Conversational Code Generation: a Case Study of Designing a Dialogue System for Generating Driving Scenarios for Testing Autonomous Vehicles

Rimvydas Rubavicius*, Antonio Valerio Miceli-Barone, Alex Lascarides and Subramanian Ramamoorthy

School of Informatics, University of Edinburgh 10 Crichton Street, Edinburgh EH8 9AB

Abstract

Cyber-physical systems like autonomous vehicles are tested in simulation before deployment, using domain-specific programs for scenario specification. To aid the testing of autonomous vehicles in simulation, we design a natural language interface, using an instruction-following large language model (LLM), to assist a non-coding domain expert in synthesising the desired scenarios and vehicle behaviours. We show that using it to convert utterances to the symbolic program is feasible, despite the very small training dataset. Human experiments show that *dialogue* is critical to successful simulation generation, leading to a 4.5 times higher success rate than generation without engaging in extended conversation.

Keywords

Code Generation, Autonomous Vehicles, Simulation, Human-computer Interaction

1. Introduction

Testing autonomous vehicles exclusively on public roads, especially in near-crash scenarios, is not feasible [1]. Iterative development and testing in simulation is essential [2], especially in the early stages of the development process. Simulators like CARLA [3] support this iterative development process for autonomous vehicles (AVs). Using CARLA, engineers can generate driving scenarios using Scenic [4]: probabilistic scenario description languages such as Scenic specify a distribution over the driving scenarios that satisfy the description, and then sample many simulation instances for evaluation. In this way, challenging driving scenarios can be generated [5, 6], which can then be used for evaluation of control algorithms for autonomous vehicles without a physical system, safely and cost-effectively.

Writing programs to specify driving scenarios in Scenic, or more generally programming in domain-specific languages (DSLs), is challenging with steep learning curves. Domain experts with deep knowledge of edge cases and desired driving behaviours may lack proficiency in Scenic or other DSLs, or even when they do have such proficiency, they may still benefit from some assistance in their workflow. A natural language interface in the form of a dialogue system in which engineers converse with a chatbot that knows how to program in Scenic is a useful tool for facilitating this. In this paper, we aim to develop such a dialogue system, and by doing so increase engineer access to simulation-driven testing, by allowing them to interactively synthesise scenarios using natural language dialogue.

This is a challenging task for two reasons. Firstly due to data scarcity (only 32 examples of Scenic programs with natural language descriptions are available online), approaches in developing a dialogue system using instruction-following large language models (IFLLM) [7] are not directly applicable. Secondly, the scenario description provided by the user may be *underspecified*: missing some details about the desired driving scenario in the description that is only exposed through interaction with

GeCoIn 2025: Generative Code Intelligence Workshop, co-located with the 28th European Conference on Artificial Intelligence (ECAI-2025), October 26, 2025 — Bologna, Italy

*Corresponding author.

rimvydas.rubavicius@ed.ac.uk (R. Rubavicius); amiceli@ed.ac.uk (A. V. Miceli-Barone); alex@inf.ed.ac.uk (A. Lascarides); s.ramamoorthy@ed.ac.uk (S. Ramamoorthy)

0000-0003-4270-7154 (R. Rubavicius); 0000-0002-9904-8869 (A. V. Miceli-Barone); 0000-0003-1704-1864 (A. Lascarides); 0000-0002-6300-5103 (S. Ramamoorthy)

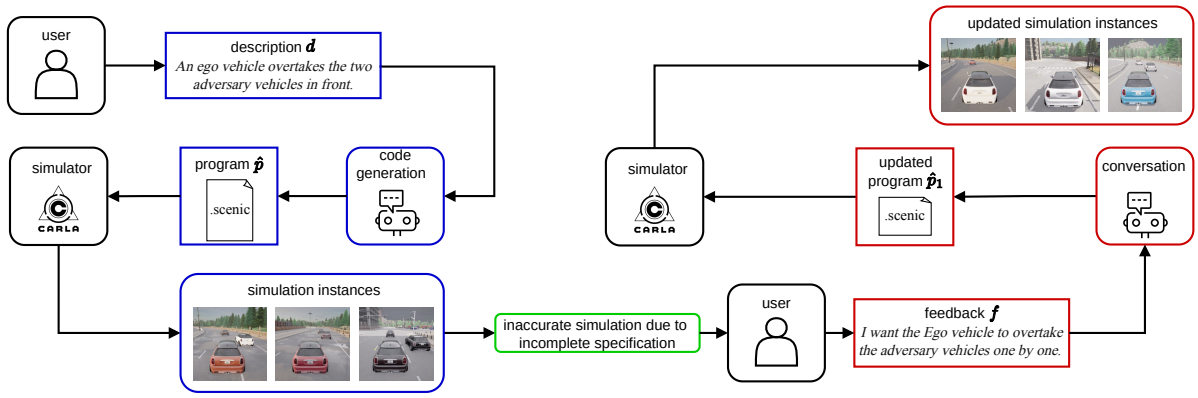


Figure 1: Dialogue System Overview. It facilitates driving scenario generation using natural language in code generation (§ 3.1) and conversation (§ 3.2).

