Designing and Developing DirectorAI: An AI Assistant for Generating Vehicle Simulation Scenarios

Creating an effective AI Assistant for complex, domain-specific tasks without domain-specific training data

Master's Thesis in Interaction Design & Technologies
Master's Thesis in Computer Science – Algorithms, Languages and Logic

Adam Telles and Hannes Raaholt Larsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Designing and Developing DirectorAI: An AI Assistant for Generating Vehicle Simulation Scenarios

Creating an effective AI Assistant for complex, domain-specific tasks without domain-specific training data

Adam Telles, Hannes Raaholt Larsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Designing and Developing DirectorAI: An AI Assistant for Generating Vehicle Simulation Scenarios
Creating an effective AI Assistant for complex, domain-specific tasks without domain-specific training data
Adam Telles, Hannes Raaholt Larsson

Supervisor: Morten Fjeld, Interaction Design and Software Engineering
Advisor: Anders Tell, Volvo Cars
Examiner: Staffan Björk, Interaction Design and Software Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Stylized mockup of the DirectorAI chat interface, overlaying a scene generated with DirectorAI and shown in Volvo Cars' Product Simulator.

Typeset in LaTeX
Gothenburg, Sweden 2025

Designing and Developing DirectorAI: An AI Assistant for Generating Vehicle Simulation Scenarios
Creating an effective AI Assistant for complex, domain-specific tasks without domain-specific training data
Adam Telles, Hannes Raaholt Larsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis investigates the use and potential benefits of AI in automating the generation of vehicle simulation scenarios. Focused on enhancing the usability of Director, a scripting tool for Volvo Cars' Product Simulator software, this project involved designing and developing DirectorAI, an AI assistant featuring a chatbot interface. Using large language models, the research explored how to create effective and reliable scenarios without domain-specific model training. Moreover, the project identified limitations that emerge when deploying general-purpose language models in complex, domain-specific environments, alongside design patterns that enable these models to function effectively as assistants. The primary research question addressed was: "What design choices or patterns enable general-purpose language models to function as effective assistants in complex, domain-specific software environments without domain-specific training?"

The outcome and findings demonstrated that the success of general-purpose LLMs in domain-specific environments relies less on model modification and more on how the system is designed to supply the model with relevant information. By iteratively crafting system prompts that embed domain context, constraints, and examples, DirectorAI was able to perform effectively without the need for custom training or fine-tuning. Through prototyping and user evaluation, several key design patterns were identified that enabled the assistant to support complex workflows within the existing simulation software.

This research emphasized the importance of interaction design in shaping the utility and usability of AI-assisted systems. By identifying and analyzing the design choices and patterns that facilitate the effective use of general-purpose LLMs in domain-specific environments, this thesis contributes to the understanding of how AI-assisted tools can be developed for complex simulation scenarios, offering valuable insights for future applications. Ultimately, this study demonstrated the potential of AI to significantly improve the efficiency and reliability of vehicle simulation scenarios, with implications for the automotive industry and beyond.

Keywords: Large Language Models (LLMs), Chatbots, Interaction Design, Computer Science, Vehicle Simulation Scenarios, AI Assistant, AI Adaptation, Prompt Engineering.

Acknowledgements

First and foremost, we are grateful for the opportunity to conduct our Master's thesis at Volvo Cars and for the valuable industrial context provided. We would particularly like to express our gratitude to our industry advisor, Anders Tell, whose support and guidance were instrumental in the completion of this thesis. He not only helped set up this project but also provided invaluable technical insights, practical advice, and encouragement throughout the process. His deep understanding of the complex systems underlying ProSim and Director greatly shaped our work.

We would also like to thank the ProSim team at Volvo Cars for their technical support and for sharing their expertise. Their insights into the practical workflows and use cases for Director and ProSim were crucial for grounding our work in real-world needs.

Additionally, we extend our thanks to the other employees at Volvo Cars who participated in our user studies. Their willingness to share their experiences and provide feedback was essential for our understanding of the challenges faced by users, significantly shaping the direction of our design work.

Finally, we would like to thank our academic supervisor, Morten Fjeld, for his guidance and support during the academic aspects of this thesis, providing us with valuable feedback and helping us navigate the research process.

Adam Telles and Hannes Raaholt Larsson, Gothenburg, 2025-06-23

Contents

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
Director	Product Simulator Director
FSM	Function State Machine
JSON	JavaScript Object Notation
LLM	Large Language Model
MIDI	Musical Instrument Digital Interface
ML	Machine Learning
MLM	Masked Language Model
NLP	Natural Language Processing
ProSim	Product Simulator
RAG	Retrieval-Augmented Generation
TF-IDF	Term Frequency-Inverse Document Frequency
GPT	Generative Pre-trained Transformer

List of Figures

List of Tables

1

Introduction

Recent advances in large language models (LLMs) such as OpenAI's GPT-4 have sparked growing interest in their potential to support complex digital workflows [1]. Certain AI-driven assistants have shown how LLMs can help with implementation details and suggestions. An example of this is Github Copilot, an AI-driven coding assistant. However, these systems often rely on vast amounts of domain-specific training data in order to achieve their efficiency. This thesis explores a different question: can general-purpose LLMs, without fine-tuning or additional domain-specific training, still provide effective assistance within a highly technical, domain-specific tool?

The case study of this project is Director, a proprietary scripting tool used at Volvo Cars for configuring and creating scenarios for a 3D vehicle simulator. While powerful, Director presents a range of usability challenges, particularly for new or less technical users. Through a user-centered design process, this project develops and evaluates an AI-driven chat interface for Director, using GPT-4 to help users find simulation commands, configure parameters, and generate scripts conversationally.

