# Vibe Coding in Practice: Building a Driving Simulator Without Expert Programming Skills

Margarida Fortes-Ferreira
m.raposo.fortes.ferreira@umail.leidenuniv.nl
Leiden University
Leiden, The Netherlands

Md Shadab Alam*
m.s.alam@tue.nl
Eindhoven University of Technology
Eindhoven, The Netherlands

Pavlo Bazilinskyy
p.bazilinskyy@tue.nl
Eindhoven University of Technology
Eindhoven, The Netherlands

## Abstract

The emergence of large language models has introduced new opportunities in software development, particularly through a revolutionary paradigm known as vibe coding or "coding by vibes", in which developers express their software ideas in natural language and where the LLM generates the code. This paper investigates the potential of vibe coding to support novice programmers. The first author, without coding experience, attempted to create a 3D driving simulator using the Cursor platform and Three.js. The iterative prompting process improved the simulation's functionality and visual quality. The results indicated that LLM can reduce barriers to creative development and expand access to computational tools. However, challenges remain: prompts often required refinements, output code can be logically flawed, and debugging demanded a foundational understanding of programming concepts. These findings highlight that while vibe coding increases accessibility, it does not completely eliminate the need for technical reasoning and understanding prompt engineering.

## Keywords

Vibe Coding, Large Language Models (LLMs), Driving Simulator, Prompts

**ACM Reference Format:**
Margarida Fortes-Ferreira, Md Shadab Alam, and Pavlo Bazilinskyy. 2025. Vibe Coding in Practice: Building a Driving Simulator Without Expert Programming Skills. In *17th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI Adjunct '25), September 21–25, 2025, Brisbane, QLD, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3744335.3758482

## 1 Introduction

The emergence of Artificial Intelligence (AI) technologies has significantly impacted individuals, organisations and society in various fields, from education [28] to healthcare [26], and from scientific research [27] to computer science [12]. In the contemporary data-driven landscape, Generative AI (Gen AI), particularly Large Language Models (LLMs), has emerged as a significant technological advancement [16]. Serving as a subset of AI that seeks to understand and generate human-like language, LLMs have expanded the scope of Natural Language Processing (NLP), enabling new functionalities in a wide range of applications, such as machine translation [24], text summarisation [13] question answering [29], and, since recently, code generation [23]. These models are trained in vast repositories of publicly available texts, including books, articles, and websites, enabling them to produce coherent responses and engage in complex linguistic tasks [21]. By enabling more intuitive, human-centred computing, LLMs are reshaping how people interact with digital systems and transforming the way developers approach their work [18]. This shift toward user-centric design is also evident in automated mobility interfaces, where clear and accessible information presentation plays a key role in improving user experience [5].

Early work on using LLMs for code generation began with models like Code2vec, trained on millions of code from GitHub repositories [7]. These early systems showed that machine learning could effectively capture and represent the structure of code. Later research tackled some of the more difficult problems in translating natural language into programming syntax, such as dealing with the huge variety of variable names and the abstraction of logic, leading to more capable models such as CodeBERT [14] and Code2Seq [6]. This sets the stage for a new generation of large, general-purpose models that can take on a wide range of development tasks. In practice, this has led to tools like Codex (https://openai.com/index/introducing-codex/), GitHub Copilot (https://github.com/features/copilot), Replit Ghostwriter (https://replit.com/learn/intro-to-ghostwriter), and Cursor (https://www.cursor.com) often described as "AI Pair Programmers". These tools vary in approach: some act as general-purpose plugins for existing code editors, others are built for specific platforms or companies, and some introduce entirely new ways of coding [19]. While they can boost productivity, code quality, and efficiency, they also have limitations, including potential bias and reliability issues arising from their non-human nature [18].

Building upon these trends, a novel paradigm in software development has emerged: vibe coding. This concept was popularised by Andrej Karpathy in his tweet in February 2025 (https://x.com/karpathy/status/1886192184808149383). According to Maes [19], vibe coding is an AI-assisted development approach in which the developer articulates their software idea using natural language and the AI automatically generates the corresponding code. Rather than manually writing code, developers express their ideas using natural language, often in the form of concise, high-level prompts, or informal descriptions, and the AI translates this input into executable code. Hence, this paradigm emphasises intent over implementation, thereby shifting the developer's role from coder to conceptual

*Corresponding Author


*AutomotiveUI Adjunct '25, Brisbane, QLD, Australia*

ACM ISBN 979-8-4007-2014-7/2025/09
https://doi.org/10.1145/3744335.3758482

guide. Although early forms of this approach emerged nearly a decade ago using non-LLM AI [20], the current generation of tools has made vibe coding more practical and accessible. Developers now provide succinct high-level prompts, known as "vibes", such as single-sentence feature requests, which AI interprets and transforms into functional code. This reduces the barriers to entry for software development and has the potential to democratise the field, making it more accessible to individuals without formal programming training.

