HASCO: A Hybrid AI Simulation Compiler for Semantic Accident Reconstruction

Edin Jelačić
Department of Computer Science and Engineering, Mälardalen University, Sweden

Rong Gu
Prover Technology AB, Stockholm, Sweden

Cristina Seceleanu
Department of Computer Science and Engineering, Mälardalen University, Sweden

Ning Xiong
Department of Computer Science and Engineering, Mälardalen University, Sweden

Peter Backeman
Department of Computer Science and Engineering, Mälardalen University, Sweden

Tiberiu Seceleanu
Department of Computer Science and Engineering, Mälardalen University, Sweden

Zhennan Fei
Volvo Cars, Göteborg, Sweden
Chalmers University of Technology, Göteborg, Sweden

Ali Nouri
Volvo Cars, Göteborg, Sweden
Chalmers University of Technology, Göteborg, Sweden

Abstract

The validation of Automated Driving Systems (ADSs) has shifted from distance-based metrics to Scenario-Based Testing (SBT). Large Language Models (LLMs) have emerged as powerful tools with potential for generating vehicular scenarios at scale. However, generative models, used for direct simulation synthesis, produce inadequate output, therefore necessitating a more structured compilation approach. In this regard, we present **HASCO** (**H**ybrid **A**I **S**imulation **CO**mpiler), a system that translates natural-language driving scene specifications into executable simulation artifacts (XOSC/XODR files) for the esmini/OpenSCENARIO ecosystem. While LLMs excel at narrative parsing, we demonstrate that direct synthesis of simulation artifacts fails in the vast majority of cases due to hallucinated physics or schema violations. To resolve this, HASCO treats scenario creation as a **compilation task** rather than a generative one. The pipeline supports three compilation paths: direct synthesis, a Python intermediate (via `scenariogeneration`), and an ontology-guided path that grounds intent into an intermediate representation (IR) before compilation. We further evaluate a self-judging mechanism for automated repair. Across six operating modes evaluated on 40 real-world accident reports, the ontology-guided compiler and Python-based compiler achieve 95% and 90% executability rates, respectively (compared to 5% for direct synthesis). Additionally, we evaluate outputs on *semantic fidelity*, positioning HASCO as a robust tool for forensic scene reconstruction.

2012 ACM Subject Classification Computing methodologies → Natural language generation

Keywords and phrases Autonomous Driving, OpenSCENARIO, Large Language Models, Scenario Generation, Semantic Reconstruction

Digital Object Identifier 10.4230/OASIcs.AEiC.2026.4

Funding We gratefully acknowledge the Swedish Knowledge Foundation via the PerFlex (*Performant and Flexible digital Systems through Verifiable Artificial Intelligence*) project, grant nr. 20220033, as well as Mälardalen University via the TSS-INSID (*In-Silico Driving Assurance Using Machine-Learning-Powered Verification and Synthesis for Safe Autonomous Vehicles*) project, of the Trusted Smart Systems initiative, which supported this research.

1 Introduction

Automated Driving Systems (ADSs) represent a paradigm shift in modern transportation, promising to significantly reduce traffic accidents. A recent *National Highway Traffic Safety Administration* study showed that Advanced Driver Assistance Systems (ADAS) features like automatic emergency braking reduce rear-end crashes by up to 49% [2]. While ADSs offer potential for optimized traffic flow and enhanced mobility, deploying these systems in unconstrained operational design domains (ODDs) remains a significant engineering challenge [13].

Unlike conventional software operating within bounded logical environments, ADSs must make safety-critical decisions in stochastic, open-world settings. Traditional "distance-based" validation of these mechanisms has proven practically unsuitable [24]. This realization has driven the industry toward Scenario-Based Testing (SBT), validating ADSs against libraries of concrete, critical traffic situations rather than random mileage [16]. The advent of generative artificial intelligence (AI) pairs well with this paradigm [27].

Natural-language scene descriptions, such as police accident reports or crash news reports, are expressive but frequently underspecified for direct simulation [11]. Dynamic vehicular simulations are deterministically rigid and tend to require significantly more information for execution than is provided in the aforementioned reports. The industry standard for describing dynamic driving maneuvers is ASAM OpenSCENARIO [1], which mandates a strict XML schema to define actor behaviors and environmental states. Converting natural language reports into executable OpenSCENARIO files requires solving two distinct problems: *Semantic Parsing* (meaning of the scenario) and *Syntactic Engineering* (producing valid XML with correct coordinate geometry).

We demonstrate that due to the semantic and syntactic complexity of OpenSCENARIO, generating XML directly from prompts is fundamentally brittle. Specifically, stochastic models struggle to maintain the long-range dependencies required by the standard, where a single maneuver requires coordinated updates across position, speed, and orientation tags that must remain mathematically consistent over time. Therefore, we propose **HASCO**, an LLM-based structured pipeline that decouples semantic reasoning from syntactic generation. Based on our experience, LLMs are proficient at extracting meaning from natural language text. The pipeline effectively functions as a semantic compiler: the LLM acts as a parser that extracts high-level logic, while the deterministic pipeline handles the "assembly code" generation of the XML.

Currently, translating textual reports into simulation requires significant manual engineering effort to close the gap between narrative events and the rigid scenario reconstruction syntax.

We position HASCO not as a replacement for expert judgment, but as a Human-in-the-Loop co-generation tool versus the manual process, as can be seen in the workflow diagram at Figure 1. By automating the syntactic manual processing of XML engineering, HASCO allows experts to focus on the "semantic validity" of the scenario, accelerating the digitization of crash databases for later use in ADS testing and crash reconstruction.

1.1 The specification gap

Menzel et al. [15] categorize vehicular scenarios into three distinct levels based on their granularity and machine readability: functional, logical, and concrete, extending upon the foundational scenario definitions provided by Ulbrich et al. [23]. The hierarchy begins with **functional scenarios**, which consist of a verbal description focusing on actions and events,

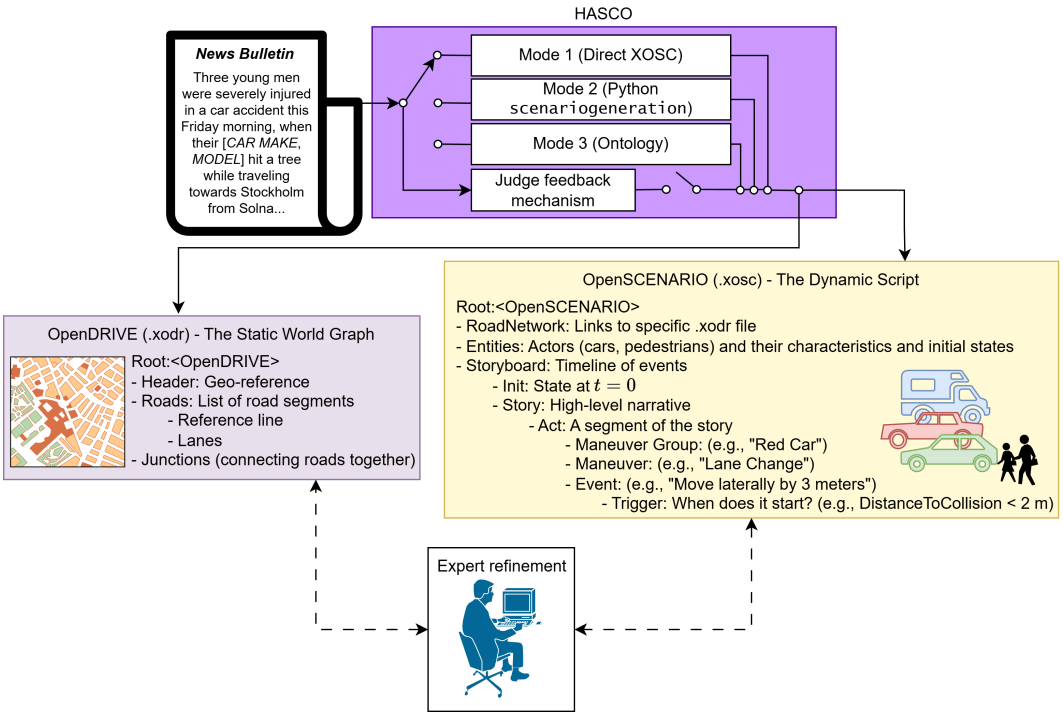


Figure 1 High-level architecture of the HASCO pipeline.

such as "maneuvers like cut-ins and following a vehicle ahead." This level corresponds directly to the nature of news reporting, where the emphasis is on semantic understanding rather than mathematical precision. As the level of detail increases, the machine readability increases. The hierarchy progresses to **logical scenarios**, defined by parameter ranges and distributions and finally to **concrete scenarios**, which are defined by exact parameter values.

The "specification gap" addressed in this work can be understood as the translational distance between these levels. News articles are rich in context but underspecified. Conversely, standards like OpenSCENARIO typically operate at the concrete layer, requiring precise coordinates, trajectories, and triggers to function. This work links these varying aspects by processing the linguistic descriptions of functional scenarios and refining them, alongside imputation, into the exact parameters required for concrete, executable simulations.

