2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Traffic Scenario Orchestration from Language via Constraint Satisfaction

Frieda Rong[1], Chris Zhang[1,2], Kelvin Wong[1,2], and Raquel Urtasun[1,2]

***Abstract*— Autonomous vehicles (AVs) require extensive testing in simulation, but test case generation for driving scenarios is laborious. The desired scenarios are often out-of-distribution and have precise requirements on interactions with the AV policy under test. Manually programming scenarios allows for precise controllability but is difficult to scale. On the other hand, statistical models can leverage compute and data, but struggle with precise controllability when out-of-distribution. We cast scenario orchestration as a constraint-solving problem and present a language-in, simulation-out scenario orchestrator for closed-loop testing AVs. Our approach leverages foundation model reasoning to translate general, natural language descriptions into a set of constraints as a scenario representation. This then allows us to leverage off the shelf solvers to solve for actor behaviors which meet precise testing intentions in closed-loop. Under a benchmark of carefully crafted and diverse scenario descriptions, our approach greatly outperforms our baselines in orchestration success rate. We further show that our closed-loop approach is especially important for scenarios which require ego-reactive specifications.**

I. INTRODUCTION

Simulation for testing has become an industry standard for autonomous vehicle (AV) development. By creating scenarios in virtual environments, simulation offers a safe, efficient and precise way to test and verify AV behavior. A key aspect of simulation is the *scenario orchestrator*, the mechanism by which the simulation engine places and controls agents in the scene, in order to fulfill some user specified testing goal.

Orchestration is a difficult and multifaceted task. Consider the simple goal of testing the AV's reaction to a cut-in with some particular time-to-collision (TTC). First and foremost, the orchestrator must be *controllable*, in that there must exist an interface for the user to be able to specify the test intention. In this example, the intention involves an *interaction* between an actor and the AV, so the orchestrator must also be *reactive* to the AV's behavior in closed-loop. The orchestrator must reason on a high-level when determining initial conditions of the actors in order to ensure the test is feasible (e.g. traveling in the same direction, adjacent lanes, etc.). It must also ensure that the low-level trajectories are realistic and kinematically feasible. Finally, this is only one possible test intention; the orchestrator should be generalizable to support a wide diversity of potential testing scenarios, many of which may be out-of-distribution.

One approach towards meeting the aforementioned requirements is to specify scenarios using a domain-specific language (DSL), such as OpenScenario 2.0 [1]. Then, typically programmatic behavior models consisting of template maneuvers, triggers, etc. can be composed in order to control actors to meet the scenario intent. The main challenge with this structured and programmatic approach is its rigidity and lack of scalability. Maintaining a scenario framework or DSL and updating abstractions to meet new demands from the messy, unstructured real world is costly, and this approach scales neither with more data nor compute.

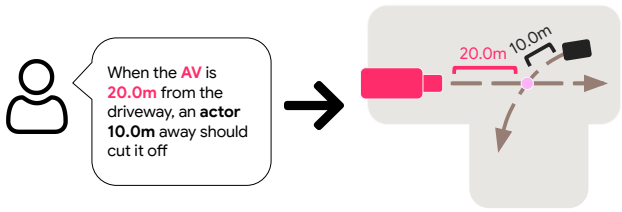

Fig. 1: We present a language-in, simulation-out framework for testing autonomous vehicles. We leverage LLMs in conjunction with SMT solvers to orchestrate scenarios that meet precise user test intentions.

Alternatively, recent work opt for more unstructured and learning based solutions towards orchestration. For instance, [2], [3] learns a natural language conditioned deep learning based initialization or actor model. While flexible, these approaches can struggle to orchestrate complex scenarios that require more reasoning, e.g. [3] cannot support more complex agent interactions like cut-in or overtake. More generally, we find that leaving reasoning entirely to an LLM with no tools or assistance can be inefficient and unreliable, as today's frontier models are still prone to making simple mistakes. Furthermore, the interface between the LLM and actor model can be overly lossy or coarse. For instance, specifying coarse spatial bins or relying on latent representations can make more precise requirements difficult to fulfill. Finally, there is an inherent tension where data driven methods are trained to produce in-distribution scenarios, but testing scenarios are often designed to be edge-case, or contain out-of-distribution commands.

In this work, we propose a *neurosymbolic* approach that combines the frontier commonsense reasoning capabilities of LLMs with the precise and controllable reasoning of classical artificial intelligence algorithms. In particular, we use LLMs in conjunction with Satisfiability Modulo Theory (SMT) solvers to orchestrate scenarios. The LLM is tasked to

Work done while at Waabi.
F.R. acknowledges support from an NSERC CGS-D fellowship.
[1]University of Toronto [2]Waabi