1.1 Project Aim and Research Questions

The broader aim of the project is to investigate how general-purpose LLMs can reduce the cognitive load and technical barriers inherent in domain-specific software. We focus particularly on the design decisions that enable such assistants to feel helpful, intuitive, and efficient, despite the utilized AI model having no task-specific training.

This leads to our main research question:

Main Research Question (MRQ):

> *What design patterns enable general-purpose language models to function as effective assistants in complex, domain-specific software environments without domain-specific training?*

To support the investigation of this main question, two supporting research questions are proposed. These help clarify aspects of the problem space and structure the evaluation of the results:

Supporting Research Question 1 (SRQ1):

What limitations emerge when using general-purpose language models as assistants in complex domain-specific environments without domain-specific training data?

Supporting Research Question 2 (SRQ2):

What constitutes "effective assistance" in the context of AI-supported domain-specific workflows?

Together, these questions guide the exploration of both the opportunities and the constraints involved in integrating general-purpose AI into specialized technical domains. Through this work, our aim is to contribute with insights on how AI can be meaningfully integrated into specialized workflows, and what considerations are required to make such integration useful and usable in practice.

1.2 Project Context

Designing and implementing simulation scenarios for Volvo Cars in Unity currently involves a detailed, hands-on process where every aspect of a vehicle's operation, such as opening doors, adjusting the camera, or triggering animations, must be meticulously programmed. For instance, creating a simple replayable simulation showcase of a car requires repeatedly switching between the 3D simulation software and Director to manually retrieve and input position and rotation values for the camera. Timing animations, such as doors opening or headlights activating, is similarly tedious, as users must determine exact signal names, many of which are inconsistently labeled, and adjust timing sequences without clear visual feedback. This fragmented workflow makes even straightforward tasks unintuitive and time-consuming, acting as a barrier to creativity and efficiency in validation.

The recent years' advancements in AI, particularly in NLP and generative AI, have made it more feasible to dynamically generate accurate results from user input, as well as the option of using cloud computing for this purpose, not relying on on-premise hardware and thus making it broadly accessible. With the ever-growing complexity of vehicle software, the need for efficient and user-friendly simulation software is becoming more important. With access to Volvo Cars' experts and proprietary tools and data, we will work closely with an industry-grade testing environment that ensures our research is both practical and impactful.

This thesis addresses the challenge of automating the scenario creation workflow in a way that users find effective and helpful. The goal is to enable users to describe scenarios using natural language either at a high, abstract level or with low-level, intricate detail. By offloading the cognitive load of learning and remembering system syntax and commands to the LLM and its supporting structure, users can instead focus on creativity and solution exploration. By embedding interaction design principles, the assistant aims to reduce cognitive load while preserving user agency and empowering new and non-technical users. All this allows the assistant to support more dynamic and fluid workflow, moving away from the current rigid ones.

1.3 Simulation Software Environment: An Overview of Director, ProSim, and FSM

The technical ecosystem for this project consists of three main components: Product Simulator (ProSim), Function State Machine (FSM), and Director. These systems work together to provide an interactive, simulation-driven environment for testing vehicle behaviors and features.

ProSim is the core simulation software and is developed in Unity [2]; an example of ProSim's use can be seen in Figure 1.1. It serves as the real-time simulation engine, responsible for rendering 3D environments and vehicles, simulating physical interactions, and allowing direct user control. ProSim includes a high-fidelity 3D car model, complete with functional elements such as doors, compartments, lights, and interior features. Users interact with the simulation through a first-person character, navigated via keyboard and mouse in a manner reminiscent of first-person video games. This setup allows users to explore the vehicle in a realistic and intuitive way, and interact with different components as if inside a physical prototype.

Figure 1.1: An example scenario showcasing a Volvo car driving in a city environment, rendered in Product Simulator.

FSM introduces a layer of logical control on top of the raw physics simulation provided by ProSim. While ProSim handles the physical effects of actions (such as opening a door), FSM manages system logic, rules, and interdependencies that regulate whether certain actions are allowed. For example, FSM enforces rules such as "a door cannot be opened if it is locked" or "the car cannot start if the key is not present." FSM is designed to reflect, as closely as possible, the real software logic that would run in the actual vehicle. This makes the simulation not only visually realistic, but also behaviorally accurate from a systems standpoint. Although the

details of the FSM team's implementation are outside the scope of this project, it is important to note that FSM is tightly integrated with ProSim, and receives input from it to enforce these logical constraints.

The third and most relevant component to this project is Director. Director is a Flutter-based application and serves as an interface for scripting and controlling ProSim. Communication between Director and ProSim occurs via Message Queuing Telemetry Transport (MQTT), a lightweight publish-subscribe messaging protocol often used in IoT systems [3]. Upon startup, Director establishes a connection with ProSim and receives a list of available signals. These signals represent actions that can be triggered in the simulation, such as opening a door, toggling a light, or adjusting climate control. Each signal corresponds to an MQTT topic (e.g., ego/door/open/FL for opening the front left door) and is used to send specific commands to ProSim. In figure 1.2, an overview of Director is presented.

