcal{T}_{layer}[E_{next}] \leftarrow (\mathcal{S}_{new}, C_{new})$
23: $\quad Q \leftarrow \mathcal{T}_{layer}$
24: **return** $\mathcal{P}^*$

to null ($\perp$), and allocating an empty global cache $\mathcal{M}_{smt}$ for SMT solver memoization (Line 3). After designating the ego object e_{ego} as the primary reference (Line 4), the active queue Q is seeded with the initial sketch and a starting cost of zero (Line 5). From this initial state, the algorithm systematically expands the search tree layer by layer using the active queue Q (Lines 6-23). Each layer corresponds to a specific depth of the sketch tree (*i.e.*, the number of instantiated statements). While BFS demands a larger memory footprint to store the active frontier, its strictly layer-wise expansion naturally aligns all partial programs of identical depths. This structural uniformity makes it exceptionally well-suited for the pruning technique introduced below.

Specifically, different declaration orders of the same group of entities result in the identical set E_{curr}. Because the context for future SMT solver depends solely on *which* entities have been declared rather than the *order* of their declaration, these distinct paths can be abstracted into an identical search state. To exploit this, we introduce **Common Prefix Abstraction-based Pruning**. At the beginning of each depth iteration, we initialize an empty layer-wise optimal prefix table $\mathcal{T}_{layer}$ (Line 7). This table is a hash map that associates each set of declared entities E with the lowest-cost partial sketch $(\mathcal{S}, C)$ that has declared exactly those entities. During the expansion phase (Lines 8-22), if a sketch has successfully instantiated all entities ($\mathcal{E} \setminus E_{curr} = \emptyset$), we evaluate it to update the global optimal program $\mathcal{P}^*$(Lines 10-12). If the sketch remains incomplete, the algorithm branches out to explore all valid single-step extensions by iterating over the pool of unassigned entities $\mathcal{E} \setminus E_{curr}$ (Line 13). For each candidate $e_i \in \mathcal{E} \setminus E_{curr}$, we bind it to the next statement hole and invoke the SynthesizeStatement procedure (using $\{e_{ego}\} \cup E_{curr}$ as references) to deduce the extended sketch $\mathcal{S}_{new}$ and its incremental penalty Δc(Line 14-17). After calculating the total accumulated cost C_{new} and updating the defined entity set E_{next} (Lines 18-19), the algorithm enforces the Common Prefix Abstraction. Specifically, the unordered entity subset E_{next} serves as a unique hash key to index the layer-wise optimal prefix table $\mathcal{T}_{layer}$. Using the unordered subset E_{next} as a unique hash

Algorithm 3 SketchBasedSearchDFS($\mathcal{S}, \sigma_{in}$)

```
1:  input: The input sketch S and the concrete scene σ_in containing a finite set of entities E.
2:  output: The optimal program P* such that P* ⊨ σ_in.
3:  P* ← ⊥, C* ← ∞, T_global ← {}, M_smt ← {}                          ▷ ⊥ denotes null
4:                ▷ T_global: global hash table E_curr ↦ minimal cost; M_smt: SMT query cache
5:  Let e_ego ∈ E be the designated ego entity
6:  Stack ← [(S, 0)]
7:  while Stack ≠ ∅ do
8:      (S_curr, C_curr) ← Stack.pop()
9:      E_curr ← Entities(S_curr)                          ▷ Extract declared entities from S_curr
10:     if E_curr ∈ T_global and C_curr ≥ T_global[E_curr] then
11:         continue                                  ▷ Common Prefix Abstraction-based Pruning
12:     T_global[E_curr] ← C_curr
13:     if C_curr ≥ C* then                                       ▷ Branch & Bound Pruning
14:         continue
15:     if E \ E_curr = ∅ then
16:         P* ← S_curr, C* ← C_curr                          ▷ P* ⊨ σ_in by construction
17:         continue
18:     for each e_i ∈ E \ E_curr do
19:         □_stmt ← GetNextStatement(S_curr)          ▷ Fetch next uninstantiated statement hole
20:         □_stmt.□_entity ← e_i      ▷ Dot notation accesses sub-holes of the statement template
21:         E_ref ← {e_ego} ∪ E_curr
22:         S_new, Δc ← SynthesizeStatement(□_stmt, e_i, E_ref, S_curr, σ_in, M_smt)
23:         Stack.push((S_new, C_curr + Δc))
24: return P*
```

key, $\mathcal{T}_{layer}$ is updated only if the new path achieves a strictly lower cumulative cost C_{new} (Lines 20-22). Consequently, any sub-optimal permutations that reach the identical entity abstraction are mathematically dominated and seamlessly pruned. Finally, after fully expanding the current layer, all optimal surviving states stored in $\mathcal{T}_{layer}$ are extracted to form the new active queue Q for the subsequent depth iteration (Line 23). This pruning mechanism drastically truncates the breadth of the search tree while strictly preserving global optimality. Ultimately, at the maximum depth, all valid paths instantiate the full entity set $\mathcal{E}$. Consequently, $\mathcal{T}_{layer}$ collapses to a single key $\mathcal{E}$, naturally isolating the globally optimal program $\mathcal{P}^*$ (Lines 10-12) and terminating the search.

Depth-First Search (DFS) and Branch-and-Bound.

Alg. 3 details the DFS variant. Sharing identical inputs and objectives with BFS, DFS utilizes a Last-In-First-Out (LIFO) *Stack* to systematically explore the permutation space (Line 6). While standard DFS is renowned for its memory efficiency by only maintaining the active path, we introduce a global optimal prefix table $\mathcal{T}_{global}$ to support the aforementioned Common Prefix Abstraction across all explored paths, deliberately trading memory overhead for substantial search space reduction.

At each step, the algorithm pops a state and sequentially evaluates it against our dual pruning mechanisms. First, we adapt the **Common Prefix Abstraction-based Pruning** into the DFS framework to combat structural redundancies across divergent branches. If the traversal encounters an already visited state E_{curr} but with a cost $C_{curr} \geq \mathcal{T}_{global}[E_{curr}]$, the branch is safely pruned as a sub-optimal prefix (Lines 10-11). Otherwise, DFS explicitly updates $\mathcal{T}_{global}$ to continuously record the minimum cost required to reach this unordered entity subset signature (Line 12). Following this, the state undergoes **Branch-and-Bound Pruning**. Because the semantic penalty increment is strictly non-negative ($\Delta c \geq 0$), the accumulated cost along any path is monotonically non-decreasing. Thus, if any partial sketch $\mathcal{S}_{curr}$ retrieved during traversal accrues a cost $C_{curr} \geq C^*$,

its entire subsequent sub-tree is guaranteed to be sub-optimal and is immediately discarded (Lines 13-14). If the state survives both pruning checks and represents a fully synthesized scenario program ($\mathcal{E} \setminus E_{curr} = \emptyset$), the algorithm directly establishes or tightens the valid upper bound C^* for the optimal cost (Lines 15-17). Finally, for incomplete states surviving all checks, the algorithm proceeds to the macro-micro expansion phase. Iterating through the unassigned entities, it binds each e_i to a new statement hole and invokes the SynthesizeStatement procedure. The resulting extended sketches and their cumulatively updated costs are subsequently pushed onto the stack (Lines 18-23). Together, these dual pruning strategies effectively constrain the combinatorial explosion, ensuring rapid convergence to the globally optimal program $\mathcal{P}^*$.

Lemma 4.1. *Under the assumption of a perfect SMT oracle, we guarantee the soundness of our pruning strategies. Any partial program discarded during the search is either definitively suboptimal or safely redundant, ensuring that the discovery of a global optimal solution is never compromised.*

Proof. Our exploration involves two primary pruning heuristics: common prefix abstraction and branch-and-bound. We formalize the soundness (optimality-preserving property) of these strategies as follows:

- **Soundness of Prefix Abstraction-based Pruning**: Pruning intermediate programs based on entity-set equivalence preserves the optimal solution. We define an equivalence relation over partial programs where two programs are equivalent if they declare the identical set of entities, irrespective of the declaration order. While the declaration permutation affects the current accumulated cost, appending identical subsequent entities to these equivalent programs will result in the same marginal cost increase. Thus, within an equivalence class, strictly higher-cost programs are dominated. For cost ties, we simply retain the first explored program to eliminate redundancy. By safely discarding these dominated and redundant variants , the optimality of the final synthesis is mathematically guaranteed.
- **Soundness of Branch-and-Bound Pruning**: Pruning partial programs whose cost exceeds the best-known upper bound preserves the optimal solution. Let $\mathcal{P}^*$ denote the fully synthesized program with the minimum cost discovered so far, acting as our global upper bound. As established in Definition 3.6, the cost function $C(P)$ is monotonically *non-decreasing* with respect to the addition of entities. If a partial program $P_{partial}$ currently under evaluation satisfies $C(P_{partial}) \geq C(\mathcal{P}^*)$, any complete extension $P'_{partial}$ derived from it will inherently satisfy $C(P'_{partial}) \geq C(P_{partial}) \geq C(\mathcal{P}^*)$. Thus, no sequence of subsequent derivations from $P_{partial}$ can yield a strictly better solution than $\mathcal{P}^*$. Pruning $P_{partial}$ eliminates only suboptimal sub-spaces or redundant ties, rendering the Branch-and-Bound Pruning strategy profoundly safe.

□

4.3 SMT-Based Statement Synthesizer

While the Sketch Based Search dictates the macro-level declaration order of entities, the exact geometric and semantic constraints are resolved by the SMT-Based SynthesizeStatement, as detailed in Alg. 4. The primary objective of this module is to instantiate the sub-holes of a given statement template ($\square_{stmt}$) by validating candidate spatial relations against the physical ground truth provided in σ_{in}.

As outlined in the algorithm, each uninstantiated statement $\square_{stmt}$ is systematically decomposed and resolved across three distinct phases: the semantic type ($\square_{type}$), the positional placement ($\square_{pos}$), and the heading ($\square_{heading}$).

- **Semantic Type Resolution.** Before invoking any geometric solvers, the synthesizer first deterministically extracts and binds the entity's semantic type directly from the concrete scene σ_{in} (Line 4).