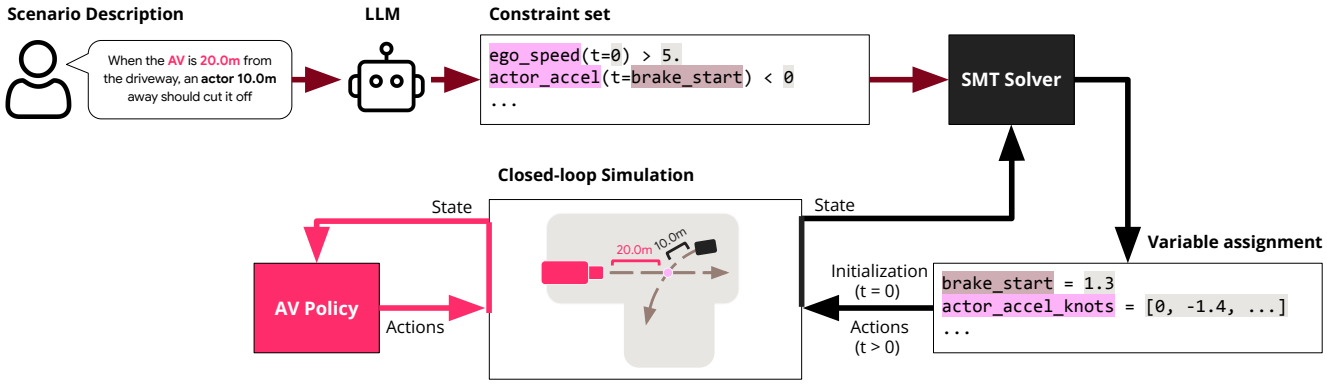

Fig. 2: Our overall pipeline. A user inputs a scenario description in natural language, which gets converted by an LLM into declarative constraints along with actor routes and symbolic motion profiles (not shown). The SMT solver runs in closed loop against the uncontrolled AV policy to search for satisfying assignments to the constraints which determine the concrete actor motion profiles that the simulation framework uses to simulate the scenario.

translate natural language descriptions to a set of constraints, where an off-the-shelf solver can be used to determine the free variable assignments.

Our approach has several advantageous properties. Firstly, the use of a powerful solver relieves the LLM of the need to reason about fine-grained interactions and tedious calculations while applying their strengths in high-level spatial and temporal reasoning. Conversely, we take advantage of the low-level functionality of SMT solvers while avoiding the more open-ended difficulty of formulating the constraint programming problem by using LLMs to produce the problem formulation. Our approach efficiently models actors as piece-wise polynomials over space and time, allowing for a joint modelling of both actor initialization and behaviors. It also allows for smooth and physically realizable trajectories, and for constraints to be specified in continuous time and space with arbitrary levels of precision. Finally, our approach is naturally reactive in closed-loop, as the SMT solver can easily be re-run at an arbitrary re-planning frequency to take into account new AV actions.

We evaluate our approach on a set of 80 diverse and challenging natural language scenario descriptions. Our experiments highlight our approach's efficacy for controllable, precise, and reactive scenario orchestration. Compared to a state-of-the-art learning-based baseline, our approach achieves a higher orchestration success rate across a range of scenario families, especially those involving multiple interactions. We also demonstrate the importance of our closed-loop approach for orchestrating scenarios with precise interactions between the ego and a hero actor.

II. RELATED WORK

A. Programmatic Scenario Generation

In the self-driving industry, simulation scenarios are commonly created by programming in domain specific languages [1], [4], [5], [6]. These are typically frameworks that have been designed with composable abstractions and primitives meant to aid in scenario description or generation. For instance, OpenScenario 2.0 [1] provides actions like `vehicle.drive()` that can be used to program more complex scenarios. Scenic [5] is a probabilistic programming language, allowing for programs to define a generative model over scenarios through sampling individual scenario parameters. Different approaches support a mixture of imperative or declarative descriptions of scenarios, and vary in how generation is implemented. In some cases, the language is meant to be purely descriptive, and generation methodology is left to the user [1]. In others, a rejection sampling approach is taken, generating scenes at random from the imperative descriptions until all declarative constraints are met [5]. Closest to our approach are methods which use solvers to find parameter assignments which satisfy all declarative constraints [7], [8], [9], [10].

We aim to address several challenges with programmatic scenario generation faces. While these frameworks offer controllability for expert designers, manually writing program specification is difficult to scale. Our work leverages advances in LLMs to support an auto-formalization of scenarios from natural language descriptions. Our constraint approach provides an efficient way to generate scenarios without relying on rejection sampling. Yet unlike existing constraint based approaches which are open-loop in nature [7], [10], our method takes a closed-loop approach where agents adapt to the actions of an external uncontrolled ego vehicle.

B. LLM-Based Scenario Generation

The survey paper [11] discusses the use of foundation models in autonomous driving for scenario generation and analysis. They identify LLM-based scenario generation to be a notable area for research and review related works. They distinguish between open- and closed-loop scenario generation, where the latter enables interaction with and reactivity to the ego vehicle. In our work, we present a solution to LLM-based scenario generation for both modes of execution. We now highlight a few papers on prior work.

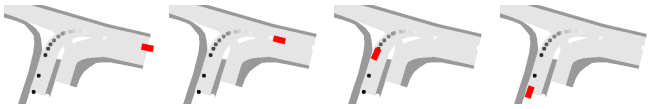

Fig. 3: While the solver reasons in 1-D by modelling the kinematics with a piecewise polynomial in Frenet frame, the resulting motion profile is projected back into the driving path.

The recent works [12], [13] define a similar problem of text to scenario generation. The work [13] in particular targets "on-demand" scenario execution, allowing a user to program the system and use LLMs to decompose the user request into scripts and subscripts for individual rule-based actors to execute at each step of simulation. Compared to [13], we invoke the LLM only in the translation stage and not for decision-making during the scenario execution. Moreover we target and measure precise adherence to reactive behavior, which neither [12] (which is not reactive) nor [13] does.