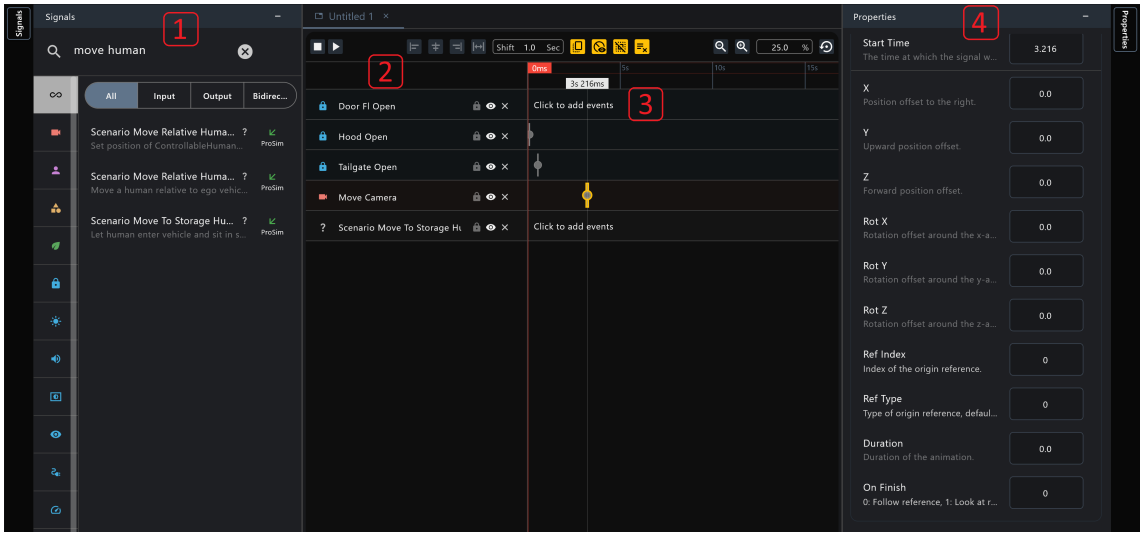

Figure 1.2: Overview of the Volvo Cars scenario scripting tool Director with numbered annotations.
(1) Signal search bar and different sections of signal types that are toggleable on the left-hand side. The list of signals appears on the right, with signal type (input, output, and bidirectional) as filters. Signals can be dragged onto the timeline (the section between signals and properties), creating a track.
(2) Column of signal tracks, each track occupying one row. The scenario can be played using the play button, paused by clicking it again, or stopped using the stop button.
(3) Timeline section, where each track can contain an unlimited number of actions. Actions are added by right-clicking the timeline and selecting *Add action*. Actions can be edited by selecting and moving, hiding, or deleting them.
(4) The *Properties* pane is shown when an action is selected. Property values can be manually edited. In this image, the signal action *MoveCamera* is selected, with eleven parameters including 3D coordinates and direction.

Director's user interface is divided into three main sections: the signal list, the timeline editor, and the properties pane. The signal list (displayed on the left-hand

side) shows all available signals retrieved from ProSim. These are organized by category, where the two most important categories for this project are ego and user input. Signals in the ego category directly affect the simulation without regard to the FSM's logic. For instance, an ego door open signal will open the door regardless of whether the vehicle is locked. On the other hand, user input signals simulate user-initiated actions, such as pulling the door handle, which respect the constraints enforced by FSM and could thus fail if the door is locked. This distinction is essential when designing test cases that should reflect realistic interactions.

Signals from the list can be added as tracks in the timeline editor. Each track corresponds to a specific signal, and users can place actions along the timeline. When the timeline is played, a red playhead moves across the screen. As it passes over an action, the associated signal is sent to ProSim, triggering the desired effect in the simulation. This timeline-based system is conceptually similar to music production software, where multiple tracks are layered and events are scheduled in time.

On the right-hand side of Director is the parameters pane, where users can configure signal-specific values for each action. For example, when triggering a light signal, parameters such as RGB values might be required to define the color or intensity. This allows for nuanced and dynamic scripting of simulation scenarios.

Although Director does not communicate directly with FSM, it plays a key role in invoking behaviors that may be subject to FSM rules. Director sends signals only to ProSim, which then relays those commands to FSM as needed. For the purposes of this project, understanding the structure and behavior of signals, particularly the distinction between ego and user input categories, is sufficient to design meaningful scenarios and scripts within Director.

2

Background

In order to better understand how DirectorAI fits into the broader technical landscape, this chapter briefly explores the history of chatbots, AI in creative and technical workflows, and related academic works.

2.1 Chatbots

Chatbots, also known as artificial conversation entities, interactive agents and digital assistants, are AI programs designed to simulate conversation with human users. Chatbots use NLP and sentiment analysis to communicate in human language through text or oral speech with humans or other chatbots [4].

The foundational concept behind chatbots is often credited to Alan Turing, who in 1950 questioned whether a computer could converse with people without revealing its artificial nature, known as the Turing test. The first chatbot named ELIZA was created in 1966, which simulated a psychotherapist by rephrasing user statements as questions. Its abilities were rather basic, relying on pattern matching and template responses, but it significantly inspired future chatbot development [5].

A great advancement happened in 2001 with the creation of SmarterChild which could be found on e.g. Microsoft's MSN, which was the first chatbot that could assist with practical daily tasks by retrieving information from databases on subjects such as movie schedules, sports results, and news [5]. The development continued with intelligent personal voice assistants such as Apple Siri, Amazon Alexa, and Google Assistant, which can understand voice commands, manage tasks via the internet, and generate relevant responses quickly. The use of chatbots increased particularly after 2016, which coincided with social media platforms allowing developers to create chatbots for their services [5].

