

Politecnico di Torino

Politecnico di Torino

Master's Degree in Data Science and Engineering

A.a. 2025/2026

Graduation Session September 2026

RAG-Enhanced Webots World Generation with LLMs

Supervisors:

Prof. Alessio Burello
Prof. Daniele Jahier Pagliari
PhD. student Luca Benfenati

Candidate:

Yasaman Golshan

Abstract

Large Language Models (LLMs) can generate code and structured artifacts from natural-language specifications, suggesting a possible interface for automatically constructing robotic simulation environments. However, simulator-world generation requires more than syntactically valid output: generated worlds must execute correctly in the target simulator while also satisfying the requested objects, attributes, and spatial relations.

This thesis investigates the generation of Webots world files (`.wbt`) from natural-language scene descriptions. Two generation architectures are compared: a direct pipeline that generates the Webots world in a single LLM call, and a structured pipeline that first produces a JSON scene specification and subsequently translates it into WBT. Both architectures are combined with Webots-specific knowledge retrieval, structural prompting, few-shot demonstrations, and simulator-grounded validation. A two-level repair mechanism further separates deterministic correction of known Webots-specific mechanical failures from feedback-driven LLM repair of semantic and spatial errors. General layout remains LLM-generated, while deterministic repair applies only bounded corrections to existing object poses for some limited relations.

The approach is evaluated on a 90-scene benchmark organized into four difficulty tiers and across five LLMs. Correctness is assessed through static parsing, execution in Webots, and semantic and geometric inspection of the generated scene, using Load Success Rate, Object Correctness, Attribute Correctness, Spatial Correctness Rate, and a strict end-to-end success criterion. On the 30-scene held-out T4 benchmark, the complete direct pipeline increases strict end-to-end success from 0/30 under zero-shot generation to 17/30 with Gemini-2.5-Flash and 22/30 with Gemini-2.5-Pro. With Gemma-3 27B, direct generation matches or exceeds the structured pipeline on three of four difficulty tiers while requiring 1.3–2.9× lower generation cost.

The results show that introducing a JSON intermediate representation does not provide a consistent accuracy advantage and that successful simulator execution alone is not sufficient to establish scene correctness. A diagnostic experiment replacing generated JSON with a hand-written ground-truth specification shows that substantial errors remain during JSON-to-WBT generation itself: on the T4 development set, the resulting worlds achieve only 21.5% spatial correctness and 0/20 strict end-to-end successes without repair. Overall, the results identify spatial realization as the main remaining bottleneck once simulator-specific syntactic and mechanical failures are addressed.

Acknowledgements

I would like to express my sincere gratitude to everyone who supported me during my master's degree and the development of this thesis.

First and foremost, I would like to thank my supervisors, Prof. Alessio Burello and Prof. Daniele Jahier Pagliari, for their guidance, valuable feedback, and support throughout this research. Their expertise and encouragement were essential to the development of this work.

I am also deeply grateful to my co-supervisor, Luca Benfenati, for his availability, patience, constructive advice, and practical assistance during the research and experimental work.

I would also like to thank the professors and colleagues at Politecnico di Torino who contributed to my academic and personal growth during these years.

Finally, I am deeply grateful to my family and friends for their constant love, encouragement, and support. Their patience and belief in me made this journey possible.

Table of Contents

List of Figures VI

1 Introduction 1

2 Background 4
2.1 LLMs and Structured Generation 4
2.2 Webots World Representation 6
2.3 Spatial Representation in Simulation Worlds 7
2.4 Simulator-Grounded Validation 8

3 Related Work 10
3.1 LLMs for Programs and Structured Artefacts 10
3.2 Language-Guided 3D Scene Generation 11
3.3 Scenario Languages and Robotics Simulation 12
3.4 LLM-Based Design Automation 13

4 Methodology 17
4.1 Overall Framework 17
4.2 Prompting Strategies and Models Evaluated 18
4.3 Node and Field Catalogue 20
4.4 Scene Generation Pipelines 21
4.4.1 Direct Pipeline 21
4.4.2 Compiled Pipeline 21
4.4.3 Compiler JSON Generation 22
4.5 Validation and Evaluation Framework 23
4.5.1 Object Matching 23
4.5.2 Isolating Compiler and Generation Error 24
4.5.3 Evaluator Reliability 24
4.6 Repair Mechanisms 25
4.6.1 Deterministic Repair 25
4.6.2 Feedback-Driven LLM Repair 27

5 Experimental Setup 29
5.1 Dataset Design . . . 29
5.2 Evaluation Metrics . . . 31

6 Results 40
6.1 Performance Comparison . . . 41
6.1.1 Gemma-27B on the main T1–T3 and held-out T4 sets . . . 41
6.1.2 Cross-model generality on T4 . . . 41
6.1.3 Strategy component ablation on the T4 development set . . . 43
6.2 Spatial Correctness and Pipeline Comparison . . . 44
6.3 Error Analysis . . . 46
6.4 Impact of Repair and Validation . . . 47

7 Discussion 49
7.1 Main Findings . . . 49
7.1.1 Generation guidance and differences between models . . . 49
7.1.2 The value of a JSON intermediate representation . . . 50
7.1.3 Structural rules, schema retrieval, and few-shot examples . . . 51
7.1.4 What automated repair contributes . . . 51
7.1.5 Separating the different forms of correctness . . . 52
7.2 Limitations . . . 53

8 Conclusions and Future Work 56
8.1 Conclusions . . . 56
8.2 Future Work . . . 57

Bibliography 59

List of Figures

4.1 Overview of the evaluated system. The generated world passes through syntax and load checks before object, attribute, relation, and collision evaluation. Solid arrows show the main flow; the dashed arrow shows the gold pipeline's diagnostic bypass of the compiler. . 18
4.2 Construction pipeline for the node and field catalogue used throughout this thesis. Base-node and PROTO definitions are crawled from the official Webots repository, parsed with a VRML97-aware regular-expression parser, and filtered to retain only genuine `SF`/`MF` field declarations, yielding a single 398-entry reference catalogue. 20
4.3 The combined repair loop. Deterministic repair is applied immediately after initial generation and again after every LLM repair attempt, ensuring catalogue-driven and bounded geometric corrections are applied before the next `e2e_success` check. Up to three LLM repair attempts are performed per sample before the loop terminates. If the budget is exhausted without a pass, the current world is returned as an unsuccessful output. 25

5.1 Schematic illustration of representative spatial constraints from Table 5.3, grouped by category: (a) cardinal and compound cardinal directions; (b) support and vertical relations; (c) containment and proximity relations; and (d) the composite `between` relation. Full definitions, including `facing`, are given in the table above. 37

6.1 `e2e_success` as a percentage of tier size; darker fill indicates more successful samples. (a) Gemma-27B: historical T1–T3 results under the earlier criterion without collision checking, and held-out T4 under the current criterion. These cells are not a controlled cross-tier comparison. (b) S4 with repair on held-out T4. Full twenty-scene-per-tier T1–T3 results are shown only for Gemma-27B; smaller exploratory runs for other models are outside this figure. 43

Chapter 1

Introduction

Simulation gives robotics researchers a practical way to develop and test ideas without beginning every experiment on physical hardware. It is particularly useful for repeatable work on navigation, manipulation, control, and robot learning, although results obtained in simulation must still be validated on the intended real system [1]. Webots is a general-purpose robot simulator that supports the modelling of robots, sensors, actuators, and their environments [2]. A Webots environment is described in a `.wbt` world file. The format is based on VRML97 and extends it with Webots-specific nodes and fields [3].

Writing a world file by hand can be laborious, even for a modest scene. The user must choose valid node types, supply correctly typed fields, define geometry and appearance, and place each object relative to the others. An LLM could make this process more accessible: rather than editing WBT directly, a user could write "a red box on top of a green cylinder" and ask the model to construct the scene.

The difficulty is that a plausible-looking answer is not necessarily a usable world. The generated text must follow the WBT/VRML97 grammar, refer only to nodes and fields supported by the target Webots installation, resolve external `PROTO` definitions, and preserve the meaning of the user's description. A file can be syntactically valid and still contain the wrong objects, colours, sizes, or positions. It can also load successfully while failing to place the red box on the cylinder.

This combination of formal and spatial requirements makes the task suitable for simulator-grounded evaluation. The HumanEval benchmark evaluates generated programs by running them against functional tests, so correctness depends on what the program does rather than on how convincing its source code appears [4]. A Webots world can be assessed in a similar spirit. The file can first be parsed and loaded; the resolved scene graph can then be inspected through the Supervisor API. These checks reveal different kinds of failure, including invalid syntax, unsupported fields, missing or hallucinated objects, incorrect attributes, unsatisfied spatial relations, and collisions. Crucially, successful loading is only one

stage of correctness, not the final criterion.

Existing scene-generation research also shows that there is no single way to turn language into a spatial layout. LayoutGPT treats the LLM as a layout planner and asks it to generate numerical spatial parameters for 2D and 3D layouts [5]. Holodeck takes a different route: the LLM proposes spatial constraints, and a dedicated search or optimisation procedure places the objects while enforcing boundary and collision conditions [6]. SceneCraft builds relational scene graphs, translates their relations into numerical constraints in Blender code, and refines rendered scenes using visual feedback [7]. SceneGenAgent uses a structured, calculable layout together with verification and iterative refinement for industrial scenes [8]. SceneWeaver and SAGE broaden this idea into agentic pipelines that coordinate several generation and critique tools [9, 10].

Work in robotic simulation has explored a related but distinct problem. GenSim uses LLMs to generate executable task code and expert demonstrations for tabletop-manipulation simulation, with the resulting tasks intended to increase the diversity of data available for policy training [11]. The focus there is task generation and successful demonstration production. The focus of this thesis is the correctness of a declarative scene itself: whether a natural-language description is faithfully realised as a native Webots world.

The distinction that matters most is how spatial realisation is handled. In this thesis, the LLM generates the initial object poses and remains responsible for general layout reasoning. The evaluator checks whether the resulting scene satisfies the requested constraints, but it does not compute a replacement layout. A separate deterministic repair stage corrects mechanical defects, such as recoverable syntax, duplicate declarations, and invalid catalogue references. It also applies bounded geometric corrections for facing, support placement, and collision penetration. These routines can modify existing object poses, but their restricted scope does not make them a general-purpose layout solver. Remaining structural, semantic, spatial, and collision failures are reported to an LLM, which must revise the candidate world. The research gap is therefore broader than the absence of a Webots-specific generator: it concerns how reliably an LLM can construct an executable simulator artefact when model-driven layout generation is assessed through independent spatial validation and supplemented by a restricted set of deterministic corrections.

The thesis examines this problem from several angles. It compares direct text-to-WBT generation with a two-stage pipeline that uses JSON as an intermediate representation. It studies the effects of structural rules, schema retrieval, and few-shot examples. It then evaluates how much deterministic repair and feedback-driven LLM repair recover from different failure types. Throughout the study, parsing, loading, object content, attributes, spatial relations, hallucinations, collisions, and generation cost are measured separately so that improvements in one area do not conceal failures in another.

This thesis makes the following contributions:

1. **A dual-pipeline generation framework with complementary repair mechanisms.** The framework supports both direct text-to-WBT generation and a two-stage, compiler-inspired pipeline. In the latter, one LLM call produces a JSON intermediate representation and a second LLM call translates it into WBT. "Compiler-inspired" refers only to this use of an intermediate representation: the JSON-to-WBT stage remains model-generated rather than a deterministic translation, and therefore provides no compiler-like correctness guarantee. The framework evaluates structural prompting rules, retrieval from a catalogue of Webots nodes and fields, and few-shot demonstrations. Deterministic repair addresses mechanical defects and bounded geometric failures involving facing, support placement, and collision penetration, while simulator-feedback-driven LLM repair handles remaining structural, semantic, spatial, and collision failures. A gold-intermediate diagnostic helps separate errors introduced in the JSON stage from those introduced during JSON-to-WBT generation.

2. **A simulator-grounded, multi-dimensional evaluation framework and benchmark.** The evaluator measures parsing, Webots loading, object precision and recall, attribute correctness, relation correctness, object hallucination, invalid fields, and collision freedom separately. A strict `e2e_success` criterion combines the required correctness checks; field-name validity remains a separate diagnostic rather than a success gate. The core corpus contains 110 scenes: 80 main scenes across tiers T1–T4, with 20 scenes per tier, and 30 additional held-out T4 scenes constructed from furniture `PROTO`s. The Results chapter reports 60 historical main-set scenes from T1–T3 and 30 held-out T4 scenes. The historical runs use an earlier success definition without collision checking, so these 90 scenes are not pooled into one score under a common criterion. The 20 main T4 scenes are used for development and diagnostics rather than final held-out evaluation. A historical ten-scene T4 strategy-ablation version is also distinguished from this current corpus. Additional diagnostic suites are kept separate.

Chapter 2

Background

The task studied in this thesis sits between language generation and robot simulation. On one side, an LLM must produce a structured file rather than an open-ended passage of text. On the other, that file describes objects whose geometry and position acquire meaning only after Webots loads the scene. This chapter introduces the main ideas needed on both sides of that process. It covers structured generation with LLMs, the WBT/VRML97 scene representation, the geometric interpretation of spatial relations, and the different levels at which a generated world can be validated.

2.1 LLMs and Structured Generation

An autoregressive language model produces text one token at a time. Each new token is chosen from a probability distribution conditioned on the prompt and on the tokens already generated. This process can produce fluent and convincing text, but fluency is not enough when the output must be read by another program. A single missing bracket, an unsupported name, or a value of the wrong type may be enough to make the whole artefact unusable. Conversely, an output can be perfectly well formed and still describe the wrong scene.

The prompt is the main way of guiding the model towards the required structure. In a zero-shot setting, the model receives instructions and a description of the expected output, but no worked example. Few-shot prompting adds examples of the mapping that the model is expected to learn. For a specialised format, these examples can clarify conventions that are difficult to express as rules alone. They are not free, however: instructions, documentation, examples, repair messages, and the desired output all share the model's finite context window. A longer prompt may provide more information, but it can also make the relevant guidance harder to identify.

Retrieval-augmented generation (RAG) addresses a related problem. Instead of expecting the model to remember every detail of a specialised language, a retrieval system selects relevant external material and places it in the prompt at inference time [12]. In a structured-generation task, that material might include a schema, a node definition, a field signature, or a similar code example [13]. Retrieval gives the model access to useful information, but it does not force the model to use that information correctly. The generated result must still be checked independently.

Another option is to restrict what the model is allowed to generate. Constrained decoding rules out token sequences that would violate a grammar, regular expression, or schema. PICARD, Outlines, and XGrammar are examples of this broad approach [14, 15, 16]. These methods are useful for enforcing formal properties, such as valid JSON with a known set of keys. Their guarantee stops at the chosen constraint: a valid JSON object may still contain the wrong values, refer to an unsuitable Webots node, or encode a layout that contradicts the user's description. In this thesis, that gap between formal validity and semantic correctness is especially important because JSON is evaluated as an intermediate representation rather than as the final output.

When the first candidate is incorrect, feedback can be used to guide another attempt. A parser, program, or simulator can return information about the failure, and the model can use that information when generating a revision. Research on self-debugging and feedback-driven agents has shown how such diagnostics can support iterative improvement [17, 18]. Not every error needs another model call. If the correction is mechanical and unambiguous, a deterministic rule is usually more predictable. Deterministic correction can also be appropriate for bounded geometric failures when a local correction can be determined from the requested constraint and measured geometry. In this thesis, such corrections address facing orientation, support placement, and collision penetration. Their scope is restricted: they can adjust existing object poses, but they do not solve an arbitrary scene layout. Errors that require interpreting the scene description or reconsidering the broader arrangement of objects remain the responsibility of the model.