Algorithm 4 SynthesizeStatement($\square_{stmt}, e_{new}, E_{ref}, \mathcal{S}_{curr}, \sigma_{in}, \mathcal{M}_{smt}$)

1: **input:** The assigned statement hole $\square_{stmt}$, target entity e_{new}, reference set E_{ref}, sketch $\mathcal{S}_{curr}$, concrete scene σ_{in}, and global SMT cache $\mathcal{M}_{smt}$.
2: **output:** The updated sketch $\mathcal{S}_{new}$ and accumulated semantic penalty Δc.
3: $\Delta c \leftarrow 0$ ▷ Phase 0: Resolve Semantic Type ($\square_{stmt}.\square_{type}$)
4: $\square_{stmt}.\square_{type} \leftarrow \sigma_{in}(e_{new}).type$
5: $pos\_found \leftarrow$ False ▷ Phase 1: Synthesize Positional Relation ($\square_{stmt}.\square_{pos}$)
6: **for each** $e_{ref} \in E_{ref}$ **do**
7: **for each** $rel \in$ {**ahead of**, **behind**, **left of**, **right of**} **do**
8: $sat, vals \leftarrow$ SMTPosRelSolve($rel, e_{new}, e_{ref}, \sigma_{in}, \mathcal{M}_{smt}$)
9: ▷ Queries $\mathcal{M}_{smt}[(rel, e_{new}, e_{ref})]$ first; on miss, calls solver and caches $(sat, vals)$
10: **if** sat **then**
11: $\square_{stmt}.\square_{pos} \leftarrow rel\ e_{ref}$ **by Range**($vals.a, vals.b$)
12: $pos\_found \leftarrow$ True
13: **if** pos_found **then break**
14: **if not** pos_found **then**
15: $\Delta c \leftarrow \Delta c + 1$
16: $sat, vals \leftarrow$ SMTLocalXYSolve($e_{new}, e_{ego}, \sigma_{in}, \mathcal{M}_{smt}$)
17: ▷ Queries $\mathcal{M}_{smt}$[("local_xy", e_{new}, e_{ego})]; caches $(sat, vals)$ on miss
18: $\square_{stmt}.\square_{pos} \leftarrow$ **on Range**($vals.a, vals.b$)@**Range**($vals.c, vals.d$) **relative to** e_{ego}
19: $head\_found \leftarrow$ False ▷ Phase 2: Synthesize Heading Relation ($\square_{stmt}.\square_{heading}$)
20: **for each** $e_{ref} \in E_{ref}$ **do**
21: **for each** $rel \in$ {**toward**, **away_from**} **do**
22: $sat \leftarrow$ SMTHeadingRelSolve($rel, e_{new}, e_{ref}, \sigma_{in}, \mathcal{M}_{smt}$)
23: ▷ Queries $\mathcal{M}_{smt}[(rel, e_{new}, e_{ref})]$; caches sat on miss
24: **if** sat **then**
25: $\square_{stmt}.\square_{heading} \leftarrow$ **facing** $rel\ e_{ref}$
26: $head\_found \leftarrow$ True
27: **if** $head\_found$ **then break**
28: **if not** $head\_found$ **then**
29: $\Delta c \leftarrow \Delta c + 1$
30: $sat, vals \leftarrow$ SMTRelativeAngleSolve($e_{new}, e_{ego}, \sigma_{in}, \mathcal{M}_{smt}$)
31: ▷ Queries $\mathcal{M}_{smt}$[("relative_heading", e_{new}, e_{ego})]; caches $(sat, vals)$ on miss
32: $\square_{stmt}.\square_{heading} \leftarrow$ **facing Range**($vals.a, vals.b$) **relative to** e_{ego}
33: $stmt \leftarrow \square_{stmt}$.assemble() ▷ Construct complete statement from resolved holes
34: $\mathcal{S}_{new} \leftarrow \mathcal{S}_{curr} \oplus \{\square_{stmt} \mapsto stmt\}$
35: **return** $\mathcal{S}_{new}, \Delta c$

- **Synthesize Positional Relation.** For $\square_{pos}$ (Lines 5-18), the module evaluates e_{new} against each reference $e_{ref} \in E_{ref}$ using spatial primitives (*e.g.*, **ahead of**). It invokes SMTPosRelSolve to verify geometric validity against the constraints in Eq. 2. To optimize performance, $\mathcal{M}_{smt}$ acts as a memoization table implemented as a hash table. Each SMT solver invocation uses a three-element key: (query type, target entity, reference entity). For example, positional relation queries use keys like ("ahead", e_{new}, e_{ref}), while fallback solvers use keys like ("local_xy", e_{new}, e_{ego}). The stored result is either a tuple $(sat, vals)$ containing satisfiability and boundary values, or a boolean sat for high-level heading relation queries. Before invoking the underlying solver, each function checks whether the query exists in $\mathcal{M}_{smt}$ and reuses cached results to bypass redundant solver calls across different search branches. If SAT, $\square_{pos}$ is instantiated with the resulting boundary intervals $vals$ (Lines 10-12). If no high-level relation is valid, the synthesizer falls back to resolving

coordinates in the ego's local frame (Eq. 3). By Definition 3.6, this lower abstraction level is penalized by incrementing the semantic cost $\Delta c \leftarrow \Delta c + 1$ (Lines 14-18).

- **Synthesize Heading Relation.** For $\square_{heading}$ (Lines 19-32), the synthesizer follows a similar paradigm by verifying constraints against Eq. 4. It first seeks high-level relations (*e.g.*, **toward**) via SMTHeadingRelSolve; if unsuccessful, it defaults to a relative heading angle. Following Definition 3.6, this fallback incurs a penalty $\Delta c \leftarrow \Delta c + 1$ (Lines 28-32).

Upon resolving all sub-holes, the instantiated statement is integrated into $\mathcal{S}_{curr}$ using the state extension operator $\oplus$, yielding the next sketch $\mathcal{S}_{new}$ (Lines 33-34). Finally, the algorithm returns $\mathcal{S}_{new}$ and the accumulated penalty Δc to guide the macro-level pruning logic (Line 35).

4.4 Theoretical Analysis

Before presenting the theoretical analysis, we assume the existence of a perfect SMT oracle. This idealized assumption ensures that the underlying solver is unconditionally sound and complete, allowing us to cleanly isolate and formally establish the theoretical guarantees of our algorithm.

4.4.1 Soundness. In this section, we prove that the proposed scenario synthesis algorithm is sound with respect to the problem formulation (Definition 3.7). That is, if the algorithm successfully returns a scenario program, this program is guaranteed to not only geometrically encapsulate the input concrete scene, but also mathematically minimize the semantic penalty cost.

Theorem 4.1 (Soundness). *Given an input concrete scene σ_{in} representing the deterministic state configuration of all n entities, if Alg. 1 terminates and returns a scenario program $\mathcal{P}^*$, then $\mathcal{P}^*$ is the optimal consistent program. Formally, $\mathcal{P}^*$ satisfies both consistency and optimality:*

$$\sigma_{in} \in \mathbb{D}(\mathcal{P}^*, \Omega_\emptyset) \quad \textit{and} \quad \forall P \; s.t. \; \sigma_{in} \in \mathbb{D}(P, \Omega_\emptyset),\; C(\mathcal{P}^*) \leq C(P) \tag{6}$$

Proof. The proof is bipartite: we first establish the consistency of the synthesized program, and subsequently prove its optimality.

Consistency ($\sigma_{in} \in \mathbb{D}(\mathcal{P}^*, \Omega_\emptyset)$). Let the synthesized program be a sequence of statements $\mathcal{P}^* = S_1\ S_2\ \ldots\ S_n$, where each S_i declares entity e_i. Let σ_i denote the partial concrete scene restricting σ_{in} to the first i entities. We proceed by induction on the incremental construction of the scenario $\Omega_i = \mathbb{D}(S_1\ \ldots\ S_i, \Omega_\emptyset)$.

Base Case: Before any statement is evaluated, $\Omega_0 = \{\sigma_\emptyset\}$ and the 0-entity scene is $\sigma_0 = \sigma_\emptyset$. Thus, $\sigma_0 \in \Omega_0$ trivially holds.

Inductive Step: Assume that after synthesizing $i-1$ statements, the partial scene σ_{i-1} is valid within the current scenario, *i.e.*, $\sigma_{i-1} \in \Omega_{i-1}$. For the i-th statement S_i (e_i =**new** $type_i\ pos_i, heading_i$), let the concrete state of entity e_i in the input scene be $\sigma_{in}(e_i) = (t_i, p_i, \theta_i)$. Let $s_{pos} = \sigma_{i-1}(ref_{pos})$ and $s_{heading} = \sigma_{i-1}(ref_{heading})$ denote the states of the reference entities specified in pos_i and $heading_i$ respectively (or undefined for absolute coordinates). According to our relational statement semantics $\mathbb{V}$, the updated partial scene $\sigma_i = \sigma_{i-1} \cup \{e_i \mapsto (t_i, p_i, \theta_i)\}$ belongs to $\Omega_i = \mathbb{V}(S_i, \Omega_{i-1})$ if and only if:

$$t_i = \mathbb{T}(type_i) \;\wedge\; p_i \in \mathbb{P}(pos_i, s_{pos}) \;\wedge\; \theta_i \in \mathbb{H}(heading_i, s_{heading}, p_i) \tag{7}$$

Alg. 4 invokes the SMT-based synthesizer to resolve these constraints, satisfying three conditions:

- **Semantic Type ($t_i = \mathbb{T}(type_i)$):** The synthesizer directly extracts the exact entity type from σ_{in}; thus, the type equality strictly holds.
- **Position ($p_i \in \mathbb{P}(pos_i, s_{pos})$):** The pos_i is synthesized by querying the SMT solver with the hypothesized positional relation. Because the solver uses the exact coordinates of the previously declared reference entities from σ_{in} (identically preserved in σ_{i-1}) as known constants, a SAT response guarantees that the synthesized boundaries legally enclose p_i. Therefore, $p_i \in \mathbb{P}(pos_i, s_{pos})$.