1.2 The need for code co-generation

While the ultimate ambition of generative AI in automotive engineering is fully automated scenario synthesis, current limitations necessitate a more guarded approach. Manual translation of forensic reports into simulation is prohibitively slow, yet direct end-to-end generation by LLMs remains unreliable for safety-critical applications. As noted by Nouri et al. [17], while LLMs demonstrate significant potential in accelerating software development, their stochastic nature introduces distinct risks, including hallucinations and non-compliance with safety requirements. Consequently, relying on them as autonomous creators of forensic scenarios is insufficient.

To address this, we adopt the "code co-generation" paradigm, positioning HASCO as a "Human-in-the-Loop" compiler. This methodology aligns with the "Generate fast, Eliminate fast" principle proposed by Nouri et al., which emphasizes integrating rapid,

automated sanity checks directly into the generation pipeline to filter out invalid outputs before they reach human review. By subjecting generated artifacts to preliminary assessment via a deterministic feedback loop (Software-in-the-Loop simulation), we mitigate the risk of generating syntactically valid but semantically illogical code.

In this framework, the role of the LLM shifts from an autonomous agent to a semantic compiler. By automating the syntactic complexity of OpenSCENARIO (the "boilerplate"), HASCO allows human experts to shift their focus from debugging XML tags to validating the semantic fidelity of crash parameters. This "Human-in-the-Loop" workflow reduces the reconstruction burden from manual coding to mere high-level overview and supervision.

To validate this paradigm, we conduct a dual-layer analysis on real-world accident reports. We evaluate performance not only on executability (the rate of valid code generation) but also on semantic fidelity, utilizing a graded qualitative rubric to measure the alignment between the generated simulation and the original forensic narrative. We demonstrate that high executability does not guarantee forensic accuracy, reinforcing the necessity of the expert-in-the-loop for final semantic validation.

To address this gap and validate the "Code Co-Generation" paradigm proposed in this work, we formulate the following research questions:

- **RQ1 (Feasibility)**: To what extent can an LLM-based tool automate the generation of syntactically valid OpenSCENARIO files from unstructured forensic text?
- **RQ2 (Fidelity)**: How does the integration of a deterministic validation mechanism (the Judge) impact the semantic fidelity of the generated scenarios compared to direct "one-shot" LLM synthesis?

2 Background

2.1 The OpenSCENARIO standard, the `scenariogeneration` library and esmini

ASAM OpenSCENARIO serves as the industry standard for describing dynamic content in driving simulations. OpenSCENARIO XML (v1.x) relies on a rigid, hierarchical Storyboard architecture to orchestrate events within a static map definition. The map definition is part of the corresponding OpenDRIVE standard. A scenario is not merely a sequence of frames, but a complex state machine where `Acts`, `Maneuvers`, and `Events` are triggered by specific run-time conditions (e.g., *"start lane change when Time-to-Collision < 2s"*).

This structure creates a high degree of syntactic coupling. A single semantic action, such as an "overtaking maneuver," requires synchronized updates across multiple nested tags, such as defining trajectories, speed profiles and trigger conditions that must remain mathematically consistent relative to the static road network. The existence of these dependencies pose a specific challenge for probabilistic models like LLMs, which are prone to losing context over long token sequences.

Manually generating machine-readable `.xosc` and `.xodr` files is highly complex. A useful intermediary framework for generating OpenSCENARIO files is the `scenariogeneration` Python package [20]. This package provides bindings between Python and the standard, enabling the generation of `.xosc` and `.xodr` files using a well-established and user-friendly syntax.

To execute these files, we utilize esmini [8], the standard open-source reference implementation. The esmini tool functions not only as a visualizer but as a deterministic constraint solver. Unlike high-fidelity simulators such as CARLA [6] or LGSVL [22], which prioritize realism and sensor rendering and require significant computational overhead, esmini focuses

exclusively on logical validation and trajectory execution. This lightweight, headless architecture allows for fast validation cycles, making it the ideal simulator for our iterative feedback loop. In our pipeline, esmini serves as the ground-truth validator: if the generated XML contains logical contradictions or syntax errors, esmini provides the explicit failure signals required for the feedback loop.

2.2 Scenario-based testing and reconstruction

The validation of ADAS/AD systems has shifted from distance-based validation (driving billions of miles) to Scenario-Based Testing (SBT). As established by Wachenfeld et al. [24], physical test driving alone is statistically infeasible for validating safe autonomy due to the rarity of critical events. To verify safety, manufacturers must therefore test systems against rare, safety-critical events (more specifically, accidents) which are statistically scarce in naturalistic driving data [16, 21].

This methodology was standardized by the PEGASUS project [25], as part of which Menzel et al. established the aforementioned widely adopted layer model (functional, logical, concrete) for highway automation. Its successor, the VVM project [9], subsequently extended these verification frameworks to complex urban environments, cementing simulation as the cornerstone of industrial safety argumentation.

Currently, populating these test libraries relies on two primary methods:

1. **Manual Engineering:** Experts manually script scenarios using simulation environment tools (e.g., IPG CarMaker, dSPACE ASM Traffic, VIRES Virtual Test Drive). This ensures high fidelity and expert control but is unscalable and labor-intensive, as every parameter must be explicitly defined and maintained [15, 3].
2. **Sensor-Based Reconstruction:** Pipelines that convert log data (Lidar/Camera) from test vehicles into simulation files. While accurate, this method is strictly limited to events the fleet has actually encountered during operation [10].

This creates a "data scarcity" gap, as thousands of critical crashes occur daily and are documented in textual reports (police records, news), but this data remains inaccessible to simulation pipelines due to its unstructured nature. Automating the translation of these texts into executable scenarios offers a scalable third path for generating the critical edge cases required for robust ADAS validation [11, 12].

3 Related Work

Recent works have attempted to automate the pipeline of transforming human-readable functional scenarios into machine-readable logical scenarios using Large Language Models (LLMs). Deng et al. [5] introduced TARGET, which generates test scenarios from traffic rules. While effective for regulatory compliance, TARGET focuses on explicit rule violations rather than the complex, causal narratives inherent in accident reconstruction. Similarly, Zhao et al. [26] proposed Chat2Scenario, utilizing LLMs to extract scenario parameters from naturalistic datasets. Crucially, however, Chat2Scenario is primarily an extraction framework; it does not address the "syntactic engineering" challenge of generating valid, complex OpenSCENARIO XML structures from scratch when no pre-existing dataset match exists.

Closer to our domain, Elmaaroufi et al. [7] introduced ScenicNL to convert police reports into simulations. Their approach targets the *Scenic* probabilistic programming language. While Scenic is concise, the automotive industry standard remains ASAM OpenSCENARIO (XML), which is significantly more verbose and prone to long-dependency syntax errors that ScenicNL does not address.

Guo et al. [11] and Ji et al. [12] utilize LLMs to parse reports into OpenSCENARIO, but treat the extraction merely as a seed for fuzzing. Txt2Sce mutates scenarios for stress-testing, and SoVAR prioritizes generalization over forensic reconstruction. Our approach differs fundamentally in intent: HASCO is a forensic reconstruction tool. We do not aim to diverge from the text via mutating instances, but to converge on it via validation.

Finally, Nouri et al. [17] highlight the stochastic risks of using LLMs for safety-critical software, arguing that LLMs should be treated as "Code Co-generators" rather than autonomous agents. We adopt this philosophy directly. While the works above address parts of the pipeline, none successfully integrate a *deterministic validation loop* (a Judge) to automatically correct the specific syntactic and semantic hallucinations common when generating verbose OpenSCENARIO XML from forensic text. More specifically, a significant gap remains in ensuring semantic fidelity of forensic reconstructions, targeting that the generated simulation adheres not only to the laws of physics but to the specific causal narrative of a news report.

4 Methodology: the HASCO compiler

We introduce HASCO (Hybrid AI Simulation Compiler), a command-line interface (CLI) based toolchain designed to bridge the specification gap between unstructured traffic reports and executable OpenSCENARIO files. The architecture is composed of four synchronized pipelines: the Geospatial Pipeline for static world generation, the Vehicle Extraction Module for actor dimensioning, the Semantic Pipeline for dynamic scenario synthesis, and the Validation Loop for reaching forensic fidelity. The pipelines in this work were executed using OpenAI's **gpt-5-mini** LLM, but remain model-agnostic, and other models may be utilized. The utilization of different models may lead to varying token counts, due to varying underlying tokenizer architectures.

4.1 Static world generation (geospatial pipeline)

To ensure environmental exactness, HASCO first extracts location strings from the report and queries the Nominatim API for geocoordinates. It then downloads the corresponding OpenStreetMap (OSM) data for the defined bounding box. This OSM data is converted into an OpenDRIVE (`.xodr`) network using a dockerized CARLA conversion script, ensuring the static road topology matches the real-world location described in the text. Finally, the XODR map is "summarized" into a textual description that gets passed later into a combined prompt for the scenario synthesis pipeline. The pipeline may be observed at Figure 2.

4.2 Vehicle extraction module

In parallel with this, the vehicle extraction module resolves actor dimensions. Rather than relying on a static lookup table, this module operates dynamically per scenario. It parses vehicle descriptions (e.g., "2015 Volvo V60" or "small silver hatchback") and queries an external knowledge base from Wikipedia to retrieve precise physical dimensions (length, width, wheelbase). For vague descriptions, it approximates dimensions based on the nearest real-world vehicle class, ensuring that collision physics in the simulation are grounded in realistic bounding boxes. This output too, in the form of an OpenSCENARIO vehicle catalog, is passed to the combined prompt for later use. The pipeline may be observed in Figure 3.