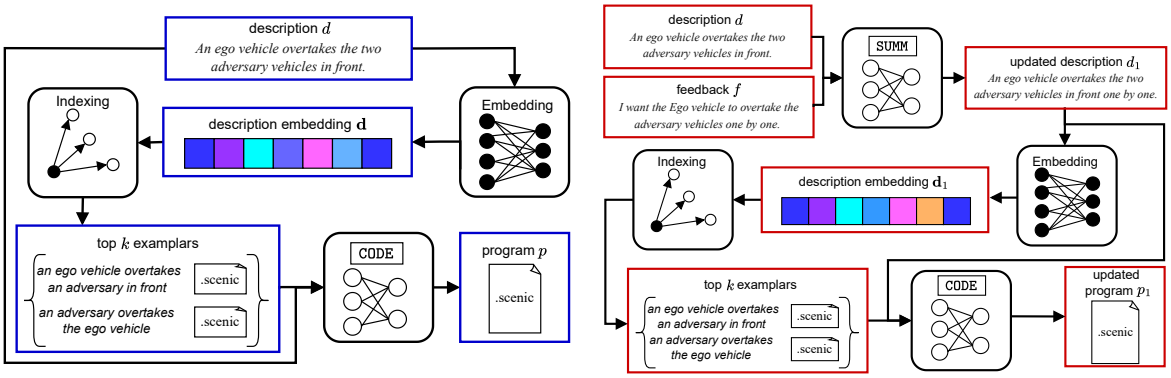
the simulator. Figure 1 illustrates this phenomenon. When a user utters "an ego vehicle overtakes the two adversary vehicles", the dialogue system can synthesize the simulation instance satisfying the description, but the description itself does not provide all information about the situation, which in this case is the manner of overtaking (could be one by one, or both at the same time). The user only becomes aware of the necessity to provide such details by observing simulation instances. In this context, it would be helpful for a natural language interface to allow the user to express feedback to the system, which can then be used to update the program and, in turn, generate simulation instances.

To this end, this paper makes the following contributions: 1) we create a dataset of English description-Scenic program pairs for a variety of driving scenarios; 2) we investigate how IFLLM can be used to synthesize Scenic programs from natural language expressions in an embodied conversation where the user converses with a chatbot and reacts to driving simulation instances; and 3) we conduct human trials to evaluate the dialogue system, and in particular the value of having multiple dialogue turns. Our results demonstrate that the dialogue system can be a valuable tool for facilitating the generation of interactive driving scenarios.

2. Related Work

Code Generation The problem of converting natural language to executable programs has been widely studied as semantic parsing [8, 9, 10]. Recently, using IFLLMs has been explored for this task and also the wider problem of arbitrary code generation, including auto-completion [11] to assist the software development process. In this paper, we concentrate on developing a dialogue system that generates Scenic programs using LLMs. It is a low-resource domain where only a handful of examples are available. This is in contrast to other popular programming languages like Python, which benefit from many code repositories for training [12]. Furthermore, the code for high-resource programming languages follows style guidelines, making it easier for models to learn generalisations from the well-structured data. In contrast, the Scenic program repository was written in different styles by different software engineers.

Testing AVs Testing is a fundamental process within software engineering, which is particularly important for designing safety-critical systems, such as AVs [13, 14]. There are a variety of methods for testing and validating AV design, even with black-box components [15]. This paper focuses on testing by generating driving scenarios as a complementary procedure to formal verification. Such testing involves interaction with expert users, which is essential for human-centred design [16]. Driving scenario generation has been explored before: both direct scenario generation [17, 18] or by utilising Scenic [19, 20]. What is unique about this work is: (i) human experiments; and (ii) utilising dialogue to align the generated driving scenarios with the user's intent. We view this latter contribution as



(a) Code generation using RAG [25] (§ 3.1) (b) Conversation for updating description d_1 (§ 3.2).

Figure 2: Dialogue System for Conversational Driving Scenario Generation.

essential, given the ubiquitous phenomenon of *natural language pragmatics* [21]: speakers of natural language often intend to convey a meaning that goes beyond what they make linguistically explicit, relying instead on their interlocutor's capacity to decode the intended content using the linguistic and non-linguistic context. There is no guarantee that the sampling processes in Scenic match how competent interlocutors perform this task, and so errors in decoding the intended message via Scenic sampling may happen, prompting the need for user feedback.

Dialogue Systems and Tools Natural language interfaces in the form of a dialogue system have become a popular means for users to interact with systems without needing to know the underlying system or how to program it[22, 23]. Broadly, dialogue systems are categorised by their function as goal-oriented or for chit-chat. We are designing a goal-oriented dialogue system with tool use [24], which in this case is the CARLA simulator. What is unique about our work is that the driving scenarios generated by the simulator are not observed in the dialogue system and are latent, with observations coming only from the user's feedback after observing simulation instances.

3. The Dialogue System

This section describes the dialogue system for generating driving simulations via conversations. Figure 2 demonstrates the two modes of the system: code generation (§3.1); and conversation (§3.2), in which the user responds to simulation instances as depicted in Figure 1.