LeGEND [14] presents an LLM-based prompt engineering workflow for converting abstract and functional scenarios described in natural language into logical and concrete scenarios expressed in a structured format, while ScenicNL [15] converts crash reports in natural language into programs written in a custom DSL using a sophisticated prompting strategy. [16] converts scenario descriptions into state charts. These works are bottlenecked by their DSL or structured format. They do not consider closed-loop reactivity either.

LCTGen [2] and ProSim [3] are language-conditional data-driven methods for open- and closed-loop scenario generation, respectively. In comparison to our approach, these methods are based on deep learning models and, in the case of ProSim, offer a statistical prior that can be conditioned on input prompts or other sources of conditioning (e.g. actions, waypoints or route goals). While the neural network learns to reconstruct the data, it is only approximately sensitive to conditional input and may ignore the instructions. Moreover, their closed-loop approach does not model complex interactions between agents.

C. Automated Motion Profiles

An alternative line of work stemming from the fields of computer graphics and animation uses formal specifications to generate solutions for motion profiles. This idea dates back to the 1980s in the classic "Spacetime Constraints" [17]. In this paradigm, the user specifies the "how" and "what" of the motion, while a solver produces a motion satisfying that input specification. Beyond self-driving applications like [7], our orchestration approach also shares similarities to the animiation system in [18] in which a series of rewrite rules converts an initial specification into a lower-level representation to be consumed by an optimization program or solver based upon kinematic equations of motion. Our rewrite rules are specific to our polynomial formulation and we present the details of our term expansion rules in Algorithms 1 and 2.

Sample Natural Language Description of a Scenario

Ego is on the road and it is initial_ego_distance_behind_intersection_m behind the intersection. It wants to lane follow along the road travelling westbound. There is a hero vehicle on the road. It wants to turn left into the driveway. When the ego is ego_distance_behind_conflict_point_m behind the the conflict point and the hero vehicle is hero_distance_behind_conflict_point_m behind the conflict point, the hero vehicle begins to decelerate to a complete stop at end_hero_distance_behind_conflict_point_m behind the conflict point.

Box 1: A sample description fed to the LLM. We parameterize the scenario with variables (indicated in snake case) to be assigned values later in the pipeline.

D. Constraint-based Scenario Autoformalization

The approach we take is a combination of the above and shares similarities with [10]. The idea is to represent a scenario as a constraint-based specification and search for a solution to that specification. An LLM can aid by autoformalizing a natural language description, such as an accident report, into the expected specification format. The solution is typically actor trajectories. In [10], however, the scenario is not closed loop in nature but rather fixed.

III. METHODOLOGY

We propose a neurosymbolic approach to scenario orchestration that combines an LLM with an SMT solver to orchestrate scenarios from natural language. Given a scenario description and a high definition (HD) map, we use an LLM to identify the actors in the scenario and the routes they drive along. The LLM also specifies each actor's motion profile along its route and a corresponding set of constraints that govern its kinematics and interactions with other actors and the map. Next, an SMT solver solves for the free variables in each actor's motion profile, determining its initial state and how it moves along its route. Finally, to orchestrate a closed-loop scenario, we iteratively reinvoke the solver with additional constraints on all actors' current state to replan each actor's motion profile. See Fig. 2 for an illustration.

A. Problem Formulation

A scenario trace $\mathcal{S}$ consists of an HD map $\mathcal{M}$ and N actor states $\{s_i^{0:T}\}_{i=1}^{N}$ over a finite time horizon T. The i-th actor's state s_i^t at time t consists of its 2D position (x_i^t, y_i^t), heading θ_i^t, velocity v_i^t, and acceleration a_i^t. The HD map $\mathcal{M}$ is represented by a lane graph $G = (V, E)$. Each vertex $u \in V$ is a lane segment and is annotated with various properties; e.g., whether it is a road or driveway, whether it is a left or right turn, etc. The edges $(u, v) \in E$ indicate that that v is a successor, predecessor, or left/right neighbor of u.

Given a scenario description and an HD map, the goal of *scenario orchestration* is to generate a scenario trace $\mathcal{S}$ that satisfies the description. The description declares what should occur; see Box 1 for an example. While it may be given in natural language, we assume that there exists an underlying function $I\colon \mathcal{S} \mapsto \{0,1\}$ that can verify whether the trace $\mathcal{S}$ satisfies the description. Therefore, the orchestrator's goal is to generate $\mathcal{S}$ such that $I(S) = 1$.

In this work, we assume an HD map is given and focus on generating the actor states $\{s_i^{0:T}\}_{i=1}^N$. Note that the orchestrator may not have control over every actor in the scene. In particular, the ego vehicle is typically controlled by the policy under test. The orchestrator must therefore anticipate and react to the ego's behavior to generate a valid trace.

B. Scenario Representation

We represent an actor's trajectory by its motion profile along its driving path; ie, in Frenet coordinates (Fig. 3). We compute the actor's path by traversing the lane graph along its route and fitting a cubic spline to the centerlines of the lane segments traversed. From the lane graph, we derive locations of interest along the path; e.g., when the path enters/exits a left (resp., right) turn lane segment, the intersection, etc. We also identify any conflict points where two actors' paths intersect.