- **Heading ($\theta_i \in \mathbb{H}(heading_i, s_{heading}, p_i)$):** Similarly, the heading relation $heading_i$ is synthesized by encoding the geometric assertions in Eq. 4. The solver utilizes the reference entity's state and the newly confirmed position p_i as constants. The instantiation of $heading_i$ only occurs upon a SAT verification against the concrete orientation θ_i, ensuring $\theta_i \in \mathbb{H}(heading_i, s_{heading}, p_i)$.

Since all conditions are satisfied, $\sigma_i \in \Omega_i$ holds for step i. By mathematical induction, upon evaluating the final statement S_n, the complete input scene $\sigma_n = \sigma_{in}$ is guaranteed to be a member of the final scenario $\Omega_n = \mathbb{D}(\mathcal{P}^*, \Omega_\emptyset)$.

Optimality ($C(\mathcal{P}^*) \leq C(P)$). Having established the synthesized program $\mathcal{P}^*$ is consistent, we must prove it minimizes the cost function C. Our search algorithm systematically explores the finite combinatorial space of entity declarations. Following Lemma 4.1, our pruning strategies (Common Prefix Abstraction and Branch-and-Bound) are provably optimality-preserving. They exclusively discard infeasible, strictly suboptimal, or safely redundant partial programs, ensuring the globally optimal solution is never erroneously pruned. Consequently, upon termination, the consistent program $\mathcal{P}^*$ that the search identifies and returns strictly minimizes the semantic cost. □

4.4.2 Completeness. In this section, we prove the completeness of the synthesis algorithm with respect to Definition 3.7. Our algorithm provides a strong guarantee: it will always terminate and successfully return the optimal consistent scenario program $\mathcal{P}^*$ for any given input scene.

Theorem 4.2 (Completeness). *Given any concrete input scene σ_{in} containing a finite set of n entities, Alg. 1 is guaranteed to terminate and return the optimal consistent scenario program $\mathcal{P}^*$ as defined in Definition 3.7.*

Proof. The proof relies on three foundational properties of the synthesis framework.

Existence Guarantee of a Consistent Program. Our DSL provides an absolute semantic cover: for any exact entity state $(t, p, \theta) \in \sigma_{in}$, it accommodates either high-level relative primitives or explicit numerical fallbacks that can formulate bounding regions encompassing any coordinate in $\mathbb{R}^2$ and angle in $[0, 360)$. Thus, the existence of at least one semantically consistent program is theoretically guaranteed. Given the discrete state space, a global cost minimum must inherently exist among these consistent candidates, ensuring the existence of the optimal target program $\mathcal{P}^*$.

Finite Abstraction of the Search Space. Although spatial parameters reside in the continuous real domain, our algorithm maintains a discrete search tree by delegating constraint resolution to the SMT solver. Assuming the solver acts as an idealized oracle returning satisfying assignments for consistent spatial primitives, the search complexity becomes strictly bounded by the finite number of entities and categorical relations, rendering the exploration finite and tractable.

Safety of Pruning Strategies. By Lemma 4.1, our pruning strategy is rigorously safe. While aggressively evaluating semantic costs to discard strictly suboptimal or safely redundant branches, it is mathematically proven to never erroneously exhaust the space of consistent solutions. By strictly retaining at least one globally optimal pathway at every step, the strategy guarantees that a canonical optimal target program $\mathcal{P}^*$ remains persistently reachable within the search tree.

In conclusion, because $\mathcal{P}^*$ inherently exists, the SMT delegation keeps the search space strictly finite, and the pruning guarantees the reachability of $\mathcal{P}^*$, the algorithm is complete. It will unconditionally terminate and return $\mathcal{P}^*$. □

4.4.3 Deterministic Search Space Reduction in BFS. Beyond soundness and completeness, the Common Prefix Abstraction yields a deterministic, strictly bounded state reduction specifically within the Breadth-First Search (BFS) framework, a theoretical guarantee absent in DFS.

Theorem 4.3 (Complexity Bound of BFS with Common Prefix Abstraction). *For a scene with $N = |\mathcal{E}|$ entities, the naive combinatorial search explores $O(N!)$ permutations. By applying Common*

Prefix Abstraction-based Pruning within layer-wise BFS, the maximum number of active states at depth k is strictly bounded by $\binom{N}{k}$, reducing the overall search complexity (in terms of SynthesizeStatement invocations) to $O(N \cdot 2^{N-1})$.

Proof. At any search depth $k \in \{0, \ldots, N\}$, a naive search maintains $P(N, k) = \frac{N!}{(N-k)!}$ active permutations. However, our abstraction defines states as equivalent if they share the same unordered subset of k declared entities. Because BFS strictly expands layer-by-layer, it exhaustively evaluates all depth-k permutations before proceeding. The prefix table $\mathcal{T}_{layer}$ deterministically collapses these into at most $\binom{N}{k}$ unique canonical states (the combinations of N entities taken k at a time), retaining only the optimal path for each subset.

Consequently, when expanding from depth k to $k+1$, each of the bounded $\binom{N}{k}$ states attempts to instantiate the remaining $(N-k)$ unassigned entities. The maximum total number of SynthesizeStatement invocations across the entire search is exactly bounded by:

$$\sum_{k=0}^{N-1} \binom{N}{k}(N-k) = N \cdot 2^{N-1}$$

This mathematically proves a deterministic complexity reduction from a factorial bound $O(N!)$ to an exponential bound $O(N \cdot 2^{N-1})$. □

Efficiency Contrast with DFS. Although DFS also employs pruning via $\mathcal{T}_{global}$, it lacks BFS's deterministic efficiency. By exploring depth-first, DFS may reach an entity subset E_{curr} via a sub-optimal permutation and exhaustively explore its sub-tree before discovering the optimal path. This triggers numerous redundant SynthesizeStatement calls and SMT evaluations. Consequently, DFS's practical state reduction is highly sensitive to branch ordering, whereas BFS mathematically guarantees optimal structural reduction.

4.5 Discussion

4.5.1 Trade-offs Between BFS and DFS. While Theorem 4.3 establishes a strict mathematical bound for BFS, DFS offers complementary advantages, creating a distinct time-memory trade-off.

1) Time Complexity. BFS guarantees deterministic branch reduction via Common Prefix Abstraction-based Pruning. Conversely, DFS is highly sensitive to exploration order; reaching a subset via a sub-optimal path early can trigger redundant sub-tree exploration. However, DFS offsets this theoretical vulnerability by rapidly diving to leaf nodes. This quickly establishes a tight global upper bound C^*, empowering aggressive Branch-and-Bound pruning for subsequent branches.

2) Space Complexity. BFS isolates memory overhead to the current depth. Its queue Q and layer-wise prefix table $\mathcal{T}_{layer}$ rebuild iteratively, peaking at $O(\binom{N}{\lfloor N/2 \rfloor})$. In contrast, while DFS features a lightweight $O(N)$ call stack, its global prefix table $\mathcal{T}_{global}$ must permanently cache visited states across all depths. Consequently, this global map grows monotonically, demanding up to $O(2^N)$ memory in the worst case.

4.5.2 Alternative Search Algorithms. In addition to BFS and DFS, other search algorithms can also be integrated into our framework. For instance, A* search [Hart et al. 1968], a classical best-first search algorithm, can be combined with Prefix Abstraction-based Pruning to guide the exploration towards lower-cost programs. Similar to BFS, A* can leverage the prefix abstraction to merge equivalent states and reduce redundant exploration. Prior work has successfully applied A* to program synthesis with synthesis-specific heuristics [Mell et al. 2024; Shah et al. 2020]. We empirically evaluate and compare these alternatives in Section 5.2.

5 Evaluation

This section evaluates the effectiveness and efficiency of our proposed method. We compare our approach against several LLM-based baselines. For the efficiency evaluation, we conduct an internal ablation of B&B and CPA within the DFS- and BFS-based configurations and compare these configurations with A*-based search[Hart et al. 1968]. Our evaluation aims to answer the following research questions:

- **RQ1** How effective is our method in synthesizing scenario programs compared to existing approaches?
- **RQ2** How efficient is our proposed algorithm, how effectively do the two pruning strategies improve the computational scalability, and how does it compare with A*-based search?

Benchmark. To address RQ1 and RQ2, we derive 20 scenario generation tasks from the annotated, real-world nuScenes [Caesar et al. 2020] dataset. To systematically evaluate performance across varying levels of difficulty, these scenes are structured to feature a strictly increasing number of entities, scaling incrementally from a relatively simple configuration of 5 entities up to a highly complex scene containing 24 entities.

Implementation and Experimental Setup. Our synthesis algorithm leverages dReal [Gao et al. 2013] as the underlying constraint solver to handle nonlinear formulas over the reals. Specifically, dReal implements the framework of δ-complete decision procedures [Gao et al. 2012]. This allows the tool to systematically evaluate various nonlinear real functions, including the trigonometric functions that are essential for our spatial layout synthesis. Since the baseline approaches primarily rely on cloud-based LLM API invocations, their effectiveness evaluations (RQ1) are hardware-independent. To rigorously answer RQ2, all efficiency and ablation experiments were executed sequentially in isolated Docker containers running on a single Linux server (AMD RT PRO 7995WX CPU), with each explicitly constrained to 4 cores and 16GB of RAM.

5.1 Results of RQ1

To address RQ1 and evaluate our method's effectiveness, we compare our approach against the following three scenario program synthesis techniques.