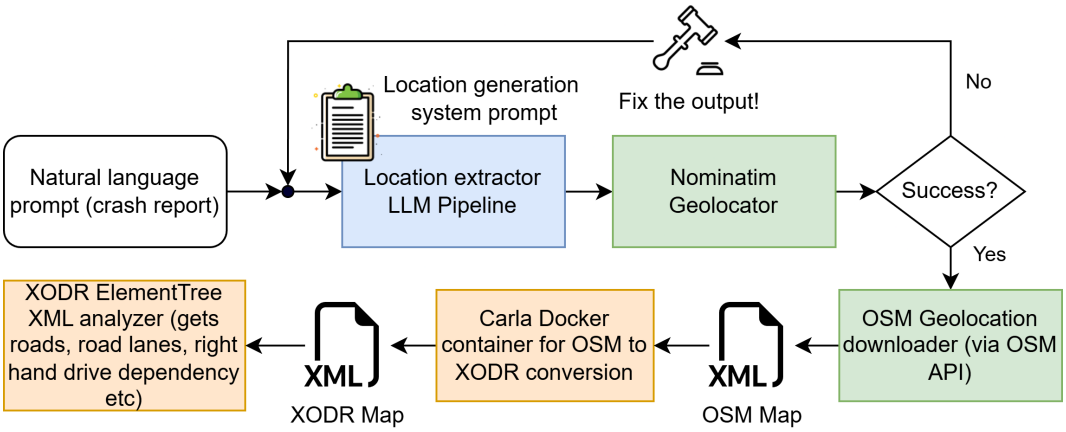


Figure 2 Geospatial pipeline.

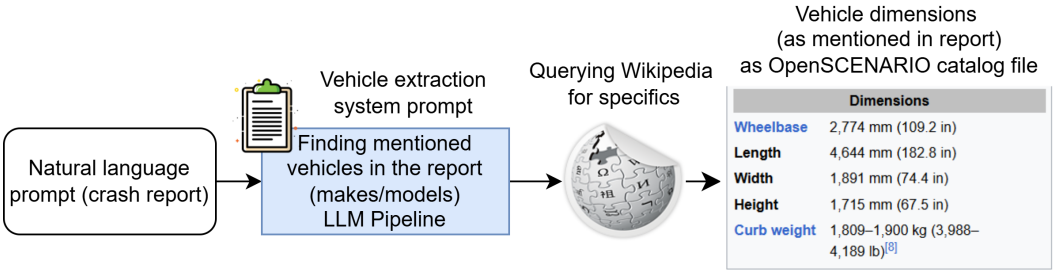


Figure 3 Vehicle extraction module.

4.3 Dynamic scenario synthesis (semantic pipeline)

Simultaneously, the system synthesizes the dynamic scenario, the temporal script that dictates the behaviors of moving actors, their specific maneuvers and event triggers over time. To ensure the generated code is both syntactically valid and compliant with industry standards, the pipeline employs a hybrid Retrieval-Augmented Generation (RAG) module. RAG enhances the language model's accuracy by actively retrieving relevant technical documentation to ground its output, pulling from a composite knowledge base containing:

- The ASAM OpenSCENARIO Standard (v1.x) schema definitions,
- The esmini documentation and validator rules,
- The complete `scenariogeneration` Python library repository.

A re-ranking mechanism filters retrieved snippets to prioritize pertinent API calls (e.g., specific `ManeuverGroup` classes), which are then injected into the system prompt to guide the LLM's code generation.

4.4 Multi-strategy synthesis

To evaluate the trade-off between various levels of scenario representation, HASCO implements three distinct synthesis strategies. Each strategy operates on a different level of specification, employing a unique feedback loop to ensure validity before the final output reaches the Judge. The structural overview of these different strategies is visible in Figure 4, and the execution in Figure 5.

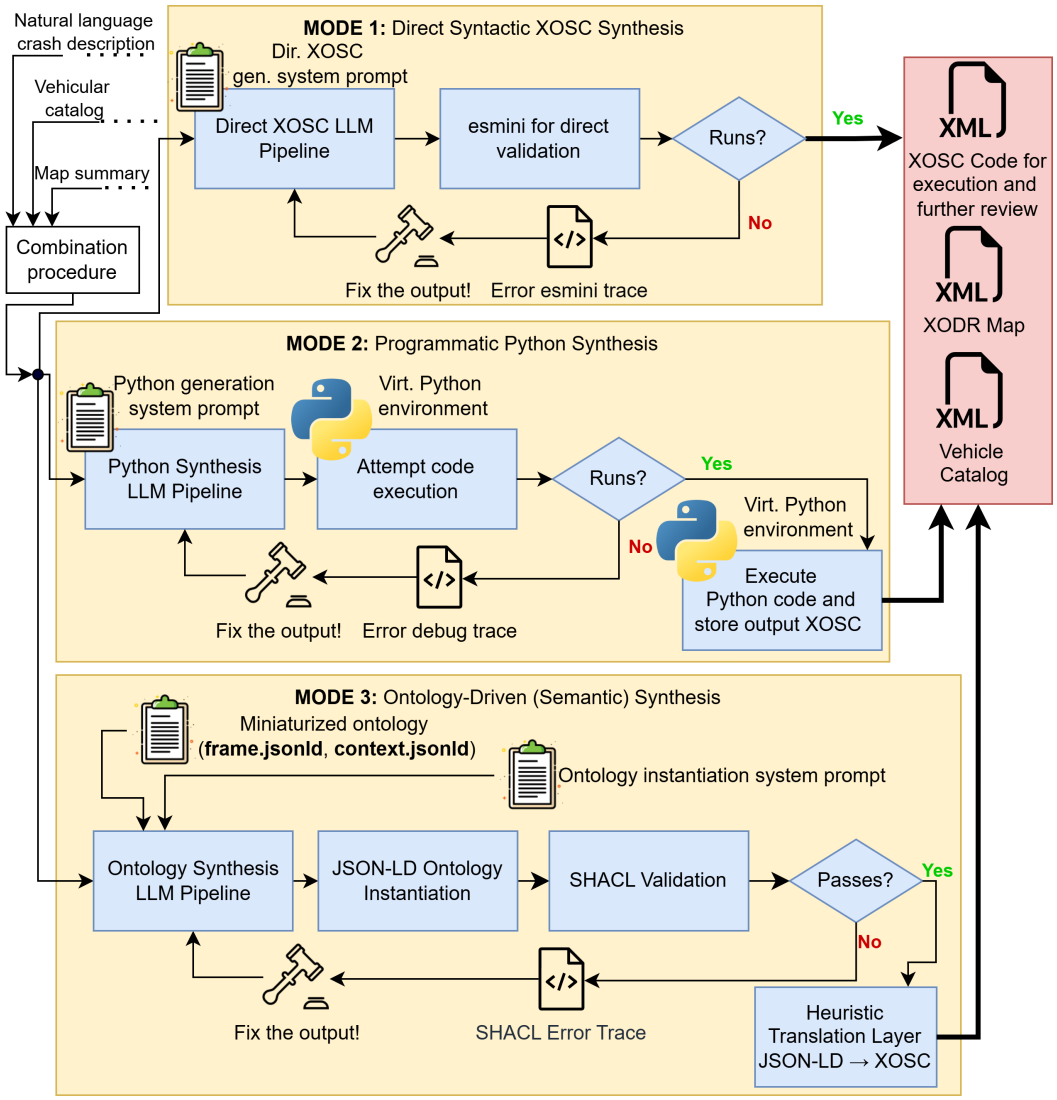


Figure 4 The Multi-Strategy Synthesis Architecture. HASCO accepts a unified prompt and routes it through one of three synthesis strategies. While Mode 1 (XML) and Mode 2 (Python) rely on execution-based feedback loops (via esmini and the Python interpreter, respectively), Mode 3 (Ontology) employs a semantic constraint loop (via SHACL) before compiling the intent into simulation code. All paths converge at the Judge for final forensic verification.