According to Pajo [22], despite its promising benefits, such as increased accessibility, improved efficiency, and greater creative freedom, vibe coding also presents significant challenges. These challenges encompass concerns about the quality and maintainability of AI-generated code, potential security vulnerabilities, as well as changes in the developer skill set that may prioritise high-level design and communication over traditional coding expertise. As the adoption of vibe coding escalates, continuous research and critical evaluation will be imperative to ensure that this paradigm effectively integrates human creativity with the generative power of AI while adequately addressing its inherent risks.

Recent developments in AI-assisted software development, epitomised by Pieter Levels' *fly.pieter.com*, a browser-based MMO flight simulator built in just 30 minutes using AI tools, highlight the creative potential of vibe coding. This emerging methodology involves telling an AI what you want instead of writing detailed code, enabling rapid prototyping and iteration. Levels claims that the game now earns more than $50,000 a month (https://www.404media.co/this-game-created-by-ai-vibe-coding-makes-50-000-a-month-yours-probably-wont/), and another report notes a record of 31,000 players online in a single day, with a consistent base of around 50 players online at all times (https://dev.ua/en/news/kalichna-shi-hra-zarobyla-vzhe-87k-baksiv-na-reklami-1741866359). This shows how quickly a prototypical project can become a revenue-generating product. Still, critics caution that such success is often tied to the developer's own audience and brand, and that issues like security, scalability, and replicability are far from settled. A similar direction is exemplified by Mirage [8], an AI-driven generative game engine prototype developed by Dynamics Lab and described as the world's first real-time "world model engine", which enables video game worlds to be created and modified in real time. This increasing accessibility also presents new opportunities for sectors beyond entertainment. Specifically, driving simulators have long been recognised as valuable tools for human factors research on automated driving and traffic safety [9–11, 15].

These developments underscore the creative opportunities and practical limitations of AI-assisted programming for simulation environments. Although AI tools enable the development of interactive 3D experiences using natural language, there is limited research understanding of how accessible and effective these workflows are for nonexperts, particularly when applied to complex systems requiring realistic physics and graphics, such as driving simulators [2]. Moreover, advances in deep learning, including generative adversarial networks (GAN), have enabled the creation of highly realistic synthetic traffic scenarios, further enhancing the fidelity and applicability of simulation-based research [3, 4].

### 1.1 Aim of Study

The aim of this study was to explore how vibe coding, a novel AI-assisted natural language programming approach, can support non-expert developers in creating complex driving simulator. The research question guiding this work was: *How can vibe coding support non-expert developers in creating complex driving simulator?* We conducted an exploratory case study in which the first author with no prior coding experience attempted to build a 3D driving simulator with an infinite, procedurally generated world using the Cursor platform. This study investigated how effectively prompt-driven development reduces barriers to entry in simulator creation while highlighting the challenges, advantages, and iterative strategies involved.

## 2 Method

The study tested the development of an interactive 3D driving simulator using a locally installed version of the Cursor platform (Version 1.3.9, based on VS Code Version 1.99.3, Electron 34.5.1, Chromium 132.0.6834.210, Node.js 20.19.0). Cursor incorporates LLMs into a customised version of the Visual Studio Code editor, enabling users to write and enhance code through natural language prompts. Furthermore, this platform promotes conversational workflows, allowing users to describe what they want the programme's functionalities to be, with the model generating or modifying code in response. In addition, the platform aids debugging by giving recommendations and alternatives when faced with challenges, making it particularly beneficial for users with limited programming backgrounds.

The simulator was developed by the first author without a background in programming. The first author used an iterative trial-and-error approach to guide the LLM using various natural language prompts, experimenting with different wordings, levels of detail, and instructional strategies, referred to as "vibes". For example, the prompts included sentences such as "Make the car more realistic" or "I want to have a more realistic city". Cursor responded with code suggestions, which the first author tested in real time and reviewed through follow-up prompts. When the initial outputs were incomplete or flawed, the first author refined the prompt by rephrasing instructions, adding specific requirements, or copy-pasting the execution error messages into Cursor to request targeted corrections.

This project involved Three.js, a JavaScript library for creating and rendering 3D environments in the browser (https://threejs.org). The library was selected for its lightweight structure and compatibility with web-based rendering, making it a popular choice for interactive visual applications. Three.js also offers an extensive set of built-in geometries, materials, and lighting options, which accelerates the development process. Its large and active community provides tutorials, examples, and open source resources, making it easier for beginners to find solutions and integrate advanced features compared to many alternative 3D libraries.