- **ChatScene:** We select ChatScene [Zhang et al. 2024] as a baseline, a recent work that leverages LLMs and a pre-built knowledge base to synthesize Scenic code from natural language. Its direct assembly mode is designed around a pre-built knowledge base and therefore cannot directly cover unseen layouts that are absent from that knowledge base. We consequently use the alternative few-shot prompting mode provided by ChatScene, which retrieves relevant code examples while allowing new Scenic programs to be generated for the layouts in our benchmark.
- **ScenicNL:** An existing work [Elmaaroufi et al. 2024] that utilizes a compositional prompting strategy to generate Scenic code from natural language. Specifically, it integrates various advanced techniques, such as Tree of Thoughts, few-shot prompting, constrained decoding, and a compiler-in-the-loop approach, to fully leverage the generation capabilities of LLMs.
- **DirectLLMGen:** We employ a multimodal LLM to directly synthesize scenario programs. The input context consists exclusively of a raw scene image, its corresponding label data file, and a supplementary image with rendered labels, deliberately omitting any textual description of the scene itself. Instead, our carefully designed natural language prompt is used strictly to enforce the syntax format of the target scenario language. This setup forces the LLM to rely on multimodal reasoning abilities to extract spatial layout from the visually augmented inputs, outputting the final program according to the predefined syntactic constraints.

To evaluate the LLM-based baselines, we utilize three frontier models: `GPT-4o`, `GPT-5.4`, and `Gemini-3.1-Pro`. Since ScenicNL's LMQL-based constrained decoding [Beurer-Kellner et al. 2023]

Table 1. Evaluation results of RQ1. The metrics in each cell represent **Syntax Correct** / **Semantic Correct** / **Average Program Cost** (Syn / Sem / Cost). For LLM-based methods, syntax success is achieved if at least one out of five attempts generates valid code. Semantic success is subsequently evaluated only on syntactically correct outputs, requiring at least one attempt to satisfy the semantic criteria. The average semantic cost is then computed exclusively over these semantically successful attempts. ✓ indicates correct, × indicates failure, and - indicates not applicable.

Task ID (# Ent.)	DirectLLMGen			ChatScene			ScenicNL			Ours
	GPT-4o	GPT-5.4	Gemini	GPT-4o	GPT-5.4	Gemini	GPT-4o	GPT-5.4	Gemini	
Task 1 (5)	✓ / × / -	✓ / × / -	✓ / ✓ / 10.0	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 20.0	✓ / ✓ / 20.0	✓ / ✓ / 20.0	✓ / ✓ / 0
Task 2 (6)	✓ / × / -	✓ / × / -	✓ / × / -	✓ / × / -	✓ / × / -	✓ / × / -	✓ / ✓ / 24.0	✓ / ✓ / 24.0	✓ / ✓ / 24.0	✓ / ✓ / 5.0
Task 3 (7)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 28.0	✓ / ✓ / 28.0	✓ / ✓ / 28.0	✓ / ✓ / 1.0
Task 4 (8)	✓ / × / -	✓ / × / -	✓ / ✓ / 32.0	× / - / -	✓ / × / -	✓ / × / -	✓ / ✓ / 32.0	✓ / ✓ / 32.0	✓ / ✓ / 32.0	✓ / ✓ / 7.0
Task 5 (9)	✓ / × / -	✓ / × / -	✓ / ✓ / 36.0	× / - / -	✓ / × / -	✓ / × / -	✓ / ✓ / 36.0	✓ / ✓ / 36.0	✓ / ✓ / 36.0	✓ / ✓ / 3.0
Task 6 (10)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	✓ / × / -	✓ / ✓ / 40.0	✓ / ✓ / 40.0	✓ / ✓ / 40.0	✓ / ✓ / 1.0
Task 7 (11)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	✓ / × / -	× / - / -	✓ / ✓ / 44.0	✓ / ✓ / 44.0	✓ / ✓ / 8.0
Task 8 (12)	✓ / × / -	✓ / × / -	✓ / × / -	✓ / × / -	✓ / × / -	✓ / × / -	✓ / ✓ / 48.0	✓ / ✓ / 48.0	✓ / ✓ / 48.0	✓ / ✓ / 6.0
Task 9 (13)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 52.0	✓ / ✓ / 52.0	✓ / ✓ / 52.0	✓ / ✓ / 9.0
Task 10 (14)	✓ / × / -	✓ / × / -	✓ / ✓ / 56.0	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 56.0	✓ / ✓ / 56.0	✓ / ✓ / 56.0	✓ / ✓ / 4.0
Task 11 (15)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 60.0	✓ / ✓ / 60.0	✓ / ✓ / 60.0	✓ / ✓ / 5.0
Task 12 (16)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 64.0	✓ / ✓ / 64.0	✓ / ✓ / 64.0	✓ / ✓ / 13.0
Task 13 (17)	✓ / × / -	✓ / × / -	✓ / × / -	✓ / × / -	✓ / × / -	✓ / × / -	✓ / ✓ / 68.0	✓ / ✓ / 68.0	✓ / ✓ / 68.0	✓ / ✓ / 8.0
Task 14 (18)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	✓ / × / -	✓ / ✓ / 72.0	✓ / ✓ / 72.0	✓ / ✓ / 72.0	✓ / ✓ / 7.0
Task 15 (19)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	✓ / × / -	✓ / ✓ / 76.0	✓ / ✓ / 76.0	✓ / ✓ / 76.0	✓ / ✓ / 7.0
Task 16 (20)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 80.0	✓ / ✓ / 80.0	✓ / ✓ / 80.0	✓ / ✓ / 0.0
Task 17 (21)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 84.0	✓ / ✓ / 84.0	✓ / ✓ / 84.0	✓ / ✓ / 12.0
Task 18 (22)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 88.0	✓ / ✓ / 88.0	✓ / ✓ / 88.0	✓ / ✓ / 0.0
Task 19 (23)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	✓ / × / -	✓ / ✓ / 92.0	✓ / ✓ / 92.0	✓ / ✓ / 92.0	✓ / ✓ / 10.0
Task 20 (24)	✓ / × / -	✓ / × / -	✓ / × / -	× / - / -	✓ / × / -	× / - / -	✓ / ✓ / 96.0	✓ / ✓ / 96.0	✓ / ✓ / 96.0	✓ / ✓ / 7.0
Overall	100.0% / 0.0% / -	100.0% / 0.0% / -	100.0% / 20.0% / 33.5	15.0% / 0.0% / -	100.0% / 0.0% / -	50.0% / 0.0% / -	95.0% / 95.0% / 58.7	100.0% / 100.0% / 58.0	100.0% / 100.0% / 58.0	**100.0% / 100.0% / 5.7**

requires token-level logit control unavailable in modern chat APIs, we adopt its alternative configuration[1]. For our proposed method, we employ the BFS strategy equipped with Common Prefix Abstraction-based Pruning.

In our evaluation, all methods are provided with the same nuScenes scene labels, including the entity types, positions, and headings. For R2SGen, these labels directly constitute the structured concrete scene. DirectLLMGen receives the raw image together with the label file and an image with rendered labels, while ChatScene and ScenicNL receive textual descriptions containing the same label information. All LLM-based configurations are evaluated with at most five generation attempts per task: ChatScene and DirectLLMGen are run five times, and ScenicNL uses one initial generation followed by at most four compiler-feedback retries.

Table 1 presents our experimental results evaluated across syntax correctness, semantic correctness, and program cost. Regarding **syntax correctness**, all programs are strictly validated via the official Scenic compiler. Our method, DirectLLMGen, and ScenicNL achieve near-perfect syntax pass rates, whereas ChatScene exhibits high variance. Our symbolic synthesis fundamentally guarantees strict syntax by design. DirectLLMGen and ScenicNL ensure validity via strict prompt constraints and ToT with compiler-in-the-loop feedback, respectively. Conversely, lacking explicit formatting or feedback loops, ChatScene relies solely on few-shot examples, making its performance highly unstable and dependent on the LLM's prior knowledge.

For syntactically valid programs, we evaluate **semantic correctness** (defined as whether the generated program's distribution space accurately encompasses the input scene). We use an automated evaluation pipeline for our method and DirectLLMGen (restricted to a Scenic subset), while ChatScene and ScenicNL (using full syntax) require a two-step evaluation: an initial automated check to ensure the compiled entity count matches the input (as any mismatch strictly invalidates the scene), followed by expert assessment. Results show our method and ScenicNL achieve complete semantic correctness, whereas DirectLLMGen only occasionally succeeds using `Gemini-3.1-Pro`. Crucially, all baselines achieve this merely by transcribing absolute coordinates from the labels. Even the sole exception (DirectLLMGen with `Gemini-3.1-Pro` on Task 1) simply calculated local relative positions.

[1]We utilize ScenicNL's `PREDICT_TOT_THEN_HYDE` configuration (combining ToT and few-shot prompting). Additionally, we ported the authors' compiler-in-the-loop feedback mechanism into this pipeline.

To quantify this, we evaluate the **program cost** (Def. 3.6). As theoretically established, our symbolic synthesis guarantees the minimum-cost program. Under our cost function, a lower cost indicates that a program relies more on relational descriptions and less on absolute coordinates. The evaluated baselines frequently reproduce the input coordinates directly, tying their outputs to the original coordinate frame. In contrast, R2SGen generates relational and re-samplable programs that preserve the encoded spatial layout while allowing controlled scenario variations. We discuss the practical implications of these programs in Section 6.

Impact of Natural Language Bias (User Study). The RQ1 evaluation above used expert-crafted prompts with explicit absolute coordinates. However, real-world user descriptions are inherently abstract and relational. To evaluate this bias, 8 computer science graduate students were provided with raw scene images and corresponding labels, and asked to describe the spatial layouts of 5 representative scenes in natural language without any structural constraints.

As Table 2 shows, ChatScene's syntax generation remains highly model-dependent. Notably, ScenicNL's syntax success rate declined significantly; lacking direct coordinates to copy, it struggled to generate valid complex structures. Crucially, the semantic performance of both baselines deteriorated sharply. This shows that, under the evaluated user-generated descriptions, the LLM-based configurations did not generate programs that correctly captured the corresponding spatial layouts.

Table 2. Comparison of generation performance between Expert-crafted descriptions (Expert) and User-generated descriptions (User). For Expert, results indicate success or failure. For User, results represent the average pass rate across 8 participants. 'Syn' and 'Sem' denote Syntax and Semantic outcomes, respectively.