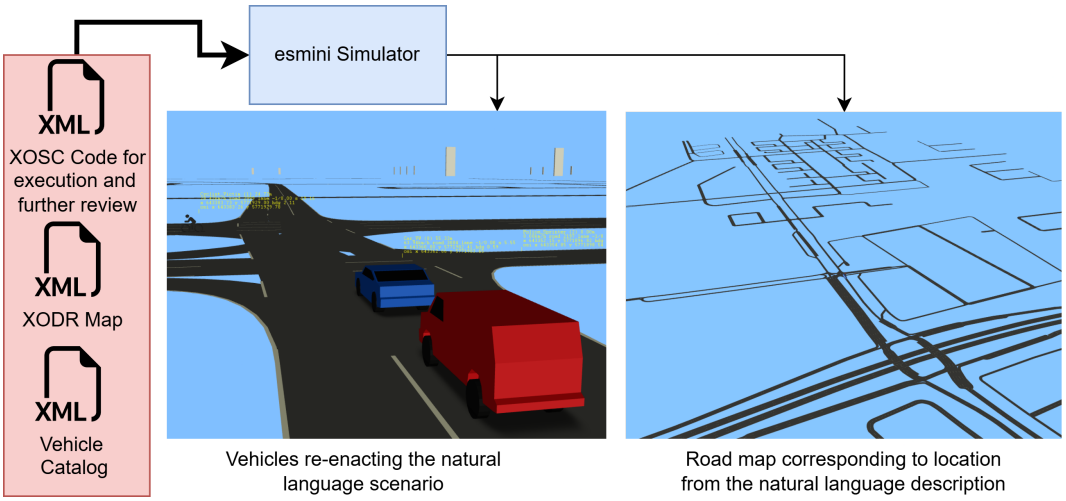


Figure 5 Overview of esmini being utilized to simulate a vehicular scenario. The `.xosc` is passed as a "story" of what happened in the scenario, in which way and by what actor; the `.xosc` vehicle catalog details the actors involved and the `.xodr` file represents the bounding-box map of where the incident took place, reconstructed in XML formatting.

1. **Direct syntactic synthesis:** The LLM attempts to generate raw ASAM OpenSCENARIO (`.xosc`) XML. The output is fed immediately into **esmini** in headless mode. If parsing fails (e.g., missing parameter definitions), the error log is fed back to the LLM for self-correction.
2. **Programmatic synthesis using Python:** The LLM generates a Python script utilizing the `scenariogeneration` library. This acts as an IR of a scenario. The script is executed in a sandboxed environment; if the Python interpreter throws an error (e.g., `AttributeError`), the traceback is returned to the LLM to repair the API usage.
3. **Ontology-driven synthesis:** To accommodate the difference between unstructured text and rigid syntax, we employ a semantic intermediate layer adapted from the framework by Bogdoll et al. [4], a choice made based on analyzing the ontology framework survey from Zipfl et al. [28] and selecting a framework that would match the needs of our generative pipeline without being overly detailed. We distilled a lightweight JSON-LD (JSON for linked data) profile focusing on core classes (`Actor`, `Event`, `StartPose`). This mode uses a deterministic SHACL [19] (Shapes Constraint Language) data consistency validation loop:
   - *Generation:* The LLM outputs a JSON-LD instance representing the semantic graph of the crash.
   - *Validation:* The instance is validated against a SHACL schema (`shapes.ttl`) to ensure logical consistency (e.g., verifying that all actors referenced in events exist in the actor list).
   - *Feedback:* Specific violations are injected into the next prompt. Only upon passing semantic validation is the graph passed to the heuristic translation layer for compilation.

4.5 The heuristic translation layer

A significant component of the ontology mode for converting from the ontology instantiation to an XOSC is the decoupling of semantic intent from syntactic implementation. Once a valid JSON-LD instance is produced, HASCO employs a deterministic compiler to convert

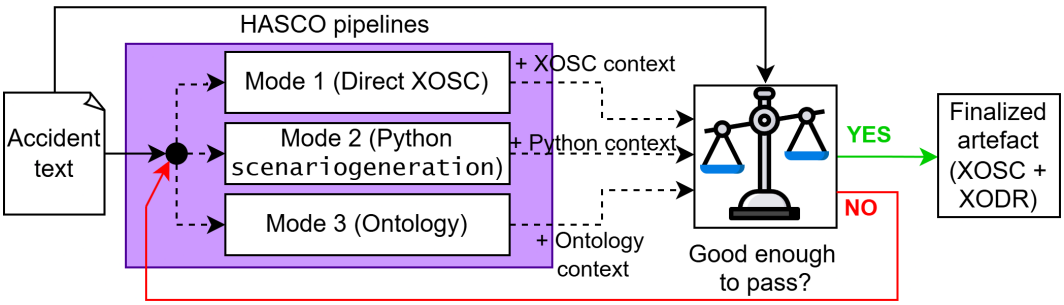


Figure 6 Forensic judge procedure in HASCO.

the high-level event graph into a concrete OpenSCENARIO XML file. This layer handles the low-level syntactic engineering that LLMs typically struggle with.

The translation layer parses the sequence of Event nodes for each actor. Unlike direct generation, which must output frame-by-frame coordinates, our compiler reconstructs a continuous polyline trajectory from sparse keyframes. It extracts the StartPose ($t = 0$) and subsequent event timestamps (t_n). If an event implies a lane change (inferred via regex matching of laneId tokens in the event summary), the compiler automatically injects intermediate waypoints to smooth the lateral transition, ensuring kinematic feasibility without requiring the LLM to calculate vector math.

To translate natural language summaries into executable triggers, the layer utilizes a keyword-based heuristic classifier:

- **Stop Logic:** Events containing tokens such as *"stop"*, *"stationary"*, or *"halt"* are automatically mapped to an OpenSCENARIO SpeedAction with a target speed of 0.0.
- **Crash Inference:** Events containing *"collision"*, *"impact"*, or *"rear-end"* (provided they lack disqualifiers like *"near-miss"*) are treated as stop triggers.

This compiler ensures that the complex ConditionGroup and SimulationTimeCondition XML blocks required to synchronize these actions across multiple actors are mathematically consistent, effectively functioning as a "type system" for physical interactions.

4.6 The forensic judge (validation loop)

A critical innovation of HASCO is the Judge loop, which operates as a self-healing feedback mechanism to enforce both runtime robustness and forensic fidelity. The process is visualized in Figure 6.

Unlike "fire-and-forget" generators, this component employs a two-stage validation process:

1. **Syntax and runtime repair:** Before the scenario is semantically evaluated, HASCO attempts to compile the generated artifact (Python or Ontology) into a `.xosc` file. If the generation process throws a runtime exception or violates static guardrails (e.g., using invalid enums), the error trace is immediately fed back to the synthesis model for repair. This ensures that the system recovers from brittle syntax errors before they cascade into downstream failures.
2. **Semantic and functional alignment:** Once a valid `.xosc` artifact is produced, it is parsed into a structured event log (e.g., *"Actor A: ChangeLane at $t = 5s$"*). The Judge LLM compares this log against the original police report. Crucially, this step detects silent failures, scenarios where the code executes successfully but produces an empty or inert simulation (e.g., actors spawning but never moving). If the Judge detects a

Table 1 Executability Success Rates ($N = 40$).

Synthesis Strategy	No Judge (%)	Judge (%)	Judge Impact (%)
Direct XOSC (Mode 1)	5.0	7.5	**+2.5**
Intermediate Python (Mode 2)	80.0	90.0	**+10.0**
Ontology-driven (Mode 3)	92.5	95.0	**+2.5**

misalignment or a functional void, it generates a natural language critique (e.g., *"The simulation is inert; the report describes a collision, but no trigger was activated."*). This critique acts as a semantic constraint for the next synthesis iteration, often converting a "technically valid but broken" scenario into a fully executable one.

5 Evaluation methodology

We evaluated HASCO on a diverse "stress test" dataset of $N = 40$ unstructured accident reports collected from open-web sources. Unlike curated datasets (e.g., NMVCCS) which provide structured metadata, our corpus consists of raw, unstructured text scraped from news articles, legal blogs, and public police logs. This dataset was specifically curated to maximize entropy and test the system's contextual imputation capabilities.

- **Linguistic Diversity:** The corpus includes reports in English, German, Swedish, Dutch, French, Japanese, Spanish and Bosnian, requiring the semantic pipeline to perform cross-lingual extraction and reasoning.
- **Information Sparsity:** Reports vary from hyper-detailed legal texts to vague news summaries (e.g., "A car hit a pedestrian near the station"). This forces the system to impute missing parameters (e.g., vehicle dimensions, exact speeds) based on context rather than explicit retrieval.

Representative examples of input reports and their corresponding generated artifacts are provided in Appendix A/B.

5.1 Quantitative analysis: executability

We first analyzed binary executability (pass/fail) and failure causes to quantify the validity of the overall procedure and measure the Judge's contribution. Table 1 presents the success rates across the three synthesis strategies. The results strongly validate the "Intermediate Representation" hypothesis.

Direct LLM generation (Mode 1) proved unreliable for complex scene engineering, suffering a 95% failure rate. The verbose nature of OpenSCENARIO XML led to frequent "long-dependency" errors, where the model failed to synchronize coordinate systems across nested `ManeuverGroup` tags. In contrast, the Ontology-guided compiler (Mode 3) achieved 95% validity. By offloading the syntactic complexity to the deterministic heuristic translation layer, the LLM only needed to produce a valid JSON graph. The remaining 5% of failures were attributed to external geocoding timeouts (Nominatim API) rather than generation logic errors; resolving these infrastructure issues could potentially raise this method to near 100% executability. Two notable examples of outright failures across the board are the report in Japanese (which failed to geocode in all pipelines, owing to the language barrier for the Nominatim OSM pipeline) and another report in Spanish (comprising of a complex textual report) which failed compilation across the board due to failures to extract the location, mentioned vehicles and the events that occurred.