The development process was structured around a series of milestones, each focused on implementing a particular feature or functionality, such as city realism, road generation, camera control, or physics and function of cars. These milestones were arranged in a logical progression, beginning with the creation of a basic scene,

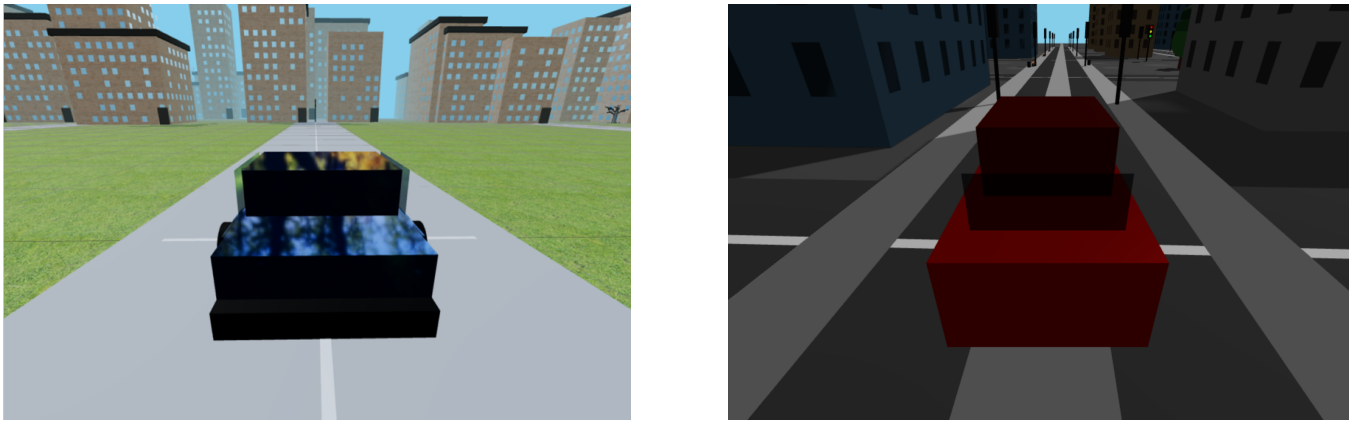

**Figure 1: Scenes generated with Prototype 1 (left) and Prototype 2 (right). The left image shows a blue car with a glossy, reflective surface viewed from behind in a stylised city featuring realistic grass, buildings, and trees. Although this prototype offered a more photorealistic environment, it lacked interactivity, with the car and other elements remaining static. The right image depicts a red car navigating an AI-generated city with buildings, roads, traffic lights, and other environmental details. This prototype included functional driving mechanics, such as acceleration, but its overall visual polish was lower than that of Prototype 1.**

followed by enhancements to realism, interactivity, and data integration. The order was chosen to ensure that the fundamental elements of the simulator (scene setup, vehicle behaviour, and city layout) were functional before adding more complex or resource-intensive features (e.g., photorealistic rendering, API integration). Two separate prompt sequences were performed during development, each representing a distinct build approach within Cursor. The first sequence exemplified in Appendix A started with creating a minimal scene and progressively adding complexity in small iterative steps. This approach prioritised testing each new feature (e.g., vehicle physics, city layout and environment, materials) before moving on to the next, with many refinements focused on realism (e.g., buildings, trees, car dimensions). The second sequence as exemplified in Appendix B started from a higher-level city scene prompt, requesting more elements at once (i.e., roads, sidewalks, buildings, trees) and then refining specific details such as street' features, vehicle controls, and physics.

The simulator was developed and tested on a MacBook Air (Retina, 13-inch, 2020) running macOS Monterey 12.7.6, with a 1.2 GHz Intel Core i7 quad-core processor, 8 GB 3733 MHz LPDDR4X RAM, and Intel Iris Plus Graphics (1.5 GB). The development was carried out using the locally installed Cursor application, while testing the games in the Google Chrome browser (version.138.0.7204.184).

## 3 Results

### 3.1 Prototypes of the Simulator

Two interactive 3D driving simulator prototypes were developed by the first author using the Cursor platform and Three.js. Each version reflected a distinct stage in the prompt-based workflow, differing in visual aesthetics, camera perspective, vehicle physics and functionality, and environmental complexity. These variations resulted from the LLM's differing interpretations of natural language directives. Table 1 compares six core features—camera perspective, road generation, vehicle functionality, vehicle physics, environment, and interactivity—across both versions. Representative examples are shown in Figure 1.

Each version was generated using natural language prompts in Cursor, without direct code manipulation. An example of prompts included "Create a basic Three.js scene with a ground plane, ambient light, and a perspective camera", "Make the car more realistic, with the right materials and a blue colour", and "Add traffic lights, trees, and sidewalks to the city". The differences in output emerged from the way the prompts were interpreted, refined, or corrected in dialogue with the AI assistant. For example, prompts emphasising realism led to adding buildings and lights, while more open-ended prompts resulted in abstract landscapes. The final version of the simulator is available at https://github.com/Shaadalam9/vibe-simulator.