3.1. Code Generation

To generate Scenic programs from natural language descriptions, the dialogue system uses retrieval-augmented generation (RAG) [25] with in-context learning [26] to construct the `CODE` prompt.

To generate a program $\hat{p}$ from natural language description d, we constructed a dataset $\mathcal{D} = \{(d_i, p_i)\}_{i=1}^{N}$ of size $N = 105$ description-program pairs[1], via manual augmentation of the few exemplars available on the web.[2] Additionally, in our manual augmentation process, we standardise the codebase to minimise variability and ease the learning process in this low-resource regime. Each program is standardised in its coding style, with five distinct code blocks—map and model, constants, agent's behaviour, spatial relations, and scenario specification—to minimise the variation between the exemplars. The dataset $\mathcal{D}$ is indexed via embeddings of descriptions.[3]

[1] https://huggingface.co/datasets/assistive-autonomy/scenic-driving-scenarios
[2] https://github.com/BerkeleyLearnVerify/Scenic/tree/2.x/examples
[3] https://huggingface.co/BAAI/bge-small-en-v1.5

`CODE` prompt for code generation.

[INST] You are a helpful assistant that translates English descriptions to Scenic programs. Scenic is a domain-specific probabilistic programming language for creating distributions over specified scenarios. For driving scenarios, each program has the following blocks:

- MAP AND MODEL: importing town assets and enabling simulator;
- CONSTANTS: specifying vehicle blueprint and other constants like vehicle speed, brake intensity and safety distance;
- AGENT'S BEHAVIOUR: describing how individual vehicles behave in the scenario;
- SPATIAL RELATIONS: outlining the type of road the scenario needs to be synthesized in (e.g. having or not having intersections)
- SCENARIO SPECIFICATION: creating individual vehicles and pedestrians in the specified roads, together with constraints that are required to be true for the full simulation as well as the termination condition.

Here are examples of descriptions and programs: {DESCRIPTION-PROGRAM PAIRS}
Now, please translate the following English description to a Scenic program. Just give the program. No extra information. Description: {DESCRIPTION}[/INST]

`SUMM` prompt for description update.

[INST] You are a helpful assistant that creates updated driving scenario descriptions. Given a driving scenario description and corresponding feedback regarding the simulation, your task is to generate an updated description that incorporates the feedback. Please ensure that the updated description accurately reflects the intended action based on the feedback received and does not introduce additional information or lose information, like the number of vehicles or pedestrians in the situation. The description will outline a specific scenario or context, while the feedback will provide information about how the described scenario could have been improved or modified. Be sure to maintain the original meaning and intent of both the initial description and the feedback. Be brief and only return the updated driving scenario description. Description: {DESCRIPTION} Feedback: {FEEDBACK}[/INST]

Figure 3: Interactive driving simulation generation prompts. Curly brackets denote task-specific information.

At inference time, the embedding $\mathbf{d}$ of the user's description d is used to find the most similar $k = 3$ description-program pairs from $\mathcal{D}$ using maximum inner-product search:

$$\mathcal{D}_k(\mathbf{d}) = \{(d, p) \in \mathcal{D} \mid (d, p) \in top(k, \mathbf{d}, \mathcal{D})\} \tag{1}$$

The description with the retrieved exemplars are then used to create the `CODE` prompt (see Figure 3 for the prompt specification) which is then used to generate the program $\hat{p}$ using an `IFLLM`:

$$\hat{p} = \texttt{IFLLM}(\texttt{CODE}(\mathcal{D}_k(\mathbf{d}), d)) \tag{2}$$

There is no guarantee that the program $\hat{p}$ will execute. To fix this, we perform error feeding: the program $\hat{p}$ is passed to the simulator to attempt to create simulation instances; and if an error occurs, the natural language description, $\hat{p}$ and its errors are passed back to the `IFLLM`, to attempt to self-correct the output. If, after three attempts of error feeding, the generated program doesn't execute, the user is asked to provide an alternative description.

3.2. Conversation

When $\hat{p}$ executes, $n = 3$ instances of the driving scenario simulations are shown to the user, who is given the opportunity to react to these, either by indicating successful generation or by providing natural language feedback f that corrects or refines the results, in an attempt to provide simulations that better align with the user's communicative intent.

Feedback is a response to the *simulations* and not to $\hat{p}$ (which is not observable to the user): it can feature and refer to information that is not present in the Scenic program but is present in the simulation due to the nature of random sampling. Because of this, feeding only f to `IFLLM` is suboptimal due to its missing context, which cannot be given in a principled way (e.g. conditioning on the simulation video or some domain-specific trace would lead to further data scarcity, which is already problematic in this domain). To alleviate this issue, we feedback an updated description by summarising d and f, using a `SUMM` prompt (see the Figure 3 for the prompt specification), to generate the updated description d_1. With this description, our aim is to guide `IFLLM`, minimise the overall context, and not introduce hallucinations and unwanted behaviours.

$$d_1 = \texttt{IFLLM}(\texttt{SUMM}(d, f)) \tag{3}$$

The updated description d_1 is used to generate the updated program $\hat{p}_1$ following the previously described code generation procedure.

$$\hat{p}_1 = \texttt{IFLLM}(\texttt{CODE}(\mathcal{D}_k(\mathbf{d}_1), d_1)) \tag{4}$$

The conversation continues up to $m = 4$ turns. If the user remains unsatisfied with the simulations, the conversation is deemed unsuccessful.