The addition of a Judge Loop improved executability modestly across modes 1 and 3, but the effect was most pronounced in the Python mode (+10.0%). This gain is attributed to the Judge's ability to detect silent failures, scenarios where the generated Python script executes successfully (i.e., no runtime exception) but produces a functionally inert or empty OpenSCENARIO file. By critiquing these empty outputs (e.g., *"The scenario duration is 0s"*), the Judge essentially acts as a **semantic linter**, forcing the model to correct the underlying logic during the retry phase and converting invalid outputs into executable artifacts.

5.2 Qualitative analysis: semantic fidelity

To quantify the performance of HASCO beyond a binary pass/fail, we adopted a 4-point Likert scale (0–3) evaluating the simulation's utility. This rubric classifies scenarios based on the intervention required to make them usable:

- **Bin 0: Syntax failure (indication of compilation error).** The pipeline failed to produce a valid, runnable `.xosc` file. We categorize these failures into two distinct groups: *infrastructure errors* (e.g., geocoding API timeouts) and *generative failures*. The latter reveals specific limitations in the LLM's ability to generate rigid XML structures, which we classify as a "Failure mode catalogue" for future research:

  Structural malformation (broken XML format): The LLM fails to generate a well-formed XML tree, often missing closing tags or root definitions due to context window limits. For example, the parser returns "Couldn't find OpenSCENARIO element," indicating a truncated file. This may be potentially fixed by enforcing constrained decoding (grammar-based sampling) or increasing token limits for verbose XML output.

  Asset hallucination (catalog integration failures): The model generates syntactically valid code that references non-existent external assets, failing to ground the generation in the local file system. One example of this is a log showing "Couldn't locate catalog file: VehicleCatalog," because the model hallucinated a standard library path. A potential fix is to explicitly inject available asset paths and catalog entry names into the system prompt context.

  Simulator incompatibility (semantic violations): The model generates valid standard OpenSCENARIO, but utilizes conditions or actions not supported by the specific target simulator (e.g., `esmini`). One common error of this type is "Exception: Unsupported condition: SimulationTimeCondition", occurring because the engine does not implement that specific standard feature. The model may be fine-tuned or prompt-engineered specifically on the target simulator's documentation rather than the general ASAM standard.

  Convergence failure: Cases where the self-correction loop exhausted all attempts without reaching a valid state. A common instance is that the pipeline times out after the maximal number of retries, unable to resolve a complex dependency chain. One way of remediating this may be to decompose complex scenarios into smaller, modular generation steps rather than attempting single-shot generation.
- **Bin 1: Semantic failure (narrative/physics hallucination).** The scenario executes, but the core causal event is missing or fundamentally incorrect. For example, the report describes a T-bone collision at an intersection, but the simulation depicts two vehicles passing on a highway without contact.
- **Bin 2: Draft quality (necessitates a revision by a Human-in-the-Loop).** The scenario captures the correct core event, but exhibits kinematic artifacts or asset mismatches. For example, the collision occurs as described, but the target vehicle "teleports"

(jerky movement) or the vehicle type is generic (e.g., a sedan instead of a bus). These files are usable but require manual "polishing", either via a GUI tool or by manually editing the generated XML.

- **Bin 3: Production quality (One-Shot).** The scenario is semantically faithful, kinematically smooth, and requires no or very little human intervention. Turns taken, events and timing align with the report; the collision physics are plausible, with potentially minor manual expert updates.

The results are visible in Table 2, and the distribution is visualized in Figure 7. These results give us key insights into the qualitative benefits of various HASCO modalities. To begin with, we may observe that Mode 1, both with Judge and without Judge, despite initially having non-zero executability, displays no possibility of achieving syntax compatibility with OpenSCENARIO. Those rare several instances where Mode 1 executes result in a static environment only determined by the `.xodr` map file within esmini. Therefore, the entire generative corpus of Mode 1 within HASCO may be discarded (bin 0), with the conclusion that recreating vehicular scenarios by way of generating direct `.xosc` files is not a task achievable without great difficulty. Modes 2 and 3 provide much more interesting nontrivial results.

Half of all the scenarios in Python mode without Judge bin as either Draft (42.5%) or Production quality (7.5%). Generation quality rises when the Judge is added into the pipeline, with Draft bin encompassing **32.5%** of cases and Production bin encompassing **30%** of total cases. Therefore, with Judge, a total of 62.5% cases are of "acceptable" Draft or Production quality. As per Figure 8 on the left, this indicates that adding the Judge significantly increases the quality of produced scenarios via Python intermediate representation, with a 300% relative increase in Production quality scenarios when compared to the non-Judge mode.

With regards to the Ontology mode, the non-Judge version initially bins 37.5% of all scenarios as either Draft (17.5%) or Production quality (20%), a result worse than Python, but shows a significant uptick when Judge is introduced, with acceptable case numbers going to 57.5% of either Draft (35%) or Production quality (22.5%). As visualized on Figure 8 on the right, the Judge reduces the amount of scenarios that bin as semantic failures by 36.3%, a significant drop.

Table 2 Qualitative rubric results of HASCO across different operating modalities ($N = 40$).

Synthesis Strategy	Bin 0 (Syntax failure)	Bin 1 (Semantic failure)	Bin 2 (Draft quality)	Bin 3 (Production quality)
XOSC - Judge	40	0	0	0
XOSC - NO Judge	40	0	0	0
Python - Judge	6	9	13	12
Python - NO Judge	9	11	17	3
Ontology - Judge	3	14	14	9
Ontology - NO Judge	3	22	7	8

5.3 Cost-benefit analysis

We tracked token consumption to determine the cost-fidelity trade-off (Figure 9). Here, the quality score S_{avg} is a weighted mean of the rubric bins ($S_{avg} = \frac{1}{N}\sum w_i n_i$, with weights $w \in \{0, 1, 2, 3\}$ corresponding to syntax failure, semantic failure, draft, and production quality).

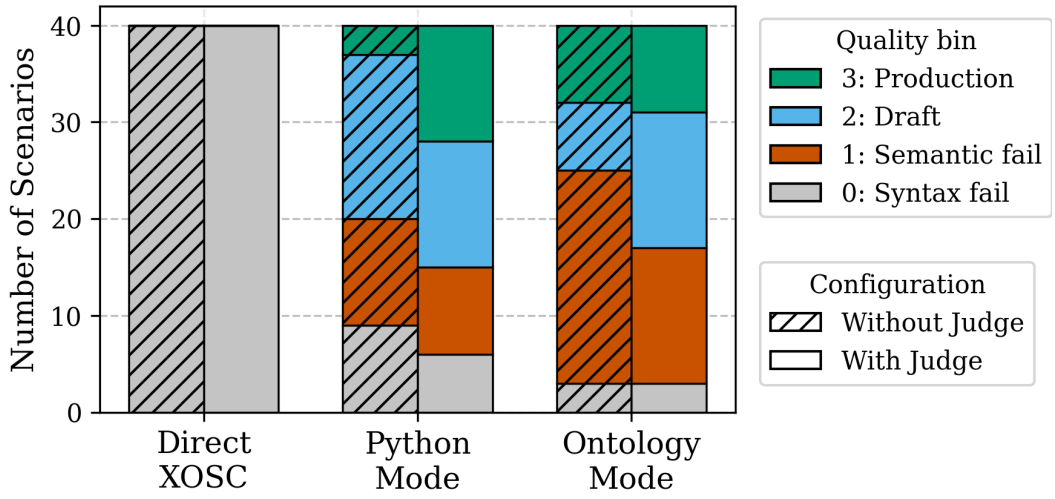


Figure 7 Qualitative distribution of generated scenarios ($N = 40$). In Python Mode, the Judge loop successfully converts Draft quality (Bin 2) scenarios into Production quality (Bin 3), tripling the high-fidelity yield. In Ontology Mode, the Judge reduces Semantic failures (Bin 1), shifting the distribution toward valid execution. Direct XOSC remains dominated by Syntax failures (Bin 0) regardless of validation.

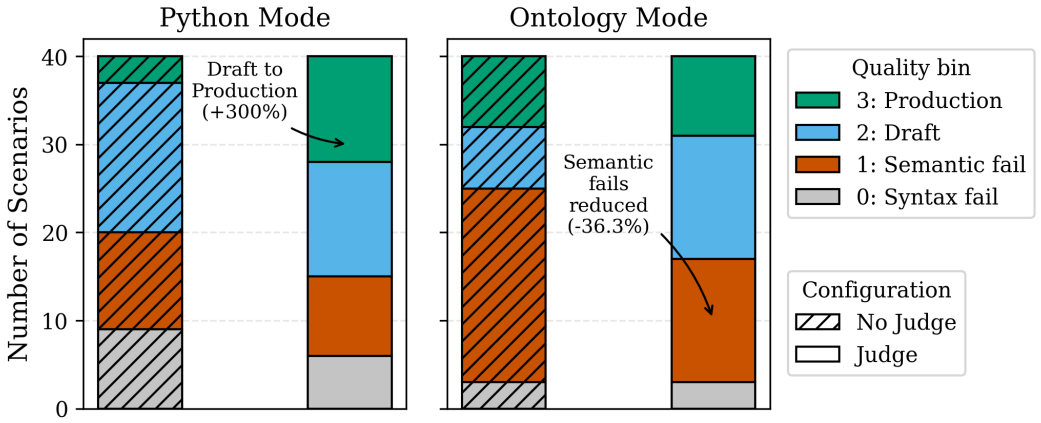


Figure 8 Qualitative impact of the Judge Loop on scenario fidelity ($N = 40$). The stacked bars illustrate the shift in quality distribution when the Judge loop is enabled. Left (Python mode): The Judge primarily acts as a physics refiner and converts technically functional "Draft" scenarios into "Production" quality (a 300% increase) by correcting simulation parameters. Right (Ontology mode): The Judge significantly reduces Semantic failures (-36.3%).