The probabilistic nature of generation also affects how results should be read. Decoding parameters such as temperature determine how narrowly the next token is selected from the model's probability distribution. With non-deterministic decoding, the same prompt may produce different files and different errors across runs. A single run therefore records one outcome of the generation process; it does not measure the full range of possible outcomes. This variation becomes more noticeable in long structured outputs, where an early change can alter much of the sequence that follows.

2.2 Webots World Representation

Webots is a robot simulation platform for modelling robots, sensors, actuators, environments, and physical interactions [2]. A simulation world is stored in a `.wbt` file derived from VRML97. Webots supports a subset of the standard VRML97 nodes and adds its own nodes for robots, devices, joints, physics, and other simulator features [3]. A WBT file is therefore neither unrestricted VRML97 nor an ordinary program. It is a typed, declarative description of a scene that Webots must turn into simulator objects.

The file follows the same hierarchy that appears in the Webots *scene tree*. A node may represent a transform, a visible shape, a geometric primitive, a physical object, a light, or a robot. Each node has a defined set of fields, and some of those fields contain further nodes. Position in the hierarchy matters. A valid geometry node, for example, still cannot be inserted into a field that does not accept geometry. This is one reason why balanced brackets alone are not enough to make a world valid.

Field types provide another layer of structure. Types beginning with `SF` hold a single value, whereas those beginning with `MF` hold multiple values. Common examples are `SFBool`, `SFFloat`, `SFString`, `SFVec3f`, `SFRotation`, and `SFNode`, along with list types such as `MFString`, `MFVec3f`, and `MFNode`. A translation is normally written as an `SFVec3f`; a rotation uses the axis-angle values of an `SFRotation`; and a collection of child nodes is commonly stored in an `MFNode` field. A generated file may therefore use a genuine field name and still be invalid because its value has the wrong type. The model may also invent a plausible-sounding field that the node does not define at all.

Object pose is expressed through transformations. A `Transform` introduces a local coordinate frame for its children using fields such as `translation`, `rotation`, and `scale`. Since each child is positioned relative to its parent, its final world pose depends on every transform above it in the scene tree. The distinction between `Transform` and `Solid` is also important. A `Transform` can group and display geometry, while a `Solid` represents a physical scene entity and adds fields used for identification, collision geometry, and physics. A generated world can consequently look reasonable while using a structural wrapper where a physical object was required. Such a scene may load, but the intended object may not be available to the evaluator in the expected form.

Not every object has to be defined directly in the world file. Webots uses `PROTO`s as reusable, typed node definitions. An `EXTERNPROTO` declaration makes an external definition available to the current world, which is how catalogue objects such as furniture and robots can be instantiated without copying their full implementations into the WBT file. Webots must locate the definition, match the supplied fields to its interface, and instantiate the nodes inside it. A line can therefore be valid

VRML97 text but still fail when the external definition is missing or its interface is used incorrectly. The `DEF` and `USE` keywords provide a related form of reuse within the scene: `DEF` names a node, and `USE` refers back to that definition within its permitted scope.

Webots also distinguishes between what an object looks like and how it interacts physically. Visible geometry is usually placed under a `Shape`, whereas the `boundingObject` of a `Solid` supplies geometry for collision handling. Dynamic behaviour additionally depends on the relevant physics fields. It is therefore possible to render an object that has no suitable collision volume, or to load an object that would not behave as intended once dynamics are enabled. The present study concentrates on the loaded scene structure, resolved geometry, and static spatial arrangement. Successful loading is not taken as proof that all possible dynamic behaviour has been validated.

Section 4.3 describes how the node and field catalogue used in this thesis was built. The catalogue provides the node and field definitions used to guide generation and check compatibility with Webots. Because its source branch was not pinned to an R2025a tag, the catalogue is not itself a guarantee of exact release compatibility.

2.3 Spatial Representation in Simulation Worlds

The worlds in this thesis use an east-north-up (ENU) coordinate frame. Positive X points east, positive Y points north, and positive Z points upward. At first glance, this seems to make a spatial relation easy to express: an object east of another should simply have a larger X coordinate. Coordinates are only part of the answer, however. An object's translation gives the origin of its local frame, not the full region occupied by its geometry. Its orientation and all parent transformations must also be taken into account.

The size and internal structure of the object matter as well. A primitive may be described by a box size, a cylinder radius and height, or a sphere radius. A `PROTO` can be more complicated, containing several pieces of geometry with their own offsets and rotations. Once the complete object is expressed in world coordinates, useful geometric summaries include its centre, its world-space bounding box, and its horizontal footprint. The footprint is the projection of the object's relevant geometry onto the X–Y plane. Two objects can have apparently reasonable centre coordinates and still overlap, miss one another, or extend outside the expected region.

For this reason, a linguistic relation has to be translated into a geometric condition. If a chair must be east of a desk, the evaluator requires a positive X displacement exceeding a size-relative margin, not merely an arbitrarily small difference between their positions. This check does not itself require their footprints

to be disjoint; collision freedom is assessed separately. A relation such as "on" also depends on the objects' vertical surfaces and on overlap between their horizontal footprints. "Above" involves vertical separation and, in the evaluator used here, horizontal overlap. "Center inside" depends on the size of the containing object, while "between" places the subject relative to the line segment joining two reference objects. Nearness and collision freedom likewise depend on geometry; metric distance is checked between the objects' world positions.

It is useful to distinguish the requested relation from the coordinates that make it true. A natural-language description or a structured relation graph provides a *spatial specification*. Choosing object poses and dimensions that satisfy it is the problem of *spatial realisation*. Some scene-generation systems hand this problem to a constraint solver or a geometric planner; others rely on the LLM to produce the numerical layout. This thesis follows the latter approach for initial generation and general layout reasoning. The LLM generates the initial object poses, while geometry-aware evaluation checks whether the result meets the specification. A separate deterministic repair stage can subsequently adjust orientations or placements for a restricted set of facing, support, and collision failures. The evaluator itself does not compute and substitute a replacement layout, and the repair stage does not act as a general-purpose layout solver. Broader layout revisions remain model-driven. Section 5.2 gives the exact definitions and tolerances used for each supported relation.

2.4 Simulator-Grounded Validation

Checking a generated world is not a single pass-or-fail operation. A file may be valid at one level and fail at the next, and those failures reveal different things about the generation process.

The first level is *static syntax checking*. Here the question is whether the text follows the expected grammar and can be parsed into a structural representation. This catches problems such as unbalanced delimiters and malformed literals. It cannot confirm that Webots supports every referenced node and field, or that external resources can be resolved.

The next level is *simulator loading*. At this point Webots attempts to resolve and instantiate the world. It may reject a file that passed the standalone parser because a node type is unsupported, a field is incompatible with its parent, or an external definition is unavailable. A successful load is an important result, but it shows only that Webots accepted the scene. It says nothing yet about whether the scene matches the request.

Once the world has loaded, the resolved scene can be inspected. The Supervisor API gives programmatic access to nodes, fields, global poses, and other properties

of the instantiated world [3]. This matters for nested transformations and reusable PROTOs, whose final form may not be obvious from the source text alone. The loaded geometry can then be used to derive bounding boxes and footprints, evaluate spatial relations, and identify collisions.

The final level considered here is *semantic validation*. Properties observed in the simulator are compared with an independent description of the intended scene. The evaluator checks whether the requested objects are present, whether extra objects have appeared, whether attributes have the expected values, and whether the required spatial relations hold. This comparison explains why loading cannot serve as the only success measure. A world may be accepted by Webots and still be entirely wrong with respect to the original request. Conversely, a world with a small syntax error may contain much of the intended content but never reach the later checks.

These validation stages can also support repair. Instead of asking the model to judge its own output without evidence, an external checker can describe the observed failure and return that description to the model. A new generation can then focus on the object, field, or relation that failed. Clear separation between validation stages also makes it possible to use different repair methods for different errors: deterministic rules for mechanical defects and bounded geometric corrections, and LLM-based repair for remaining failures that require semantic or broader layout reasoning. The geometric routines operate on existing objects and have a restricted scope, covering facing, support placement, and collision penetration. Chapter 4 presents the validators, repair rules, feedback format, and stopping criteria used in the proposed framework.

Simulator-grounded validation is not the same as full behavioural testing. The framework checks whether a world parses, loads, and contains the intended static scene structure and geometry. Webots may need to start or step the simulation so that these properties can be inspected, but this does not amount to testing a robot controller over a complete trajectory or studying the long-term dynamics of the scene. Those forms of evaluation fall outside the scope of this thesis.

Chapter 3

Related Work

This chapter positions the thesis within four areas of research: LLM-based generation of programs and structured artefacts, language-guided 3D scene generation, scenario languages for robotic simulation, and LLM-based design automation. These areas overlap, but they make different choices about a central problem in this thesis: how a linguistic description is turned into concrete object geometry and numerical poses, and how the resulting artefact is checked.

3.1 LLMs for Programs and Structured Artefacts

Work on code generation provides the clearest precedent for treating an LLM output as an executable artefact rather than as ordinary text. HumanEval evaluates generated programs by running them against functional tests [4]. The same principle motivates simulator-grounded evaluation in this thesis: a generated world must be accepted by its target system and must satisfy objective checks after loading. The analogy is not complete, because the correctness of a scene also depends on geometry and spatial relations, but it establishes the value of testing what an artefact does rather than judging only how plausible its source appears.

Retrieval and constrained decoding address two different sources of failure. Retrieval-augmented generation supplies external information at inference time [12]; RepoCoder combines retrieval and generation iteratively for repository-level code completion [13]. In the present work, retrieval has a narrower, simulator-specific role: it supplies Webots node, field, and `PROTO` definitions that may not be reliably represented in the model's parameters. PICARD, Outlines, and XGrammar instead restrict generation so that the output follows a grammar or other formal structure [14, 15, 16].

JSONSchemaBench treats schema coverage, efficiency, and output quality as separate concerns [19]. This helps explain why formal acceptance alone is insufficient.

In this thesis, an accepted JSON representation must still be compared with the intended scene, and its translation into WBT must be evaluated separately.

Feedback-based methods are relevant once an initial artefact has failed. Reflexion uses linguistic feedback from previous attempts to guide subsequent decisions, while self-debugging asks a model to inspect generated code and its execution results before producing a revision [17, 18]. The repair loop in this thesis follows the same broad idea, but its feedback is derived from parsing, Webots loading, resolved scene inspection, and spatial evaluation. Deterministic rules address mechanical defects and a restricted set of geometric failures involving facing, support placement, and collision penetration. The LLM repair stage handles remaining failures that require semantic reasoning or broader changes to the scene layout.

3.2 Language-Guided 3D Scene Generation

Language-guided scene generation differs most clearly in the way it performs *spatial realisation*. LayoutGPT treats the LLM as a numerical layout planner for 2D and 3D scenes, using in-context examples to guide the generation of object locations and sizes [5]. DirectLayout also produces numerical 3D layouts, but decomposes the process into a bird's-eye-view plan, a lifting stage, and asset-aware iterative refinement [20]. Both place numerical reasoning close to the language model, although DirectLayout adds a more explicit staged refinement process.

Holodeck takes a different approach. The LLM proposes objects and qualitative spatial constraints, while a dedicated layout procedure searches for poses that satisfy those constraints. Soft relations guide the arrangement, and hard conditions prevent objects from leaving the room or colliding [6]. Spatial realisation is therefore shared between the LLM, which constructs the relation graph, and a solver, which assigns the numerical placements. Final spatial correctness cannot be attributed to the LLM alone.

SceneCraft similarly uses an intermediate relational representation. It constructs a scene graph and generates Blender Python that translates relations into numerical constraints. Relation-scoring functions guide layout optimisation, and vision-language review of rendered images supports iterative refinement. A reusable function library allows learned scene-design procedures to be retained [7]. SceneCraft therefore combines structured planning, numerical optimisation, executable design code, and visual feedback rather than relying on a single text-to-layout call.

Recent systems extend this idea through broader agentic toolchains. SceneWeaver uses a language-model planner to select among scene-synthesis tools and revises scenes through a reason–act–reflect loop informed by physical, visual, and semantic self-evaluation [9]. SAGE targets simulator-ready environments for

embodied tasks. Its agent coordinates scene initialisation, asset placement, and editing tools, while visual and Isaac Sim-based physics critics guide refinement [10]. In both, spatial realisation depends on the tools and critics selected by the agent rather than one fixed coordinate-generation procedure.

The approach in this thesis separates model-driven generation from explicit simulator-grounded evaluation. The LLM generates the initial WBT structure and object poses, either directly or through a JSON intermediate representation. The evaluator measures whether the resulting scene satisfies the requested constraints, but it does not calculate and substitute a replacement layout. A separate deterministic repair stage can modify existing orientations or placements through bounded facing, support, and collision corrections. Feedback then prompts the LLM to revise remaining failures. General layout reasoning remains model-driven, while final scene correctness reflects both generation and the applied repair stages. The architecture supports matched comparisons of generation routes, prompting components, and repair stages; their evaluated scope is defined in Chapter 5.

Related spatial-reasoning benchmarks ask whether a model can identify or infer relationships in an existing scene. A qualitative benchmark based on simulated room layouts examines spatial relations, while 3DSRBench evaluates visual reasoning about object height, orientation, location, and multi-object relationships [21, 22]. In these settings, the scene is provided and the model reasons about it. This thesis reverses that setting: the model must construct an executable scene whose objects and numerical poses realise the requested relationships. Spatial reasoning is assessed through the resulting simulator artefact rather than only a textual or symbolic answer.

3.3 Scenario Languages and Robotics Simulation

Scenario languages provide a more formal route from an abstract description to a simulator instance. Scenic is a probabilistic programming language for describing distributions over scenes. It offers domain-specific spatial syntax and supports hard and soft requirements when concrete scenes are sampled [23]. Here, the language and its sampler carry out part of the spatial realisation. The author specifies relations and distributions rather than assigning every final coordinate manually.

ChatScene connects scenario representation with an LLM. Given a description of a safety-critical driving situation, it decomposes the request into details such as vehicle locations and behaviours, retrieves relevant description–code pairs, and assembles Scenic code executed in CARLA [24]. Its generated scenarios are evaluated through their effect on autonomous-driving agents, including collision outcomes. The target is an interactive traffic scenario for safety testing, not a static scene whose object content and pairwise relations are compared with an

independent specification.

GenSim addresses robotic manipulation rather than driving. It uses LLMs to generate task code, simulation environments, and expert demonstrations, and checks whether generated programs execute successfully and produce demonstrations [11]. These checks are simulator-grounded, but answer a different question. GenSim expands the diversity of tasks available for policy training; success concerns the executability and usefulness of a manipulation task. The present work evaluates whether a declarative scene description has been realised faithfully in Webots.

Together, Scenic, ChatScene, and GenSim show the value of executable scenario representations and simulator feedback. They also clarify the scope of this thesis. A WBT world describes the simulator's scene graph directly. It is not a probabilistic scenario specification, a driving-test generator, or a manipulation task program. Its evaluation therefore focuses on the loaded objects, fields, geometry, relations, and collisions that constitute the world itself.

3.4 LLM-Based Design Automation

LLMs have also been used as interfaces to professional 3D design tools. 3D-GPT organises procedural modelling around task-dispatch, conceptualisation, and modelling agents. They select functions and infer parameters for a Blender-based procedural-generation system, turning natural-language instructions into editable code and rendered content [25]. The procedural function library supplies much of the geometric construction logic.