We model an actor's motion profile with a piecewise polynomial function that represents the actor's position, velocity, and acceleration along its path over time. For simplicity, we assume there are no lateral deviations. The function can have an arbitrary number of pieces K, and each piece k is defined by its time duration t_k and a polynomial function ℓ_k given by the kinematic equations of motion

$$\ell_0(t) = x_0 \tag{1}$$

$$\ell_k(t) = \ell_{k-1}(t_{k-1}) + v_k(t - t_{k-1}) + \tfrac{1}{2}a_k(t - t_{k-1})^2 \tag{2}$$

where x_0 is the actor's initial position along its path, v_k is its velocity for piece k, a_k is its acceleration for piece k, t_k is the time duration of the piece, and t ranges from $t_1 + \ldots + t_{k-1}$ to $t_1 + \ldots + t_k$ for each piece k. Thus, an actor's motion is parameterized by the set of variables $\tau = \{x_0, v_k, a_k, t_k\}_{k=1}^K$ and $\mathcal{T} = \{\tau_i\}_{i=1}^N$ is the joint parameterization for all actors.

We use the Frenet transformation for the actor's path to convert between Cartesian and Frenet coordinates. Let Γ_ρ be the Frenet transformation for a path ρ. For an actor's state $(x^t, y^t, \theta^t, v^t, a^t)$ at time t, we have

$$\ell(t), \dot{\ell}(t), \ddot{\ell}(t) = \Gamma_\rho(x^t, y^t, \theta^t, v^t, a^t) \tag{3}$$

$$(x^t, y^t, \theta^t, v^t, a^t) = \Gamma_\rho^{-1}(\ell(t), \dot{\ell}(t), \ddot{\ell}(t)) \tag{4}$$

C. Scenario Constraints

Constraints form the key building block of our scenario representation. We formalize constraints as a set of logical predicates over our actor trajectory and driving path representation described in Section III-B. Let

$$\mathcal{C} = \{c_1, \ldots, c_n\} \tag{5}$$

be a set of constraints, each of which is a Boolean function $c : \mathcal{T} \times \{\rho_i\}_{i=1}^N \mapsto \{0,1\}$. A scenario is satisfiable if there exists a configuration such that all constraints in $\mathcal{C}$ hold.

When specifying constraints, standard boolean logic operators, arithmetic, functions, arrays, etc. are supported. Moreover, Algorithms 1 and 2 define functions for evaluating actor states are both symbolic (unassigned, free variable) and concrete (assigned, fixed) times respectively. This allows for both constraints like "*Actor i and actor j must have equal velocity at some time t*" (symbolic time) and "*Actor i at time* $t = 3.4$ *must have acceleration* $a = 0.1$" (concrete time). Because we have annoated locations of interest along each actor's path, we can also specify constraints relating actors to specific locations; e.g., "*Actor i must be 10m from the intersection and 5m in front of Actor j at some time t*". This allows for a flexible way to define scenarios through the composition of various simple, declarative constraints. Boilerplate constraints such as kinematic bounds, scenario length, etc. can be reused and applied to all scenarios. More complex ideas that arise from multiple constraints can be

Algorithm 1: Computing the state at a symbolic time

Data: $t \in T$, the name of the symbolic time
$o \in \{0,1,2\}$, the order of the state
Result: $f_o(t)$, the expression for the state at time t

```
i ← T.index(t);
terms_curr ← [];
for order ← pieces[i].order to o by −1 do
    terms ← [];
    for term : terms_curr do
        terms ← terms + [antiderivative(term)];
    end
    if order == pieces[i].order or i == 0 then
        expr ← pieces[i]._rates[order];
    else
        expr ← f_o[i − 1];
    end
    terms ← terms + [(expr, pieces[i]._duration, 0)];
    terms_curr ← terms;
end
f_o(t) ← 0;
for term : terms_curr do
    f_o(t) ← f_o(t) + term[0];
end
```

Algorithm 2: Computing the state at a concrete time

Data: $t \geq 0$, the value of the concrete time
$o \in \{0,1,2\}$, the order of the state
Result: $f_o(t)$, the expression for the state at time t

```
Function MakeExpr(pieces, rates, offset, depth):
    if pieces.length() == 0 then
        return rates[o];
    end
    head ← pieces[0];
    tail ← pieces[1 :];
    lo ← offset;
    hi ← offset + head._duration;
    if depth == pieces.length() − 1 then
        cond ← t ≥ lo;
    else
        cond ← And(t ≥ lo, t < hi);
    end
    then_expr ← head_o(t − lo, rates);
    for ord ← 0 to 2 do
        rates[ord] ← head_ord(head._duration, rates);
    end
    else_expr ← MakeExpr(tail, rates, hi, depth + 1);
    return If(cond, then_expr, else_expr);
return MakeExpr(pieces, [0,0,0], 0, 0);
```

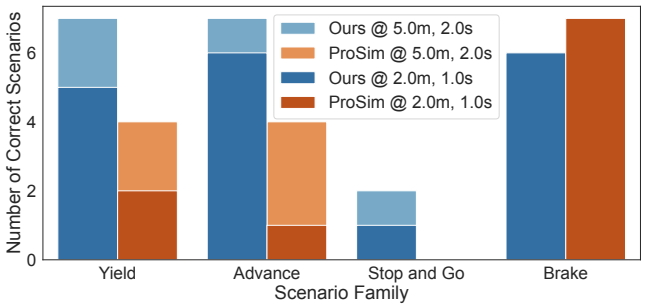

Fig. 4: **Quantitative Comparison.** More saturated in color and taller bars indicate better performance. Notice that our method achieves challenging stop-and-go scenarios.