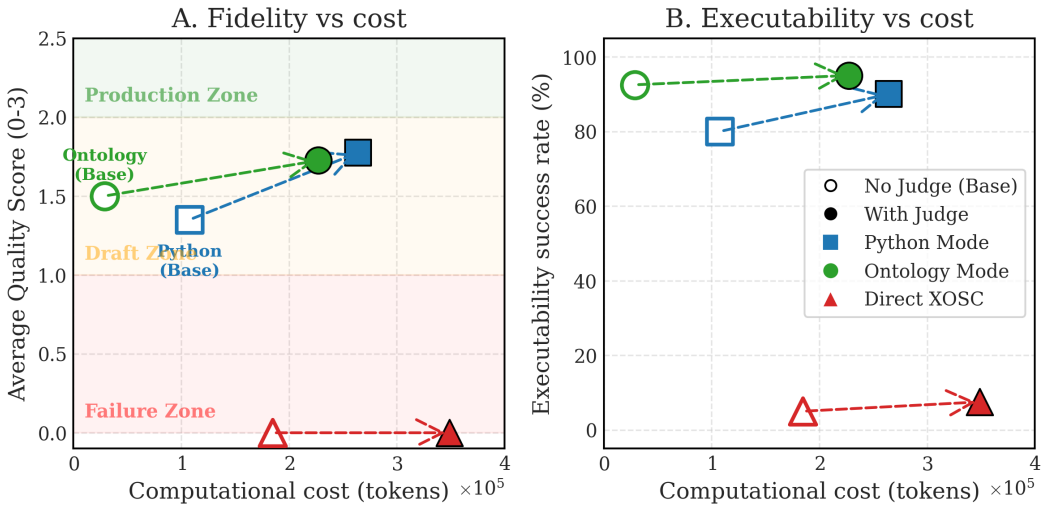


Figure 9 Cost-benefit analysis of synthesis strategies; the X-axis represents average token consumption. On (A) the Y-axis denotes the average quality score (S_{avg}). Ontology mode (Green) achieves the most favorable trade-off, delivering functional scenarios at minimal cost. On (B) the Y-axis denotes the executability rate.

Figure 9(A) shows Ontology mode achieving the most favorable trade-off. Figure 9(B) illustrates the executability rate, where Ontology mode again excels. The Judge loop provides relatively modest improvements for both Python and Ontology modes given the outsized $\approx 2.5\times$ token cost and $\approx 3\times$ latency increases (averaging 180 s per run vs. 60 s). Thus, for large-scale generation, Mode 3 (Ontology, No Judge) is the most cost-effective configuration, reserving the expensive Judge loop for complex edge cases (e.g., scenarios that execute but remain semantically inert).

6 Threats to validity

Subjectivity in evaluation. While the executability metric (bin 0) is deterministic, the semantic fidelity metric (bins 1–3) currently relies on human expert review and the "Judge" LLM's assessment. Both introduce subjectivity. The Judge LLM itself may hallucinate "alignment" (reporting OK when the simulation is flawed) or fail to catch subtle kinematic errors (e.g., impossible stopping distances). In cases of ambiguous reports, determining whether an imputed parameter (e.g., the specific lane index) is "correct" is inherently subjective. Future work will integrate deterministic safety metrics (e.g., responsibility and sensitive safety aware checks) to mathematically validate that the synthesized maneuvers align with physical crash causality, reducing reliance on qualitative grading.

Non-determinism in LLM outputs. LLMs utilize non-deterministic token sampling governed by a *temperature* parameter. Higher temperatures increase randomness, while a temperature of zero forces deterministic, greedy decoding. However, as the model we utilized did not support modifying the temperature parameter, multiple runs across the same dataset will be executed as part of future research. Additionally, all experiments were conducted using a single LLM (gpt-5-mini). While the pipeline is model-agnostic by design, we did not evaluate whether performance characteristics generalize across model families (e.g., Claude, Gemini,

open-weight models). A cross-model study is planned as part of future work to quantify how much of the observed executability improvement is attributable to the architectural decoupling versus the specific model's code-generation capability.

Imputation vs. accuracy trade-off. The use of unstructured data necessitates aggressive imputation. While this demonstrates the system's robustness, it creates a validity threat regarding ground truth. When HASCO encounters a report omitting specific details (e.g., "a truck" without a model year), it utilizes the vehicle extraction module to infer a "most likely" candidate (e.g., Generic Heavy Truck). While this allows for valid simulation, the resulting scenario is a probabilistic reconstruction rather than a forensic reproduction. Users must treat HASCO outputs as representative edge cases rather than exact digital twins of the specific historical event.

External dependencies. The system's fidelity is bound by the quality of external APIs. A valid scenario generation can still fail if the Nominatim geocoder fails to resolve a vague location string, the OpenStreetMap data for a region is incomplete or lacks lane-level detail or simply the LLM generating request times out. These external failures account for the majority of the errors in our Ontology and Python modes and some of the errors in Direct XOSC mode. This underlines the need for offline fallback mechanisms in production environments.

7 Discussion

Our findings validate the "Code Co-Generation" paradigm by directly addressing our research questions through the data presented in Section 5. Regarding **RQ1**, Table 1 demonstrates that direct "one-shot" generation of verbose OpenSCENARIO XML is highly unreliable (95% failure rate). However, automated synthesis becomes highly feasible when utilizing an intermediate representation. By decoupling semantic intent from syntactic formatting – using intermediate Python scripts or JSON-LD ontologies, the LLM is relieved of complex XML synchronization. This allows the pipeline to achieve up to 95% executability. Therefore, we confirm that LLM-based automation is viable for unstructured text if the architectural abstraction is managed properly. Addressing **RQ2**, the integration of the Judge loop fundamentally shifts the pipeline from a fragile "fire-and-forget" system to a robust, self-healing process. As shown in Figure 8, the Judge significantly elevates semantic fidelity. In Ontology mode, it acted as a logical linter, reducing semantic mismatches by 36.3%. In Python mode, it refined physical parameters to promote scenarios from "Draft" to "Production" quality by 300%. Furthermore, it improved raw executability by catching and repairing "silent failures," proving that iterative feedback is essential for bridging the specification gap.

8 Conclusion and future work

In this work, we presented HASCO, a Human-in-the-Loop compiler designed to bridge the gap between unstructured forensic accident reports and executable OpenSCENARIO simulations. By treating scenario generation as a multi-strategy compilation task rather than a purely generative one, we demonstrated that utilizing Intermediate Representations (such as Python scripts and JSON-LD ontologies) alongside an iterative Judge feedback loop resolves the inherent brittleness of LLM-generated XML. This approach drastically reduces the manual engineering burden of Scenario-Based Testing while maintaining semantic fidelity.

Future work will focus on integrating constraint-based motion planning to enhance the kinematic realism of the heuristic translation layer, as well as introducing deterministic safety metrics to automatically quantitatively evaluate and verify the physical causality of the synthesized collisions.

References

1 ASAM e.V. ASAM OpenSCENARIO®: Standard for Dynamic Content in Driving Simulation, 2022. Version 1.x (XML format). URL: `https://www.asam.net/standards/detail/openscenario/`.

2 Amy Aukema, Kate Berman, Travis Gaydos, Ted Sienknecht, Chou-Lin Chen, Chris Wiacek, Tim Czapp, and Schuyler St. Real-world effectiveness of model year 2015-2020 advanced driver assistance systems. In *27th International Technical Conference on the Enhanced Safety of Vehicles (ESV) National Highway Traffic Safety Administration.*, 2023.

3 Gerrit Bagschik, Till Menzel, and Markus Maurer. Ontology based scene creation for the development of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1813–1820. IEEE, 06 2018. `doi:10.1109/IVS.2018.8500632`.

4 Daniel Bogdoll, Stefani Guneshka, and J Marius Zöllner. *One ontology to rule them all: Corner case scenarios for autonomous driving*, pages 409–425. Lecture Notes in Computer Science. Springer Nature Switzerland, 2023.

5 Yao Deng, Jiaohong Yao, Zhi Tu, Xi Zheng, Mengshi Zhang, and Tianyi Zhang. TARGET: Automated scenario generation from traffic rules for testing autonomous vehicles via validated LLM-guided knowledge extraction. *arXiv [cs.SE]*, 05 2023. `arXiv:2305.06018`.

6 Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017. URL: `https://proceedings.mlr.press/v78/dosovitskiy17a.html`.

7 Karim Elmaaroufi, Devan Shanker, Ana Cismaru, Marcell Vazquez-Chanlatte, Alberto Sangiovanni-Vincentelli, Matei Zaharia, and Sanjit A Seshia. ScenicNL: Generating probabilistic scenario programs from natural language. *arXiv [cs.SE]*, 2024. `arXiv:2405.03709`.