Task	Model	ChatScene				ScenicNL			
		Expert		User		Expert		User	
		Syn	Sem	Syn	Sem	Syn	Sem	Syn	Sem
Task 1	GPT-4o	×	-	0.0%	0.0%	✓	✓	75.0%	0.0%
	GPT-5.4	✓	×	87.5%	0.0%	✓	✓	87.5%	0.0%
	Gemini-3.1-Pro	×	-	37.5%	0.0%	✓	✓	62.5%	0.0%
Task 5	GPT-4o	×	-	12.5%	0.0%	✓	✓	50.0%	0.0%
	GPT-5.4	✓	×	100.0%	0.0%	✓	✓	75.0%	0.0%
	Gemini-3.1-Pro	✓	×	37.5%	0.0%	✓	✓	75.0%	0.0%
Task 10	GPT-4o	×	-	12.5%	0.0%	✓	✓	50.0%	0.0%
	GPT-5.4	✓	×	75.0%	0.0%	✓	✓	87.5%	0.0%
	Gemini-3.1-Pro	×	-	62.5%	0.0%	✓	✓	87.5%	0.0%
Task 15	GPT-4o	×	-	12.5%	0.0%	✓	✓	75.0%	0.0%
	GPT-5.4	✓	×	100.0%	0.0%	✓	✓	87.5%	0.0%
	Gemini-3.1-Pro	✓	×	50.0%	0.0%	✓	✓	87.5%	0.0%
Task 20	GPT-4o	×	-	12.5%	0.0%	✓	✓	50.0%	0.0%
	GPT-5.4	✓	×	87.5%	0.0%	✓	✓	100.0%	0.0%
	Gemini-3.1-Pro	×	-	25.0%	0.0%	✓	✓	62.5%	0.0%

Answer to RQ1. Under the evaluated inputs, R2SGen achieves complete syntax and semantic correctness on all 20 scenes, whereas the LLM-based baselines attain such correctness only by transcribing absolute coordinates. Under our cost function, R2SGen produces substantially lower-cost, and therefore more relational and re-samplable programs than the baselines.

5.2 Results of RQ2

RQ2 evaluates the efficiency and scalability of our synthesis algorithms from two perspectives. We first conduct an ablation study to quantify the effects of Branch-and-Bound (B&B) and Common Prefix Abstraction-based Pruning (CPA) on the DFS- and BFS-based configurations. We then compare these configurations with A* search as an alternative to explore the same synthesis space.

We measure execution time and peak memory consumption across the 20 tasks. The evaluation includes nine configurations: one monolithic SMT baseline, six DFS/BFS configurations for the internal ablation, and two A*-based comparison configurations.

1) **Pure-SMT**: Our custom baseline encoding both combinatorial permutations and geometric constraints into a single monolithic SMT formula.
2) **DFS-None**: A naive Depth-First Search baseline without any pruning.
3) **BFS-None**: A naive Breadth-First Search baseline without any pruning.
4) **DFS-B&B**: DFS with only B&B.

Table 3. Detailed Performance across All Tasks (N = 5 to 24), where N denotes the number of entities within the scene, reflecting the increasing complexity of the scenarios. All reported metrics are averaged over 5 independent runs. TO = Timeout (> 7200s), OOM = Out of Memory (> 8GB).

Task	N	Pure-SMT		DFS-None		BFS-None		DFS-B&B		DFS-CPA		DFS-B&B-CPA		BFS-CPA		A*		A*+CPA	
		Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)
Task 1	5	TO	1449.9	0.47	43.7	0.44	43.7	0.43	43.8	0.50	46.0	**0.40**	47.6	0.47	43.6	0.42	**9.63**	0.42	9.66
Task 2	6	TO	2700.1	1.09	45.1	0.93	44.6	1.16	50.5	1.11	48.5	1.01	49.7	1.02	44.4	**0.90**	**13.99**	0.93	14.11
Task 3	7	TO	4799.7	1.55	47.3	1.41	49.8	1.21	49.1	1.47	49.4	1.17	49.5	1.40	46.3	**1.10**	**14.01**	1.15	14.13
Task 4	8	TO	7801.1	4.13	52.0	4.15	77.6	3.55	49.8	3.11	53.4	3.01	53.9	2.82	53.5	2.67	17.73	**2.46**	**14.19**
Task 5	9	5029.83	OOM	8.73	50.4	13.61	533.4	5.86	47.7	1.82	53.2	1.73	51.3	1.77	49.2	3.88	71.58	**1.43**	**14.20**
Task 6	10	6236.34	OOM	63.46	51.2	111.11	5005.9	3.28	49.0	3.10	56.1	2.74	55.9	2.91	51.7	3.10	46.10	**1.91**	**14.26**
Task 7	11	6083.55	OOM	1223.93	54.1	219.19	OOM	416.44	51.3	5.62	57.2	5.73	58.5	5.41	53.3	49.28	1129.74	**4.74**	**14.68**
Task 8	12	5392.32	OOM	TO	53.1	188.82	OOM	607.34	54.5	7.30	57.4	6.55	58.3	6.73	54.2	336.55	7106.95	**5.83**	**15.64**
Task 9	13	3580.38	OOM	TO	52.7	209.74	OOM	TO	53.8	9.09	58.5	8.24	59.0	7.89	54.5	OOM	OOM	**6.71**	**19.83**
Task 10	14	307.64	OOM	TO	54.5	163.66	OOM	1498.79	53.2	12.30	63.0	9.34	58.7	10.82	54.8	OOM	OOM	**7.93**	**16.37**
Task 11	15	8.03	OOM	TO	53.7	170.63	OOM	TO	52.5	15.44	71.7	10.65	66.9	11.93	61.8	245.85	5152.39	**7.03**	**17.97**
Task 12	16	8.03	OOM	TO	53.8	166.62	OOM	TO	52.8	23.81	102.6	23.77	103.1	15.39	**74.7**	OOM	OOM	**12.12**	84.70
Task 13	17	8.68	OOM	TO	54.3	169.90	OOM	TO	54.2	58.06	155.6	43.44	150.2	26.24	100.8	OOM	OOM	**13.91**	**99.63**
Task 14	18	9.03	OOM	TO	54.6	142.23	OOM	TO	54.4	89.31	261.5	72.53	254.2	39.20	**152.2**	OOM	OOM	**15.56**	153.83
Task 15	19	9.44	OOM	TO	52.6	143.92	OOM	TO	53.4	169.83	480.5	97.75	420.4	72.49	258.1	OOM	OOM	**13.28**	**125.81**
Task 16	20	10.05	OOM	TO	53.3	126.19	OOM	5.31	**53.8**	374.19	886.2	**5.15**	63.2	101.36	438.1	OOM	OOM	51.52	862.39
Task 17	21	10.24	OOM	TO	51.4	133.70	OOM	TO	53.3	615.63	1735.3	615.73	1736.0	279.37	**874.8**	OOM	OOM	**238.07**	2472.01
Task 18	22	11.09	OOM	TO	52.4	131.46	OOM	6.68	**52.7**	1775.63	3286.0	**6.15**	62.5	445.23	1593.1	OOM	OOM	213.80	3666.35
Task 19	23	12.44	OOM	TO	54.2	137.77	OOM	TO	53.2	3849.99	6704.4	3461.34	6681.6	1145.22	**3140.4**	OOM	OOM	**650.49**	7408.19
Task 20	24	13.13	OOM	TO	54.1	141.69	OOM	TO	54.7	3630.59	OOM	1851.33	OOM	**2515.80**	**6129.6**	OOM	OOM	OOM	OOM

5) **DFS-CPA**: DFS with only CPA.
6) **DFS-B&B-CPA**: DFS equipped with both B&B and CPA.
7) **BFS-CPA**: BFS equipped with CPA.
8) **A***: An A* search baseline without CPA.
9) **A*+CPA**: A hybrid comparison configuration that combines A* with CPA.

A*-based comparison. Prior work has applied A* to program synthesis using synthesis-specific heuristics. Shah et al. [Shah et al. 2020] derive an approximately admissible neural heuristic from a continuous relaxation to guide A* and iterative-deepening search, while Mell et al. [Mell et al. 2024] derive an abstract-interpretation-based heuristic that bounds the best achievable objective within subtrees of partial programs. Motivated by these studies, we instantiate A* for our synthesis space using $f = g + h$, where g is the accumulated cost of the current partial program and h is a relaxed lower bound on the cost of declaring the remaining entities. For each undeclared entity, the relaxation permits references to any entity in the scene and uses the minimum possible declaration cost. Because these relaxed choices include all references available to the actual search, h cannot overestimate the true remaining cost and is admissible. Pure A* retains different declaration orders as distinct states, whereas A*+CPA merges equivalent prefixes with the same declared entity set.

Table 3 reports the execution time and peak memory consumption of all nine configurations. We analyze the results in terms of the necessity of decoupled synthesis, the effectiveness of CPA after decoupling, and the time–memory trade-offs among the search strategies.

1) Necessity of Decoupled Synthesis. The monolithic Pure-SMT baseline jointly encodes entity permutations and geometric constraints in a single formula and fails on all evaluated tasks, timing out for $N \leq 8$ and reaching the memory limit for $N \geq 9$. In contrast, after separating declaration-order search from SMT-based geometric reasoning, even DFS-None and BFS-None complete several small tasks within seconds. This contrast demonstrates that decoupling the discrete and geometric parts is essential for making the synthesis problem tractable. However, the unpruned decoupled configurations still fail as the number of entities grows, indicating that decoupling alone does not resolve the combinatorial search space.

2) Effectiveness of CPA after Decoupling. Within the decoupled architecture, the main bottleneck is the repeated exploration of declaration orders. DFS-None times out from Task 8 onward, and BFS-None reaches the memory limit from Task 7 onward. Pure A* changes the expansion order and is more competitive on smaller tasks, but it still retains declaration orders and reaches the memory limit on most tasks from $N = 13$ onward. CPA directly addresses this redundancy. DFS+B&B remains sensitive to branch order and times out on many tasks, whereas DFS+B&B-CPA completes every task through $N = 23$. The A* comparison shows the same effect: at

$N = 12$, A*+CPA reduces execution time from 336.55 s to 5.83 s and peak memory from 7,106.95 MB to 15.64 MB, and also completes every task through $N = 23$. These results show that CPA effectively removes declaration-order redundancy across all three search orders: DFS, BFS, and A*.