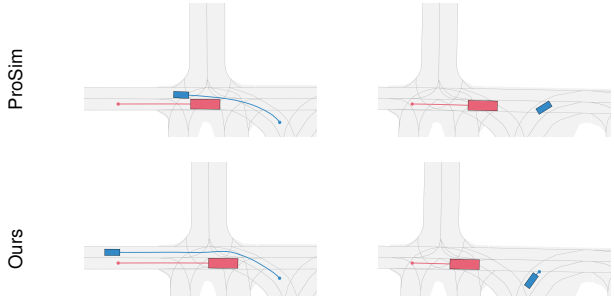

Fig. 5: **Qualitative Comparison.** We show the baseline (**top**) and our approach (**bottom**) on two different scenario descriptions. The baseline fails to accelerate past the conflict point. The line behind each actor indicates a 5s history.[1]

saved as template and reused as well.

D. Closed-loop Replanning

In most cases, the purpose of orchestration is to test the behavior an AV policy that is not under the control of the orchestrator. Thus, the orchestrator must be able to react to new AV behavior over the course of a closed-loop rollout of the scenario. In our framework, constraints can naturally specify closed-loop behavior, e.g. "*Actor i maintains the same speed as ego*". To support an unknown AV policy, we can simply update the ego's current state as variable assignments or additional constraints and re-solve at some frequency. We also found it helpful to add constraints[2] on the ego's future motion as well. Intuitively, these constraints act as the solver's model of the future ego behavior. Due to our efficient scenario representation, the solve-time itself is relatively quick compared to the initial constraint generation phase which is either manually written by humans or completed by an LLM.

E. Foundation model

While writing constraints manually is still reasonably ergonomic, the ability to specify scenarios directly in natural language can have many advantages, such as allowing non-technical team members to directly create scenarios themselves, or potentially opening up the possibility of leveraging scenario description databases available online. In this work, we simply prompt an LLM with an in-context example of a language description of a scenario and the corresponding scenario constraints and program. We found it helpful to invoke the LLM in separate stages corresponding to different components of our scenario program. Specifically, we first ask for a pseudocode representation of the constraints before programming them into Python. In this work, we use GPT-5[19] to leverage the Z3 [20] API to specify constraints before calling the solver, but our approach is agnostic to the specific LLM or solver. An example of the scenario description and prompt can be found in the appendix.

IV. EXPERIMENTS

A. Experiment Setting

1) Scenario Descriptions: We conduct experiments on a set of 80 scenario descriptions. We sample 20 scenarios for development (e.g., prompt engineering) and hold out the rest for testing. The scenarios describe the interaction between the ego and an actor at an intersection and vary in:

- *Intended Route:* We vary the ego and the actor's starting locations and their intended routes; e.g., lane-following along the road, turning left to enter the driveway, turning right to exit the driveway, etc. This dictates the conflict geometry between the ego and the actor–whether they are travelling along the parallel lanes, converging to the same lane, or crossing each others' lanes.
- *Desired Interaction:* We vary the desired interaction between the ego and the actor; e.g., the actor should yield, not yield, or stop-and-go to the ego turning across its path, the actor should hard brake to a stop, etc.
- *Trigger Condition:* We vary the trigger condition for the desired interaction; e.g., the ego's distance behind the actor, the ego and the actor's distance or time-to-arrival at a conflict point, the ego and the actor's distance or time-to-arrival to the intersection, etc.

2) Metrics: Our primary metric is orchestration success rate. For each scenario description, we define the success criteria as a programmatic function of the scenario trace. Specifically, we implement a function that, given a scenario description and trace, evaluates whether the scenarios's intended route, desired interaction, and trigger conditions are met. We use a best-effort approach to ensure that the intent in the descriptions are accurately captured. Our success criteria is hand-written, but defining this function based on natural language is an interesting and open research question.

Our success criteria are defined with respect to an error tolerance. For example, if the trigger condition is based on the ego and the actor's distance (resp., time-to-arrival) to a conflict point, we evaluate whether that trigger condition is met with respect to an error tolerance on the distance

[1]Note that these results use GPT-5 with a high reasoning effort.
[2]e.g. preventing the ego from reversing

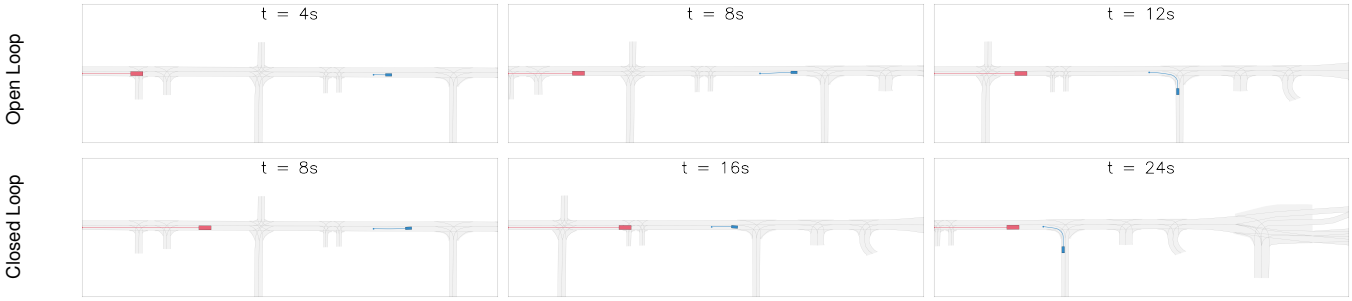

Fig. 6: Side-by-side comparison of open loop vs. closed loop orchestration for a lead actor turn into driveway scenario. In the open loop mode, the lead actor makes the turn while the ego actor remains distant, whereas in the closed loop mode, the lead actor makes the turn as the ego actor approaches it. (top) Open loop orchestration at every 4s with 4s long history. (bottom) Closed loop orchestration at every 8s with 8s long history. The lead actor waits for the ego, which decelerates due to the lead actor, to approach it in order to meet the reactive specification of making the turn 50m ahead of the ego.