SceneGenAgent applies a related coding-agent approach to industrial simulation. It converts descriptions into C code for Tecnomatix and uses a structured, calculable layout representation, layout verification, and iterative refinement [8]. Its emphasis on precise measurements and coordinate differences is particularly relevant here. The main difference is architectural: SceneGenAgent adds an explicit layout-analysis process for precise industrial arrangements. In this thesis, the LLM generates initial poses and remains responsible for general layout reasoning, while simulator-grounded evaluation checks correctness and a separate deterministic stage applies bounded geometric corrections.

SceneCraft also belongs partly to this design-automation line because its output is executable Blender Python and its feedback loop uses rendered images [7]. Across these systems, code acts as an interface between language and a complex design application. It provides editability and access to tool APIs, but does not itself guarantee correct geometry. This thesis brings that general idea to Webots while making failure attribution a primary objective: syntax, simulator loading, object content, attributes, spatial relations, and collisions are reported separately.

Tables 3.1 and 3.2 summarise the closest systems. The comparison focuses on

architectural properties rather than treating the target application as the main difference. The first table shows how a request becomes a concrete scene. The second shows how the result is checked and whether a failed candidate can be revised.

Table 3.1: Generation and spatial-realisation strategies in closely related systems. "Native artefact" denotes the executable representation consumed by the target design or simulation environment.

System	Grounding source	Native artefact	Spatial realisation	Generation stack
Holodeck [6]	Prompt and Objaverse assets	Interactive 3D environment	LLM relation graph followed by constraint-based layout search	LLM pipeline, asset retrieval, and layout solver
SceneCraft [7]	Prompt and retrieved assets	Blender Python and rendered scene	Relation graph, scoring functions, and numerical optimisation	LLM agent, Blender, optimiser, and visual reviewer
SceneGenAgent [8]	Prompt and Tecnomatix API	C code and Tecnomatix scene	Calculable layout with explicit coordinate reasoning	Coding agent, structured layout, and industrial API
ChatScene [24]	Prompt and description–code pairs	Scenic code executed in CARLA	LLM decomposition and scenario-language instantiation	LLM agent, retrieval, Scenic, and CARLA
SceneWeaver [9]	Prompt and scene tools	Synthesised 3D scene	Planner selects layout and scene-generation tools	Reflective LLM agent and extensible tool suite
SAGE [10]	Task description and generator tools	Simulator-ready Isaac Sim scene	Agent coordinates placement and editing tools	Agent, generators, and visual and physics critics
This thesis	Prompt, Webots catalogue, and demonstrations	Native `.wbt` world	LLM poses with bounded geometric repair and simulator-grounded validation	Direct or JSON-mediated generation with deterministic and LLM repair

Table 3.2: Validation and refinement in closely related systems. The repair column concerns a separately described rule-based stage for mechanical defects or bounded geometric corrections. "Not reported separately" does not imply that a system's tools perform no deterministic geometric operations.

System	Validation or feedback	Candidate refinement	Separate bounded repair	Asset scope
Holodeck [6]	Constraint satisfaction, human evaluation, and navigation tests	Solver searches placements	Not reported separately	Objaverse retrieval
SceneCraft [7]	Constraint scores and visual review of renders	Visual-feedback loop and library learning	Not reported separately	Retrieved assets
SceneGenAgent [8]	Code execution and layout verification	Iterative agent refinement	Not reported separately	Industrial domain
ChatScene [24]	Safety and collision outcomes against driving agents	No candidate-level repair reported	Not reported separately	Driving domain
SceneWeaver [9]	Physical, visual, and semantic self-evaluation	Reason–act–reflect refinement	Not reported separately	Tool-dependent
SAGE [10]	Visual critic and Isaac Sim physics critic	Critic-guided refinement	Geometric checks integrated into tools	Generated assets
This thesis	Parser, Webots inspection, specification, and collision checks	Bounded correction and diagnostic LLM repair	Catalogue fixes plus facing, support, and collision repair	Webots catalogue

The comparison exposes the main research distinction. Several related systems improve spatial quality through a solver, a calculable layout representation, or an agent that selects specialised tools. This thesis does not claim novelty merely because its target is Webots. Its contribution is a comparative framework in which the model generates the WBT structure and initial placement and remains responsible for general layout reasoning. A separate deterministic stage corrects mechanical defects and bounded geometric failures, while simulator feedback guides LLM repair of the remaining errors. The evaluator makes syntactic, semantic, spatial, and collision failures visible across these stages. The bounded corrections are not a general-purpose layout solver, and final performance reflects the complete

generation-and-repair system.

Chapter 4

Methodology

4.1 Overall Framework

The system converts a natural-language description of a static three-dimensional scene into a `.wbt` world file for Webots R2025a. Each benchmark prompt is paired with a structured YAML specification that records the expected objects, attributes, and spatial relations. The evaluator uses this specification as ground truth, rather than inferring the intended scene from the prompt itself.

Valid syntax is only the first requirement for a correct world. A file may pass the VRML97 parser but still fail to load correctly in Webots. Malformed nodes can be discarded silently, while unresolved external PROTO declarations can consume most of the inspection timeout. A world that loads may still contain the wrong objects or give them incorrect colours, positions, or spatial relations. Objects may also overlap physically even when the overall scene looks plausible.

These aspects are therefore evaluated separately. The final `e2e_success` criterion requires the file to parse, load cleanly in Webots, contain the correct objects, satisfy all declared attributes and relations, avoid hallucinated object types, and pass collision checking. Field validity is reported as a separate diagnostic because it does not directly determine end-to-end success. Section 5.2 gives the complete metric definitions.

Two generation architectures are compared. In the *direct pipeline*, an initial language model call produces the complete `.wbt` file from the scene description. The *compiled pipeline* uses two sequential calls: the first converts the description into a structured scene representation, and the second turns that representation into a `.wbt` file. A third configuration, the *gold pipeline*, is used only for diagnostic purposes and is described in Section 4.5.

Figure 4.1 summarises the complete system: the two generation routes, the diagnostic gold configuration, and the shared validation, evaluation, and repair

workflow.

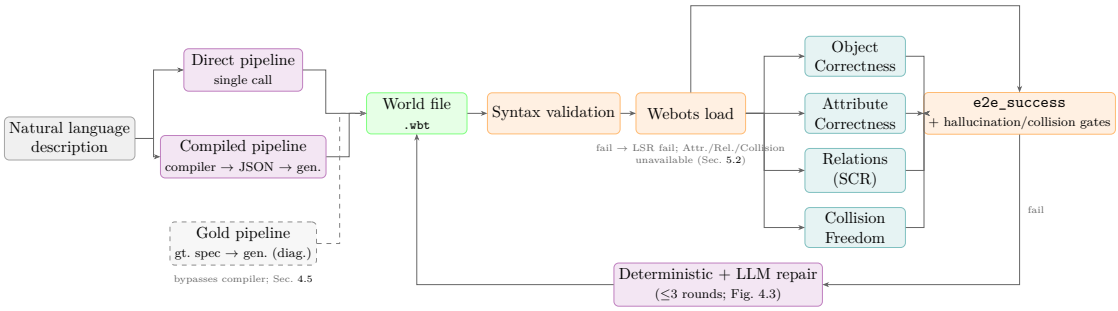


Figure 4.1: Overview of the evaluated system. The generated world passes through syntax and load checks before object, attribute, relation, and collision evaluation. Solid arrows show the main flow; the dashed arrow shows the gold pipeline's diagnostic bypass of the compiler.

Both pipelines share the same prompting infrastructure. The system message and base user-message template remain fixed; each prompting strategy changes only the guidance added to the user message. Keeping this framing constant helps isolate the component being evaluated, following the approach used in language-model experiments such as HELM [26]. It also reduces the risk that differences in prompt framing affect comparisons between strategies or models. Section 4.2 describes the five strategies used in this work.

After generation, outputs from either pipeline may pass through two repair stages before final evaluation. Section 4.6 describes these mechanisms.

4.2 Prompting Strategies and Models Evaluated

Both generation pipelines support five prompting strategies, S0 through S4. As described in Section 4.1, they differ in the guidance added to the user message, not in the system message or base template. The historical component ablation compares all five strategies on the development set without repair. The held-out comparison uses S0 without repair as the baseline and S4 with the full repair stack as the full-system condition, reported in Chapter 6.

- **S0** (zero-shot): provides the task description without additional guidance.
- **S1** (+ structural rules): adds a fixed set of structural and grammar rules: the required VRML header, a prohibition on inventing node types, correct SF/MF bracket syntax, and the ENU coordinate convention. Field-level schema information is deliberately withheld.

- **S2** (+ schema retrieval): adds, on top of S1, the relevant node schemas selected from the 398-entry catalogue.

- **S3** (+ few-shot): adds, on top of S1, three hand-written demonstrations of scene structure and spatial relations. The direct pipeline uses natural-language-to-`.wbt` examples; the compiled pipeline uses JSON-to-`.wbt` examples. These are fixed guidance examples, not held-out benchmark scenes. Their colours are not unique to the demonstrations, so colour choice alone does not establish separation from the dataset.

- **S4** (combined): combines the structural rules, retrieved schema, and few-shot examples before presenting the task.

The direct pipeline also includes a chain-of-thought control. Like the compiler's multi-step variant (Section 4.4.2), it is an exploratory control outside the headline evaluation. The reported results do not establish its effect on final performance.

Table 4.1 lists the models used in the thesis. All are accessed through OpenRouter, as described in Section 4.4.3. The historical S0–S4 ablation uses four of these models on a historical ten-scene T4 development version, with the direct pipeline and no repair. This version is distinct from the twenty-scene T4 development set used for the gold-pipeline diagnostic; it is not a subset of that set. Chapter 6 distinguishes the two protocols.

Table 4.1: Language models evaluated in this thesis, identified by the short label used to refer to each model throughout the remainder of this work.

Label	Model	OpenRouter identifier
Gemma-4B	Gemma 3, 4B parameters	`google/gemma-3-4b-it`
Gemma-12B	Gemma 3, 12B parameters	`google/gemma-3-12b-it`
Gemma-27B	Gemma 3, 27B parameters	`google/gemma-3-27b-it`
Gemini-Pro	Gemini 2.5 Pro	`google/gemini-2.5-pro`
Gemini-Flash	Gemini 2.5 Flash	`google/gemini-2.5-flash`

Gemma-27B is the main model in the results and discussion. The other models allow the findings to be examined across model families and sizes. The main T1–T4 datasets were used during development to identify and correct system issues. The final cross-model evaluation uses a separate 30-sample held-out dataset reserved for final evaluation. The held-out descriptions were not used for debugging or prompting-strategy selection. Gemini-Flash appears in this held-out comparison, but not in the historical S0–S4 strategy ablation.

Inference configuration. All calls use OpenRouter's OpenAI-compatible endpoint. Temperature is set to 0, and the maximum output length is 8,192 tokens for both compiler JSON and `.wbt` generation. Other sampling parameters, including `top_p`, are left at the provider defaults. Provider routing also uses OpenRouter's defaults: no provider order, allow-list, or fallback policy is specified. Reasoning output is excluded, and low reasoning effort is requested where supported by the routed provider. Within each run, the compiler, initial world generator, and feedback-driven repair calls use the same model and decoding settings.

4.3 Node and Field Catalogue

One shared catalogue of Webots node and PROTO definitions supports several parts of the system: schema retrieval (Section 4.2), invalid-field detection and repair, PROTO-URL correction, and the insertion of missing `EXTERNPROTO` declarations. This section describes how the catalogue was built and how it is used during generation.

The target simulator is Webots R2025a. The catalogue was built from the `released` branch of the official `cyberbotics/webots` repository. The crawler references this branch rather than a version-pinned R2025a tag. A crawl through the GitHub Tree API identified all base-node definitions (`.wrl` files in `resources/nodes/`) and PROTO object definitions (`.proto` files in `projects/objects/`). These files were downloaded concurrently, and a VRML97-aware regular-expression parser extracted their field declarations.

The initial greedy extraction also captured some English text from source comments. A second pass removed entries whose declared type was not a genuine VRML97 field type, retaining declarations with the `SF` and `MF` prefixes for single- and multi-value fields. Figure 4.2 summarises the process.

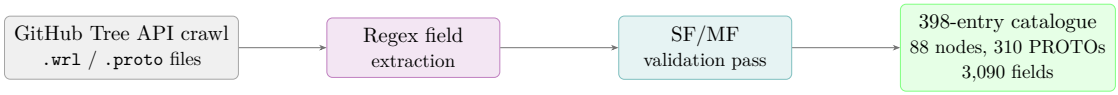


Figure 4.2: Construction pipeline for the node and field catalogue used throughout this thesis. Base-node and PROTO definitions are crawled from the official Webots repository, parsed with a VRML97-aware regular-expression parser, and filtered to retain only genuine `SF`/`MF` field declarations, yielding a single 398-entry reference catalogue.

After cleaning, the catalogue contains 398 entries: 88 base nodes and 310 PROTOs, with 3,090 fields in total. It is the shared reference for schema retrieval and for the node, field, and canonical PROTO-URL corrections used in deterministic repair.

During direct generation, retrieval tokenises the scene description and matches one- or two-word object names against the catalogue. Matching ignores case and uses simple singularisation for plurals. The compiled pipeline reads object types directly from the intermediate JSON instead. Both routes combine the selected object schemas with a fixed set of infrastructure, lighting, wrapper, and appearance nodes.

The retrieved entries are added to the user message as documentation-style blocks. Each block gives the canonical node name, permitted fields, their SF/MF types, and concise defaults or dimensional placeholders. The schema material appears after the structural rules and before the task description. Only the selected entries are included, rather than the full catalogue.

4.4 Scene Generation Pipelines

4.4.1 Direct Pipeline

The direct pipeline makes one initial language model call per sample, before any optional repair calls. The model receives the scene description with the guidance supplied by the selected prompting strategy and returns a complete `.wbt` world file. With no intermediate representation, the same call must interpret the description, choose a spatial layout, and express it in valid VRML97 syntax.

4.4.2 Compiled Pipeline

The compiled pipeline separates generation into two sequential stages. Each stage uses an independent language model call with its own conversation history.

The first stage, called the compiler, extracts the scene's objects, visual attributes, and spatial relationships into JSON. Object and attribute entries follow the ground-truth schema. The default compiler keeps the same outer relation-list structure but relaxes the relation-type discriminator, allowing ordinary relation phrases to be copied from the description without being normalised at this stage. Before diagnostic comparison, these free-text phrases are reconciled with the ground-truth relation vocabulary.

The second stage receives this JSON through a JSON-input variant of the selected generation strategy and translates it into the final `.wbt` world file.

Two compiler prompting variants are implemented. The *default* variant extracts objects, attributes, and spatial relationships directly into JSON. The implemented exploratory *multi-step (chain-of-thought) compiler variant* breaks this process into entity extraction, placement ordering, per-object spatial reasoning, and final JSON generation. It was motivated by chain-of-thought methods for spatial layout generation [20].

All headline compiled-pipeline results use the default compiler. The multi-step variant remains an exploratory control outside those comparisons. The reported evidence does not establish whether it improves extraction or final scene correctness, or whether repair reduces differences between the variants.