3) Time–Memory Trade-offs. DFS+B&B-CPA can be exceptionally fast when depth-first search finds a low-cost complete program early, as on $N = 20$ and $N = 22$, where it finishes in 5.15 s and 6.15 s, respectively. BFS-CPA avoids a persistent global prefix table by releasing states layer by layer and is the only configuration that completes $N = 24$ within the memory limit. A*+CPA also achieves strong runtime performance: it is the fastest completed configuration for $N = 16$–19 and $N = 23$, and it is consistently faster than BFS-CPA from $N = 16$ through $N = 23$. This speed comes with higher memory consumption at the largest scales. From $N = 20$ to $N = 23$, A*+CPA uses approximately two to three times the peak memory of BFS-CPA; at $N = 23$, the two configurations consume 7,408.19 MB and 3,140.40 MB, respectively. A*+CPA then reaches the memory limit at $N = 24$, whereas BFS-CPA completes the task. These results show that A*+CPA often favors execution time, BFS-CPA provides stronger memory efficiency at the largest scales, and DFS+B&B-CPA benefits most when an effective upper bound is established early. To further investigate how the distribution of complete-program costs in the search space relates to search performance, we conduct a supplementary analysis (Appendix B). Since R2SGen searches for a low-cost program, we summarize each scene's cost distribution by its mean program cost, where a lower mean means that more low-cost programs exist. The runtime of DFS+B&B-CPA varies widely across scenes and is markedly shorter on scenes whose cost distribution admits more low-cost programs, whereas A*+CPA and BFS-CPA show no clear trend with the mean cost.

Answer to RQ2: Pure-SMT confirms the necessity of decoupling, while the results across DFS, BFS, and A* show that CPA is critical for removing declaration-order redundancy. DFS+B&B-CPA benefits from early upper bounds, BFS-CPA provides the strongest memory scalability and is the only configuration that completes $N = 24$, and A*+CPA is competitive in runtime but consumes more memory at larger scales.

5.3 Threats to Validity

The RQ1 comparison is subject to differences in task formulation and input modality. ChatScene and ScenicNL were designed to translate natural-language descriptions into Scenic, whereas R2SGen synthesizes a relational program from a structured concrete scene. Providing all methods with the same quantitative spatial facts enables comparison of their generated programs, but does not reproduce the native end-to-end use case of each system. In addition, program cost measures the preference for relational descriptions defined by our cost function; it is not a general measure of program quality or language understanding. Therefore, the RQ1 results should be interpreted as an output-level comparison of syntax validity, consistency with the supplied scene, and relational abstraction under the evaluated inputs.

6 Discussion

6.1 Practical Implications and Benefits

The practical benefit of R2SGen is that it transforms a structured concrete scene into a relational and re-samplable Scenic program. Replaying absolute coordinates reproduces only one concrete scene, whereas the synthesized program can generate multiple scenario variations while preserving the spatial relations encoded from the input scene. The input scene itself remains encompassed by the program, while relational primitives and `Range` expressions allow controlled variations in

entity positions and headings. RQ2 further shows that such programs can be synthesized for the evaluated scenes containing up to 24 entities.

These scenario variations can support synthetic data generation and the targeted expansion of rare or difficult scenes. Johnson-Roberson et al. [Johnson-Roberson et al. 2017] showed that synthetic images generated in a virtual world could train a vehicle detector that transferred to real-world images, and Fremont et al. [Fremont et al. 2019] showed that Scenic-generated partial-occlusion scenes improved detector performance on difficult cases when added to training data. However, the Scenic programs used by Fremont et al. were hand-authored, while some existing Scenic generation methods start from natural-language descriptions. R2SGen takes a different path: it synthesizes relational programs directly from real-world structured scenes. Our preliminary evaluation (Appendix A) further shows that these structured scenes can potentially be obtained automatically from unlabeled images. For users, this means providing a single real-world image suffices; no manual program writing or spatial description is needed.

6.2 Integration with Perception, VLMs, and NL-based Methods

We further consider how R2SGen can be integrated with perception systems, VLMs, and NL-based methods, and preliminarily test the feasibility of the image-based frontend. In such an integrated workflow, a detector or VLM extracts the entity types, positions, and headings from images or videos, forming the structured concrete scene required by R2SGen. R2SGen then uses the scene as the synthesis specification, applying SMT to check candidate spatial relations and synthesize their numerical ranges while searching for a low-cost relational Scenic program.

NL-based methods can also assist the symbolic search after a structured concrete scene is available. Pairwise position and heading relations can be derived from its labels and expressed as stylized NL, from which an LLM can propose a declaration order and candidate reference relations as a high-level sketch. Such guidance may be particularly useful for DFS, because a promising declaration order may produce a low-cost complete program earlier, tighten the global upper bound, and enable stronger pruning. Our preliminary experiment below evaluates the image-based frontend; implementing and evaluating this NL-guided search strategy remains future work.

The preliminary image-to-structured-scene evaluation covers 50 images from the nuScenes validation split; the complete protocol and results are reported in Appendix A. Both GPT-5.5 and Claude opus-4.8 show limited accuracy in recovering vehicle positions and headings. The domain-aligned PGD detector [Wang et al. 2021] trained on nuScenes performs substantially better, while the same architecture trained on KITTI degrades sharply, confirming that the advantage of a perception model depends critically on domain alignment. These results indicate that a domain-aligned perception model is the most promising source of the structured scene required by R2SGen, while current general-purpose VLMs are not yet accurate enough for direct use.

7 Related Work

While many computer vision approaches prioritize *visual realism* (*e.g.*, textures), we focus on *structural realism*, which ensures physical plausibility for real-world scenarios. Consequently, appearance-centric generation is beyond our scope. We categorize relevant literature into: (1) Data-Driven Scenario Generation, (2) LLM-based Scenario Generation, and (3) Sketch-based Program Synthesis.

Data-Driven Scenario Generation. Recent data-driven works leverage neural architectures like VAEs [Kingma and Welling 2014], GANs [Goodfellow et al. 2014], Diffusion Models [Ho et al. 2020], and Autoregressive models to learn spatial distributions. These methods typically synthesize scenes through four intermediate representations [Wen et al. 2025]: explicit parameters [Feng et al. 2023; Lu et al. 2024; Tan et al. 2021; Tang et al. 2024], scene graphs [Devaranjan et al. 2020; Kar

et al. 2019], implicit latent spaces [Li et al. 2024a; Ren et al. 2024], and semantic layouts [Chen et al. 2025; Xie et al. 2024]. While representations like scene graphs and explicit parameters successfully preserve the high-level semantics of scenario topologies (*e.g.*, categorical relationships and rough spatial layouts), they inherently leave out low-level geometric precision. In contrast, our scenario program-based approach perfectly bridges this gap, enabling mathematically precise editability over scenario structure. Furthermore, unlike data-driven methods that risk generating physically invalid scenarios, our symbolic paradigm utilizes SMT as a rigorous constraint solver to ensure physical and semantic validity. This precise constraint capability also enables high-fidelity real-world scene reconstruction, such as traffic accident restoration, aiding accident analysis and safety validation.

LLM-based Scenario Generation. Leveraging the prior knowledge of LLMs, recent approaches broadly follow two paradigms. The first generates intermediate structural representations like scene graphs [Fu et al. 2024; Gao et al. 2024; Li et al. 2024b; Yang et al. 2024]. To achieve more precise editability, the second paradigm employs LLMs for procedural code generation. Some works generate Python-like scripts [Hu et al. 2024; Sun et al. 2025; Zhang et al. 2025b], while others synthesize domain-specific programs [Aguina-Kang et al. 2024; Gumin et al. 2025]. Notably, works like ChatScene [Zhang et al. 2024] and ScenicNL [Elmaaroufi et al. 2024] target Scenic. While they excel at generating diverse scenarios with dynamic behaviors from open-vocabulary inputs, they struggle to capture accurate spatial relationships in static scenes. Consequently, they easily produce physically impossible configurations, causing critical issues for downstream tasks. In contrast, our approach grounds the synthesis process directly in real-world scene data. By leveraging an SMT-based symbolic framework, we rigorously extract and encode precise geometric relationships from physical environments into scenario programs, thereby guaranteeing structural realism.

Sketch-based Program Synthesis. Sketch-based synthesis is a classic paradigm pioneered by Solar-Lezama et al. [Solar-Lezama 2009], which allows programmers to write partial programs with "*holes*" to reduce the search space. Recent works advance both the automated generation and efficient solving of sketches [Chen et al. 2023, 2020; Wang et al. 2018b; Zhang et al. 2025a]. Specifically, generation methods extract sketches from natural language or input-output examples [Chen et al. 2023, 2020; Zhang et al. 2025a], while solving techniques predominantly employ enumerative search to instantiate holes [Gulwani et al. 2017; Lee 2021; Reger and Zohar 2024]. To improve efficiency, abstraction techniques are often leveraged to verify intermediate states and prune infeasible spaces [Guo et al. 2020; Liang and Naik 2011; Wang et al. 2018a,b; Zhang et al. 2025a]. In contrast, our approach directly generates template-based sketches using semantic information from real world scenes. For solving, we combine enumerative search [Barke et al. 2020] with SMT solving: the search phase instantiates discrete high-level spatial relations, while the SMT solver verifies their geometric feasibility and synthesizes the corresponding continuous numerical parameters, significantly accelerating the overall process.

8 Conclusion and Future Work

In this paper, we presented R2SGen to address the sim-to-real gap in scenario generation. Instead of relying on monolithic constraint solving (which we showed to be intractable even for simple cases), we demonstrated that a decoupled search-and-solve architecture can efficiently navigate the vast permutation space of spatial relationships. By pruning the search tree, our approach successfully scales to highly dense environments with up to 24 entities.