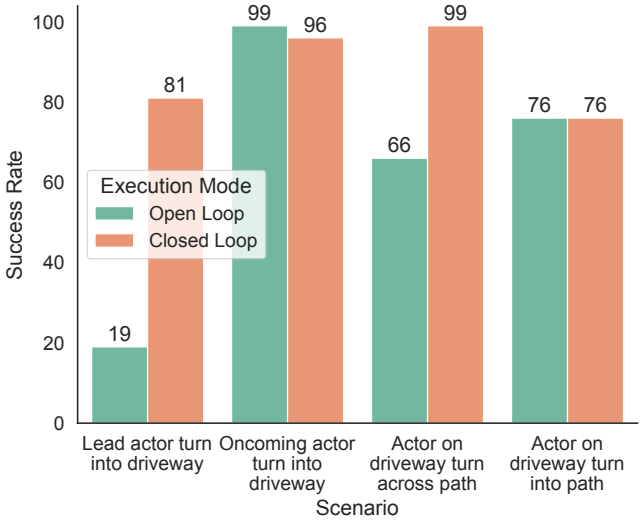

Fig. 7: **Importance of Closed-loop.** We find closed-loop replanning in order to react to ego behaviors is important, especially for scenarios that require actor/ego interaction.

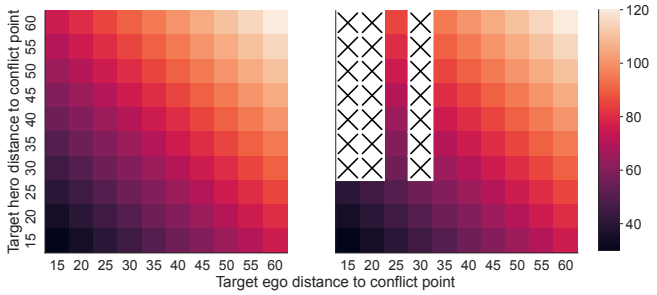

Fig. 8: (left) Ours. (right) Baseline. The cell color represents the sum of the x and y values. A cell with an "x" indicates that the reactive trigger was not met. We evaluate both methods at a distance threshold of 2.0m. We relax the validation criteria for the baseline so that the actor does not have to yield at the specified distance but note that this is part of the test intention and that most of the baseline trials fail for this reason.

(resp., time-to-arrival). We report success rate over two error tolerance levels. Overall, this metric allows us to measure the degree of controllability/precision a method is able to offer.

B. Benchmarking Natural Language to Scenario

We evaluate our method's ability to orchestrate scenarios from natural language descriptions. In all cases, the ego vehicle is equipped with an AV policy that the orchestrator does not have control over.

1) Baseline: We use ProSim [3], a state-of-the-art deep learning based conditional traffic simulation model as our baseline, with some adaptations to our setting. Firstly, because ProSim does not handle actor initialization and also requires 1 second of state history, we use our method's output at from $t = 0$ to $t = 1$ to initialize and warmup ProSim. Secondly, as ProSim's language conditioning does not handle arbitrary prompts, we found that the most effective approach was to use our method's plans as intermediate (x, y, t) waypoints for ProSim conditioning. Specifically, we re-run the solver at 1hz and give ProSim the waypoint planned (x, y) 4 seconds from the current time.

2) Implementation Details: We use GPT-5 [19] as the LLM. We tried reasoning efforts of low, medium, and high and found medium to work the best. For the external, off-the-shelf SMT solver we use Z3 [20] via the Python API.

3) Results: We show quantitative results in Fig. 4 and qualitative results in Fig. 5. Overall, we find that our approach outperforms the baseline in orchestration success rate. Despite using the plans from our approach as goal points, the ProSim baseline has a lower success rate. We found that the most common failure mode was ProSim not executing the scenario to the precise timing of the solver. For instance, if a constraint is to meet the ego at a certain conflict point, ProSim may fail to accelerate fast enough, or in another case, ProSim may fail to go ahead and not yield. This suggests that while statistical models may be beneficial in generating more in-distribution scenarios, precise or slightly out-of-distribution scenarios remain a challenge.

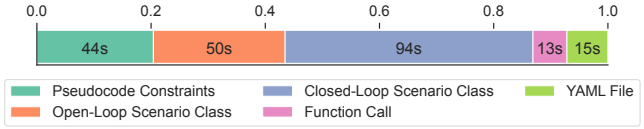

Fig. 9: **LLM Generation Runtime.** We measure the average runtime of the one-time LLM generation phase. In our experiments, we use GPT-5 (medium reasoning effort).