Chatbots are deployed across multiple sectors, driven by their ability to automate tasks, enhance user engagement, and provide scalable solutions. The primary applications include:

- **Customer Service and Support**: Chatbots facilitate 24/7 customer support by addressing frequently asked questions (FAQs), tracking orders, and providing multilingual assistance. Their scalability reduces operational costs. Omnichannel integration enables seamless interactions across web, mobile, and social media platforms [6].

- **Data Collection and Analytics**: By capturing user preferences and behavioral data, chatbots enable businesses to refine marketing strategies and personalize offerings. In e-commerce, recommendation systems driven by chatbot interactions enhance sales [7].
- **Education**: Chatbots serve as virtual tutors, offering personalized learning experiences and administrative support. Tools like Ada and Socratic exemplify their role in subjects such as mathematics and computer science, promoting experiential learning [8].
- **Healthcare**: Chatbots provide remote patient support, mental health assistance, and health education, improving care accessibility. However, their deployment requires stringent compliance with regulations, such as the american HIPAA [9].
- **Marketing and Sales**: Through conversational marketing, chatbots engage customers via digital ads and messaging apps, streamlining lead generation and sales workflows [6].
- **Administrative Automation**: Chatbots automate repetitive tasks such as scheduling and invoice processing, enhancing operational efficiency in sectors like finance and retail [10].

2.2 AI in 3D Modeling, Video Editing and Content Generation

One example of adding AI features into an existing editing program is **Blender Open MCP**, which integrates the open-source Blender software with various AI models (e.g., Claude AI, Ollama) to enable natural language control over complex 3D model creation and scene manipulation [11]. While this primarily focuses on the generation of static assets, the methodology for rapidly creating and modifying intricate 3D environments and characters via AI is highly pertinent. Furthermore, **DeepMotion** is another related software example that has a large library of motion capture and uses AI to translate text prompts into 3D animations [12].

There are also many other generative AI tools such as **Adobe Firefly** and **Runway ML** that offer capabilities for text-to-video and image-to-video generation [13], [14]. **Synthesia**, for instance, uses AI to generate videos directly from text scripts, complete with AI avatars and voiceovers, resulting in AI-generated content with a clear chronological sequence [15]. Similarly, **Descript**, does video editing through text manipulation, allowing users to intuitively cut or rearrange video segments by simply editing the corresponding transcribed text [16]. While these pieces of software work great in their domain, the implementation of AI is specific to each application and may often not be generalizable to other applications.

Academic research on **LLM-Assisted Video Editing with Unified Language Representations** is particularly relevant to the concept of scripting. This work explores using LLMs to facilitate natural language interaction for complex editing

tasks, such as generating shot lists or refining sequences based on textual prompts [17]. This research underscores the potential for linguistic commands to directly influence the temporal flow and content of a generated sequence.

2.3 Related Academic Work

The 18 design guidelines for human-AI interaction from Amershi et al. [18] serve as a foundational reference for designing user-friendly AI systems. These guidelines, developed through iterations and expert review, aim to help practitioners build AI tools that are both effective and intuitive. For our work in Director, where users work with complex scripting logic and often unpredictable signals, several principles stood out, especially those related to clarifying AI capabilities, supporting user learning, and enabling corrections. We used these as a guide to design an assistant that emphasizes transparency, context-awareness, and trust. The guidelines also stress the importance of efficient interactions and graceful handling of errors. Drawing from these guidelines helped us align DirectorAI with current best practices in human-AI interaction, particularly in making the assistant more transparent and allowing for better recoverability.

Certain studies explore how LLMs can improve usability in more specific contexts. For example, Varma et al. [19] developed an AI-powered librarian assistant that helps students interact with library systems using natural language. Their assistant allows users to ask about books, availability, and recommendations through conversation, reducing the friction of navigating traditional interfaces. Although the domain differs from our work, the core idea is the same: using natural language to bridge the gap between user goals and complex backend systems. Their success in integrating LLMs into an existing workflow supports our approach of applying a general-purpose model to a domain-specific tool like Director without requiring custom retraining.

DirectorAI is reactive in the sense that it mainly responds to user input, but there is also research into more proactive uses of LLMs in technical settings. For example, Chen et al. [20] built a conversational programming assistant that suggests improvements and anticipates user needs without being prompted. While DirectorAI does not go that far, their work is relevant in how it shows the potential of context-aware, conversational agents to ease work in syntax-heavy environments. Their focus on shared context and fluid collaboration also resonates with our use case, where users can gradually build or refine simulation scripts through ongoing dialogue with the assistant.

Another important factor is how users perceive the AI, especially their mental model of what it can and can't do. Rismani et al. [21] studied this in the context of AI writing assistants and found that user understanding of the AI's behavior significantly affects how effectively they use it. If users assume the assistant is smarter than it is, or interpret its suggestions too literally, it can lead to frustration or misuse. This applies directly to DirectorAI. If users assume the assistant is always right, they may not critically evaluate signal suggestions or generated scenarios. On the other

hand, if they understand it as a helpful but imperfect assistant, users are more likely to engage critically, which ultimately could lead to more satisfactory results. This highlights the need for clear, transparent communication from the assistant, as well as features that help users build a realistic understanding of its capabilities.

Khurana et al. [22] offer a cautionary perspective. In their study, users interacted with an LLM tailored to a specific software system. Surprisingly, the results showed little improvement over a general baseline, partly because users had trouble understanding how their prompts related to the responses they received. More concerning was that many users followed incorrect AI-generated instructions without questioning them, especially users with less technical experience. This highlights the need to design AI systems that users can understand and trust, not just those that generate correct answers.