4. Experiments

We conduct experiments for code generation and human evaluation for the overall dialogue system. NVIDIA RTX A6000 is used for serving IFLLM via a text generation inference server. For human trials, we use an Alienware Area-51m laptop with NVIDIA RGX 2080 to generate simulation instances in CARLA.

4.1. Code Generation Experiments

4.1.1. Experimental Setup

We experiment with several open-source IFLLMs with 7B parameters: Mistral [27], Gemma [28], and CodeLlama [29] that used $k = 3$ exemplars to construct `CODE` that's retrieved using RAG, or at random with and without error feeding. To cope with the small dataset bias when evaluating different IFLLMs, we perform leave-one-out validation [30]. We record three metrics: (a) BLEU [31, 32] to measure generation precision; (b) ROUGE-L [33] to measure generation recall; and (c) Execution (EXEC) to measure the percentage of the generated code that is executable, all measured with error feeding (if required). More sophisticated code generation evaluation metrics like CodeBLEU [34], to abstract away from the surface form and focus on the syntactic similarity, are not considered because such metrics require adaptation of the parser to Scenic programs. This is partially circumvented by program standardisation performed in the dataset construction.

4.1.2. Results and Discussion

Table 1 records the results for code generation. We observe that IFLLM that uses `CODE` prompt constructed using RAG is better than a random selection, which most of the time results in non-executable programs. Error feeding boosts performance for IFLMM that are not fine-tuned on code (Mistral and Gemma), yet it does increase inference time, which for Gemma leads to non-termination in a reasonable time. High BLEU and ROUGE-L indicate a high correspondence between prediction and reference, as expected after program standardisation. For EXEC, CodeLlama is better than Mistral and Gemma, which is as expected due to the supervised fine-tuning on code. Based on code generation experiments, we chose CodeLlama with RAG and error feeding to use for human evaluation.

Table 1
Code generation evaluation. RAG notes that CODE prompt was constructed using RAG while RAND notes it was constructed using random exemplars. EF notes error feedback while symbol - denotes cases when IFLLM did not terminate in a reasonable time (more than 10 min.).

CodeLlama	BLEU	ROUGE-L	EXEC	Gemma	BLEU	ROUGE-L	EXEC	Mistral	BLEU	ROUGE-L	EXEC
RAG+EF	64.32	63.23	**71.43**	RAG+EF	56.86	55.86	40.95	RAG+EF	84.54	79.94	41.90
RAG	**89.66**	**85.96**	70.48	RAG	10.44	16.79	0.00	RAG	87.80	84.03	37.14
RAND+EF	0.09	5.80	0.00	RAND+EF	-	-	-	RAND+EF	10.41	30.74	1.90
RAND	0.45	17.29	0.00	RAND	-	-	-	RAND	4.46	23.05	0.00