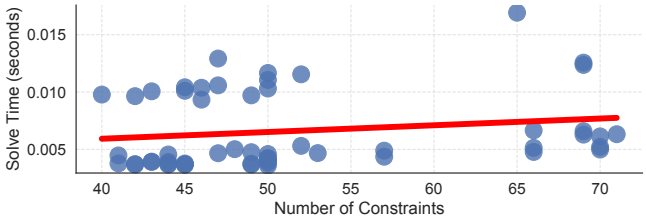

Fig. 10: **Solver Runtime.** We measure the runtime of the solver when given varying number of constraints.

C. Precision & Controllability

To assess the precision and controllability of our method, we analyze its performance on a fixed scenario across a discretized grid of parameter values in comparison to the baseline (Fig. 8). The scenario description is shown in Box 1. We find that the baseline systematically fails when the target ego distance to the conflict point is low. This is problematic for a testing framework which must have coverage over the range of possible scenario parameterizations in order to provide a reasonable guarantee on safety.

D. Open loop vs. Closed loop Execution

Our next experiment analyzes the importance of our closed-loop approach to scenario orchestration. Specifically, we evaluate our approach on four representative scenarios with closed-loop replanning disabled ("open-loop execution") and enables ("closed-loop execution"). In both cases, we emphasize that the ego is controlled by a different AV policy executing in closed-loop. To control for any errors attributable to the LLM, we manually implement constraints for each scenario. These scenarios span the possible intended routes for the actor at an intersection. As before, each scenario also contains a trigger condition that relates the ego and non-ego actors which the scenario framework needs to achieve in closed-loop. Qualitative and quantitative results are shown in Figs. 6 and 7 respectively.

We find that for half of the scenarios, closed-loop execution makes a significant difference over open-loop execution. For the remaining half of the scenarios, we believe that the open- vs. closed-loop versions are equivalent due to the semantics of the scenarios involving a trigger point that does not change over time and thus the scenario is not sensitive to the behavior of the ego vehicle. Due to the complexity of closed-loop, some errors may result such as a failure to solve the constraint satisfaction problem, leading to the slightly worse performance of closed-loop on the oncoming actor scenario. Overall, closed-loop orchestration achieves a 23% higher success rate over open-loop orchestration.

E. Runtime Analysis

In Figs. 9 and 10, we analyze the runtime of our approach. Overall, the one-time LLM generation phase takes on the order of minutes while the solver, which is run in closed loop at the re-planning frequency, runs within a fraction of a second. During closed loop execution, we solve at various tolerances until a solution is found, or raise a failure at the maximum tolerance, with a solve timeout of 10s. Solving can take longer than a few seconds during closed loop execution depending on the difficulty of the problem.

V. CONCLUSIONS & DISCUSSION

In this paper we introduced a neurosymbolic framework for closed-loop traffic scenario orchestration By delegating high-level spatial-temporal reasoning and natural-language understanding to an LLM, and precise continuous-time motion planning to an SMT solver, our method achieves controllable, reactive, and physically realizable scenarios from free-form textual specifications. Our experiments on a diverse set of scenario descriptions show that this hybrid design consistently outperforms a state-of-the-art learning-only baseline, particularly for multi-actor interactions and edge-case conditions that have precise spatial temporal requirements. The solver's ability to re-plan in closed loop enables seamless adaptation to the ego vehicle's evolving trajectory, especially important in scenarios with reactive constraints.

We now discuss some limitations. Our use of an LLM means that errors such as hallucinations can propagate through our system. Our implementation features a single hero actor and map which should be scaled up to greater real-world complexity. Lastly, our experiments did not measure statistical realism. Measuring realism is difficult since our descriptions do not correspond to any existing ground truth execution. Most realism metrics are based on data likelihood, but we are also not focused on orchestrating the most likely scenarios. Thus, as a proxy, we have resorted to ensuring all methods create kinematically feasible trajectories. However, exploring additional metrics and methods which consider statistical realism is a promising line of future work.

APPENDIX

A. Supplementary Algorithms

We include subroutines referenced in Algorithms 1 and 2.

Algorithm 3: Calculating the antiderivative of a term

Data: $expr$, the term expression
var, the variable that the term is with respect to
$o \in \{0,1,2\}$, the order of the term
Result: $\int expr$, the antiderivative of the term
o, the next order of the term
$\int expr \leftarrow \frac{max(1,o)}{o+1} \cdot expr \cdot var$;
$o \leftarrow o+1$;

Algorithm 4: Computing the piece at a time

Data: t, the time
$o \in \{0,1,2\}$, the order of the state
piece, the piece
rates, the cumulative value at each order so far
Result: $piece_o(t, rates)$, the expression for the piece at time t

$s \leftarrow 0$;
if *rates* **then**
  $rates\_next \leftarrow []$;
  for $order \leftarrow 0$ **to** $piece.order + 1$ **do**
    $\_r \leftarrow piece.\_rates[order]$;
    $r \leftarrow rates[order]$;
    if $order < piece.order$ **then**
      $rates\_next \leftarrow rates\_next + [\_r + r]$;
    else
      $rates\_next \leftarrow rates\_next + [\_r]$;
    end
  end
  $rates \leftarrow rates\_next$;
else
  $rates \leftarrow piece.\_rates$;
end
for $k \leftarrow o$ **to** $piece.order + 1$ **do**
  $power \leftarrow k - o$;
  if *power* **then**
    $val \leftarrow t^{power}$;
  else
    $val \leftarrow 1$;
  end
  $s \leftarrow s + \frac{1}{power!} \cdot rates[k] \cdot val$;
end

B. Prompts

We include excerpts of our prompts for multiple stages of our LLM-based generation.