At the same time, other research shows that even imperfect AI systems can still be helpful. Schoenegger et al. [23], for instance, studied how LLMs influence human forecasting performance. They found that even when users were given advice from a deliberately flawed LLM, their forecasts still improved compared to those who had no AI support. This suggests that LLMs can provide meaningful cognitive support even when their output isn't perfect. That idea aligns well with our goals. DirectorAI isn't meant to be an expert system that replaces the user's judgment. Instead, it's a tool to help users get unstuck, reduce friction, and better understand how to interact with the system. As Schoenegger et al. put it, the LLM acts more like a 'decision aid', which we believe is especially valuable in the kind of complex, technical workflows users face in Director.

AI-based assistants have significantly enhanced the efficiency of existing software, particularly in software development. [24] review AI-driven innovations, presenting a case study on GitHub Copilot, a generative AI tool that provides real-time code suggestions and completions. The study reports a 55% reduction in task completion time and higher rates of passing test cases on the first attempt, demonstrating how AI assistants streamline coding processes. By automating repetitive tasks and offering context-aware suggestions, GitHub Copilot improves productivity and reduces errors, making it a valuable tool for developers working on enterprise applications. The review emphasizes the need for clean data and user preparation to maximize these efficiency gains, underscoring the design considerations for embedding AI assistants into development workflows.

3

Theory

In this chapter, we present the technical resources used in this thesis, along with the theoretical foundations and related works that informed our approach.

3.1 Limitations of Chatbots

Despite their versatility, chatbots face significant theoretical and practical constraints, rooted in the limitations of current NLP and ML frameworks. These include:

- **Limited Contextual Understanding**: Chatbots struggle with ambiguous or complex queries due to finite NLP capabilities. Rule-based systems rely on keyword matching, while AI-driven models may misinterpret nuanced inputs, leading to irrelevant responses [25].
- **Lack of Emotional Intelligence**: The inability to interpret emotions, humor, or sarcasm restricts chatbots' capacity to foster emotional connections, critical for customer loyalty [26]. This stems from their reliance on statistical language models rather than human-like emotional understanding.
- **Inability to Handle Complex Queries**: Chatbots excel in routine tasks but falter in scenarios requiring reasoning or creativity. For instance, providing personalized advice or resolving multifaceted technical issues remains challenging [27].
- **Security and Privacy Risks**: Handling sensitive data exposes chatbots to vulnerabilities like hacking or data breaches. Compliance with privacy regulations is critical, particularly in healthcare [28].
- **Hallucinations and Inaccuracy**: LLMs may generate false information, known as hallucinations, due to overfitting or biased training data. Examples include fabricated citations or nonsensical responses to ambiguous inputs [29].
- **Ethical and Bias Concerns**: Biases in training datasets can lead to discriminatory outputs, undermining fairness in applications such as education and healthcare. Ethical challenges also arise from potential emotional manipulation [30].
- **Environmental Impact**: The computational resources required for training and operating chatbots contribute to significant energy consumption and

carbon emissions, raising sustainability concerns [31].

3.1.1 Implementation of Chatbots

Chatbot implementations typically rely on two core approaches: pattern matching and machine learning.

Pattern matching uses rule-based systems to compare user input against predefined templates, selecting fixed responses accordingly. This method, exemplified by ELIZA, is effective for predictable interactions but struggles with ambiguous or novel input due to its reliance on scripted responses [5].

Machine learning-based chatbots, by contrast, leverage natural language processing (NLP) to understand context and intent, allowing them to generate responses dynamically. This approach powers virtual assistants like Siri, Alexa, and Google Assistant, which continuously adapt to user behavior through models such as deep learning and LLMs [5].

A leading example is ChatGPT, which uses the GPT architecture to produce coherent and contextually relevant dialogue. It can handle nuanced prompts and is capable of more than casual conversation, such as generating scenario-based simulations for educational purposes by interpreting structured prompts and delivering dynamic multi-turn responses [32], [33].

3.2 Large Language Models

LLMs are neural networks, typically based on transformer architectures, trained on vast corpora of text data to predict and generate sequences of words [34]. The transformer model, introduced by [34], leverages self-attention mechanisms to capture long-range dependencies in text, allowing LLMs to process and generate coherent language. Prominent examples, such as BERT and GPT, demonstrate the power of pre-training on diverse datasets followed by task-specific adaptation [35], [36]. Pre-training equips LLMs with broad linguistic knowledge, which can be fine-tuned for tasks like text classification, translation, or question answering [35].

The scale of LLMs, often comprising billions of parameters, enables them to model complex linguistic patterns but introduces challenges, including high computational costs and ethical concerns related to bias and misinformation [37]. Despite these limitations, LLMs have transformed fields such as scientific research, healthcare, and education by facilitating advanced text analysis and generation [38]. Their ability to generalize across tasks underscores their potential as foundational tools in NLP.

3.2.1 GPT Architecture and Transformer Mechanisms

The GPT family is based on the Transformer architecture introduced by Vaswani et al. [34], which uses self-attention to model long-range dependencies in sequences and allows for efficient parallel computation.

GPT uses an autoregressive approach to generate text one token at a time, conditioning each token on all previous ones. The probability of a sequence is expressed as shown in Equation (3.1).

$$P(x_1, x_2, ..., x_n) = \prod_{t=1}^{n} P(x_t | x_1, ..., x_{t-1}) \tag{3.1}$$

These conditional probabilities are modeled using deep neural networks [39]. GPT follows a decoder-only Transformer design, stacking layers of masked self-attention and feed-forward components optimized for generative tasks like dialogue and summarization.