8 esmini development team. esmini: Basic OpenSCENARIO player and validator. `https://github.com/esmini/esmini`, 2025. Accessed: 2026-01-20.

9 Roland Galbas, Marcus Nolte, Ulrich Eberle, Hardi Hungar, Henning Mosebach, Nayel Fabian Salem, Helmut Schittenhelm, Jan Reich, Thomas Kirschbaum, and Lukas Westhofen. VV methods safety assurance position paper. resreport, Robert Bosch GmbH und BMW AG, 2024.

10 Yuan Gao, Mattia Piccinini, Korbinian Moller, Amr Alanwar, and Johannes Betz. From words to collisions: LLM-guided evaluation and adversarial generation of safety-critical driving scenarios. *arXiv [cs.AI]*, 2025. `arXiv:2502.02145`.

11 An Guo, Yuan Zhou, Haoxiang Tian, Chunrong Fang, Yunjian Sun, Weisong Sun, Xinyu Gao, Anh Tuan Luu, Yang Liu, and Zhenyu Chen. SoVAR: Build generalizable scenarios from accident reports for autonomous driving testing. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 268–280. ACM, 10 2024. `doi:10.1145/3691620.3695037`.

12 Pin Ji, Yang Feng, Zongtai Li, Xiangchi Zhou, Jia Liu, Jun Sun, and Zhihong Zhao. Txt2Sce: Scenario generation for autonomous driving system testing based on textual reports. *arXiv [cs.SE]*, 2025. `arXiv:2509.02150`.

13 Hoseon Kim, Jieun Ko, Cheol Oh, and Seoungbum Kim. Evaluation of autonomous driving safety by operational design domains (ODD) in mixed traffic. *Sustainability*, 16:9672, 11 2024.

14 Stefan Jacobs Madlen Haarbach. Lieferwagen stand auf Radweg: Radfahrerin stirbt bei Unfall in Berlin-Friedrichshain–Mahnwache am Freitag, May 2021. URL: https://www.tagesspiegel.de/berlin/polizei-justiz/radfahrerin-stirbt-bei-unfall-in-berlin-friedrichshain--mahnwache-am-freitag-6174754.html [Online; accessed 2026-04-16].

15 Till Menzel, Gerrit Bagschik, and Markus Maurer. Scenarios for development, test and validation of automated vehicles. *arXiv [cs.SE]*, 01 2018. arXiv:1801.08598.

16 Christian Neurohr, Lukas Westhofen, Tabea Henning, Thies de Graaff, Eike Mohlmann, and Eckard Bode. Fundamental considerations around scenario-based testing for automated driving. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 121–127. IEEE, 10 2020.

17 Ali Nouri, Beatriz Cabrero-Daniel, Zhennan Fei, Krishna Ronanki, Håkan Sivencrona, and Christian Berger. Large language models in code co-generation for safe autonomous vehicles. *arXiv [cs.SE]*, 05 2025. arXiv:2505.19658.

18 Sebastian Pagel, Flo, Michi Scholz, Muhammed Kerem Kahraman, Pranav Ashok, victorjarlow, cxruan, Egan, Eric Xue, boettol, Emil Knabe, Hüseyin Aydın, and m0uH. grepthat/libOpenDRIVE: v0.5.0.

19 Paolo Pareti and George Konstantinidis. *A review of SHACL: From data validation to schema reasoning for RDF graphs*, pages 115–144. Lecture Notes in Computer Science. Springer International Publishing, 2022.

20 pyoscx. scenariogeneration: A python library for generating openscenario and opendrive files, 2024. Accessed: 2025-02-11. URL: https://github.com/pyoscx/scenariogeneration.

21 Stefan Riedmaier, Thomas Ponn, Dieter Ludwig, Bernhard Schick, and Frank Diermeyer. Survey on scenario-based safety assessment of automated vehicles. *IEEE Access*, 8:87456–87477, 2020. doi:10.1109/ACCESS.2020.2993730.

22 Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. LGSVL simulator: A high fidelity simulator for autonomous driving. *arXiv [cs.RO]*, 2020. arXiv:2005.03778.

23 Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988. IEEE, 09 2015. doi:10.1109/ITSC.2015.164.

24 Walther Wachenfeld and Hermann Winner. *The release of autonomous vehicles*, pages 425–449. Springer Berlin Heidelberg, 2016.

25 Hermann Winner, Karsten Lemmer, Thomas Form, and Jens Mazzega. *PEGASUS—first steps for the safe introduction of automated driving*, pages 185–195. Springer International Publishing, 2019.

26 Yongqi Zhao, Wenbo Xiao, Tomislav Mihalj, Jia Hu, and Arno Eichberger. Chat2Scenario: Scenario extraction from dataset through utilization of large language model. In *2024 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 06 2024.

27 Ji Zhou, Yongqi Zhao, Yixian Hu, Hexuan Li, Zhengguo Gu, Nan Xu, and Arno Eichberger. Can AI generate more comprehensive test scenarios? review on automated driving systems test scenario generation methods. *arXiv [cs.SE]*, 12 2025. arXiv:2512.15422.

28 Maximilian Zipfl, Nina Koch, and J Marius Zöllner. A comprehensive review on ontologies for scenario-based testing in the context of autonomous driving. In *2023 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7. IEEE, 06 2023. doi:10.1109/IV55152.2023.10186681.

A Prompts and LLM Context

Listing 1 System prompt for the ontology mode.

```
You are HASCo's ontology specialist. Convert police/incident reports plus OpenDRIVE
↪ context into detailed JSON-LD instances that conform to the provided context/frame
↪ and can later be rendered into OpenSCENARIO.

When the user prompt includes "### Reference docs ... ### End docs", treat those
↪ snippets as authoritative.

OUTPUT RULES
- Respond with EXACTLY one JSON object (no markdown fences, no prose before/after).
- Copy the @context block exactly as provided in the user prompt and keep all
↪ existing prefixes.
- Ensure the instance passes SHACL: include actors, assets, start poses, and events
↪ as required.
- Every actor must have a unique hasco: identifier, a concrete ontology class
↪ (Vehicle, Pedestrian, Bicycle, etc.), a catalog asset, and a startPose with
↪ xodrRoadId/laneId/s, headingDeg, speedMps.
- Events describe concrete interactions: include subject, target, timeSec, and
↪ eventSummary text that narrates the collision/interaction.
- Use OpenDRIVE IDs from the map summary (lane signs follow the local traffic side).
↪ Never invent roadId=0 or omit lane polarity.
- For multi-vehicle crashes, instantiate each participant separately (e.g., ego car,
↪ struck vehicle, bus, pedestrian).
- Populate eventSummary with meaningful prose so later tools can recover intent
↪ ("Tanker rear-ends slowing sedan near junction J2").
- Add additional derived actors if the report implies them (e.g., parked truck,
↪ waiting pedestrian, lead vehicle).
- Choose realistic speeds for the road type; if the report states someone stopped,
↪ set speedMps ≈ 0 and describe the reason in eventSummary.
- Prefer chronological ordering of events using timeSec.
- Never include TODOs or placeholders. If uncertain, pick the most defensible option
↪ and explain the assumption briefly in eventSummary.
```

Listing 2 System prompt for the direct XOSC mode.

```
You are HASCo's OpenSCENARIO architect. Convert report text and map summaries
↪ directly into complete .xosc XML files.

OUTPUT RULES
- Respond with EXACTLY one XML document (no markdown fences, comments, or
↪ explanations).
- Use the literal map/catalogn paths provided in the user prompt without modification.
- Instantiate every actor implied in the report using <Entities><ScenarioObject> with
↪ CatalogReference entries from the provided catalog list.
- Include a rich <Storyboard> structure with Init → Story → Act → ManeuverGroup →
↪ Event(s), plus <StartTrigger> and <StopTrigger>.
- Actions must be deterministic: use SpeedAction, LaneChangeAction,
↪ FollowRelativeDistance, CollisionCondition, etc. to choreograph the described
↪ crash/interaction.
- Ground LanePosition/WorldPosition coordinates in the map context; never invent
↪ roadId=0 or impossible laneIds.
- Respect traffic-side semantics: negative laneIds for right-hand driving, positive
↪ for left-hand driving.
- Always stop or park every actor after the event concludes; include a
↪ simulation-wide StopTrigger.
- Reference catalog entries exactly; do not fabricate new vehicle definitions.
- When uncertain, choose the most believable interpretation and proceed -- do not
↪ omit the participant.
```