### 3.2 Observations on Prompt Iteration

During the initial stages of the iterative process, the Cursor platform repeatedly delayed processing and responding to the prompts. This system was slow to provide outputs or generate appropriate code suggestions, temporarily hindering the rapid iteration that was expected with vibe coding. As the tool, the workflow, and the familiarity of participants with the Cursor was developed, its interaction model improved both in responsiveness and efficiency. Thus, this evolution enabled for more seamless experimentation and development in the later stages. Furthermore, during all attempts to develop prototypes of a driving simulator with a single prompt, the prompt did not produce the desired result. Many core features—particularly those that needed the integration of multiple systems, such as city realism and functioning vehicle controls—were developed through an iterative process. The first author often provided follow-up prompts (as illustrated in Appendix A and Appendix B). It took multiple iterations to achieve a usable result for each feature in diverse cases.

**Table 1: Feature comparison between Prototypes 1 and 2 of simulator in Figure 1.**

| Feature | Left Image | Right Image |
|---|---|---|
| Camera Perspective | - Global overview of the scenario | - Third-person perspective |
| Road Generation | - Consistent geometry across map | - Self-generated road network<br>- Includes intersections and traffic lights |
| Vehicle Functionality | - Static visual model<br>- No driving input or interactivity | - W-A-S-D commands-based movement<br>- No collision detection |
| Vehicle Physics | - Include wheels, reflective properties<br>- Blue | - Red<br>- Simple with a cube shape |
| Environment | - Realistic materials on grass and buildings<br>- Static trees, no interaction | - Procedurally generated city<br>- Dynamic street elements and complex layout |
| Interactivity | - Fully static scene<br>- Primarily visual demonstration | - User-driven navigation |

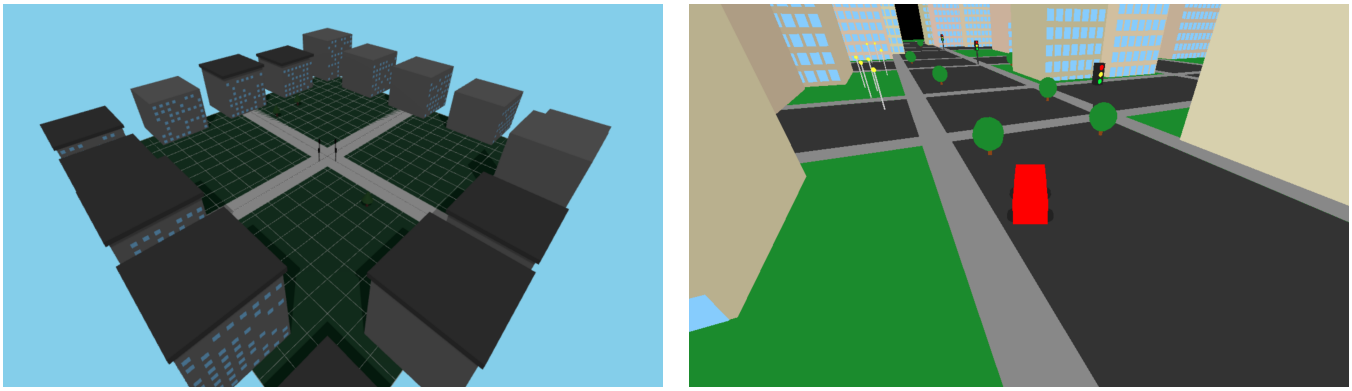

**Figure 2: The in-process outputs are presented in Appendix A and Appendix B.**

## 4 Discussion

The results of this study demonstrate the significant potential of vibe coding to transform how non-experts engage in software development, particularly in the context of building complex simulation environments. Using LLM and the Cursor platform, the first author was able to produce two functional prototypes of a 3D driving simulator without writing any manual code. This process highlighted several key outcomes related to the effectiveness and nuances of prompt-driven development.

First, vibe coding substantially reduced the traditional barriers to entry for software and game development. The first author, without any prior coding background, successfully developed functional simulation scenarios by iteratively refining natural language prompts. This demonstrates that LLMs can enable users from diverse, non-technical backgrounds to translate their ideas into working applications, thereby lowering the barrier to entry for coding and expanding participation in creative computational tool development.

Second, the quality and complexity of the prototypes were closely tied to the specificity and clarity of the instructions. For example, a more abstract prompt, such as "Make the car even more realistic!" (P10, see Appendix A) often yielded unstable or incomplete results, such as missing textures or misaligned proportions. Similarly, prompts that combined technical and aesthetic goals, such as "Add pedestrian elements, benches, and bus stops to the city" (P4, see Appendix B), produced more visually complex scenes, but also introduced instability, including non-functional traffic lights and frame-rate drops. By comparison, narrower single focus prompts such as "Use realistic acceleration and turning logic" (P2, Appendix A) generated clearer and more reliable outcomes, although with less visual sophistication. These observations highlight the importance of clarity, specificity, and iterative refinement when using LLMs for simulation development. The achievement of advanced interactivity and realism required multiple refinement cycles, which underscores the importance of clear communication and iterative "dialogue" with the AI system. This highlights the limitations and advantages of LLM-assisted coding: achieving both interactivity and realism often requires multiple trial-and-error prompting.