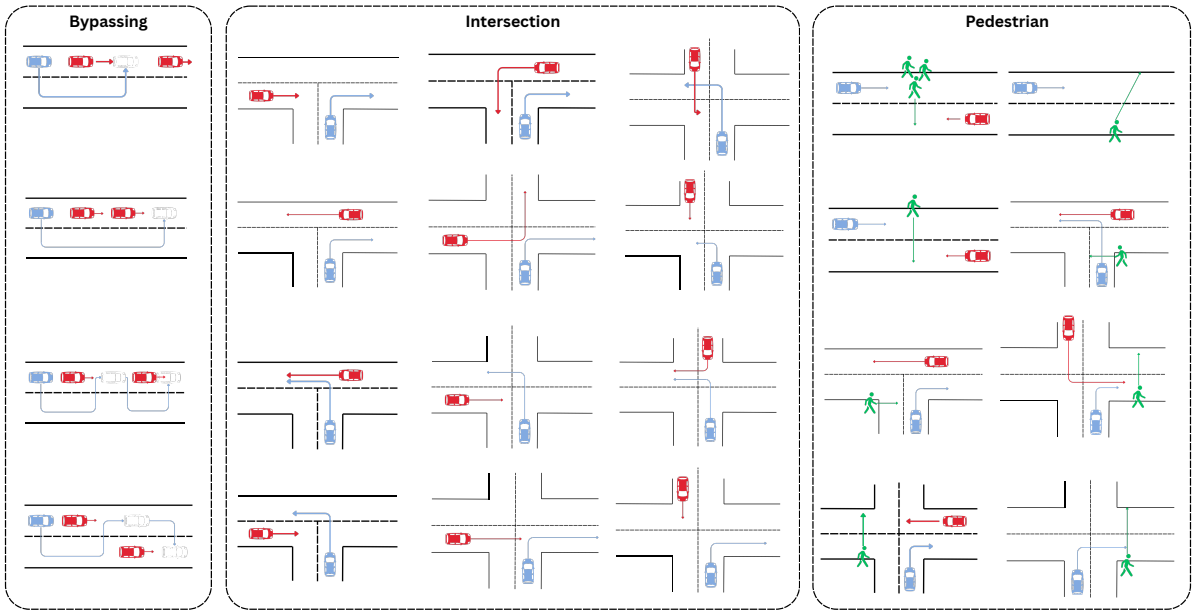


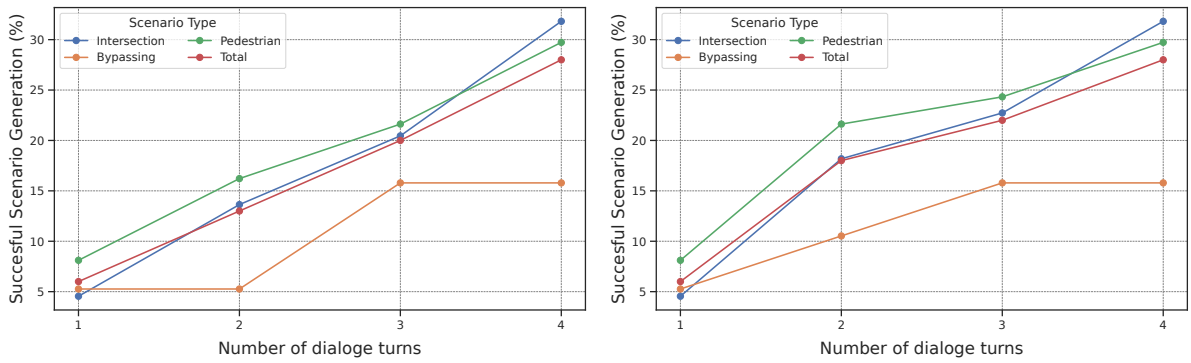
Figure 4: Visual stimuli for generating driving scenarios using dialogue system in human evaluation. In each stimulus, blue elements denote an ego vehicle, red elements denote adversary vehicles, and green elements denote pedestrians.

4.2. Human Evaluation

4.2.1. Experimental Setup

We perform a human evaluation of the dialogue system. Visual stimuli used for human trials are given in Figure 4. The user is presented with a (diagrammatic) stimulus and asked to describe it. After that, CARLA is run to produce 3 simulation instances.[4] After observing it, the user may respond with further natural language input to what they see, or check "satisfied with the scenario produced" and move to the next stimulus. The user can have up to $m = 4$ dialogue turns. If code generation does not produce executable code, the user is asked to paraphrase their description as if it is a fresh interaction. The stimuli are categorised into three types: bypassing (e.g. the user stimulus may yield the user description "an ego overtakes an adversary"), intersections (e.g. "an ego goes left at a 4-way intersection while the adversary approaches from behind."), and pedestrians (e.g."an ego goes left and yields to the pedestrian"). The experiments were conducted with 20 users who have a driving license (as a proxy for domain proficiency), with each asked to describe 5 of the 25 stimuli. In total, 100 conversations were recorded: 44 intersections, 19 bypassing, and 37 pedestrian scenarios.

[4]The dialogue system can produce many (different) instances, but to keep the stimuli contained, we limit it to 3.



(a) With execution errors

(b) without execution errors

Figure 5: Commutative scenario generation success over the number of dialogue turns in human evaluation with (5a) and without (5b) execution errors. For example, if, on the 3rd turn, the system fails to produce executable code and the user is asked to paraphrase, leading to successful scenario generation, we count this as success on 4 dialogue turns with execution errors and 3 dialogue turns without execution errors. The results show a positive correlation between the successful scenario generation and the number of dialogue turns for all types of scenarios considered.

4.2.2. Results and Discussion

Figure 5 outlines the results of human evaluation. As the number of dialogue turns increases, so does the percentage of successfully generated scenarios: without dialogue, only 6% of the code generations were successful, while conversations of up to four turns raise this to 28%. This holds per scenario type, with or without execution errors. This suggests that the *extended conversation enables successful scenario generation*. Table 2 gives examples of a successful and failed scenario generation. Error analysis shows that failed scenario generation conversations had one or more of these errors:

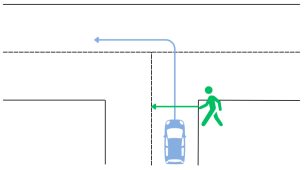
- Execution error (62.5%): the code produced does not execute, and the user is asked to paraphrase one or more times. This happens due to a missing correspondence between Scenic API and the description. E.g. the description features "T-junction", while the Scenic API would expect 3-way intersection, and the code features `junction` as a primitive rather than `intersection`.
- Pass error (82%): the code executes but does not pass as a correct simulation. It is because the user's description diverges from descriptions in the dataset: in length (training data descriptions were 30-50 tokens while user descriptions were up to 200 tokens); in syntax (training data descriptions were single sentences while users expressed multiple sentences and fragments); and in narrative style (training data descriptions were ego-centric while users expressed other perspectives).
- Summarization error (27.8%): errors arising due to summarization (§3.2) like failure to include the feedback in the description or the updated description has extra information.