In contrast, bidirectional models such as BERT [35] use an encoder-only architecture trained via masked language modeling (MLM), making them well-suited for classification and extraction tasks.

Retrieval-Augmented Generation (RAG) enhances transformer models by incorporating retrieved external documents into the context window, improving factual accuracy and domain-specific performance [40].

3.3 Natural Language Processing

Natural Language Processing (NLP) focuses on enabling machines to understand and generate human language in a contextually meaningful way [41]. Applications span translation, summarization, and sentiment analysis.

Modern NLP has evolved from rule-based and statistical methods to deep learning architectures, especially transformers, which outperform RNNs and LSTMs in modeling long-term dependencies.

A key task is text classificationused in this project to categorize user input. Traditional approaches like TF-IDF require manual feature engineering and struggle with generalization. In contrast, transformer-based models such as BERT leverage pretraining and transfer learning to improve accuracy on unseen inputs [35].

3.4 Transformers

Transformers, introduced by Vaswani et al. [42], replaced recurrence with an attention mechanism, improving parallelization and the ability to model long-range relationships. Their core innovation is self-attention, which computes new token embeddings as weighted combinations of all tokens in a sequence.

The self-attention mechanism, defined in Equation (3.2), is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.2}$$

where Q, K, V are learnable matrices and d_k is the dimension of the key vectors.

Encoder and Decoder Roles

The encoder processes input tokens into contextual embeddings using stacked layers of multi-head self-attention and feed-forward networks. The decoder, used in generative tasks, predicts output tokens iteratively, incorporating encoder outputs and prior predictions. GPT models use decoder-only stacks, while BERT employs encoder-only stacks [43].

BERT and Bidirectional Context

BERT (Bidirectional Encoder Representations from Transformers) was proposed by Devlin et al. [35] to pretrain models that leverage context from both directions. It uses the MLM objective to predict randomly masked tokens using surrounding context and is ideal for classification and question answering.

3.5 Software

This section introduces the main software applications and technologies used in this project.

Dart and Flutter

Dart is an open-source, general-purpose programming language developed by Google, optimized for building high-performance, client-side applications across web, mobile, desktop, and embedded platforms [44]. With an object-oriented syntax and features like sound null safety, Dart enhances code reliability by preventing null reference errors [45]. It supports ahead-of-time (AOT) compilation for efficient native code execution and just-in-time (JIT) compilation with hot reload, enabling rapid development cycles [45]. Dart's platform-independent virtual machine and standard library, extended by the Pub package repository, make it a versatile tool for modern application development [46].

Flutter, a UI software development kit (SDK) built on Dart, enables developers to create natively compiled, visually consistent applications from a single codebase for multiple platforms, including iOS, Android, web, and desktop [47]. Flutter's widget-based architecture allows hierarchical composition of UI components, known as widgets, to build responsive interfaces [48]. Its rendering engine, powered by Skia (or Impeller on iOS), bypasses platform-specific UI components to ensure uniform visuals and performance across devices [47]. Flutter leverages Dart's AOT compilation for fast execution and JIT compilation for hot reload, streamlining development workflows [46]. Supported by a rich ecosystem of Pub packages and pre-built widgets, Flutter reduces development time and is used in applications such as Google Pay, making it suitable for cross-platform development [46].

Azure OpenAI Service

Microsoft Azure OpenAI Service provides enterprise-grade access to advanced AI models developed by OpenAI, including the GPT-4 model family, integrated with Azure's secure cloud infrastructure [49]. Launched in 2021, this service enables organizations to leverage powerful language models for tasks such as content generation, summarization, code generation, and conversational interfaces, while ensuring compliance with enterprise requirements such as data privacy, security, and regional availability [49], [50]. The GPT-4 models, including GPT-4, GPT-4 Turbo, and GPT-4o, are multimodal, capable of processing text and images, and excel in complex reasoning, coding, and multilingual tasks [51].

To interact with GPT-4 models, Azure OpenAI Service provides a REST API, accessible via endpoints for chat completions, embeddings, and other capabilities [52]. Developers create an Azure OpenAI resource in the Azure portal, deploy a GPT-4 model and authenticate API calls using either API keys or Microsoft Entra ID tokens [52], [53]. For example, a chat completion API call involves sending a POST request to an endpoint with a JSON payload containing a system message, user prompt, and parameters such as `max_tokens` to control output length [53]. The response includes a model-generated completion, token usage, and metadata, enabling multi-turn conversations or single-turn tasks [52]. Azure's enterprise features, such as private networking, role-based access control, and content filtering, ensure secure and compliant API usage, while global standard deployments dynamically route traffic for low-latency performance [49], [54]. Additionally, Azure OpenAI supports RAG for grounding responses in enterprise data, enhancing accuracy for domain-specific applications [55].

Unity

Unity is a real-time 3D development platform developed by Unity Technologies, widely used for creating interactive simulations, games, and extended reality (XR) applications across industries such as automotive, robotics, and manufacturing [56]. Launched in 2005, Unity supports the creation of both two-dimensional (2D) and three-dimensional (3D) environments, using the C# programming language for scripting and a visual editor for designing scenes, physics, and animations [56], [57].

Its component-based architecture allows developers to attach behaviors, such as physics simulations or artificial intelligence (AI), to game objects, enabling rapid prototyping and iteration [58]. Unity's cross-platform capabilities support over 19 platforms, including Windows, macOS, Linux, iOS, Android, and XR devices like HoloLens and Oculus, making it versatile for deploying simulation environments [56].