In addition, the study revealed that effective use of vibe coding requires an emerging skill set centred on prompt literacy. The ability of the first author to analyse the results, recognise when the results did not match their intentions, and iteratively adjust their commands was crucial to achieving satisfactory results. Thus, while vibe coding shifts software development away from traditional programming, it also elevates the role of design thinking, system analysis, and structured communication. This creates an important reflection: Although LLMs can support the coding process, they do not eliminate the need for computational thinking and knowledge. *As programming becomes increasingly mediated by natural language*

*systems, future developers will need new forms of coding literacy, including the ability to translate ideas into precise prompts.* Coding may transform from a hands-on technical endeavour to a high-level design conversation, but the skills to analyse systems, identify errors, and organise logic remain crucial.

## 4.1 Limitations and Future Work

This study involved a single participant with no prior programming experience, who acted as both the developer and the evaluator of an LLM-assisted development workflow ("vibe coding") implemented in Cursor, providing natural language prompts to generate and iteratively refine Three.js based simulator code. Although this allowed for a detailed qualitative exploration of vibe coding, it inherently limits the generalisability of the findings. Also, it is not possible to determine whether the results observed in this study were caused specifically by vibe coding or other factors present in the development setup. The development process combined several elements, such as the features and interface of the Cursor platform, the choice of libraries (e.g., Three.js), the performance of the hardware used, and the way the prompts were written and refined. Because these factors were not varied, controlled, or tested independently, their individual impacts cannot be separated. As a result, the influence of vibe coding itself cannot be isolated from the broader combination of tools, configurations, and workflows that shaped the development process.

A further limitation concerns the complexity and fidelity of the simulation environments generated. Although Cursor enabled the creation of functional prototypes, achieving high levels of realism such as photorealistic visuals and nuanced city dynamics proved challenging. The development process often involved repeated prompt refinement, trial-and-error, and substantial user guidance, especially when prompts were vague or expressed only high-level creative ideas. The quality of the AI-generated output depended heavily on the clarity and specificity of the prompts. Simple instructions generally produced more reliable results, while abstract or complex requests often resulted in incomplete or flawed code that required user intervention and multiple reformulations. Performance limitations also became apparent. For example, adding numerous glass elements to building windows significantly slowed loading and rendering, and attempts to integrate a 3D map of Leiden (The Netherlands) to produce a realistic city layout were difficult, requiring extensive prompt adjustments and still producing incomplete and unrealistic output.

These observations underscore the importance of prompt literacy, which means the ability to communicate ideas in ways that AI systems can reliably translate into effective code. Although vibe coding reduces the need for explicit programming syntax, it does not remove the need for technical reasoning, logical thinking, or understanding of dependencies. The study was also limited by a short development period, which limited the depth of exploration and refinement of advanced simulator features. In addition, the exclusive use of the Cursor platform represents only one example of an LLM-assisted development environment, so the findings may be specific to this tool.

Future research should expand participation to include people from a variety of backgrounds and levels of technical expertise to better evaluate the applicability, consistency, and accessibility of AI-assisted coding between user groups. Incorporating questionnaires and semi-structured interviews could provide deeper insight into user experience, perceived usability, satisfaction, and trust. The perceived usability here refers to the subjective assessment of the user of how easy and pleasant the system is to use, often measured through standardised tools such as the System Usability Scale (SUS), which captures opinions on learnability, efficiency, and overall satisfaction.

To address causal ambiguity, we recommend controlled comparisons. One practical approach is a within-subjects, counterbalanced 2×2 design that manipulates (i) the coding environment (e.g., Cursor vs. widely used LLM-assisted tools such as ChatGPT or Perplexity, compared against a standard IDE without AI-assisted inline code generation) and (ii) prompt style. Designed in a conversational language (e.g., "Make the city more realistic"), while structured criteria-driven prompts specify measurable requirements in detail, often as numbered criteria (e.g., "Create a city scene with at least 10 unique building models, each textured with windows, doors, and rooftops; add a road network with intersections and functioning traffic lights; place trees every 10 metres; maintain a minimum frame rate of 30 FPS on mid-range laptops; and simulate daylight with shadows enabled"). Keeping libraries, hardware, and model configurations constant, primary outcome measures could include task success and time-to-completion, while secondary measures could include error counts, number of LLM interactions, workload (e.g., NASA-TLX [17]), perceived usability (SUS) [25] and blinded code quality ratings [1], where independent reviewers evaluate correctness, readability, maintainability, and efficiency of code without knowing which tool produced it.

Longer longitudinal studies would clarify how extended iterative prompting and continued human–AI collaboration affect sophistication, functionality, and overall quality. Comparative evaluations across AI coding environments (e.g. Cursor, GitHub Copilot, and other LLM-assisted tools) should examine differences in usability, code quality, error handling, and user trust.