5. Conclusions

We have presented a dialogue system for generating conversational driving scenarios. Our results show that users can, in principle, generate desired driving scenarios with conversation being a crucial element in reducing scenario generation errors.

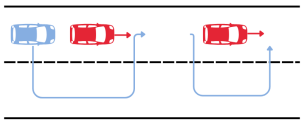
For future work, we envision several developments. Firstly, there is a gap in generating scenarios, despite the extended conversation. To address this, further techniques could be considered, including data augmentation [35], constrained decoding [36], prompt engineering [37], and memory-based techniques for explicit state tracking [38]. Second, the current setup does not dynamically change the controller and use the interface to test it. This should be evaluated and considered testing of such new controllers in tandem with formal verification methods [39].

Table 2
Example conversation from human trials (§4.2). Description in red signified that it was generated using previous description and the user's feedback using `SUMM` prompt (§3.2)



Turn	Feedback f	Description d	Simulation
1	-	*at a 3-way intersection, a pedestrian crosses the road. An ego vehicle waits to make a left turn.*	
2	*first the pedestrian crosses, then ego vehicle moves.*	*At a 3-way intersection, a pedestrian crosses the road and an ego vehicle waits to make a left turn.*	
3	*the pedestrian crosses the road on which the ego vehicle enters*	*At a 3-way intersection, an ego vehicle waits to make a left turn, and a pedestrian crosses the road.*	
4	*the pedestrian is to the right of the vehicle*	*At a 3-way intersection, an ego vehicle waits to make a left turn, and a pedestrian crosses the road to the right.*	

Outcome: simulation generated successfully



Turn	Feedback f	Description d	Simulation
1	-	*An ego vehicle overtakes an adversary vehicle, then overtakes another adversary vehicle*	
2	*The ego vehicle is on a dual carriageway behind two adversary vehicles in the left lane*	*An ego vehicle overtakes an adversary vehicle, then overtakes another adversary vehicle on a dual carriageway behind two adversary vehicles in the left lane.*	
3	*The road has two lanes and all vehicles begin in the left lane*	*An ego vehicle overtakes an adversary vehicle, then overtakes another adversary vehicle on a dual carriageway behind two adversary vehicles in the left lane.*	
4	*The road only has two lanes*	*An ego vehicle overtakes an adversary vehicle, then overtakes another adversary vehicle on a dual carriageway behind two adversary vehicles in the left lane.*	

Outcome: failed to generate the simulation

Acknowledgments

This work was supported by UKRI Strategic Priorities Fund to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regulation (grant EP/V026607/1) and UKRI Turing AI World Leading Researcher Fellowship on AI for Person-Centred and Teachable Autonomy (grant EP/Z534833/1)

Declaration on Generative AI

During the preparation of this work, the author(s) used Grammarly in order to do the following: Grammar and spell checking; generating a paraphrase or alterantive word. After using this tool, the author(s) reviewed and edited the content as needed, and we take full responsibility for the publication's content.

References

[1] N. Kalra, S. M. Paddock, Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?, Transportation Research Part A: Policy and Practice 94 (2016) 182–193. URL: https://www.sciencedirect.com/science/article/pii/S0965856416302129. doi:https://doi.org/10.1016/j.tra.2016.09.010.

[2] H.-P. Schöner, Simulation in development and testing of autonomous vehicles, in: M. Bargende, H.-C. Reuss, J. Wiedemann (Eds.), 18. Internationales Stuttgarter Symposium, Springer Fachmedien Wiesbaden, Wiesbaden, 2018, pp. 1083–1095.

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, V. Koltun, CARLA: an open urban driving simulator, in: 1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings, volume 78 of *Proceedings of Machine Learning Research*, PMLR, 2017, pp. 1–16. URL: http://proceedings.mlr.press/v78/dosovitskiy17a.html.

[4] D. J. Fremont, E. Kim, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, S. A. Seshia, Scenic: a language for scenario specification and data generation, Mach. Learn. 112 (2023) 3805–3849. URL: https://doi.org/10.1007/s10994-021-06120-5. doi:10.1007/S10994-021-06120-5.

[5] Q. Li, Z. Peng, L. Feng, Q. Zhang, Z. Xue, B. Zhou, Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning, IEEE Trans. Pattern Anal. Mach. Intell. 45 (2023) 3461–3475. URL: https://doi.org/10.1109/TPAMI.2022.3190471. doi:10.1109/TPAMI.2022.3190471.

[6] Q. Li, Z. Peng, L. Feng, Z. Liu, C. Duan, W. Mo, B. Zhou, Scenarionet: Open-source platform for large-scale traffic scenario simulation and modeling, Advances in Neural Information Processing Systems (2023).