The compiler output balances two goals: staying close to the wording of the description and supplying enough structure for evaluation and world generation. The following choices define that balance:

- Colours retain the names used in the description, such as "red" or "green". The world-generation stage converts them to the RGB values required by the `.wbt` format.
- A floor is included in the compiler output only when it is mentioned in the description. Task-invariant Webots infrastructure is not added by the compiler; it is supplied by the downstream generation prompt.
- Quantified objects are represented as separate instances. For example, "three chairs" produces three individual specification entries rather than a single entry containing a count field.
- The compiler may emit `relative_to` and `offset` only by transcribing an exact, multi-axis numerical displacement stated in the description, for example "0.3 m east and 0.1 m north". It may not estimate coordinates for qualitative relations such as "near" or "east of"; that task belongs to the JSON-to-`.wbt` generator. The gold pipeline therefore removes extraction errors but still requires the generator to translate qualitative relations into coordinates.
- The default compiler preserves ordinary relation words as lower-case free text copied from the description rather than normalising them to the evaluator's closed vocabulary. `between`, `distance`, and `facing` retain their specialised field structures. Physical support is represented separately through the object-level `on_top_of` field, and a relation entry of type `on` is not permitted. Vocabulary normalisation is therefore part of the downstream generation stage.
- Architectural elements described as part of a room, such as walls, are assigned support on the floor by default even when the support phrase is implicit.
- Role or purpose phrases, such as "a crate standing in as a nightstand", remain descriptive language rather than becoming additional specification entries.

4.4.3 Compiler JSON Generation

All models are accessed through OpenRouter, which provides a common API for multiple language-model providers. The compiler generates free-form JSON, guided

by explicit prompt instructions describing the expected objects, attributes, and relation structure. The default compiler uses the permissive relation representation described in Section 4.4.2.

The generated JSON is parsed and checked locally against the compiler's representation requirements. Additional checks address colour completeness and catalogue-field compatibility. Outputs that fail these checks receive error feedback, with up to three additional compiler attempts. These retries are separate from the world-repair attempts described in Section 4.6.

Passing these checks establishes acceptance by the compiler, rather than complete correctness with respect to the requested scene. Extraction accuracy is assessed separately against the ground-truth specification, and the resulting world is evaluated after Webots loading.

4.5 Validation and Evaluation Framework

Each generated `.wbt` file is loaded into Webots, and the resulting scene graph is inspected through the Supervisor API. Static analysis alone cannot establish correctness: a syntactically valid file may fail to load, and a loaded world may still omit or misrepresent intended scene elements. Keeping syntax, loading, and semantic checks separate makes it easier to identify where generation failed.

4.5.1 Object Matching

Before object-level metrics are computed, objects resolved by Webots are matched to the ground-truth specification. A type-partitioned Hungarian algorithm finds a minimum-cost one-to-one assignment independently for each object type. Each candidate pairing has a name cost and a bounded attribute cost. The name cost is 0 for an exact identifier match, 10 when the two names contain the same normalised alphanumeric tokens, 50 when the generated object has a default or empty `Solid` name, and 100 otherwise. Attribute compatibility is averaged over the specification object's declared attributes and mapped to a cost in $[-10,0]$, keeping names as the dominant cue. Numerical and colour matching uses tolerances three times looser than those used for final scoring. Exact, inequality, range, and pattern constraints use their corresponding evaluators. If no generated object has the requested type, the specification object remains unmatched.

A relation-informed tie-break is used only for groups of two to five same-type objects whose Hungarian matches were all type-only: neither names nor attributes distinguished the candidates. The evaluator tests every permutation of the objects already assigned to the group, using the same relation predicates as the scorer. It selects the permutation satisfying the most relations involving that group; ties retain the original Hungarian assignment.

The purpose of this fallback is to avoid scoring an otherwise correct layout against an arbitrary assignment. In these ambiguous groups, spatial relations are the only specification information that distinguishes the objects. Using those relations to resolve their identities does not add a new requirement or privilege a particular naming scheme. If no relation distinguishes the objects either, retaining the original assignment does not systematically affect the score.

The evaluator records parsing, usable and clean Webots loading, attributes, spatial relations, object hallucination, field validity, and collision freedom separately. Webots inspection has a 90-second limit. If loading fails, attribute and relation checks that need simulator data are marked unavailable. Field validity is checked from the parsed WBT text because Webots may silently discard unsupported fields.

Facing is checked using the orientation reported by Webots. Collision checking examines the initial scene geometry, allowing normal support contact but not penetration beyond the configured tolerance. A scene can pass `e2e_success` only if collision checking is available and successful. Section 5.2 gives the definitions and tolerances.

4.5.2 Isolating Compiler and Generation Error

In the compiled pipeline, a failure may originate in text-to-JSON extraction or in JSON-to-`.wbt` generation. Two complementary diagnostics help separate these sources of error.

The gold pipeline tests the generator by replacing compiler output with the hand-authored ground-truth specification. It uses the compiled pipeline's JSON-input generation strategies but makes no compiler call. This is a diagnostic configuration, not a deployable architecture: it measures how the generator performs when compiler errors are removed.

The second diagnostic tests extraction directly. Compiler output is compared with the ground-truth specification, and discrepancies are classified as missing objects, hallucinated objects, incorrect object types, incorrect fields, or incorrect values. Together, these checks provide separate evidence about the compiler and generator, rather than attributing every compiled pipeline failure to one stage.

4.5.3 Evaluator Reliability

An evaluator error can be mistaken for a model failure. The framework was therefore audited before the final results were collected. The audit identified several inconsistencies in how scene representations were compared. The diagnostic comparison was updated to ignore case and formatting differences in relation strings, reconcile independently assigned object identifiers, and recognise equivalent inverse relations. These changes were in place before the final evaluation.

To guard against regressions, a calibration suite of 21 hand-crafted scenes was also added. Each scene has an expected outcome derived independently from the metric definitions, rather than from the evaluator's output. The suite allows these checks to be repeated when the evaluation code changes.

The compiled pipeline's few-shot demonstrations were checked with the live evaluator as well. One omitted the required `EXTERNPROTO` declaration for `Floor`; it was corrected before the compiled-pipeline results were collected. The equivalent audit of the direct-pipeline demonstrations had not been completed at the time of writing. This remains a limitation of the evaluation.

4.6 Repair Mechanisms

Both generation pipelines can use two repair mechanisms. Deterministic repair handles mechanical and syntactic defects, along with a bounded set of geometric corrections, without calling a language model. Feedback-driven LLM repair addresses the remaining semantic and spatial failures by regenerating the world. Figure 4.3 shows how the two mechanisms work together.

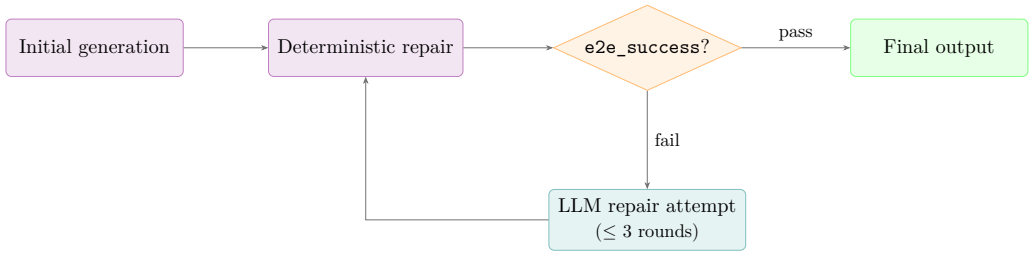


Figure 4.3: The combined repair loop. Deterministic repair is applied immediately after initial generation and again after every LLM repair attempt, ensuring catalogue-driven and bounded geometric corrections are applied before the next `e2e_success` check. Up to three LLM repair attempts are performed per sample before the loop terminates. If the budget is exhausted without a pass, the current world is returned as an unsuccessful output.

4.6.1 Deterministic Repair

Deterministic repair applies predefined corrections to the generated `.wbt` file without calling a language model. A single catalogue-driven pass uses valid Webots node types, field definitions, and canonical PROTO source URLs. Geometric routines address the specific failures listed below.

This stage runs immediately after generation and again after every LLM repair attempt. Reapplying it addresses mechanical defects and supported geometric

failures that remain or are introduced by a repair call before the next end-to-end evaluation.

The rules address failure modes observed during development. Table 4.2 summarises them.

Table 4.2: Deterministic repairs applied by the current system.

Category	Error mode	Correction
Catalogue-driven	Numeric Boolean in an `SFBool` field	Rewrite `0/1` as `FALSE/TRUE`, based on the catalogue field type
Catalogue-driven	Incorrect external PROTO URL or invalid trailing arguments	Replace them with the canonical catalogue entry
Catalogue-driven	Duplicate `EXTERNPROTO` declarations	Keep the first declaration and remove the duplicates
Catalogue-driven	Local `PROTO` block shadowing a catalogue entry	Remove the redundant local block
Catalogue-driven	`PROTO` reference with no field block	Append an empty field block
Catalogue-driven	Duplicate object block or object name	Remove the later duplicate
Catalogue-driven	Referenced `PROTO` not declared	Insert its canonical `EXTERNPROTO` declaration
Catalogue-driven	Field invalid for the node or PROTO type	Apply the corresponding catalogue-based field correction
Geometric	Incorrect facing orientation	Rotate the existing subject while preserving its translation
Geometric	Incorrect support or on-top placement	Correct the subject's vertical translation using the measured support offset; preserve horizontal placement
Geometric	Object penetration	Search a bounded set of alternative translations and retain only a collision-reducing candidate that preserves the protected evaluator checks

The 1,811-file audit recorded frequencies for only two rules. Duplicate `EXTERNPROTO` declarations appeared in 124 files (≈6.8%), and local `PROTO` shadowing occurred 16 times. Frequencies for the other rules were not tracked separately and are not reported.

Incorrect external PROTO URLs are particularly costly. Models frequently

generated invalid URLs and sometimes appended arguments that VRML97 does not permit. The deterministic rule replaces these declarations with canonical catalogue entries. Without this correction, Webots may spend most of the 90-second inspection interval trying to resolve a URL before reporting failure. In one run, this problem alone exhausted the repair budget for nine samples.

The geometric routines change orientation or placement, but their scope is deliberately limited. Facing repair handles horizontal constraints whose configured local forward axis is $+X$: it rotates an existing subject towards the measured target direction while preserving translation. Support repair changes only the subject's vertical translation using the evaluator's measured offset; it does not resolve an incorrect horizontal placement. Collision repair searches a bounded set of displacement candidates after a clean load. Each candidate is checked by the full evaluator and accepted only if collision severity decreases without a regression in the checked object, attribute, or relation constraints. These routines are not general spatial-layout solvers.

Deterministic repair improves loadability when mechanical defects are the cause of failure. It can also correct facing, support placement, and collision penetration, but it is not a general scene-layout solver. The held-out repair-stage comparison in Section 6.4 shows that these bounded corrections can produce end-to-end successes without an LLM repair call. The full repair loop increases the pass count in every reported configuration, showing that deterministic repair alone does not resolve all remaining failures. These later gains reflect LLM revision together with the deterministic corrections reapplied after each attempt, rather than LLM repair in isolation.

The two mechanisms therefore have different roles. Deterministic repair removes mechanical, syntactic, and narrowly defined geometric failures, leaving the LLM stage to address semantic errors and problems that need broader spatial reasoning.

4.6.2 Feedback-Driven LLM Repair

LLM repair is triggered when a scene fails `e2e_success`. The model receives the original task description, the current output, and a structured failure summary. This may include parser or Webots diagnostics, missing objects, incorrect attributes, failed spatial relations with numerical details, hallucinated object types, or collision pairs and penetration depths. The model is asked to return a corrected version of the complete scene.

Each attempt is an independent single-turn interaction. No conversation history is carried between rounds, so the model sees only the original task, the current world, and its current diagnostics. Each sample is allowed up to three repair attempts.

The summary identifies the violated constraint rather than simply reporting

that the scene is incorrect. For example, it may state that an object's Y-offset must be negative when its current value is +0.200. This gives the repair model a specific error to target.

Detailed diagnostics were adopted to make individual failures actionable. An earlier generic-failure formulation was explored during development, but it is not included as a separately quantified comparison in the reported results. Chapter 6 reports the performance of the evaluated repair configuration.

Detailed feedback also creates a risk: the model could satisfy a diagnostic without correcting the intended scene. It might, for example, add an object solely to satisfy a relation instead of moving the existing object. The repair prompt includes three constraints to discourage this behaviour.

First, repairs should correct existing objects by editing their fields or, when necessary, replacing an incorrect shape with an appropriate valid Webots primitive. For example, replacing a `Box` with a `Plane` is permitted when a flat panel has been represented with the wrong shape, even if `Plane` was absent from the original output. Second, the repair prompt instructs the model not to invent unsupported node types or fabricate nodes that merely declare a spatial relationship; relations must be realised through the objects' geometry and poses. It also prohibits introducing unrequested scene objects merely to satisfy a spatial constraint. This restriction does not preclude restoring a requested object missing from the generated world. Third, the model is instructed not to append a second instance of an object that already exists, but to modify that object's existing representation. This instruction complements deterministic duplicate removal. Together, these rules distinguish correcting a requested object's representation from adding fabricated or duplicate objects to satisfy a diagnostic. The prompt also asks the model to preserve already-correct objects, dimensions, attributes, and relations while resolving physical intersections. These are instructions to the model rather than guarantees of compliance; the resulting world is evaluated again after repair.

Repair feedback is derived from the requested constraints and the generated scene, rather than from hidden ground-truth absolute coordinates. It helps the model identify and correct the remaining failures. The final results therefore evaluate the complete generation-and-repair system.

It remains open whether deterministic repair should be extended to general spatial-layout constraints. Many failed relations can be expressed geometrically, so a broader optimiser could reposition several objects jointly. Such a mechanism might, however, solve the placement task from specification relations rather than correct a local defect. This raises a fairness concern about the role of repair in the evaluated system. Chapter 7 considers the scope and fairness of this extension.

This chapter has described the generation pipelines, evaluator, and repair mechanisms that make up the experimental system. The following chapters define the experimental conditions and report how the system performs under them.

Chapter 5

Experimental Setup

This chapter explains how the system introduced in Chapter 4 is evaluated. It describes the dataset (Section 5.1) and the metrics used to assess scene correctness (Section 5.2). The prompting strategies and language models are described in Section 4.2 as part of the methodology.

5.1 Dataset Design

The dataset has four difficulty tiers, each placing greater demands on spatial and compositional reasoning.

- **T1**: one or two objects, with zero or one spatial constraint, representing the simplest scenes.
- **T2**: two to four objects, with one explicit relation and, where specified, additional object-level support constraints.
- **T3**: three to five objects, combining multiple constraints, including quantitative requirements. Each sample has two to five spatial constraints when support is included.
- **T4**: the most demanding tier. Scenes use semantic furniture and household PROTOs alongside supporting geometry. Main-set scenes contain five to nine objects and seven to fourteen spatial constraints; held-out scenes contain six to eleven objects and nine to twenty-one spatial constraints. These constraint counts include object-level support as well as explicit relation-list entries.

The main development dataset contains twenty samples per tier: sixty across T1–T3 and twenty in T4. Final T4 performance is measured on a separate set of thirty held-out samples. The current corpus therefore contains fifty T4 prompts, of

which only the thirty held-out prompts contribute to the final T4 evaluation. The main samples were used to identify and correct implementation issues. The held-out descriptions were reserved for final evaluation and were not used for debugging or prompting-strategy selection. Table 5.1 summarises the two sets and their roles.

Table 5.1: Composition and role of the main and held-out datasets.