On the technical side, future work should aim to build more sophisticated and controlled simulation environments, both to advance fidelity (e.g., asset optimisation, level-of-detail strategies, material/shader efficiency, robust map-data pipelines) and to enable empirical studies of human behaviour, decision making, and traffic safety. Improving reproducibility by releasing complete prompt logs, pinned configurations, and tagged repositories will also help the community verify and extend these findings.

## References

[1] Murad Alam, NA Kim, Jillian Havey, Alfred Rademaker, Desiree Ratner, Barbara Tregre, Dennis P West, and William P Coleman III. 2011. Blinded vs. unblinded peer review of manuscripts submitted to a dermatology journal: a randomized multi-rater study. *British Journal of Dermatology* 165, 3 (2011), 563–567. doi:10.1111/j.1365-2133.2011.10432.x

[2] Md Shadab Alam and Pavlo Bazilinskyy. 2025. Cross or Nah? LLMs get in the mindset of a pedestrian in front of automated car with an eHMI. In *Adjunct Proceedings of the 17th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutoUI)*. Brisbane, QLD, Australia. doi:10.1145/3744335.3758477

[3] Md Shadab Alam, Marieke H Martens, and Pavlo Bazilinskyy. 2025. Generating Realistic Traffic Scenarios: A Deep Learning Approach Using Generative Adversarial Networks (GANs). In *13th International Conference on Human Interaction & Emerging Technologies: Artificial Intelligence & Future Applications, IHIET-AI 2025.*