[7] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, R. Lowe, Training language models to follow instructions with human feedback, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems, volume 35, Curran Associates, Inc., 2022, pp. 27730–27744. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

[8] Y. W. Wong, R. Mooney, Learning for semantic parsing with statistical machine translation, in: R. C. Moore, J. Bilmes, J. Chu-Carroll, M. Sanderson (Eds.), Proceedings of the Human Language Technology Conference of the NAACL, Main Conference, Association for Computational Linguistics, New York City, USA, 2006, pp. 439–446. URL: https://aclanthology.org/N06-1056/.

[9] L. Zettlemoyer, M. Collins, Learning context-dependent mappings from sentences to logical form, in: K.-Y. Su, J. Su, J. Wiebe, H. Li (Eds.), Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing

of the AFNLP, Association for Computational Linguistics, Suntec, Singapore, 2009, pp. 976–984. URL: https://aclanthology.org/P09-1110/.

[10] L. Dong, M. Lapata, Language to logical form with neural attention, in: K. Erk, N. A. Smith (Eds.), Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Berlin, Germany, 2016, pp. 33–43. URL: https://aclanthology.org/P16-1004/. doi:10.18653/v1/P16-1004.

[11] J. Jiang, F. Wang, J. Shen, S. Kim, S. Kim, A survey on large language models for code generation, CoRR abs/2406.00515 (2024). URL: https://doi.org/10.48550/arXiv.2406.00515. doi:10.48550/ARXIV.2406.00515. arXiv:2406.00515.

[12] D. Kocetkov, R. Li, L. B. Allal, J. Li, C. Mou, Y. Jernite, M. Mitchell, C. M. Ferrandis, S. Hughes, T. Wolf, D. Bahdanau, L. von Werra, H. de Vries, The stack: 3 TB of permissively licensed source code, Trans. Mach. Learn. Res. 2023 (2023). URL: https://openreview.net/forum?id=pxpbTdUEpD.

[13] S. Feng, X. Yan, H. Sun, Y. Feng, H. X. Liu, Intelligent driving intelligence test for autonomous vehicles with naturalistic and adversarial environment, Nature Communications 12 (2021) 748. URL: https://doi.org/10.1038/s41467-021-21007-8. doi:10.1038/s41467-021-21007-8.

[14] B. Padmaja, C. V. K. N. S. N. Moorthy, N. Venkateswarulu, M. M. Bala, Exploration of issues, challenges and latest developments in autonomous cars, Journal of Big Data 10 (2023) 61. URL: https://doi.org/10.1186/s40537-023-00701-y. doi:10.1186/s40537-023-00701-y.

[15] A. Corso, R. J. Moss, M. Koren, R. Lee, M. J. Kochenderfer, A survey of algorithms for black-box safety validation of cyber-physical systems, J. Artif. Intell. Res. 72 (2021) 377–428. URL: https://doi.org/10.1613/jair.1.12716. doi:10.1613/JAIR.1.12716.

[16] H. Rosenbrock, Designing human-centred technology: a cross-disciplinary project in computer-aided manufacturing, 1989. URL: https://api.semanticscholar.org/CorpusID:106694042.

[17] C. Chang, S. Wang, J. Zhang, J. Ge, L. Li, Llmscenario: Large language model driven scenario generation, IEEE Transactions on Systems, Man, and Cybernetics: Systems 54 (2024) 6581–6594. doi:10.1109/TSMC.2024.3392930.

[18] L. Feng, Q. Li, Z. Peng, S. Tan, B. Zhou, Trafficgen: Learning to generate diverse and realistic traffic scenarios, in: 2023 IEEE International Conference on Robotics and Automation (ICRA), 2023, pp. 3567–3575. doi:10.1109/ICRA48891.2023.10160296.

[19] A. V. Miceli Barone, C. Innes, A. Lascarides, Dialogue-based generation of self-driving simulation scenarios using large language models, in: A. Padmakumar, M. Inan, Y. Fan, X. Wang, M. Alikhani (Eds.), Proceedings of the 3rd Combined Workshop on Spatial Language Understanding and Grounded Communication for Robotics (SpLU-RoboNLP 2023), Association for Computational Linguistics, Singapore, 2023, pp. 1–12. URL: https://aclanthology.org/2023.splurobonlp-1.1/. doi:10.18653/v1/2023.splurobonlp-1.1.

[20] J. Zhang, C. Xu, B. Li, Chatscene: Knowledge-enabled safety-critical scenario generation for autonomous vehicles, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 15459–15469.

[21] H. P. Grice, Utterer's meaning and intentions, Philosophical Review 68 (1969) 147–177.

[22] H. Chen, X. Liu, D. Yin, J. Tang, A survey on dialogue systems: Recent advances and new frontiers, SIGKDD Explor. Newsl. 19 (2017) 25–35. URL: https://doi.org/10.1145/3166054.3166058. doi:10.1145/3166054.3166058.

[23] J. Ni, T. Young, V. Pandelea, F. Xue, E. Cambria, Recent advances in deep learning based dialogue systems: a systematic survey, Artificial Intelligence Review 56 (2023) 3055–3155. URL: https://doi.org/10.1007/s10462-022-10248-8. doi:10.1007/s10462-022-10248-8.