Set	**Samples**	**Role**
Main T1–T3	60	Twenty samples per tier; used during development and for the retained historical T1–T3 evaluation
Main T4	20	Used for T4 development and debugging, and for development-only diagnostics
T4 held-out	30	Reserved for final T4 evaluation; not used for debugging or prompting-strategy selection

T4 has a held-out set because it required additional PROTO-aware evaluation and repair development. Evaluating on separate prompts provides evidence beyond the examples used to develop the system. The historical S0–S4 strategy ablation uses a separate ten-scene T4 development version, the direct pipeline, and no repair. This version is not a subset of the twenty-scene T4 development set used for the gold-pipeline diagnostic. Its protocol is given with the strategy definitions in Section 4.2.

Historical and current evaluation criteria. The retained T1–T3 results were produced under an earlier evaluation version that did not include collision freedom in its end-to-end success criterion. The T4 held-out comparisons use the current collision-inclusive criterion defined below. Chapter 6 labels the historical results accordingly. They describe performance under their original protocol and should not be treated as directly comparable measurements under one unchanged criterion across all four tiers. Historical diagnostic and strategy-ablation results are also identified separately.

Held-out comparison protocol. The reported T4 held-out comparison uses S0 without repair as the baseline and S4 with the full deterministic and LLM repair stack as the full-system condition, for both the direct and compiled pipelines. Both conditions use the same thirty held-out specifications. This comparison measures the combined effect of additional generation guidance and repair; it does not isolate the effect of prompting under equal repair assistance. S0 with repair is not a reported condition. The reported baseline and full-system outcomes are presented

in Chapter 6.

5.2 Evaluation Metrics

The evaluation uses four correctness metrics, a collision check, and a strict end-to-end criterion (Table 5.2). Parsing errors, hallucinated objects, and invalid fields are recorded separately to help explain failures. Ratio-based metrics are microaveraged: the satisfied and total counts are summed across a tier before division,

$$M_{\text{tier}} \;=\; \frac{\sum_{i \in \text{tier}} n^{(i)}_{\text{satisfied}}}{\sum_{i \in \text{tier}} n^{(i)}_{\text{total}}}, \tag{5.1}$$

rather than averaging individual sample scores. Equation (5.1) applies to Attribute Correctness and Spatial Correctness. Load Success Rate is the fraction of evaluated scenes with a usable inspected world. Object Correctness uses the same approach, with precision and recall aggregated before F1 is calculated. This gives each object, attribute, or relation equal weight even when scenes contain different numbers of constraints. Averaging sample scores would instead give a simple scene the same weight as a much more complex one.

Syntactic validity. Each file is checked before Webots starts. It must be readable, contain valid UTF-8 with no unexpected control characters, and pass the VRML97 parser after all nodes have been processed. This is a binary check. On failure, the evaluator records the first condition that failed, giving a specific cause rather than a generic syntax error.

Load success and clean loading. The reported Load Success Rate counts scenes for which the Webots Supervisor returns a usable inspected scene. For N evaluated scenes,

$$\text{LSR} = \frac{\sum_{i=1}^{N} \mathbf{1}[\text{load.success}^{(i)}]}{N}. \tag{5.2}$$

Clean loading is recorded separately because Webots can skip a malformed or unrecognised node and still return the rest of the scene. Such a world counts as a usable load but not a clean load. The strict end-to-end criterion requires successful parsing, a usable load, and no hard loading error:

$$\text{parse.success} \;\wedge\; \text{load.success} \;\wedge\; \text{load.clean}. \tag{5.3}$$

Attribute and relation checks can still assess the surviving scene when loading is usable but not clean. Consequently, LSR must not be interpreted as a rate of completely error-free loading.

Table 5.2: The correctness metrics, collision check, and strict end-to-end criterion.

Metric	What it measures	Definition
LSR (Load Success Rate)	Whether Webots returns a usable inspected scene	Fraction of evaluated scenes with successful Supervisor inspection; clean loading is checked separately for end-to-end success
Object Correctness	Whether the correct object types are present	F1, computed after the type-partitioned Hungarian matching described in Section 4.5
Attribute Correctness	Whether declared colours, sizes, and similar attributes are correct	Fraction of attribute constraints satisfied, under criteria that differ by property type
SCR (Spatial Correctness Rate)	Whether the requested spatial relations hold	Fraction of the specification's stated spatial relations satisfied
Collision Freedom	Whether generated objects physically overlap	Collision checking must be available, with no penetration beyond the configured tolerance; normal support contact is allowed
`e2e_success`	Strict headline pass/fail criterion	Conjunction of parsing and load success, complete object, attribute, and relation correctness, zero hallucinated objects, and successful collision checking

Aggregation under load failure. Attribute and relation checks need world-space information from Webots. If a world does not load, these metrics are unavailable, and the sample is excluded from both the numerator and denominator. The scores are therefore conditional on loading and may appear high even when few scenes load. A stricter SCR would count relations from failed-load samples as unsatisfied. The authoritative result files do not store this aggregate, so the Results chapter reports conditional SCR alongside Load Success Rate rather than reconstructing it.

Object Correctness can still be calculated after a load failure, using the parsed syntax tree instead of the inspected Webots scene. Scores from these two sources are reported separately because they are not directly comparable.

Object Correctness. Object Correctness uses F1 rather than recall alone, so unnecessary copies of a requested object reduce the score. Let n_{spec} be the number of specified objects, n_{rel} the number of relevant generated objects, excluding structural wrappers, and n_{match} the number of correct matches found by the type-partitioned Hungarian assignment described in Section 4.5. Precision, recall, and F1 are then

$$P = \frac{n_{\text{match}}}{n_{\text{rel}}}, \qquad R = \frac{n_{\text{match}}}{n_{\text{spec}}}, \qquad \text{ObjF1} = \frac{2PR}{P+R}. \tag{5.4}$$

Tier-level F1 is calculated once, after summing the three counts. The reported source-specific aggregates omit samples whose specifications contain no requested objects. Such empty specifications are outside the scene-content comparisons reported here. If objects are requested but no generated type matches, both are set to zero. Unrequested types are recorded separately as object hallucinations, not included in this score.

Scope of attribute correctness. Attribute correctness covers colour, position, rotation, size, radius, height, transparency, and several geometry-specific properties. Common field names are mapped to canonical properties so that the same concept can be checked across different node types. Other declared field names are resolved case-insensitively against the raw fields reported by Webots, and their values are checked using the criteria below. This supports furniture-specific properties, such as texture or component depth, without a separate rule for every PROTO. Constraints may specify a single value, an inequality, a range, or a regular-expression pattern.

Matching criteria for attribute correctness. The comparison rule depends on the property. Colours pass when their Euclidean distance in normalised RGB space is at most 0.10. RGB is used because VRML97 stores colours as RGB triples. Position, size, radius, and height allow a per-component error of ten percent of the expected value. Position also has a five-centimetre absolute tolerance when an expected component is zero; non-zero components use the relative bound without that floor. Zero-valued size, radius, and height components use only a 10^{-6} numerical guard. Rotation values are compared component by component with an absolute tolerance of 0.05 in the stored axis–angle representation; this is not a comparison of the angular distance between two orientations. Other numerical properties, including transparency and PROTO-specific numerical fields, allow an absolute error of 0.01. Non-numerical values require equality. A per-constraint `tolerance` can override the default bound. Attribute Correctness is calculated as

$$\text{AttrCorrectness} = \frac{n_{\text{attr,sat}}}{n_{\text{attr,tot}}}. \tag{5.5}$$

If a sample declares no attribute constraints, this metric is undefined rather than zero.

Relation taxonomy. The evaluator supports fifteen explicit relation types in the ENU frame: $+X$ is east, $+Y$ is north, and $+Z$ is up. Support is written separately in the object-level `on_top_of` field and is converted internally into a support relation. A relation-list entry of type `on` is not accepted. Table 5.3 summarises the supported constraints, and Figure 5.1 illustrates the main groups. Matching between ambiguous objects of the same type follows the Hungarian tie-break procedure described in Section 4.5.

Exact relation thresholds. Let $h_k(o)$ be object o's half-extent on axis k. Let $\mathcal{O}$ contain every record exported by the inspector in `scene_objects`. For object o, let $p_k(o)$ be its world position and $[e_k^-(o), e_k^+(o)]$ its measured extent relative to that position, expressed along world axis k. The scene scale is

$$S = \max_{k\in\{x,y,z\}} \left[\max_{o\in\mathcal{O}} \big(p_k(o) + e_k^+(o)\big) - \min_{o\in\mathcal{O}} \big(p_k(o) + e_k^-(o)\big) \right]. \tag{5.6}$$

Thus S is the largest dimension of the world-axis-aligned bounding box containing these records. The set includes exported floors, furniture, primitives, and plain `Solid` wrappers, including unrequested objects; it is not restricted to objects matched to the specification. Infrastructure stored separately by the inspector, such as viewpoints, lights, and the injected Supervisor, does not contribute. An object without a usable geometric extent contributes its position with zero extent; an empty set uses $S = 1$ metre. Unless the specification overrides them, the relation checks in Table 5.3 use these thresholds:

- `on_top_of` requires the subject's lower surface to match the reference's upper surface within $\max(0.10h_z(r), 0.01\,\text{m})$, and at least 50% of the subject's horizontal footprint must overlap the reference footprint. A floor is treated as fully overlapping objects placed within it.
- `above` and `below` require the subject and reference centres to differ in the requested vertical direction by more than $0.10h_z(r)$, together with strictly positive horizontal-footprint overlap.
- Each simple cardinal relation requires the signed displacement on its active axis to exceed $0.50h_k(s)$. Compound cardinals apply this condition on both active axes and require the normalised axis displacements to differ by at most 0.50. For all horizontal cardinals, the centre-height difference may not exceed $h_z(r)$.
- `near` requires the shortest horizontal distance between the two axis-aligned footprints to be at most $0.10S$ and the centre-height difference to be at most $h_z(r)$. A configuration satisfying `on_top_of` is excluded.

Table 5.3: Spatial constraints recognised by the evaluator. The `on_top_of` support field is expanded internally and is not one of the fifteen explicit relation-list types.

Category	**Relation(s)**	**Definition**
Support	`on_top_of`	The subject rests on the reference's top surface, requiring substantial overlap between the two objects' footprints in the horizontal (X–Y) plane
Containment	`center_inside`	The subject's centre lies within thirty percent of the reference's half-size from the reference's own centre, evaluated independently per axis; axes along which the reference has zero extent are skipped
Simple cardinal	`north_of`, `south_of`, `east_of`, `west_of`	A single-axis comparison requiring the subject to lie beyond the reference, along the corresponding axis, by at least a minimum gap
Compound cardinal	`northeast_of`, `northwest_of`, `southeast_of`, `southwest_of`	The conjunction of the two corresponding simple cardinal conditions, both of which must hold simultaneously
Vertical	`above`, `below`	A comparison along the vertical axis that additionally requires horizontal overlap between the two objects' footprints, so that an object merely elevated but offset to the side does not satisfy the relation
Orientation	`facing`	The subject's configured local forward axis points towards a reference object or a world direction; the default horizontal-mode angular tolerance is 30°
Proximity	`near`	A qualitative relation, evaluated in the horizontal plane, satisfied when the two objects lie within a threshold distance of one another; explicitly excludes configurations satisfying `on_top_of`
Metric	`distance`	A quantitative relation requiring the measured distance between the two objects to match a specified value within a small tolerance, rather than merely satisfying a qualitative threshold
Composite	`between`	Requires two reference objects; the subject must lie close to the line segment connecting them, within a maximum allowed perpendicular deviation from that segment

- `distance` compares the Euclidean distance between object positions with the declared distance d. Its default error bound is $0.10d$; an explicit per-relation `epsilon` replaces this with an absolute bound in metres.

- `between` requires the subject to be no farther than $0.10S$ from the line through the two endpoints and to project onto the segment with longitudinal parameter $t \in [-0.05, 1.05]$.

- `center_inside` requires the subject centre to be within $0.30h_k(r)$ of the reference centre on every non-zero-extent axis. `facing` uses a default horizontal angular error of at most 30°.

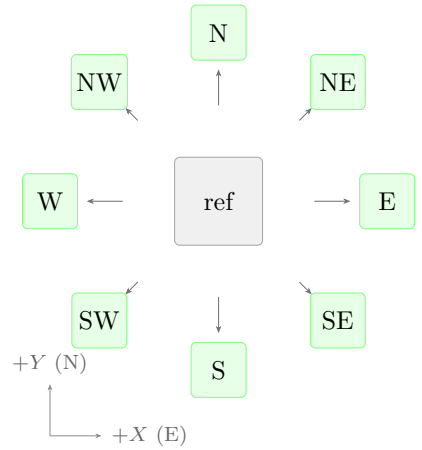
Dependence on generated scene scale. The $0.10S$ bounds for `near` and `between` depend on the output being evaluated. A larger generated floor, a more spread-out layout, or a distant extra object can increase S and relax these two checks even if the corresponding requested relation is unchanged. This is a limitation of the implemented normalization: the thresholds are not fixed independently of model output. Object Correctness, attribute checks, and the hallucination gate may penalize such changes separately, but they do not remove this potential bias from SCR. Reported SCR therefore describes correctness under these output-dependent thresholds.

Forward-axis convention and facing checks. The evaluator uses the fixed local forward vector (1,0,0), namely local $+X$, for both primitives and furniture PROTOs unless the ground-truth `facing` relation supplies a different `local_axis`. This is a coordinate convention, rather than a front inferred from geometry or appearance: even a rotationally symmetric primitive has a designated local axis for this check. There is no per-PROTO lookup mapping furniture types to their visual fronts. All fifty-seven facing constraints in the held-out specifications use the default axis. Consequently, interpreting these scores as visual-facing correctness assumes that each evaluated asset's intended front agrees with local $+X$; assets using another convention would require an explicit ground-truth override.

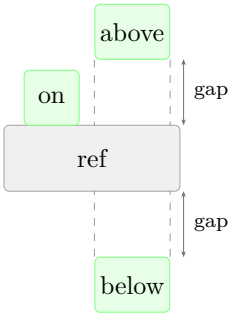
The inspector's world-orientation matrix transforms the local vector into world coordinates. In the default horizontal mode, its projection onto the X–Y plane is compared with the reference object's measured axis-aligned footprint. The minimum angular deviation towards that footprint must be at most 30°, and the direction towards the reference position must be within the forward half-plane (90°). A forward ray intersecting the footprint has zero footprint-angle error. If the footprint is unavailable, the evaluator compares directions towards object positions instead. An explicit `world_direction` is compared directly, and the optional `3d` mode compares three-dimensional directions. Missing or invalid orientation data,

or a direction whose required projection is zero, makes the check unavailable and unsatisfied; it is not omitted from SCR.

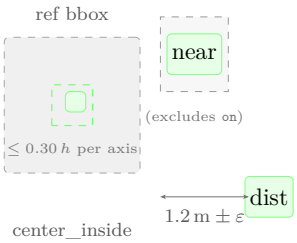
Exact relative displacements use `relative_to` and `offset` in the same ENU frame. For an expected offset vector $\mathbf{o}$, the Euclidean error between measured and expected offsets must not exceed $\max(0.10\|\mathbf{o}\|, 0.05\,\mathrm{m})$ unless the sample provides an explicit tolerance.



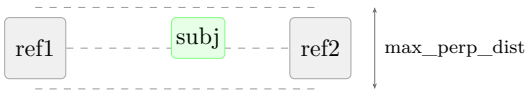
(a) Cardinal & compound cardinal



(b) Support & vertical



(c) Containment & proximity



(d) Composite: `between`

Figure 5.1: Schematic illustration of representative spatial constraints from Table 5.3, grouped by category: (a) cardinal and compound cardinal directions; (b) support and vertical relations; (c) containment and proximity relations; and (d) the composite `between` relation. Full definitions, including `facing`, are given in the table above.