Listing 3 System prompt for the Python mode.

```
You are an assistant specialized in generating OpenSCENARIO (.xosc) traffic scenarios
↪ in Python using the
scenariogeneration library (xosc/xodr).

You will receive a user request, and sometimes an inline section titled:
### Reference docs
...retrieved API snippets here...
### End docs

When present, treat the Reference docs as authoritative and prefer those APIs.

REQUIREMENTS
- Output: RETURN ONLY Python code (no markdown, no text before/after).
- Use scenariogeneration.xosc/xodr (and standard library) only.
- Load the map path provided in the user prompt (see "Map path") and do NOT build
↪ your own road graph:
    road = xosc.RoadNetwork(roadfile="<MAP PATH FROM PROMPT>", scenegraph="")
- Register the provided VehicleCatalog directory verbatim (e.g.,
↪ scenario.add_catalog("VehicleCatalog", "../catalogs")); never rewrite the path or
↪ point at the .xosc file directly.
- Include: at least one vehicle actor, an Init/setup, at least one
↪ Maneuver/Event/Action,
  and a simple Story/Act structure that reproduces the described scenario.
- When the report involves multiple vehicles, model each distinct participant (3+ if
↪ specified) instead of collapsing the crash to two actors; use CatalogReference
↪ entries for all of them.
- Avoid placeholders/dummies. Choose reasonable defaults if missing.
- Add light inline comments where helpful. When you reference an API, add a marker
↪ like <SpeedAction> or <LaneChangeAction> in a comment.
- No I/O except reading the .xodr via RoadNetwork and (optionally) writing the .xosc
↪ file.
- No external network calls, no shelling out.
- Start by inferring how every involved vehicle moves relative to the others
↪ (head-on, rear-end, crossing paths, same-direction merge, etc.) and pick
↪ roadId/laneId/orientations that physically match the report before adding Actions.
- If uncertain about specifics, make sensible assumptions and proceed (do NOT ask
↪ questions).
- If the report indicates a crash/collision/impact, YOU MUST choreograph a
↪ deterministic impact:
  * Drive the storyline through: approach (accelerate/close gap), impact
↪ (LaneChange/RelativeDistance trigger <=0.5 m or CollisionCondition), and aftermath
↪ (stop/park both entities).
  * Align geometry with the stated crash type: head-on crashes require opposing
↪ headings on the same road, rear-end crashes require a faster follower in the same
↪ lane, intersection/side impacts require perpendicular or turning approaches, etc.
  * Do not leave the crash chance-based -- script the actions/timing so contact
↪ always occurs within the Story.
- Use Position objects (WorldPosition/LanePosition/RelativeObjectPosition). No
↪ tuples/strings for positions.
- Do NOT use strings for enums (use xosc.Rule.lessThan / greaterThan / equalTo).
- Do NOT subclass any ScenarioGenerator; build a plain xosc.OpenScenario.
- Do NOT redefine vehicle characteristics; load the provided vehicle catalog using
↪ the exact directory listed in the prompt's catalog header before creating Entities:
    scenario = xosc.Scenario('PoliceReportScene', '1.0', road, entities, story)
    scenario.add_catalog('VehicleCatalog', '<CATALOG DIR FROM PROMPT>')
# Valid entries are enumerated in the catalog header section of the user prompt.

...
```

Listing 4 Judge system prompt.

```
You are HASCo's alignment judge for OpenSCENARIO and ontology outputs.

General rules:
- Your only job is to compare the incident report + map context to the candidate
↪ scene the user supplies.
- Always follow the user's immediate instructions for response format (e.g., reply
↪ exactly "OK", return critique text, emit corrected JSON-LD, or return revised
↪ OpenSCENARIO XML). Do not introduce any other format.
- Never emit Python or scenariogeneration code. You are validating completed scenes,
↪ not writing scripts.
- Keep feedback concise, factual, and actionable. When returning corrected data,
↪ output ONLY the requested JSON/XML with no markdown fences or commentary.
```

Listing 5 Geospatial pipeline system prompt.

```
    You are a natural-language geolocation assistant.

Task: Extract the most probable traffic location from a police/incident description
↪ and return a single, human-readable query string suitable for
↪ OpenStreetMap/Nominatim search.

OUTPUT RULES (STRICT)
- Return EXACTLY one line of text with no line breaks.
- Include the most specific intersection or segment if clear (e.g., "Sveavägen &
↪ Tegnérgatan, Stockholm, Sweden"), else a single street or area plus city and
↪ country.
- Include neighborhood/municipality if it disambiguates.
- No coordinates, no postal codes.
- No explanations, no metadata, no JSON/dicts, no notes.
- If you cannot identify a plausible location, return an empty string.
- If a junction of two streets is mentioned, only use one of the streets.

GOOD EXAMPLES
- Turreff Avenue, Donnington, Telford and Wrekin, England, United Kingdom
- Sveavägen, Stockholm, Sweden
- A1, Denton Burn, Newcastle, United Kingdom
- E18, Viksäng, Västerås, Sweden
```

B Worked example: cyclist–truck collision (German report)

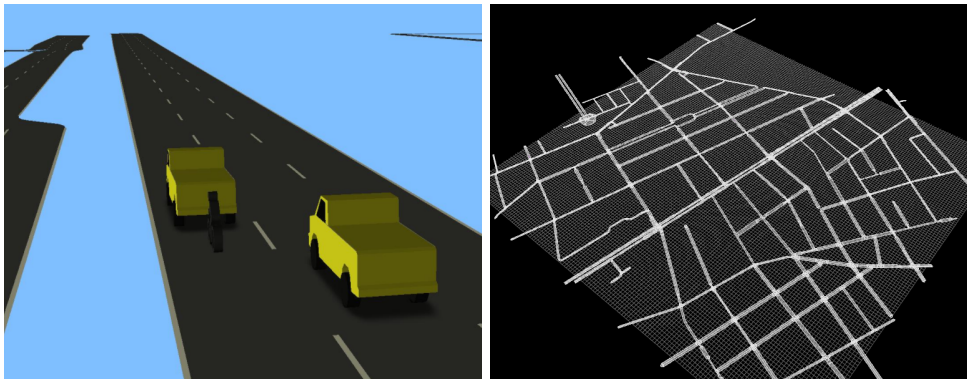
This appendix illustrates HASCO end-to-end on a real German-language report describing a cyclist-truck collision near a complex multi-lane intersection. The German source [14] is passed to the pipeline without translation, and the English rendering is provided for reader convenience only.

Listing 6 Input (original, German).

```
Update:  Lieferwagen stand auf Radweg: Radfahrerin stirbt bei Unfall in
↪ Berlin-Friedrichshain - Mahnwache am Freitag
Eine Radlerin ist auf der Frankfurter Allee von einem Lkw tödlich verletzt worden.
↪ Sie war einem Lieferwagen ausgewichen, der auf dem Pop-up-Radweg stand. Eine
↪ Radfahrerin ist bei einem Unfall auf der Frankfurter Allee in
↪ Berlin-Friedrichshain tödlich verletzt worden. Nach vorläufigen Informationen
↪ musste die Radfahrerin einem auf dem Radstreifen geparkten Lieferwagen ausweichen.
↪ Dabei wurde sie von einem von hinten kommenden Sattelzug offenbar gerammt und
↪ tödlich verletzt.Ein Feuerwehrsprecher sagte, drei Menschen, die in der Nähe
↪ waren, hätten einen Schock erlitten. Sie würden von Rettungskräften betreut. Zu
↪ Details des Unfalls äußerten sich zunächst weder die Pressestelle der Polizei noch
↪ die Einsatzleitung am Unfallort....
```

Table 3 HASCO outputs on the example report across all six configurations.

Mode	Judge	Bin	Reconstructed scene
Ontology	No	3	Cyclist and articulated truck on the left of a three-lane road exiting the intersection; truck clips the cyclist, advances, and stops.
Ontology	Yes	3	Cyclist on the right of a multi-lane road near the intersection, with a parked delivery van ahead and an articulated truck in the middle lane; cyclist strikes the stationary van as the truck moves up.
Python	No	2	Three-lane road with bystanders off-road; articulated truck in the middle lane, cyclist and delivery van in the rightmost lane; cyclist strikes van, truck continues briskly.
Python	Yes	2	Delivery van in the leftmost lane, cyclist behind it, articulated truck in the center; cyclist collides with van and continues forward, van halts, truck decelerates to a stop.
Direct XOSC	–	0	No artifact produced.



(a) esmini reconstruction (Python + Judge). **(b)** OpenDRIVE map generated using libOpenDRIVE [18].

Figure 10 Rendered output for the worked example.

Listing 7 Input (English translation).

```
Update: Delivery Van Parked on Bike Lane: Cyclist Dies in Accident in
↪ Berlin-Friedrichshain - Vigil on Friday
A cyclist was fatally injured by a truck on Frankfurter Allee. She had swerved to
↪ avoid a delivery van that was parked on the pop-up bike lane. A cyclist was
↪ fatally injured in an accident on Frankfurter Allee in Berlin-Friedrichshain.
↪ According to preliminary information, the cyclist had to swerve to avoid a
↪ delivery van parked on the bike lane. In doing so, she was apparently struck by a
↪ semi-truck approaching from behind and fatally injured. A fire department
↪ spokesperson said three people who were nearby suffered shock. They are being
↪ treated by emergency responders. Neither the police press office nor the incident
↪ command at the scene provided any details about the accident at this time...
```

Table 3 summarizes the bin assigned to each of the six configurations. Figure 10 shows the Ontology-with-Judge reconstruction and the generated OpenDRIVE map. Direct XOSC synthesis failed in both variants, consistent with its 95% corpus-wide failure rate. The Python mode introduced a delivery van absent from the source narrative and mislocated the collision target, which was an actor-hallucination failure mode undetected by the Judge because the resulting simulation is internally consistent. The Ontology mode with Judge produced the most faithful reconstruction (Bin 3): the SHACL validation step constrained the admissible actor set to those extracted from the report, preventing the hallucination observed in Python mode.