[24] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, T. Scialom, Toolformer: Language models can teach themselves to use tools, in: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), Advances in Neural Information Processing Systems, volume 36, Curran Associates, Inc., 2023, pp. 68539–68551. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

[25] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-

intensive nlp tasks, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[26] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[27] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de Las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, W. E. Sayed, Mistral 7b, CoRR abs/2310.06825 (2023). URL: https://doi.org/10.48550/arXiv.2310.06825. doi:10.48550/ARXIV.2310.06825. arXiv:2310.06825.

[28] T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, P. Tafti, L. Hussenot, A. Chowdhery, A. Roberts, A. Barua, A. Botev, A. Castro-Ros, A. Slone, A. Héliou, A. Tacchetti, A. Bulanova, A. Paterson, B. Tsai, B. Shahriari, C. L. Lan, C. A. Choquette-Choo, C. Crepy, D. Cer, D. Ippolito, D. Reid, E. Buchatskaya, E. Ni, E. Noland, G. Yan, G. Tucker, G. Muraru, G. Rozhdestvenskiy, H. Michalewski, I. Tenney, I. Grishchenko, J. Austin, J. Keeling, J. Labanowski, J. Lespiau, J. Stanway, J. Brennan, J. Chen, J. Ferret, J. Chiu, et al., Gemma: Open models based on gemini research and technology, CoRR abs/2403.08295 (2024). URL: https://doi.org/10.48550/arXiv.2403.08295. doi:10.48550/ARXIV.2403.08295. arXiv:2403.08295.

[29] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. Canton-Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, G. Synnaeve, Code llama: Open foundation models for code, CoRR abs/2308.12950 (2023). URL: https://doi.org/10.48550/arXiv.2308.12950. doi:10.48550/ARXIV.2308.12950. arXiv:2308.12950.

[30] C. M. Bishop, Pattern recognition and machine learning, 5th Edition, Information science and statistics, Springer, 2007. URL: https://www.worldcat.org/oclc/71008143.

[31] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu, Bleu: a method for automatic evaluation of machine translation, in: P. Isabelle, E. Charniak, D. Lin (Eds.), Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 2002, pp. 311–318. URL: https://aclanthology.org/P02-1040/. doi:10.3115/1073083.1073135.

[32] M. Post, A call for clarity in reporting BLEU scores, in: O. Bojar, R. Chatterjee, C. Federmann, M. Fishel, Y. Graham, B. Haddow, M. Huck, A. J. Yepes, P. Koehn, C. Monz, M. Negri, A. Névéol, M. Neves, M. Post, L. Specia, M. Turchi, K. Verspoor (Eds.), Proceedings of the Third Conference on Machine Translation: Research Papers, Association for Computational Linguistics, Brussels, Belgium, 2018, pp. 186–191. URL: https://aclanthology.org/W18-6319/. doi:10.18653/v1/W18-6319.

[33] C.-Y. Lin, ROUGE: A package for automatic evaluation of summaries, in: Text Summarization Branches Out, Association for Computational Linguistics, Barcelona, Spain, 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013/.

[34] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, S. Ma, Codebleu: a method for automatic evaluation of code synthesis, CoRR abs/2009.10297 (2020). URL: https://arxiv.org/abs/2009.10297. arXiv:2009.10297.

[35] P. Chen, G. Lampouras, Exploring data augmentation for code generation tasks, in: A. Vlachos, I. Augenstein (Eds.), Findings of the Association for Computational Linguistics: EACL 2023, Association for Computational Linguistics, Dubrovnik, Croatia, 2023, pp. 1542–1550. URL: https://aclanthology.org/2023.findings-eacl.114/. doi:10.18653/v1/2023.findings-eacl.114.

[36] S. Geng, M. Josifoski, M. Peyrard, R. West, Grammar-constrained decoding for structured NLP

tasks without finetuning, in: H. Bouamor, J. Pino, K. Bali (Eds.), Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 10932–10952. URL: https://aclanthology.org/2023.emnlp-main.674/. doi:10.18653/v1/2023.emnlp-main.674.

[37] M. L. Siddiq, B. Casey, J. C. S. Santos, A lightweight framework for high-quality code generation, CoRR abs/2307.08220 (2023). URL: https://doi.org/10.48550/arXiv.2307.08220. doi:10.48550/ARXIV.2307.08220. arXiv:2307.08220.

[38] P. Jain, M. Lapata, Memory-based semantic parsing, Transactions of the Association for Computational Linguistics 9 (2021) 1197–1212. URL: https://aclanthology.org/2021.tacl-1.71/. doi:10.1162/tacl_a_00422.

[39] Y. Wang, M. Nakamura, K. Sakakibara, Y. Okura, Formal specification and verification of an autonomous vehicle control system by the ots/cafeobj method (S), in: S. Chang (Ed.), The 35th International Conference on Software Engineering and Knowledge Engineering, SEKE 2023, KSIR Virtual Conference Center, USA, July 1-10, 2023, KSI Research Inc., 2023, pp. 363–366. URL: https://doi.org/10.18293/SEKE2023-170. doi:10.18293/SEKE2023-170.