Spatial Correctness Rate is the fraction of requested relations that are satisfied:

$$\mathrm{SCR} = \frac{n_{\mathrm{rel,sat}}}{n_{\mathrm{rel,tot}}}. \tag{5.7}$$

Support and facing constraints contribute to this total. Facing also has separate satisfied, total, and unavailable counts, allowing orientation failures to be distinguished

from placement errors.

Size-relative tolerances. Spatial tolerances scale with object size rather than using one fixed value for every scene. For `center_inside`, the allowed centre displacement is thirty percent of the reference's half-size on each non-zero axis. An axis with no extent, such as a floor's vertical axis, is skipped. Cardinal gaps similarly scale with the subject's half-size. These choices keep relations such as "centred" and "east of" consistent across small and large objects. Spatial thresholds are separate from the attribute tolerances above.

Collision freedom. Collision freedom is checked independently of the requested relations. Before physics advances, the inspector extracts each object's `boundingObject`, or its geometry children if no `boundingObject` is present. Furniture is represented by its component collision shapes, not by one enclosing box. Axis-aligned bounding boxes first rule out pairs that cannot collide. Remaining pairs undergo rotation-aware narrow-phase testing with Webots' bundled double-precision ODE library, supplemented where needed by separating-axis tests and mesh surface and containment checks.

The supported collision shapes are `Box`, `Sphere`, `Cylinder`, `Capsule`, `Cone`, finite `Plane`, and `IndexedFaceSet` geometry. A pair fails only if its measured penetration exceeds 0.01 m. Surface contact and penetration up to that tolerance pass; support and containment relations do not exempt objects from testing.

The check is unavailable if the ODE backend cannot be loaded or if any relevant object has incomplete, unreadable, malformed, or unsupported collision geometry. Unsupported branches include external meshes, an `ElevationGrid`, or a scaled `Transform` that cannot be resolved. End-to-end success requires complete collision-check coverage and no pair with penetration above the tolerance.

Object hallucination. Object hallucination counts generated objects whose types do not appear in the specification. Extra instances of a requested type are handled by Object Correctness instead. The diagnostic records hallucinated and total generated object counts, and end-to-end success requires zero hallucinations.

Object types are read from the inspected Webots scene when available, or from the parsed syntax tree after a load failure. Structural wrappers and infrastructure—including `Solid`, `Shape`, `Transform`, `WorldInfo`, and lighting nodes—are excluded. One limitation remains: Webots may silently discard a wholly unknown node type before the inspector can observe it.

Field-name validity. For each valid node type, the evaluator checks whether its generated field names are permitted. It uses the parsed WBT text because Webots

may silently discard unrecognised fields. A baseline dictionary supplies the valid names, with the node catalogue from Section 4.3 taking precedence wherever it has an entry. The score is

$$\text{FieldValidity} = 1 - \frac{n_{\text{invalid}}}{n_{\text{fields}}}, \tag{5.8}$$

The evaluator stores up to twenty-five invalid-field examples per sample. This cap limits diagnostic output only; every field contributes to the score. Field validity supports error analysis but does not affect `e2e_success`.

The end-to-end success criterion. A sample passes `e2e_success` only when every required check passes: valid syntax, successful and clean loading, all and only the requested object types, every declared attribute and relation, zero hallucinated objects, and an available collision check with no failure. A dimension without declared constraints is treated as satisfied because there is nothing to test. Field-name validity is diagnostic and is not included in this gate. If loading fails and a content-dependent metric cannot be computed, the sample cannot pass end to end.

Each evaluated scene receives a binary result, with no partial credit. The tier-level end-to-end success rate is the number of passing scenes divided by the number of evaluated scenes. For the initial stage, this gate is applied to the initially generated world; for the final stage, it is applied to the world returned after the permitted repair process. The final-stage rate therefore measures completion under the stated generation-and-repair protocol, which can involve multiple model calls.

Cost telemetry. For each sample, the system records token usage, model calls, and monetary cost for initial generation and all repair attempts. These are telemetry rather than correctness metrics, but they allow performance to be considered alongside computational cost. Results distinguish total run cost from mean cost per scene, calculated by dividing the total by the number of samples.

Chapter 6

Results

This chapter presents historical results for the sixty main T1–T3 samples and current results for the separate thirty-scene T4 held-out set described in Table 5.1. The main datasets were used during development to identify and correct implementation issues. The held-out descriptions were reserved for final evaluation and were not used for debugging or prompting-strategy selection.

Two versions of the T4 development data also appear in this chapter. The historical strategy ablation uses ten scenes, whereas the gold-pipeline diagnostic uses a twenty-scene version. They share five scenes with the same IDs and descriptions, but the ten-scene version is not a subset of the twenty-scene version. Both are separate from the thirty held-out scenes.

Gemma-27B has full twenty-scene-per-tier results on T1–T3. All five models in Table 4.1 appear in the T4 comparison. The reported full-scale cross-model comparison is therefore limited to that tier. Every reported compiled result uses the default compiler described in Section 4.4.2.

The retained T1–T3 results use an earlier evaluation version without a collision gate. Their scores are reported under their original criterion, not as verified passes under the current definition. Historical development diagnostics also retain their original protocols. The current held-out full-system condition uses S4 with deterministic repair after generation and after each of up to three LLM repair attempts (Figure 4.3). Its strict end-to-end criterion requires clean loading and collision freedom as well as scene-content correctness. The reported LSR counts usable inspected loads, which need not be clean.

Section 6.4 traces the paired held-out outputs through raw generation, deterministic repair, and the subsequent repair loop. Figure 4.1 shows the framework, including the diagnostic gold pipeline. Facing is included in SCR; collision freedom is an additional gate in the current held-out E2E criterion.

6.1 Performance Comparison

6.1.1 Gemma-27B on the main T1–T3 and held-out T4 sets

Table 6.1 compares the direct and compiled pipelines for Gemma-27B after the repair process used in each evaluation. T1–T3 use twenty historical main-set samples per tier, while T4 uses thirty held-out samples and the current protocol. Their E2E criteria differ, as explained above.

Table 6.1: Load Success Rate and `e2e_success` for Gemma-27B, direct versus compiled (default variant) pipeline after repair. T1–T3: historical main-set runs, N=20/tier, earlier E2E criterion without collision checking. T4: held-out set, N=30, current collision-inclusive criterion.

		Direct		Compiled (default)	
Dataset	**Tier**	**LSR**	**e2e_success**	**LSR**	**e2e_success**
Historical main	T1	20/20	18/20	20/20	19/20
Historical main	T2	20/20	19/20	20/20	16/20
Historical main	T3	20/20	17/20	20/20	16/20
Held-out	T4	30/30	10/30	29/30	10/30

After repair, both pipelines load every T1–T3 scene. End-to-end success shows a less consistent pattern: direct generation is higher on T2 and T3, while compiled generation is ahead by one scene on T1. The pipelines each pass ten scenes on held-out T4. These are single runs on small datasets, so small differences are not firm evidence of one pipeline's superiority. T4 also uses a different dataset split, evaluation criterion, and repair implementation. The lower held-out counts cannot therefore be attributed to tier difficulty alone.

Table 6.2 reports run costs for the main-set Gemma-27B comparisons and the held-out T4 full-system configurations.

The compiled route adds a compiler call. For Gemma-27B, its run cost is between 1.3 and 2.9 times that of direct generation. The additional cost does not bring a consistent gain in end-to-end success: compiled is one scene ahead on T1, behind on T2 and T3, and tied on held-out T4. On held-out T4, compilation also costs more for Gemma-12B and both Gemini models, while its end-to-end advantage is two scenes for Gemma-12B, one for Gemini-Pro, and zero for Gemini-Flash.

6.1.2 Cross-model generality on T4

Table 6.3 extends the held-out T4 comparison to all five models. It contrasts the S0 baseline without repair with S4 and the full repair stack.

Table 6.2: Run cost (USD), S4 with the full repair stack. Main T1–T3: N=20/tier; held-out T4: N=30. Ratio is compiled cost divided by direct cost. Monetary values are rounded to four decimal places. A dash indicates that a complete cost total is unavailable for the corresponding configuration.

Dataset	Model	Tier	Direct	Compiled	Ratio
Main	Gemma-27B	T1	$0.0084	$0.0245	2.9×
Main	Gemma-27B	T2	$0.0104	$0.0301	2.9×
Main	Gemma-27B	T3	$0.0149	$0.0310	2.1×
Held-out	Gemma-4B	T4	–	–	–
Held-out	Gemma-12B	T4	$0.0304	$0.0382	1.3×
Held-out	Gemma-27B	T4	$0.0553	$0.0717	1.3×
Held-out	Gemini-Pro	T4	$1.8334	$2.6283	1.4×
Held-out	Gemini-Flash	T4	$0.5924	$0.7810	1.3×

Table 6.3: `e2e_success` (out of 30) on the T4 held-out set. S0 uses no repair; S4 uses the full repair stack.

	S0 (baseline)		S4 + repair (full)	
Model	Direct	Compiled	Direct	Compiled
Gemma-4B	0/30	0/30	0/30	0/30
Gemma-12B	0/30	0/30	2/30	4/30
Gemma-27B	0/30	0/30	10/30	10/30
Gemini-Pro	0/30	0/30	22/30	23/30
Gemini-Flash	0/30	0/30	17/30	17/30

Every S0 entry in Table 6.3 is 0/30. On the held-out set, the baseline runs nevertheless differ in loadability: Gemini-Pro loads 17/30 direct and 24/30 compiled scenes, while Gemini-Flash loads 25/30 direct and 21/30 compiled scenes. These loads do not translate into end-to-end passes, illustrating the difference between a loadable world and a fully correct scene.

Among the S4 configurations with current load aggregates, LSR is 28/30 direct and 27/30 compiled for Gemma-12B, 30/30 and 29/30 for Gemma-27B, 28/30 and 30/30 for Gemini-Pro, and 30/30 and 29/30 for Gemini-Flash.

Gemini-Pro has the highest end-to-end success, reaching 22/30 with direct generation and 23/30 with compiled generation. Gemini-Flash reaches 17/30 with either pipeline. Gemma-27B also ties, at 10/30 for each pipeline, while Gemma-12B reaches 2/30 direct and 4/30 compiled. Compilation therefore provides small gains for two models and no gain for the other three in these runs. With one completed

run per configuration, the results do not establish a general preference for either pipeline.

Figure 6.1 shows the reported full-system end-to-end results as a visual summary; the tables provide the exact values.

	T1	T2	T3	T4
Direct	90.0%	95.0%	85.0%	33.3%
Compiled	95.0%	80.0%	80.0%	33.3%

(a) Gemma-27B: historical T1–T3; held-out T4

	Direct	Compiled
Gemma-4B	0.0%	0.0%
Gemma-12B	6.7%	13.3%
Gemma-27B	33.3%	33.3%
Gemini-Pro	73.3%	76.7%
Gemini-Flash	56.7%	56.7%

(b) Cross-model on T4

Figure 6.1: `e2e_success` as a percentage of tier size; darker fill indicates more successful samples. (a) Gemma-27B: historical T1–T3 results under the earlier criterion without collision checking, and held-out T4 under the current criterion. These cells are not a controlled cross-tier comparison. (b) S4 with repair on held-out T4. Full twenty-scene-per-tier T1–T3 results are shown only for Gemma-27B; smaller exploratory runs for other models are outside this figure.

6.1.3 Strategy component ablation on the T4 development set

The historical ablation described in Section 4.2 compares all five strategies, S0–S4, using ten T4 development samples and the four models available at that stage. Every condition uses the direct pipeline without repair, including S4. These results are development evidence, not held-out results, and are kept separate from the final T4 comparison of S4 with the full repair stack.

S1's structural rules improve loading for all three Gemma models, but their relation scores remain zero. Better loadability alone does not produce a correct layout. Adding schema guidance in S2 raises Gemma-27B to 8/10 successful loads and 30.1% SCR, but does not consistently help the two smaller models. The separate S3 few-shot condition loads all ten scenes for Gemma-4B and Gemma-12B and is the first condition in which either satisfies any relations. S2 and S3 are parallel extensions of S1; S3 does not add examples on top of S2. Only S4 combines all three components.

S4 gives Gemma-27B its highest historical SCR, 46/92 (50.0%), with all ten scenes loading. Combining the components does not consistently improve the

Table 6.4: Load Success Rate and conditional Spatial Correctness Rate for five prompting conditions on the T4 development set (N=10), direct pipeline, no repair. S2 and S3 are separate extensions of S1 rather than successive cumulative steps; S4 combines structural rules, schema retrieval, and few-shot examples. A dash indicates that no sample loaded, so no relation could be evaluated.

Model	S0	S1 (+rules)	S2 (+schema)	S3 (+few-shot)	S4 (combined)
Gemma-4B	0/10, –	1/10, 0.0%	2/10, 0.0%	10/10, 6.5%	7/10, 4.5%
Gemma-12B	0/10, –	3/10, 0.0%	0/10, –	10/10, 29.3%	2/10, 30.8%
Gemma-27B	0/10, –	5/10, 0.0%	8/10, 30.1%	9/10, 41.0%	10/10, 50.0%
Gemini-Pro	8/10, 6.0%	10/10, 26.1%	9/10, 19.8%	10/10, 33.7%	10/10, 37.0%

smaller models over S3. Gemma-4B falls to seven loads and 4.5% conditional SCR, while Gemma-12B loads just two scenes. Gemma-12B's 4/13 (30.8%) S4 SCR reflects only those two loads. It is therefore not evidence of an overall improvement over S3, which loads all ten scenes and satisfies 27/92 (29.3%) of the relations.

Gemini-Pro has a stronger S0 baseline, with 8/10 loads and 6.0% SCR. S2's relation score is lower than S1's, whereas S3 reaches 33.7%. S4 reaches 34/92 (37.0%) while retaining ten loads. The small dataset and the parallel design of S2 and S3 support broad component-level observations, not a claim of steady improvement across the columns. These historical S4 results contain no repair and should not be confused with S4 plus repair on the held-out set.

6.2 Spatial Correctness and Pipeline Comparison

Spatial Correctness is examined separately because a world may load and contain the requested objects but still place them incorrectly. Table 6.5 compares Gemma-27B's spatial performance across the four tiers.

Table 6.5: Conditional Spatial Correctness Rate for Gemma-27B after repair. T1–T3 are historical main-set results under an earlier evaluation and repair protocol; T4 is the current held-out evaluation. Failed-load samples are excluded because their relations cannot be evaluated. A separate strict SCR aggregate is not reported.

Tier	Direct	Compiled (default)
T1	8/10 (80.0%)	9/10 (90.0%)
T2	55/56 (98.2%)	52/56 (92.9%)
T3	81/83 (97.6%)	77/83 (92.8%)
T4	337/390 (86.4%)	323/377 (85.7%)

Direct generation has a higher conditional SCR on T2–T4; compiled generation is higher on T1. On held-out T4, direct satisfies 337/390 relations (86.4%), compared with 323/377 (85.7%) for compiled. However, one compiled scene fails to load, while every direct scene loads. The compiled denominator excludes that scene's thirteen requested relations: six explicit spatial relations and seven support relations expanded from the specification. This accounts for the difference between 390 and 377; it does not mean that the compiled pipeline was evaluated against fewer constraints per scene. SCR must therefore be read alongside LSR. No strict SCR is reported because the authoritative result files do not store it.

Table 6.6 separates facing and collision results. Both checks are part of the current evaluation and already affect the E2E values in Table 6.3.