Graphical simulation tools are important to visualize, test and validate different behaviors and functions in the automotive industry. For instance, Yang et al. [59] have used the Unity software to create a virtual reality driving simulation platform to examine driver behavior and depth sensor accuracy in various scenarios, which the authors mean will reduce training costs and improve the efficiency of addressing

emergency events.

3.6 Interaction Design-Related Theory

This section introduces relevant theory from the field of interaction design used in this thesis.

3.6.1 User-Centered Design

Within the field of interaction design there exist several different approaches to design, such as speculative design, participatory design, and somaesthetic design to name a few [60]–[62]. Each type of design influences what questions we ask ourselves as we work: What is the goal? Who are we designing for? Why are we designing for them? Consequently, the selected type of design affects the process we follow and the intended outcomes.

In the case of designing and developing DirectorAI, an assistant intended to address the usability issues in Volvo Cars' Director tool, we use a User-Centered Design (UCD) approach. This approach is based on the work by Norman [63], Norman & Draper [64], and Gould & Lewis [65].

UCD is not a strict methodology, but rather it is a guiding principle or design philosophy. Fundamentally, UCD emphasizes building an understanding and empathy for users' issues, pain points, and workflows. It promotes a design and development process based on user needs, and producing prototypes that are continuously evaluated by users themselves to ensure solutions stay aligned with their real-world issues.

3.6.2 Cognitive Load

Cognitive load, as introduced by Sweller et al. [66], refers to users' limited capacity to hold task-relevant information in memory. These limitations in how people learn and process information can often increase the difficulty of performing a task. In the case of software like Director, cognitive load can be alleviated through various methods, such as reducing the complexity of the task or simplifying the interface.

One of the main motivations behind DirectorAI is to reduce the cognitive load associated with using Director. The tool's interface, scripting syntax, and workflow all require users to hold a lot of information in mind while working. This is where LLMs offer an opportunity to reduce cognitive load. With their ability to understand users' natural language, we can shift some of that mental burden away from users and potentially make the system simpler to use.

3.6.3 Affordances

Affordance is a concept introduced by Norman [63] and refers to the properties of an object that suggest how it can be used. In terms of interface design, affordances are what guide user expectations. For example, if the interface contains a button

that looks clickable then it suggests an action will be performed when it is pressed. Cooper et al. [67] expand on the idea of affordances when it comes to digital interface design. They argue that the perceived functionality of an interface element may not always align with its actual functionality. Therefore, they suggest that what matters most is not the interface element's true functionality, but instead how users perceive it to work based on past experiences.

This issue becomes more complex in the case of DirectorAI, affordances extend beyond interface elements such as buttons or icons to the functionality of the underlying LLM. While the interface contains familiar elements such as a revert button that undoes actions or a chat log that logs user and AI messages, the assistant itself carries a more complex affordance. DirectorAI is presented as a helpful and intelligent assistant and if users perceive the assistant as an expert, or something they can delegate all work to, breakdowns in usability and trust will begin to occur since the assistant will inevitably fail at certain tasks.

3.6.4 Mental Models

Another important concept introduced by Norman [63] is that of mental models. Mental models refer to the internal understanding people form about how a system works. They can be shaped by prior experience, cues from the system, and guidance from other people. While these models are not always accurate, they guide how users interact with a system and what they expect it to do.

When it comes to DirectorAI, mental models become particularly important, as users may approach the assistant with preconceptions from previous experiences with other AI-based assistants or timeline-based scripting tools. When a user's mental model differs from how the assistant actually functions, especially regarding the capabilities and limitations of the underlying LLM, confusion and frustration may arise. As such, considering users' existing mental models, and supporting the formation of accurate new ones, is an important part of designing DirectorAI.

3.6.5 Usability Heuristics

Established interaction design heuristics include Nielsen's *10 Usability Heuristics for User Interface Design* [68]. While we do not consider all of them in this project, it is worth briefly mentioning a few that are particularly relevant for DirectorAI's design. These include the heuristic about visibility of system status and providing timely feedback (H1). Nielsen also emphasizes the importance of user control, such as being able to undo actions (H3). Another relevant heuristic is the availability of help and documentation (H10). Perhaps the most central one, however, is the idea that systems should speak the user's language, avoiding overly technical jargon (H2). This directly relates to the core of what we aim to address with DirectorAI, as many of Director's usability issues stem from the overuse of technical terminology.

4

Methods

This project used both design and technical methods in parallel, allowing insights from one to inform decisions in the other throughout the process. The design methods will follow an interaction design framework, ensuring that the development of AI-driven features is grounded in user needs, usability principles, and iterative refinement. Simultaneously, the technical methods will focus on implementing and integrating AI technologies, such as natural language processing, within the constraints of the existing 3D simulation software. By combining these two perspectives, the project aims to create a solution that is feasible within the system's technical limits while directly addressing user needs.

4.1 Design Methods

To begin, this project was planned with a formative methodological approach, which in this case meant that the insights, observations, and feedback from both Director and DirectorAI would be used to inform our work and how we progressed. In contrast to a summative approach, the focus was not on producing statistically generalizable results. Instead, we intended to use findings from our research to iteratively refine our prototype and address usability issues. This formative approach was chosen since it aligns well with the principles of UCD where understanding user needs, challenges, and workflows is crucial.