To scale to even larger scenes, future work will investigate more advanced heuristic pruning techniques. Furthermore, we plan to extend our framework's expressiveness by supporting more complex syntactic constructs and incorporating dynamic agent behaviors. Finally, we plan to integrate perception models and natural-language-guided search toward an end-to-end pipeline from images to Scenic programs.

Data Availability Statement

The artifact associated with this paper is available at https://doi.org/10.5281/zenodo.21476337.

Acknowledgments

This research was supported by National Key R&D Program of China (No. 2024YFF0908003) and the NSFC Program (No. 62172429). We also thank the anonymous reviewers for their valuable feedback.

A Preliminary Image-to-Structured-Scene Evaluation

Table 4. Vehicle detection and pose estimation on 50 nuScenes validation images (average 6.0 vehicles per image within 50 m). TP, FP, and FN are per-image averages; position and heading errors are medians over matched pairs.

Method	Precision	Recall	F1	TP/img	FP/img	FN/img	Pos. Err.(m)	Head Err.(°)
GPT-5.5	0.359	0.404	0.362	2.0	4.0	4.1	2.59	32.3
Claude opus-4.8	0.260	0.258	0.236	1.2	3.7	4.9	3.12	50.1
PGD-nuScenes	0.832	0.449	0.548	2.5	0.4	3.6	0.89	4.5
PGD-KITTI	0.089	0.184	0.101	0.6	6.0	5.5	3.07	96.9

R2SGen takes a structured concrete scene as input. This experiment asks whether current vision-language models (VLMs) and perception models can produce such a scene directly from images, *i.e.*, whether the upstream frontend of the R2SGen pipeline can be automated. Given an image, each frontend must detect all vehicles within 50 m of the ego vehicle and estimate their (x, y) positions and headings in an ego-centric frame ($+x$ right, $+y$ forward, heading counter-clockwise from $+y$). We use 50 images from the nuScenes validation split, containing on average 6.0 vehicles per image within 50 m. We use the validation split to ensure that PGD-nuScenes was not trained on any of the evaluated images.

Predicted and ground-truth vehicles are paired by optimal one-to-one assignment (the Hungarian algorithm [Kuhn 1955; Munkres 1957]): each prediction is matched to at most one ground-truth vehicle so as to minimize the total matching distance, and pairs farther than 5 m are rejected. A **true positive (TP)** is a prediction correctly matched to a ground-truth vehicle within 5 m; a **false positive (FP)** is an unmatched prediction (a hallucinated vehicle that does not exist in the scene); a **false negative (FN)** is an unmatched ground-truth vehicle (a real vehicle that was missed). **Precision** = TP/(TP + FP) is the fraction of predictions that are correct; **recall** = TP/(TP + FN) is the fraction of ground-truth vehicles that are detected; **F1** = $2 \times \mathrm{TP}/(2 \times \mathrm{TP} + \mathrm{FP} + \mathrm{FN})$ combines both into a single score from 0 to 1 that penalizes hallucinations and missed detections equally. For matched pairs, we report the median **position error** (Euclidean distance, in meters) and the median **heading error** (circular angular difference, in degrees).

We evaluate four configurations. GPT-5.5 and Claude opus-4.8 are two general-purpose VLMs, each given the same task-specific prompt (defining the ego-centric frame and requesting vehicle positions and headings within 50 m); each VLM is run three times per image, and the per-image metrics are averaged across the three runs. The other two use the Probabilistic and Geometric Depth (PGD) monocular 3D detector [Wang et al. 2021]: PGD-nuScenes uses a checkpoint trained on nuScenes (the same domain as the test images), while PGD-KITTI uses the same architecture trained on KITTI and applied to nuScenes images without adaptation. Both PGD configurations use a confidence threshold of 0.1.

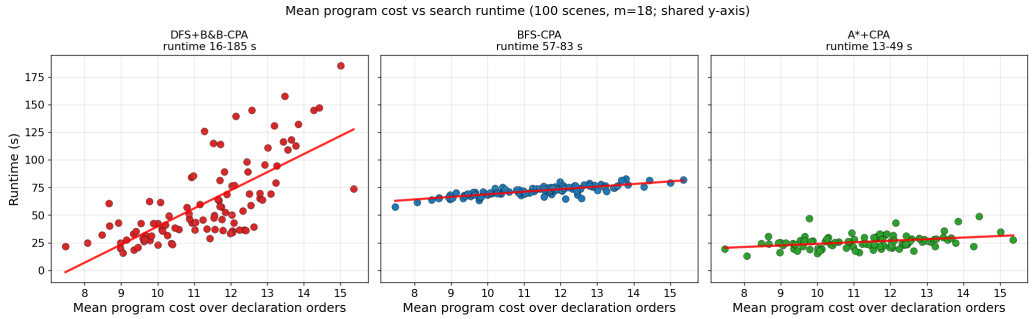


Fig. 4. Mean program cost versus search runtime over 100 controlled scenes (N = 18). The three panels show DFS+B&B-CPA, BFS-CPA, and A*+CPA; each title reports the runtime range.

Table 4 shows that PGD-nuScenes achieves the best performance across all metrics, with the highest F1 (0.548) and precision (0.832) and the lowest position and heading errors (0.89 m and 4.5°). We note that while PGD-nuScenes achieves high precision (0.832), its recall remains moderate (0.449); a further analysis shows that recall drops to only 0.21 for ground-truth vehicles beyond 30 m, reflecting the fundamental difficulty of monocular depth estimation at long range. The two VLMs are substantially worse: GPT-5.5 (F1 = 0.362, position error 2.59 m, heading error 32.3°) outperforms Claude opus-4.8 (F1 = 0.236, 3.12 m, 50.1°). PGD-KITTI, using the same architecture trained on a different domain, collapses to F1 = 0.101 with a heading error of 96.9°, showing that the advantage of a perception model depends on domain alignment.

These results suggest that a domain-aligned perception model is the most promising source of structured scenes for R2SGen, while current general-purpose VLMs are not yet accurate enough for direct use.

B Program-Cost Distribution and Search Behavior

We investigate how the distribution of complete-program costs in the search space relates to the performance of DFS+B&B-CPA, BFS-CPA, and A*+CPA. We use 100 controlled scenes, each containing 18 non-ego vehicles and a fixed ego vehicle, that cover a broad range of program-cost distributions. To summarize each scene's cost distribution, we use its mean program cost—a lower mean indicates that more low-cost programs exist.

Since the cost of declaring an entity depends only on the set of entities already declared (the property exploited by our Common Prefix Abstraction pruning), we can use a dynamic program to obtain the number of declaration orders at each total cost without enumerating all $N!$ complete orders.

We evaluate DFS+B&B-CPA, BFS-CPA, and A*+CPA on each of the 100 scenes. DFS+B&B-CPA's runtime varies widely across the 100 scenes, ranging from 16 to 185 s, and tends to increase with the mean cost. We speculate that scenes admitting more low-cost programs let the search reach a low-cost complete program earlier and thereby tighten the branch-and-bound upper bound sooner. Neither A*+CPA (13–49 s) nor BFS-CPA (57–83 s) shows a clear dependence on the mean cost.

References

Rio Aguina-Kang, Maxim Gumin, Do Heon Han, Stewart Morris, Seung Jean Yoo, Aditya Ganeshan, R. Kenny Jones, Qiuhong Anna Wei, Kailiang Fu, and Daniel Ritchie. 2024. Open-Universe Indoor Scene Generation using LLM Program

Synthesis and Uncurated Object Databases. *CoRR* abs/2403.09675 (2024). arXiv:2403.09675 doi:10.48550/ARXIV.2403.09675

Shraddha Barke, Hila Peleg, and Nadia Polikarpova. 2020. Just-in-time learning for bottom-up enumerative synthesis. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 227:1–227:29. doi:10.1145/3428295

Luca Beurer-Kellner, Marc Fischer, and Martin T. Vechev. 2023. Prompting Is Programming: A Query Language for Large Language Models. *Proc. ACM Program. Lang.* 7, PLDI (2023), 1946–1969. doi:10.1145/3591300

Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. 2020. nuScenes: A Multimodal Dataset for Autonomous Driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 11618–11628. doi:10.1109/CVPR42600.2020.01164

Minglin Chen, Longguang Wang, Sheng Ao, Ye Zhang, Kai Xu, and Yulan Guo. 2025. Layout2Scene: 3D Semantic Layout Guided Scene Generation via Geometry and Appearance Diffusion Priors. *CoRR* abs/2501.02519 (2025). arXiv:2501.02519 doi:10.48550/ARXIV.2501.02519

Qiaochu Chen, Arko Banerjee, Çagatay Demiralp, Greg Durrett, and Isil Dillig. 2023. Data Extraction via Semantic Regular Expression Synthesis. *Proc. ACM Program. Lang.* 7, OOPSLA2 (2023), 1848–1877. doi:10.1145/3622863

Qiaochu Chen, Xinyu Wang, Xi Ye, Greg Durrett, and Isil Dillig. 2020. Multi-modal synthesis of regular expressions. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 487–502. doi:10.1145/3385412.3385988

Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. 2020. Meta-Sim2: Unsupervised Learning of Scene Structure for Synthetic Data Generation. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XVII (Lecture Notes in Computer Science, Vol. 12362)*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer, 715–733. doi:10.1007/978-3-030-58520-4_42

Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings (Proceedings of Machine Learning Research, Vol. 78)*. PMLR, 1–16. http://proceedings.mlr.press/v78/dosovitskiy17a.html

Karim Elmaaroufi, Devan Shanker, Ana Cismaru, Marcell Vazquez-Chanlatte, Alberto L. Sangiovanni-Vincentelli, Matei Zaharia, and Sanjit A. Seshia. 2024. Generating Probabilistic Scenario Programs from Natural Language. *CoRR* abs/2405.03709 (2024). arXiv:2405.03709 doi:10.48550/ARXIV.2405.03709

Lan Feng, Quanyi Li, Zhenghao Peng, Shuhan Tan, and Bolei Zhou. 2023. TrafficGen: Learning to Generate Diverse and Realistic Traffic Scenarios. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*. IEEE, 3567–3575. doi:10.1109/ICRA48891.2023.10160296

Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2019. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 63–78. doi:10.1145/3314221.3314633

Rao Fu, Zehao Wen, Zichen Liu, and Srinath Sridhar. 2024. AnyHome: Open-Vocabulary Generation of Structured and Textured 3D Homes. In *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part XXXIX (Lecture Notes in Computer Science, Vol. 15097)*, Ales Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (Eds.). Springer, 52–70. doi:10.1007/978-3-031-72933-1_4

Gege Gao, Weiyang Liu, Anpei Chen, Andreas Geiger, and Bernhard Schölkopf. 2024. GraphDreamer: Compositional 3D Scene Synthesis from Scene Graphs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 21295–21304. doi:10.1109/CVPR52733.2024.02012

Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. 2012. Delta-Complete Decision Procedures for Satisfiability over the Reals. *CoRR* abs/1204.3513 (2012). arXiv:1204.3513 http://arxiv.org/abs/1204.3513

Sicun Gao, Soonho Kong, and Edmund M. Clarke. 2013. dReal: An SMT Solver for Nonlinear Theories over the Reals. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7898)*, Maria Paola Bonacina (Ed.). Springer, 208–214. doi:10.1007/978-3-642-38574-2_14

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. *CoRR* abs/1406.2661 (2014). arXiv:1406.2661 http://arxiv.org/abs/1406.2661

Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program synthesis. *Foundations and Trends in Programming Languages* 4, 1-2 (2017), 1–119.