Table 6.6: Facing and collision results on the held-out T4 set under S4 with the full repair stack. Facing "counted" totals include constraints retained in each run's relation metrics; failed-load scenes are excluded. The current held-out specifications contain 57 facing constraints in total. "Evaluable" counts available checks and "satisfied/pass" counts successes. All 30 scenes remain in the collision-check scope. Gemma-4B rows retain historical diagnostic aggregates and are supplementary rather than a paired current-stage comparison.

		Facing constraints			**Collision check**		
Model	**Pipeline**	**Counted**	**Evaluable**	**Satisfied**	**Scenes**	**Evaluable**	**Pass**
Gemma-4B	Direct	27	0	0	30	13	7
Gemma-4B	Compiled	25	9	8	30	14	1
Gemma-12B	Direct	53	45	42	30	27	5
Gemma-12B	Compiled	48	39	39	30	27	22
Gemma-27B	Direct	57	48	46	30	30	20
Gemma-27B	Compiled	56	47	47	30	29	25
Gemini-Pro	Direct	52	49	49	30	28	26
Gemini-Pro	Compiled	57	53	53	30	30	28
Gemini-Flash	Direct	57	52	52	30	30	22
Gemini-Flash	Compiled	55	51	51	30	29	23

The Gemini models satisfy every evaluable facing constraint. Gemini-Pro satisfies 49 direct and 53 compiled constraints, although three and four additional constraints, respectively, cannot be checked. Gemini-Flash satisfies all 52 evaluable direct constraints and all 51 evaluable compiled constraints, with five and four additional constraints unavailable. Gemma-27B satisfies 46 of 48 evaluable direct constraints and all 47 evaluable compiled constraints.

Collisions remain a separate source of failure. Gemini-Pro passes 26 of 28 evaluable direct scenes and 28 of 30 compiled scenes. Gemini-Flash passes 22 of 30 direct scenes and 23 of 29 evaluable compiled scenes. Gemma-27B passes 20 of 30

direct scenes and 25 of 29 evaluable compiled scenes, despite achieving the same E2E count with both pipelines.

Diagnostic: stage isolation on the T4 development set. The gold pipeline isolates the JSON-to-`.wbt` generator by supplying the hand-authored specification instead of compiler output. On the twenty T4 development samples under its historical protocol without repair, it reaches 15/20 usable loads, 15/25 correct evaluated attributes (60.0%), 31/144 satisfied evaluated relations (21.5%), and 0/20 end-to-end passes. Attribute and relation denominators exclude scenes without a usable load; they are not totals over all twenty specifications. Even with a correct intermediate specification, the generator still struggles to produce valid Webots output and correct spatial placement.

Relation-level scope. The authoritative aggregates contain overall relation totals and a separate facing breakdown, but no current pooled breakdown for every relation type. Earlier per-type percentages are omitted because they came from a different set of result artefacts.

6.3 Error Analysis

Gemma-4B's object, attribute, facing, collision, and field diagnostics below retain the historical full-system aggregates supplied for this model. They are supplementary evidence and are not used to reconstruct a current paired repair-stage comparison.

Object hallucination. Most models produce few hallucinated object types on the held-out set under S4 with repair. Gemma-12B's counts are 3/200 direct and 1/185 compiled; Gemma-27B's are 3/211 and 2/202. Neither Gemini model produces a hallucinated object in the scenes resolved by Webots. Gemma-4B is the exception, with 31/46 hallucinated objects in direct output and 16/85 in compiled output.

Object correctness. Object F1 exceeds 0.95 for every S4 configuration except Gemma-4B and Gemma-12B direct in scenes resolved by Webots. Gemma-12B reaches 0.947 direct and 0.977 compiled, while Gemma-27B reaches 0.986 with either pipeline. The Gemini models range from 0.991 to 0.995. Gemma-4B scores much lower, at 0.265 direct and 0.818 compiled. Parser-fallback scores remain separate, as defined in Section 5.2.

Attribute correctness. Attribute correctness varies substantially across the tested models. Gemma-4B satisfies 0/19 declared attributes directly and 4/16 through compilation; Gemma-12B reaches 23/34 and 31/34. Scores are substantially higher for Gemma-27B (36/39 direct and 37/39 compiled), Gemini-Pro (37/37 and 39/39), and Gemini-Flash (38/39 and 36/37). No current per-property breakdown is available in the authoritative aggregates, so earlier colour- and size-specific percentages are not reused.

Field-name validity. Invalid fields are diagnostic and do not directly determine E2E success. The largest held-out counts are 44 among 705 fields for Gemma-4B compiled and 30 among 979 for Gemma-12B direct. Among the other configurations with field totals, invalid-field counts range from zero to ten. Both pipelines record zero invalid fields in the historical main-set T1–T3 results after repair.

6.4 Impact of Repair and Validation

This comparison keeps the model, pipeline, strategy, and dataset fixed while contrasting deterministic repair alone with the full stack on held-out T4. In the latter, deterministic repair runs again after each of up to three LLM repair attempts. The raw stage is also shown to separate the contribution of deterministic repair from the subsequent LLM repair loop.

Table 6.7: `e2e_success` at the raw, deterministic-repair, and final stages on held-out T4 (N=30), S4. Columns trace the same initial outputs through repair. Initial generation or compiler failures remain in the denominator as unsuccessful scenes. Gemma-4B is omitted because a complete paired stage comparison is unavailable.

Model	Pipeline	Raw	Det. repair only	+ LLM repair
Gemma-12B	Direct	0/30	0/30	2/30
Gemma-12B	Compiled	0/30	1/30	4/30
Gemma-27B	Direct	0/30	0/30	10/30
Gemma-27B	Compiled	0/30	4/30	10/30
Gemini-Pro	Direct	0/30	11/30	22/30
Gemini-Pro	Compiled	0/30	15/30	23/30
Gemini-Flash	Direct	0/30	8/30	17/30
Gemini-Flash	Compiled	0/30	9/30	17/30

No raw output reaches end-to-end success in these S4 configurations. Deterministic repair alone produces between eight and fifteen passes for the Gemini

configurations, but between zero and four for the Gemma configurations. Adding the LLM repair loop increases the final pass count in every row. The largest net increase is Gemini-Pro direct, from 11/30 to 22/30. Gemma-27B direct increases from 0/30 to 10/30, while compiled increases from 4/30 to 10/30. These gains measure the LLM loop together with the deterministic repairs applied after its attempts, rather than LLM correction in isolation.

Repair efficiency. Mean LLM repair attempts per held-out scene are 2.90 direct and 2.40 compiled for Gemma-12B, 2.60 and 2.30 for Gemma-27B, 1.30 and 1.13 for Gemini-Pro, and 1.70 and 1.80 for Gemini-Flash. Means use all thirty scenes in each configuration; scenes without an LLM repair call contribute zero. The Gemini-Pro compiled pipeline passes fifteen scenes after deterministic repair alone, leaving fewer scenes requiring LLM repair. The full stack nevertheless leaves seven scenes unsuccessful in that configuration, compared with eight for Gemini-Pro direct, thirteen for each Gemini-Flash pipeline, twenty for each Gemma-27B pipeline, and twenty-eight direct and twenty-six compiled for Gemma-12B.

Deterministic repair's isolated contribution. The paired raw and deterministic columns show that deterministic repair can produce end-to-end passes without an LLM repair call, especially for the Gemini models. Its contribution is not limited to loadability: it also makes bounded facing, support, and collision adjustments. Repair is not uniformly beneficial for every individual output, however. For Gemini-Pro direct, all thirty scenes load at the deterministic stage but only twenty-eight load at the final stage, even though the E2E count increases from eleven to twenty-two. Aggregate gains therefore coexist with individual regressions.

Chapter 7

Discussion

This chapter considers what the results in Chapter 6 tell us about generating Webots scenes from natural language. The discussion is organised around five issues: differences between models, the value of an intermediate representation, prompting components, automated repair, and the interpretation of correctness. The main conclusions draw on the thirty-scene held-out T4 evaluation. Historical development experiments provide additional context, but they are not treated as equivalent tests under the current evaluation protocol.

7.1 Main Findings

7.1.1 Generation guidance and differences between models

The zero-shot results show that producing a loadable world and producing the requested scene are different achievements. Every S0 configuration in Table 6.3 reaches `e2e_success` 0/30, yet the Gemini models load many of their outputs. On held-out T4, Gemini-Pro loads 17/30 direct and 24/30 compiled scenes, while Gemini-Flash loads 25/30 direct and 21/30 compiled scenes. The three Gemma models load none of their direct S0 outputs. The baseline therefore does not support the claim that all models are unable to produce loadable Webots files. It shows instead that loadability alone is insufficient, and that unaided performance differs substantially between the tested models.

With S4 and the full repair stack, load success reaches 27–30 out of 30 for Gemma-12B, Gemma-27B, and both Gemini models. End-to-end performance remains much more uneven. Gemini-Pro reaches 22/30 direct and 23/30 compiled passes; Gemini-Flash reaches 17/30 with either pipeline; Gemma-27B reaches 10/30 with either pipeline; and Gemma-12B reaches 2/30 direct and 4/30 compiled passes. Gemma-4B reaches no end-to-end passes in either configuration. Its object and

attribute errors in Section 6.3 show that generation guidance and repair do not reliably recover the intended scene for this model.

These comparisons establish that the complete system improves on the zero-shot baseline for four of the five models, but not that prompting or repair alone explains the improvement. S0 has neither the additional generation guidance nor repair, whereas S4 has both. Their comparison measures the combined intervention. The paired S4 repair stages discussed below provide a more specific view of repair's contribution.

The ordering within the Gemma family is consistent with larger models handling this task more successfully. It is not evidence that parameter count alone determines performance. Architecture, training, instruction following, and provider behaviour also differ, and the comparisons with Gemini do not isolate model size. The practical finding is narrower: the same generation-and-repair framework produces markedly different outcomes depending on the model placed inside it.

7.1.2 The value of a JSON intermediate representation

The held-out results do not identify a universally better pipeline. Compilation increases the E2E count by two scenes for Gemma-12B and one scene for Gemini-Pro, while Gemma-4B, Gemma-27B, and Gemini-Flash tie between pipelines. These are small differences from a single completed evaluation per configuration, so they should not be described as clear or general accuracy advantages.

The cost comparison is more consistent. For the four models with cost totals in Table 6.2, compilation costs approximately 1.3–1.4 times as much as direct generation on held-out T4. The extra text-to-JSON stage introduces model calls and another opportunity for error; it does not remove the need to generate valid WBT or realise spatial relations. Its value must therefore be judged against both the final accuracy and the additional cost, not inferred from the presence of a structured representation alone.

The gold-pipeline diagnostic helps explain this limitation. In the historical twenty-scene T4 development experiment, the generator receives the hand-authored specification rather than compiler output, yet only 15/20 worlds load and 31/144 evaluated relations are satisfied without repair. Removing extraction errors does not remove generation errors. This is useful stage-isolation evidence, but not a numerical upper bound for the current held-out pipeline: the diagnostic uses different scenes and a historical evaluation configuration.

The retained T1–T3 comparisons offer development context. Under their earlier E2E definition, compilation is ahead by one scene on T1 and behind on T2 and T3. Those runs use tolerance version 4 and have no collision records, whereas the held-out runs use version 8 and require collision freedom. They cannot establish how current collision-inclusive success changes across tiers. Nor do the present

results establish why a particular model benefits from JSON. A targeted extraction and generation ablation would be needed to distinguish a useful decomposition from extra error introduced by the compiler.

7.1.3 Structural rules, schema retrieval, and few-shot examples

The historical strategy ablation in Section 6.1.3 provides the direct evidence about individual prompting components. It uses ten T4 development scenes, four models, the direct pipeline, and no repair. This ten-scene version is not a subset of the twenty-scene development version used by the gold diagnostic: only five scene IDs and descriptions are shared. Both experiments remain separate from the thirty-scene held-out comparison.

Structural rules improve loadability for the three Gemma models, but their S1 relation scores remain zero. This is a modest improvement rather than closure of most of the loadability gap: S1 loads only one, three, and five of ten scenes for Gemma-4B, Gemma-12B, and Gemma-27B, respectively. Schema retrieval helps Gemma-27B, while its benefit is inconsistent for the two smaller models. Few-shot examples give both smaller models ten successful loads and their first non-zero relation scores in this ablation. The examples appear particularly helpful for demonstrating the structure of a valid world, although the remaining spatial errors show that copying that structure is not enough to construct the requested layout.

The combined strategy is not uniformly best. Gemma-27B reaches its highest historical conditional SCR under S4, at 50.0%, but Gemma-4B and Gemma-12B load fewer scenes under S4 than under S3. Gemma-12B's S4 SCR of 30.8% comes from just two loaded scenes; it should not be read as an overall improvement over S3, which loads all ten. Gemini-Pro also shows a non-monotonic response: adding schema retrieval gives a lower relation score than structural rules alone, while few-shot examples and the combined strategy give higher scores.

S2 and S3 are parallel extensions of S1, not consecutive steps in a cumulative progression. The experiment supports component-level observations within this design, not a claim that each added component steadily improves performance. S4 was subsequently evaluated as a fixed full-system condition on held-out T4. Keeping strategy selection on development data is appropriate, but it leaves the interaction between prompting components and repair unresolved.

7.1.4 What automated repair contributes

The paired held-out stages in Table 6.7 show that both repair mechanisms contribute to final performance. No raw S4 output passes end to end in the eight configurations

with paired stage evidence. After deterministic repair, Gemini-Pro passes eleven direct and fifteen compiled scenes, while Gemini-Flash passes eight direct and nine compiled scenes. The corresponding Gemma counts range from zero to four. Adding the LLM repair loop increases the final count in every configuration.

The division of responsibility is therefore not simply syntax versus spatial reasoning. The implementation applies catalogue-driven fixes, but it also rotates existing objects to correct supported horizontal facing constraints, changes vertical placement using measured support offsets, and searches a bounded set of collision-reducing translations. These operations change the geometry and can improve SCR and E2E success without an LLM repair call. The correct distinction is between restricted geometric correction of an existing candidate and a general-purpose layout solver that constructs or jointly optimises a scene from its constraints.

Collision-displacement candidates are re-evaluated and accepted only when collision severity decreases without a regression in the protected checks. This safeguard applies to that candidate search; it is not a guarantee that every support correction, facing correction, or LLM rewrite preserves all previous successes. Indeed, Gemini-Pro direct loads all thirty scenes after deterministic repair but only twenty-eight at the final stage, although its E2E count rises from eleven to twenty-two. Repair improves the aggregate outcome while still damaging some individual outputs.

The increase from the deterministic stage to the final stage measures the LLM loop together with deterministic corrections reapplied after each attempt. It cannot be assigned to the LLM alone. The results should therefore be interpreted as performance of the evaluated hybrid system, not as a direct measure of unaided model spatial competence. The larger deterministic-stage pass counts for Gemini also suggest that some initial outputs are closer to scenes recoverable by the implemented routines; they do not establish a model-independent split of responsibility.

Repair feedback is derived from the requested constraints and the generated scene, rather than hidden ground-truth absolute coordinates. The final score therefore measures the complete generation-and-repair system. A broader deterministic solver could be studied as a separate solver-assisted configuration, but should not be confused with the restricted repair mechanisms evaluated here or described as an established upper bound.