AHFE International, 349–358. doi:10.54941/ahfe1005927
[4] Md Shadab Alam, Sagar Hitendra Parmar, Marieke H. Martens, and Pavlo Bazilinskyy. 2025. Deep Learning Approach for Realistic Traffic Video Changes Across Lighting and Weather Conditions. In *2025 8th International Conference on Information and Computer Technologies (ICICT)*. Hilo, HI, USA, 180–185. doi:10.1109/ICICT64582.2025.00034
[5] Md Shadab Alam, Thirumanikandan Subramanian, Marieke Martens, Wolfram Remlinger, and Pavlo Bazilinskyy. 2024. From A to B with ease: User-centric interfaces for shuttle buses. In *Adjunct Proceedings of the 16th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutoUI)*. Stanford, CA, USA. doi:10.1145/3641308.3685033
[6] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2018. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400* (2018). doi:10.48550/arXiv.1808.01400
[7] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29. doi:10.48550/arXiv.1803.09473
[8] Abdalla Bayoumi. 2025. Mirage AI Game Engine: World's First Real-Time AI Gaming Platform That Creates Games While You Play. https://aiixx.ai/blog/mirage-ai-game-engine-worlds-first-real-time-ai-gaming-platform-that-creates-games-while-you-play. Accessed: August 6, 2025.
[9] Pavlo Bazilinskyy, Md Shadab Alam, and Roberto Merino-Martınez. 2025. Pedestrian crossing behaviour in front of electric vehicles emitting synthetic sounds: A virtual reality experiment. In *Proceedings of 54th International Congress Exposition on Noise Control Engineering (INTER-NOISE)*. São Paulo, Brazil.
[10] Pavlo Bazilinskyy, Md Shadab Alam, and Roberto Merino-Martınez. 2025. Psychoacoustic assessment of synthetic sounds for electric vehicles in a virtual reality experiment. In *Proceedings of 54th International Congress Exposition on Noise Control Engineering (INTER-NOISE)*. Malaga, Spain.
[11] Pavlo Bazilinskyy, Lars Kooijman, Dimitra Dodou, and J. C. F. De Winter. 2020. Coupled simulator for research on the interaction between pedestrians and (automated) vehicles. In *Proceedings of Driving Simulation Conference (DSC)*. Antibes, France.
[12] Yogesh K Dwivedi, Nir Kshetri, Laurie Hughes, Emma Louise Slade, Anand Jeyaraj, Arpan Kumar Kar, Abdullah M Baabdullah, Alex Koohang, Vishnupriya Raghavan, Manju Ahuja, et al. 2023. Opinion Paper:"So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy. *International journal of information management* 71 (2023), 102642. doi:10.1016/j.ijinfomgt.2023.102642
[13] Jiangnan Fang, Cheng-Tse Liu, Jieun Kim, Yash Bhedaru, Ethan Liu, Nikhil Singh, Nedim Lipka, Puneet Mathur, Nesreen K Ahmed, Franck Dernoncourt, et al. 2024. Multi-LLM Text Summarization. *arXiv preprint arXiv:2412.15487* (2024). doi:10.48550/arXiv.2412.15487
[14] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020). doi:10.48550/arXiv.2002.08155
[15] Christian Gold, Daniel Damböck, Lutz Lorenz, and Klaus Bengler. 2013. "Take over!" How long does it take to get the driver back into the loop?. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 57. Sage Publications Sage CA: Los Angeles, CA, 1938–1942. doi:10.1177/15419312135714
[16] Desta Haileselassie Hagos, Rick Battle, and Danda B Rawat. 2024. Recent advances in generative ai and large language models: Current status, challenges, and perspectives. *IEEE Transactions on Artificial Intelligence* (2024). doi:10.48550/arXiv.2407.14962
[17] Sandra G Hart. 2006. NASA-task load index (NASA-TLX); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 50. Sage publications Sage CA: Los Angeles, CA, 904–908. doi:10.1177/1541931206050009
[18] Sajed Jalil. 2023. The Transformative Influence of Large Language Models on Software Development. *arXiv preprint arXiv:2311.16429* (2023). doi:10.48550/arXiv.2311.16429
[19] Stephane H Maes. 2025. The gotchas of ai coding and vibe coding. it's all about support and maintenance. doi:10.5281/zenodo.15343349
[20] Stephane H. Maes, Karan Singh Chhina, and Guillaume Dubuc. 2016. Natural language translation-based orchestration workflow generation. US Patent 11,120,217 B2, granted in 2021.
[21] Jesse G Meyer, Ryan J Urbanowicz, Patrick CN Martin, Karen O'Connor, Ruowang Li, Pei-Chen Peng, Tiffani J Bright, Nicholas Tatonetti, Kyoung Jae Won, Graciela Gonzalez-Hernandez, et al. 2023. ChatGPT and large language models in academia: opportunities and challenges. *BioData mining* 16, 1 (2023), 20. doi:10.1186/s13040-023-00339-9
[22] P Pajo. 2025. Vibe Coding: Revolutionizing Software Development with AI-Generated Code. https://doi.org/10.13140/rg.2.2.36458.22727. doi:10.13140/RG.2.2.36458.22727 Accessed: 2025-06-07.
[23] Haoran Su, Jun Ai, Dan Yu, and Hong Zhang. 2023. An evaluation method for large language models' code generation capability. In *2023 10th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 831–838. doi:10.1109/DSA59317.2023.00118
[24] Maryana Tomenchuk and Kseniia Popovych. 2024. Large Language Models and Machine Translation. (2024). doi:10.52058/2695-1592-2024-11(42)-422-431
[25] Prokopia Vlachogianni and Nikolaos Tselios. 2022. Perceived usability evaluation of educational technology using the System Usability Scale (SUS): A systematic review. *Journal of Research on Technology in Education* 54, 3 (2022), 392–409. doi:10.1080/15391523.2020.1867938
[26] Fei Wang and Anita Preininger. 2019. AI in health: state of the art, challenges, and future directions. *Yearbook of medical informatics* 28, 01 (2019), 016–026. doi:10.1055/s-0039-1677908
[27] Yongjun Xu, Xin Liu, Xin Cao, Changping Huang, Enke Liu, Sen Qian, Xingchen Liu, Yanjun Wu, Fengliang Dong, Cheng-Wei Qiu, et al. 2021. Artificial intelligence: A powerful paradigm for scientific research. *The Innovation* 2, 4 (2021). doi:10.1016/j.xinn.2021.100179
[28] Xuesong Zhai, Xiaoyan Chu, Ching Sing Chai, Morris Siu Yung Jong, Andreja Istenic, Michael Spector, Jia-Bao Liu, Jing Yuan, and Yan Li. 2021. A Review of Artificial Intelligence (AI) in Education from 2010 to 2020. *Complexity* 2021, 1 (2021), 8812542. doi:10.1155/2021/8812542
[29] Wenting Zhao, Ye Liu, Tong Niu, Yao Wan, Philip S Yu, Shafiq Joty, Yingbo Zhou, and Semih Yavuz. 2023. DIVKNOWQA: assessing the reasoning ability of llms via open-domain question answering over knowledge base and text. *arXiv preprint arXiv:2310.20170* (2023). doi:10.48550/arXiv.2310.20170

# A Cursor prompts used during development (Figure 1, left image)

The following prompts were entered into Cursor (AI code assistant) during the development of the 3D driving simulator to iteratively build and refine the scene. Each prompt represents a specific design intention or correction attempt in natural language.