Maxim Gumin, Do Heon Han, Seung Jean Yoo, Aditya Ganeshan, R. Kenny Jones, Kailiang Fu, Rio Aguina-Kang, Stewart Morris, and Daniel Ritchie. 2025. Procedural Scene Programs for Open-Universe Scene Generation: LLM-Free Error Correction via Program Search. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers, SA Conference Papers*

2025, Hong Kong, December 15-18, 2025, Taku Komura, Michael Wimmer, and Hongbo Fu (Eds.). ACM, 141:1–141:11. doi:10.1145/3757377.3763930

Zheng Guo, Michael James, David Justo, Jiaxiao Zhou, Ziteng Wang, Ranjit Jhala, and Nadia Polikarpova. 2020. Program synthesis by type-guided abstraction refinement. *Proc. ACM Program. Lang.* 4, POPL (2020), 12:1–12:28. doi:10.1145/3371080

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* 4, 2 (1968), 100–107. doi:10.1109/TSSC.1968.300136

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. 2024. SceneCraft: An LLM Agent for Synthesizing 3D Scenes as Blender Code. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024 (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR / OpenReview.net, 19252–19282. https://proceedings.mlr.press/v235/hu24g.html

Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. 2017. Driving in the Matrix: Can virtual worlds replace human-generated annotations for real world tasks?. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*. IEEE, 746–753. doi:10.1109/ICRA.2017.7989092

Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. 2019. Meta-Sim: Learning to Generate Synthetic Datasets. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 4550–4559. doi:10.1109/ICCV.2019.00465

Edward Kim, Jay Shenoy, Sebastian Junges, Daniel J. Fremont, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2022. Querying Labelled Data with Scenario Programs for Sim-to-Real Validation. In *13th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2022, Milano, Italy, May 4-6, 2022*. IEEE, 34–45. doi:10.1109/ICCPS54341.2022.00010

Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1312.6114

Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

Woosuk Lee. 2021. Combining the top-down propagation and bottom-up enumeration for inductive program synthesis. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–28. doi:10.1145/3434335

Xinyang Li, Zhangyu Lai, Linning Xu, Yansong Qu, Liujuan Cao, Shengchuan Zhang, Bo Dai, and Rongrong Ji. 2024a. Director3D: Real-world Camera Trajectory and 3D Scene Generation from Text. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/hash/89566c18d5b3e9836e8e16fde010b41d-Abstract-Conference.html

Xiao-Lei Li, Haodong Li, Hao-Xiang Chen, Tai-Jiang Mu, and Shi-Min Hu. 2024b. DIScene: Object Decoupling and Interaction Modeling for Complex Scene Generation. In *SIGGRAPH Asia 2024 Conference Papers, SA 2024, Tokyo, Japan, December 3-6, 2024*, Takeo Igarashi, Ariel Shamir, and Hao (Richard) Zhang (Eds.). ACM, 101:1–101:12. doi:10.1145/3680528.3687589

Percy Liang and Mayur Naik. 2011. Scaling abstraction refinement via pruning. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, Mary W. Hall and David A. Padua (Eds.). ACM, 590–601. doi:10.1145/1993498.1993567

Jack Lu, Kelvin Wong, Chris Zhang, Simon Suo, and Raquel Urtasun. 2024. SceneControl: Diffusion for Controllable Traffic Scene Generation. In *IEEE International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan, May 13-17, 2024*. IEEE, 16908–16914. doi:10.1109/ICRA57147.2024.10610324

Stephen Mell, Steve Zdancewic, and Osbert Bastani. 2024. Optimal Program Synthesis via Abstract Interpretation. *Proc. ACM Program. Lang.* 8, POPL (2024), 457–481. doi:10.1145/3632858

James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* 5, 1 (1957), 32–38.

Giles Reger and Yoni Zohar (Eds.). 2024. *Proceedings of the 22nd International Workshop on Satisfiability Modulo Theories co-located with the 36th International Conference on Computer Aided Verification (CAV 2024), Montreal, Canada, July, 22-23, 2024*. CEUR Workshop Proceedings, Vol. 3725. CEUR-WS.org. https://ceur-ws.org/Vol-3725

Xuanchi Ren, Jiahui Huang, Xiaohui Zeng, Ken Museth, Sanja Fidler, and Francis Williams. 2024. XCube: Large-Scale 3D Generative Modeling using Sparse Voxel Hierarchies. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 4209–4219. doi:10.1109/CVPR52733.2024.00403

Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Martins Mozeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. 2020. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. In *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*. IEEE, 1–6. doi:10.1109/ITSC45102.2020.9294422

Ameesh Shah, Eric Zhan, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. 2020. Learning Differentiable Programs with Admissible Neural Heuristics. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/342285bb2a8cadef22f667eeb6a63732-Abstract.html

Armando Solar-Lezama. 2009. The Sketching Approach to Program Synthesis. In *Programming Languages and Systems, 7th Asian Symposium, APLAS 2009, Seoul, Korea, December 14-16, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5904)*, Zhenjiang Hu (Ed.). Springer, 4–13. doi:10.1007/978-3-642-10672-9_3

Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 2025. 3D-GPT: Procedural 3D Modeling with Large Language Models. In *International Conference on 3D Vision, 3DV 2025, Singapore, March 25-28, 2025*. IEEE, 1253–1263. doi:10.1109/3DV66043.2025.00119

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. 2020. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2443–2451. doi:10.1109/CVPR42600.2020.00252

Shuhan Tan, Kelvin Wong, Shenlong Wang, Sivabalan Manivasagam, Mengye Ren, and Raquel Urtasun. 2021. SceneGen: Learning To Generate Realistic Traffic Scenes. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 892–901. doi:10.1109/CVPR46437.2021.00095

Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. 2024. DiffuScene: Denoising Diffusion Models for Generative Indoor Scene Synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 20507–20518. doi:10.1109/CVPR52733.2024.01938

Tai Wang, Xinge Zhu, Jiangmiao Pang, and Dahua Lin. 2021. Probabilistic and Geometric Depth: Detecting Objects in Perspective. In *Conference on Robot Learning, 8-11 November 2021, London, UK (Proceedings of Machine Learning Research, Vol. 164)*, Aleksandra Faust, David Hsu, and Gerhard Neumann (Eds.). PMLR, 1475–1485. https://proceedings.mlr.press/v164/wang22i.html

Xinyu Wang, Greg Anderson, Isil Dillig, and Kenneth L. McMillan. 2018a. Learning Abstractions for Program Synthesis. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10981)*, Hana Chockler and Georg Weissenbacher (Eds.). Springer, 407–426. doi:10.1007/978-3-319-96145-3_22

Xinyu Wang, Isil Dillig, and Rishabh Singh. 2018b. Program synthesis using abstraction refinement. *Proc. ACM Program. Lang.* 2, POPL (2018), 63:1–63:30. doi:10.1145/3158151

Beichen Wen, Haozhe Xie, Zhaoxi Chen, Fangzhou Hong, and Ziwei Liu. 2025. 3D Scene Generation: A Survey. *CoRR* abs/2505.05474 (2025). arXiv:2505.05474 doi:10.48550/ARXIV.2505.05474

Haozhe Xie, Zhaoxi Chen, Fangzhou Hong, and Ziwei Liu. 2024. CityDreamer: Compositional Generative Model of Unbounded 3D Cities. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 9666–9675. doi:10.1109/CVPR52733.2024.00923

Yixuan Yang, Junru Lu, Zixiang Zhao, Zhen Luo, James J. Q. Yu, Victor Sanchez, and Feng Zheng. 2024. LLplace: The 3D Indoor Scene Layout Generation and Editing via Large Language Model. *CoRR* abs/2406.03866 (2024). arXiv:2406.03866 doi:10.48550/ARXIV.2406.03866

Jiawei Zhang, Chejian Xu, and Bo Li. 2024. ChatScene: Knowledge-Enabled Safety-Critical Scenario Generation for Autonomous Vehicles. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 15459–15469. doi:10.1109/CVPR52733.2024.01464

Wenmeng Zhang, Zhenbang Chen, and Weijiang Hong. 2025a. Multi-modal Sketch-Based Behavior Tree Synthesis. *Proc. ACM Program. Lang.* 9, OOPSLA2 (2025), 3560–3587. doi:10.1145/3763178

Yunzhi Zhang, Zizhang Li, Matt Zhou, Shangzhe Wu, and Jiajun Wu. 2025b. The Scene Language: Representing Scenes with Programs, Words, and Embeddings. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*. Computer Vision Foundation / IEEE, 24625–24634. doi:10.1109/CVPR52734.2025.02293

Received 2026-03-17; accepted 2026-08-06