Natural Language to Constraint Generation Prompt

You convert descriptions of surface street driving scenarios into actor trajectories and pseudocode constraints. The scenario consists of exactly two actors, one ego vehicle (id = 0) and one other actor (id = 1).
`[... ommitted for space]`
Pseudocode constraints
To refer to the acceleration of an actor with id 'i' at knot 't{j}', use 'A{i}a("t{j}")'. To refer `[... ommitted for space]`

Constraint to Scenario Class Generation Prompt

You translate actor trajectories and pseudocode constraints into a Python class.
The Python class should inherit from the Scenario class and be named the scenario name converted to camel case and with "Scenario" appended.
`[... ommitted for space]`
In the pseudocode constraints, there are special constants 'stop_line', 'conflict_point', 'turn_start', and 'turn_end'. `[... ommitted for space]`

Open Loop to Closed Loop Scenario Class Generation Prompt

Modify the given Scenario class as follows:
- add self.search = True in the .reset() method
`[... ommitted for space]`
Output the complete, modified Scenario class after making all the requested modifications. Do not output anything else.

REFERENCES

[1] ASAM e.V., "ASAM OpenSCENARIO v2.0.0," 2022, retrieved from https://www.asam.net/standards/detail/openscenario/v200/.

[2] S. Tan, B. Ivanovic, X. Weng, M. Pavone, and P. Kraehenbuehl, "Language conditioned traffic generation," in *7th Annual Conference on Robot Learning*, 2023. [Online]. Available: https://openreview.net/forum?id=PK2debCKaG

[3] S. Tan, B. Ivanovic, Y. Chen, B. Li, X. Weng, Y. Cao, P. Krähenbühl, and M. Pavone, "Promptable closed-loop traffic simulation," in *8th Annual Conference on Robot Learning*, 2024.

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

[5] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, 2019, pp. 63–78.

[6] C. Chang, D. Cao, L. Chen, K. Su, K. Su, Y. Su, F.-Y. Wang, J. Wang, P. Wang, J. Wei, *et al.*, "Metascenario: A framework for driving scenario data description, storage and indexing," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 2, pp. 1156–1175, 2022.

[7] M. Klischat and M. Althoff, "Synthesizing traffic scenarios from formal specifications for testing automated vehicles," in *2020 IEEE intelligent vehicles symposium (IV)*. IEEE, 2020, pp. 2065–2072.

[8] A. Nonnengart, M. Klusch, and C. Müller, "Crisgen: Constraint-based generation of critical scenarios for autonomous vehicles," in *Formal Methods. FM 2019 International Workshops: Porto, Portugal, October 7–11, 2019, Revised Selected Papers, Part I 3*. Springer, 2020, pp. 233–248.

[9] K. Scheibler, A. Eggers, T. Teige, M. Walz, T. Bienmüller, and U. Brockmeyer, "Solving constraint systems from traffic scenarios for the validation of autonomous driving." in *SC-square@ SIAM AG*, 2019.

[10] A. Guo, Y. Zhou, H. Tian, C. Fang, Y. Sun, W. Sun, X. Gao, A. T. Luu, Y. Liu, and Z. 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*, 2024, pp. 268–280.

[11] Y. Gao, M. Piccinini, Y. Zhang, D. Wang, K. Moller, R. Brusnicki, B. Zarrouki, A. Gambi, J. F. Totz, K. Storms, *et al.*, "Foundation models in autonomous driving: A survey on scenario generation and scenario analysis," *arXiv preprint arXiv:2506.11526*, 2025.

[12] X. Cai, X. Bai, Z. Cui, D. Xie, D. Fu, H. Yu, and Y. Ren, "Text2scenario: Text-driven scenario generation for autonomous driving test," *arXiv preprint arXiv:2503.02911*, 2025.

[13] H. Gao, J. Wang, W. Fang, J. Xu, Y. Huang, T. Chen, and X. Ma, "Laser: Script execution by autonomous agents for on-demand traffic simulation," in *Proceedings of the 16th International Conference on Internetware*, 2025, pp. 84–95.

[14] S. Tang, Z. Zhang, J. Zhou, L. Lei, Y. Zhou, and Y. Xue, "Legend: A top-down approach to scenario generation of autonomous driving systems assisted by large language models," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1497–1508.

[15] K. Elmaaroufi, D. Shanker, A. Cismaru, M. Vazquez-Chanlatte, A. L. Sangiovanni-Vincentelli, M. Zaharia, and S. A. Seshia, "Generating probabilistic scenario programs from natural language," *CoRR*, 2024.

[16] P. Nguyen, T.-H. Wang, Z.-W. Hong, S. Karaman, and D. Rus, "Text-to-drive: Diverse driving behavior synthesis via large language models," 2024.

[17] A. Witkin and M. Kass, "Spacetime constraints," *ACM Siggraph Computer Graphics*, vol. 22, no. 4, pp. 159–168, 1988.

[18] T. Ellman, R. Deak, and J. Fotinatos, "Automated synthesis of numerical programs for simulation of rigid mechanical systems in physics-based animation," *Automated Software Engineering*, vol. 10, pp. 367–398, 2003.

[19] OpenAI, "Gpt-5," https://platform.openai.com, 2025, openAI API.

[20] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.