**P1:** `Create a basic Three.js scene with a ground plane, ambient light, and a perspective camera. Use orbit controls to inspect the scene for now.`
**P2:** `Use realistic acceleration and turning logic, not just simple position translation. The car should face the direction it is driving.`
**P3:** `I want to have a more realistic city.`
**P4:** `Create a realistic city, with buildings, trees, traffic lights, roads, and sidewalks for pedestrians.`
**P5:** `Can you create the city even more realistic?`

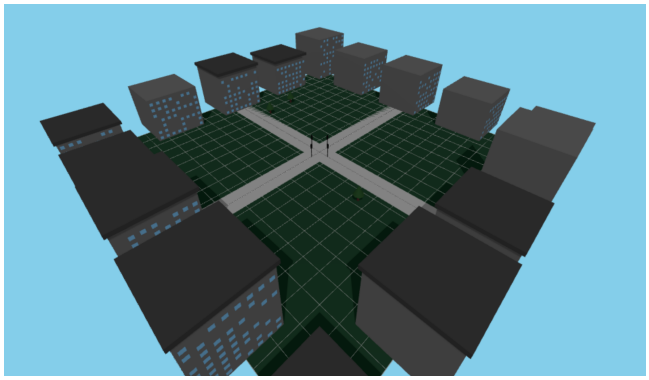

**Figure A1: Result of prompt 5.**

**P6:** `Can you create the city even more realistic?`

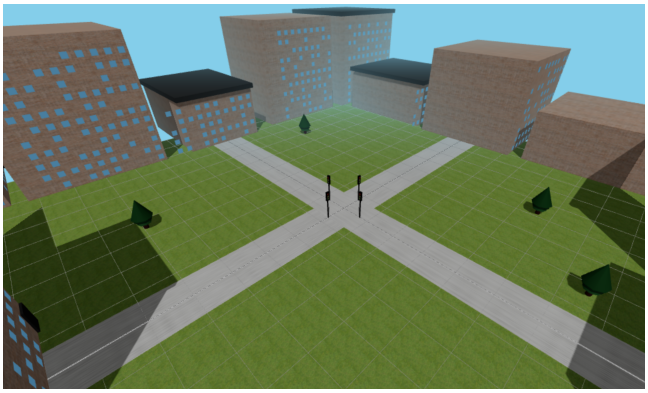

**Figure A2: Result of prompt 6.**

**P7:** Build a realistic car, with wheels, windows, and five doors.
**P8:** Make the car even more realistic, with the right materials, and the color can be blue.
**P9:** Make the car more realistic.
**P10:** Make the car even more realistic! Focus on the materials first.
**P11:** Make the car a bit bigger in terms of dimensions, and I want to properly be able to see the doors of the car in a realistic way.
**P12:** Make the car look more realistic!
**P13:** Make the car with four doors that are visible and realistic.
**P14:** Make sure that I can see the interior of the car through the windows.
**P15:** Can you make the trees even more realistic?
**P16:** Use the right materials for the trees, to make them real.
**P17:** Can you create a "photorealistic" city? Use all the datasets or open APIs you need.
**P18:** Add integration with any of these APIs (would require API keys but more realistic).
**P19:** Enhance the building data processing?
**P20:** Make a "photorealistic" city, using Google or Apple Maps.
**P21:** Using the base you already have which is good, can you: include and create a 3D city scene with real map data (e.g., Leiden), legally and freely, using only open APIs and open-source tools?

## B Cursor prompts used during development (Figure 1, right image)

The following sequence illustrates an approximate set of natural language prompts issued to Cursor (AI assistant) during the development of the 3D driving simulator. These prompts were iteratively refined to improve scene complexity, realism, and interactivity.

**P1:** Create a 3D city scene using Three.js, with roads, sidewalks, buildings, and trees.

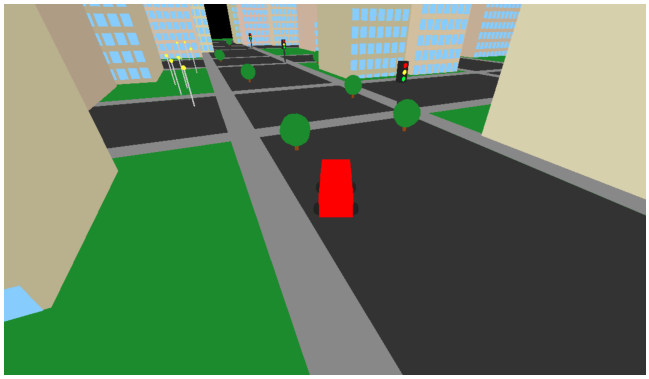

**Figure A3: Result of prompt 1.**

**P2:** Make the city look more realistic, add details like streetlights and traffic lights.
**P3:** Improve the materials and lighting to make the city more visually realistic.
**P4:** Add pedestrian elements, benches, and bus stops to the city.
**P5:** Add a realistic car to the scene.
**P6:** Make the car more realistic.
**P7:** Ensure the car faces the direction of travel, and moves using proper acceleration and turning logic.
**P8:** Add keyboard controls "WASD" to drive the car forward, backward, and turn left/right.

**Figure A4: Result of prompt 8.**

**P9:** Create realistic physics for braking and acceleration.
**P11:** Detect collisions with buildings or trees and stop the car.
*Note: Some instructions had to be reformulated multiple times due to incomplete or flawed outputs, highlighting the trial-and-error nature of prompting and the importance of clarity and precision.*