7.1.5 Separating the different forms of correctness

The multi-dimensional evaluation is valuable because different summaries answer different questions. LSR tells us whether Webots returns a usable inspected world; object and attribute metrics assess scene content; SCR assesses requested relations; and the collision gate adds a physical-validity requirement. The strict E2E criterion additionally requires clean loading and asks whether all required conditions hold in

the same scene. Field-name validity remains a diagnostic, not an additional E2E gate.

On held-out T4, Gemma-27B direct loads all thirty scenes and satisfies 337/390 relations, yet only ten scenes pass end to end. This is not a contradiction. Pooled relation correctness gives credit for individual constraints, whereas E2E gives no partial credit and can fail on a single attribute, object, relation, or collision. Likewise, a high facing score does not imply a collision-free or otherwise correct scene.

Availability must also be kept separate from failure. Conditional SCR excludes the relations of scenes that fail to load; unavailable facing checks within loaded scenes remain unsatisfied. The facing totals in Table 6.6 refer to constraints retained in each run's relation metrics, rather than all fifty-seven requested facing constraints. Collision checking can be unavailable even for a loaded world, and an unavailable check cannot establish collision freedom. Conversely, a scene that already fails another required condition is a confirmed E2E failure even if collision checking is unavailable.

These distinctions make the remaining errors easier to interpret, but they do not establish that the metrics are statistically independent or that the evaluator is infallible. Their main contribution is a more transparent account of where a generated scene falls short.

7.2 Limitations

The first limitation is the scope of the evaluated scenes. The held-out comparison covers thirty demanding T4 scenes across five models, rather than a broad sample of arbitrary Webots environments. The descriptions focus on static objects, attributes, and spatial relations. The results do not establish correctness for dynamic interactions, robot behaviour, sensor configurations, or downstream tasks. The historical T1–T3 tables cover twenty scenes per tier for Gemma-27B, while smaller exploratory runs for other models exist in the archive. Those smaller runs do not constitute the same full-tier cross-model comparison.

The evaluation also changed during development. The retained main-set T1–T3 runs use tolerance version 4 and contain no collision-check records. Their E2E scores retain the earlier criterion and are labelled historical in Chapter 6. They are not verified current E2E passes. The gold diagnostic and strategy ablation are also development evidence. Different dataset versions, thresholds, and repair implementations prevent their scores from being treated as controlled comparisons with the version-8 held-out evaluation.

The held-out descriptions were reserved for final evaluation and were not used for debugging or prompting-strategy selection. This keeps the final comparison

separate from the development experiments. It does not, however, remove the limits imposed by thirty scenes, selected completed evaluations, and the absence of independent repeated trials.

The reported configurations represent selected completed evaluations, not averages over independent repeated trials. Repair attempts within one scene are not additional trials: later candidates depend on earlier outputs and feedback. Temperature zero reduces sampling variability but does not demonstrate reproducibility across provider routing, software versions, or infrastructure conditions. No empirical run-to-run variance estimate is available. The one-scene Gemini-Pro pipeline difference and two-scene Gemma-12B difference should therefore be treated as descriptive observations rather than established advantages.

API errors, compiler failures, unavailable collision backends, and inspection timeouts can affect the outcome alongside generation quality. Initial failures remain unsuccessful scenes in the thirty-scene denominator; repair attempts are not counted as additional scenes. A scene that does not complete the required checks cannot pass E2E, but the reason for failure still matters.

The repair-stage evidence is not equally complete for every model. Gemma-4B's final E2E outcomes are reported, but it is not included in the paired repair-stage table. Its final zero-success count alone cannot quantify how much each repair mechanism contributed. The retained historical Gemma-4B diagnostic aggregates are supplementary evidence, not an independently verified current paired-stage evaluation. The component ablation is similarly restricted to four models, ten historical development scenes, one pipeline, and no repair. It does not identify the separate effects of every component in the complete held-out system.

Evaluator definitions impose additional limits on interpretation. The `near` and `between` thresholds depend on the generated scene scale. A larger floor, a more spread-out layout, or a distant extra object can relax those checks without changing the requested relation. Other metrics may penalise these changes, but do not remove the resulting dependence from SCR. Facing uses local +X unless the specification supplies an override; it is not a visual-front detector for each furniture PROTO. Its scores describe compliance with that axis convention. Collision checking covers the initial geometry under the configured penetration tolerance, not long-term stability or successful physical interaction.

Feedback and final scoring share the same evaluator. This makes repair actionable, but also means that improvements reflect compliance with its implemented constraints and diagnostics, rather than an independent assessment of scene quality. The calibration suite and implementation tests support reliability, but cannot guarantee coverage of all scene geometries and representations. The documented live audit covers the compiled few-shot demonstrations; the retained evidence does not establish an equivalent complete live audit of the direct examples. That audit remains an important verification step, not proof of an existing defect or a claim

that a hypothetical defect would affect every scene equally.

Finally, the current Results do not report a harmonised breakdown for all relation types or attribute properties. Claims about the hardest compound cardinal relation, colour-specific errors, or furniture-sizing errors cannot be inferred from the pooled totals alone. The discussion is accordingly limited to the overall relation, attribute, facing, collision, and object results that are actually reported.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis studied whether natural-language descriptions can be translated into native Webots worlds that are both executable and faithful to the requested scene. It developed two complementary contributions: a framework for direct and JSON-mediated generation with simulator-guided repair, and a benchmark and evaluation framework that distinguish loadability, scene content, spatial correctness, and collision freedom.

The held-out evaluation shows that a plausible or loadable world is not necessarily a correct one. None of the zero-shot configurations passes the strict end-to-end criterion. The complete system improves that outcome for four of the five models, but performance remains model-dependent and none achieves universal success. The central practical challenge is therefore not just writing valid WBT. It is satisfying all of the requested constraints and physical-validity checks together in the same scene.

A JSON intermediate representation does not provide a consistent accuracy advantage in the held-out comparison. It gives small gains for two models and ties for the other three, while increasing cost for the four models with cost totals. Structuring the task into two stages is a useful design option, not a correctness guarantee. The gold-specification diagnostic reinforces this distinction: removing extraction errors still leaves substantial difficulty in generating and spatially realising the world.

Both repair mechanisms matter. Deterministic repair corrects known mechanical defects and a restricted set of facing, support, and collision failures. Feedback-driven LLM repair, followed by further deterministic correction, improves the final

pass count in every configuration with paired stage evidence. This is a hybrid generation-and-repair system, and its scores should be read as such. The results do not establish that semantic and spatial correction belongs exclusively to the LLM, nor that repair always preserves an individual candidate's previous successes.

The multi-dimensional evaluation is essential to interpreting these outcomes. It makes visible the gap between a loadable world, a world that satisfies many individual relations, and a fully correct scene. Keeping these measures separate also exposes unavailable checks and the influence of the evaluator's geometric conventions. The remaining limitations, such as small datasets selected rather than repeated evaluations, and historical results under earlier definitions, restrict the strength of general claims without removing the value of the observed failure patterns.

The main implication is that reliable simulator-world generation requires more than stronger prompting or a more elaborate representation. Generation, feedback, bounded correction, and explicit validation must be considered together. The framework developed here provides a way to inspect that process and identify what remains unresolved: for the stronger configurations, the persistent gap is between producing a loadable scene and realising a scene that satisfies every required semantic, spatial, and collision check.

8.2 Future Work

Reproducible evaluation on additional descriptions. Future evaluations should preserve the separation between development and held-out testing. Saving the exact prompts, catalogue, repair code, tolerances, simulator environment, and immutable stage evidence would make each configuration easier to reproduce. Additional unseen descriptions would then test whether the observed performance extends beyond the present thirty-scene benchmark, without reusing its results for system selection.

Repeated evaluations and broader coverage. Repeating selected configurations would allow uncertainty in E2E counts, conditional relation scores, and cost to be estimated. The comparison should also extend to the simpler tiers across all five models, using the same current evaluation and repair protocol. Re-evaluating retained T1–T3 worlds would clarify their current correctness, although rescoring final worlds alone would not reproduce the original repair trajectory under the newer implementation. Additional scenes involving varied assets and more diverse descriptions would help test whether the present findings extend beyond the benchmark's static layouts.

Matched component and repair ablations. Future experiments should hold repair assistance fixed while varying structural rules, schema retrieval, and few-shot examples. Complementary ablations should vary deterministic geometric routines and LLM repair separately, with their interactions explicitly controlled. A genuinely LLM-only repair condition must disable all deterministic geometric interventions, including facing correction. These experiments would show which components contribute most to final accuracy, rather than assigning the combined gain to a single mechanism.

Targeted study of intermediate representations. The default and multi-step compiler implementations offer a starting point for a matched comparison across models. The direct pipeline's stepwise prompting control could be examined under the same repair budget. Compiler acceptance, extraction accuracy, generation accuracy, retry calls, and total cost should be reported separately. This would help determine whether an intermediate representation reduces task complexity or simply introduces another error-prone stage.

Independent evaluator and example audits. A complete live audit of both pipelines' few-shot demonstrations would remove the verification asymmetry described in Section 7.2. The evaluator also needs broader tests for ambiguous object matching, unusual collision geometry, and asset-specific forward directions. Sensitivity analysis could compare output-dependent `near` and `between` thresholds with alternatives fixed independently of the generated scene. Independent manual review of a sample of worlds would help identify cases where passing the implemented checks does not match the intended scene.

A solver-assisted reference configuration. A broader geometric optimiser could be evaluated as a separate configuration that delegates more placement work to a deterministic mechanism. The comparison should make clear which relations it solves, which inputs it receives, and whether it changes object sizes or moves multiple objects. It would provide a solver-assisted reference, not a guaranteed upper bound on E2E success: extraction, attributes, unsupported geometry, and infeasible constraints could still prevent a pass. Comparing it with the current bounded corrections would clarify the trade-off between model-driven layout generation and stronger external geometric assistance.

Bibliography

[1] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. «A Review of Physics Simulators for Robotic Applications». In: *IEEE Access* 9 (2021), pp. 51416–51431. DOI: 10.1109/ACCESS.2021.3068769 (cit. on p. 1).

[2] Olivier Michel. «Webots: Professional Mobile Robot Simulation». In: *International Journal of Advanced Robotic Systems* 1.1 (2004), pp. 39–42. DOI: 10.5772/5618 (cit. on pp. 1, 6).

[3] Cyberbotics Ltd. *Webots Reference Manual*. https://cyberbotics.com/doc/reference/index. Accessed 15 September 2026. 2026 (cit. on pp. 1, 6, 9).

[4] Mark Chen et al. «Evaluating Large Language Models Trained on Code». In: *arXiv preprint arXiv:2107.03374* (2021). DOI: 10.48550/arXiv.2107.03374 (cit. on pp. 1, 10).

[5] Weixi Feng, Wanrong Zhu, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. «LayoutGPT: Compositional Visual Planning and Generation with Large Language Models». In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023 (cit. on pp. 2, 11).

[6] Yue Yang et al. «Holodeck: Language Guided Generation of 3D Embodied AI Environments». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 16227–16237 (cit. on pp. 2, 11, 14, 15).

[7] Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. «SceneCraft: An LLM Agent for Synthesizing 3D Scenes as Blender Code». In: *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. Proceedings of Machine Learning Research. 2024, pp. 19252–19282 (cit. on pp. 2, 11, 13–15).

[8] Xiao Xia, Dan Zhang, Zibo Liao, Zhenyu Hou, Tianrui Sun, Jing Li, Ling Fu, and Yuxiao Dong. «SceneGenAgent: Precise Industrial Scene Generation with Coding Agent». In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*. 2025, pp. 17847–17875 (cit. on pp. 2, 13–15).

[9] Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. «SceneWeaver: All-in-One 3D Scene Synthesis with an Extensible and Self-Reflective Agent». In: *arXiv preprint arXiv:2509.20414* (2025) (cit. on pp. 2, 11, 14, 15).

[10] Hongchi Xia et al. «SAGE: Scalable Agentic 3D Scene Generation for Embodied AI». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2026 (cit. on pp. 2, 12, 14, 15).

[11] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. «GenSim: Generating Robotic Simulation Tasks via Large Language Models». In: *International Conference on Learning Representations*. 2024 (cit. on pp. 2, 13).

[12] Yunfan Gao et al. «Retrieval-Augmented Generation for Large Language Models: A Survey». In: *arXiv preprint arXiv:2312.10997* (2023). DOI: 10.48550/arXiv.2312.10997 (cit. on pp. 5, 10).

[13] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. «RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation». In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2023, pp. 2471–2484. DOI: 10.18653/v1/2023.emnlp-main.151 (cit. on pp. 5, 10).

[14] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. «PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models». In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021, pp. 9895–9901. DOI: 10.18653/v1/2021.emnlp-main.779 (cit. on pp. 5, 10).

[15] Brandon T. Willard and Rémi Louf. «Efficient Guided Generation for Large Language Models». In: *arXiv preprint arXiv:2307.09702* (2023). DOI: 10.48550/arXiv.2307.09702 (cit. on pp. 5, 10).

[16] Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, and Tianqi Chen. «XGrammar: Flexible and Efficient Structured Generation Engine for Large Language Models». In: *Proceedings of Machine Learning and Systems*. Vol. 7. 2025 (cit. on pp. 5, 10).

[17] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. «Reflexion: Language Agents with Verbal Reinforcement Learning». In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023. DOI: 10.52202/075280-0377 (cit. on pp. 5, 11).

[18] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. «Teaching Large Language Models to Self-Debug». In: *International Conference on Learning Representations*. 2024 (cit. on pp. 5, 11).

[19] Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. «JSONSchemaBench: A Rigorous Benchmark of Structured Outputs for Language Models». In: *arXiv preprint arXiv:2501.10868* (2025). DOI: 10.48550/arXiv.2501.10868 (cit. on p. 10).

[20] Xingjian Ran, Yixuan Li, Linning Xu, Mulin Yu, and Bo Dai. «Direct Numerical Layout Generation for 3D Indoor Scene Synthesis via Spatial Reasoning». In: *arXiv preprint arXiv:2506.05341* (2025). DOI: 10.48550/arXiv.2506.05341 (cit. on pp. 11, 21).

[21] Fangjun Li, David C. Hogg, and Anthony G. Cohn. «Reframing Spatial Reasoning Evaluation in Language Models: A Real-World Simulation Benchmark for Qualitative Reasoning». In: *arXiv preprint arXiv:2405.15064* (2024). DOI: 10.48550/arXiv.2405.15064 (cit. on p. 12).

[22] Wufei Ma, Haoyu Chen, Guofeng Zhang, Celso M. de Melo, Alan Yuille, and Jieneng Chen. «3DSRBench: A Comprehensive 3D Spatial Reasoning Benchmark». In: *arXiv preprint arXiv:2412.07825* (2024). DOI: 10.48550/arXiv.2412.07825 (cit. on p. 12).

[23] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. «Scenic: A Language for Scenario Specification and Scene Generation». In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. Association for Computing Machinery, 2019, pp. 63–78. DOI: 10.1145/3314221.3314633 (cit. on p. 12).

[24] Jiawei Zhang, Chejian Xu, and Bo Li. «ChatScene: Knowledge-Enabled Safety-Critical Scenario Generation for Autonomous Vehicles». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024 (cit. on pp. 12, 14, 15).

[25] Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. «3D-GPT: Procedural 3D Modeling with Large Language Models». In: *arXiv preprint arXiv:2310.12945* (2023). DOI: 10.48550/arXiv.2310.12945 (cit. on p. 13).

[26] Percy Liang et al. «Holistic Evaluation of Language Models». In: *arXiv preprint arXiv:2211.09110* (2022). DOI: 10.48550/arXiv.2211.09110. URL: https://arxiv.org/abs/2211.09110 (cit. on p. 18).