The planned design methodology in this project is fundamentally rooted in a UCD approach, adopted to effectively address the usability challenges of Director and ensure the AI assistant genuinely meets user needs. UCD emphasizes an iterative process involving user feedback, usability testing, and continuous refinement throughout the design and development process. As explored by Mirabdolah et al. [69], applying UCD principles is particularly valuable for enhancing interaction quality and usability within complex systems like Director, especially across diverse domains. Their work highlights how iterative feedback loops and a focus on the user perspective can lead to more efficient, effective, and less cognitively demanding solutions. Employing UCD was therefore considered essential for navigating the complexities of integrating a novel AI tool into the existing Director workflow, allowing the design to evolve based on direct user input and testing, increasing the likelihood of developing a truly helpful and usable assistant.

Intended to provide structure for this UCD approach, the Double Diamond frame-

work [70] served as the primary design framework for this project. While the Double Diamond was central, it is not the only established approach available within the field of interaction design. Alternative methodologies such as Design Thinking, Goal-Directed Design, and Lean UX each offer distinct strengths, and their relevance depends heavily on the goals, constraints, and nature of the project.

Design Thinking shares several principles with the Double Diamond, particularly the emphasis on empathy, iterative prototyping, and user feedback [71], [72]. However, it often operates in a more fluid and less formally staged manner, which can be helpful in exploratory projects. However, in our case, we needed a more structured process to support analysis and reporting. For this thesis, where clarity, traceability, and a well-defined structure for analysis and reporting were important, the Double Diamond offers a more appropriate balance between flexibility and methodological rigor.

Goal-Directed Design, introduced by Alan Cooper, focuses heavily on identifying user goals and creating solutions that align with these goals through persona-based design [73]. This approach offers deep behavioral insight and works particularly well when designing tools for specific, repeatable tasks. However, this project aimed to explore emergent creative behaviors, many of which were discovered through early exploratory studies or even appearing later in prototyping phases. As such, a rigid goal-driven approach risked narrowing the design space too early.

Lean UX, on the other hand, emphasizes rapid experimentation, minimal viable products, and continuous iteration in fast-moving product teams [74]. While ideal in agile, startup-like settings with quick user feedback loops, the lacking emphasis on discovery and exploration makes it less suitable for early-stage research into novel technologies, especially considering our primary goal is understanding user needs rather than optimizing conversion metrics or feature releases. Lean UX contrasts having a clear 'solution space' such as in the Double Diamond which enables clear understanding of user's issues before moving to prototyping, something we considered valuable.

In contrast to these alternatives, the Double Diamond model was chosen because it supported a comprehensive, structured exploration of both the problem space and the solution space [70]. It allowed the project to remain grounded in user needs while still leaving room for creative and conceptual exploration, particularly around the emerging role of AI in design workflows. Its staged structure also enabled clear documentation of each phase, which was beneficial for the transparency and reflection included in a thesis. The Double Diamond's four phases (Discover, Define, Develop, and Deliver, as seen in figure 4.1) offered a methodical yet open-ended process that aligned well with the project's aim to investigate, understand, and meaningfully contribute to how users interacted with AI assistants in creative but technically complex tools.

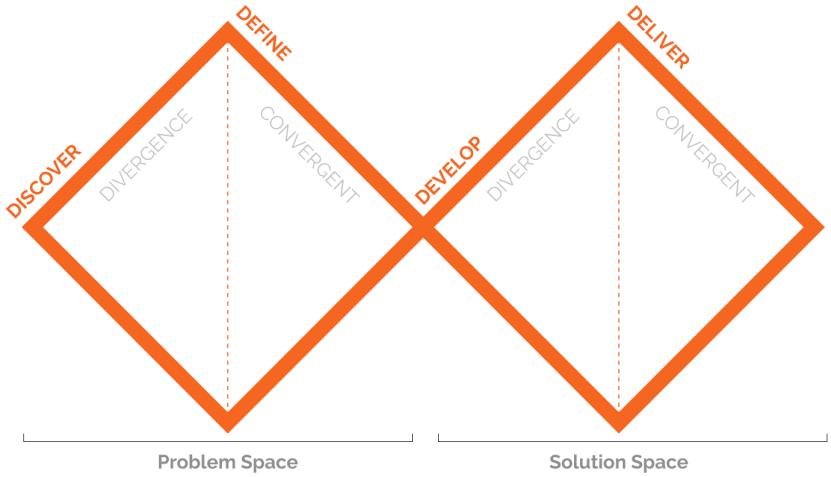

Figure 4.1: A visualization of the Double Diamond design framework. Adapted from [70].

4.1.1 Discover Phase

In the Discover phase, we planned to familiarize ourselves with Volvo Cars' 3D vehicle simulation software and Director in order to understand their functionality and limitations. This included informal observations of developers and users using the tool to identify common workflows, challenges, and workarounds. While full ethnographic studies provide deep insights into user behavior [75], they are often time-intensive and require prolonged immersion in this setting. Given the constraints of this project, we used a more lightweight ethnographic approach [76], focusing on observations and contextual inquiry rather than extensive fieldwork.

To supplement these observations with broader user data, we conducted a survey targeting existing users of the software. Surveys are useful for reaching a larger participant pool and identifying common trends [77], though they lack the depth of direct user interaction and potential for follow-up. The survey focused on understanding the software's primary use cases and challenges users faced. While interviews could have provided richer qualitative data, a survey allowed us to efficiently gather input from a larger and more diverse set of users.

We then conducted usability testing following the guidelines of Rubin and Chisnell [78]. The motivation for these studies was to identify the specific challenges that users can encounter while performing tasks in Director. During these studies, we collected qualitative data through a combination of think-aloud protocols to capture users' thought processes, direct observations of hesitations, confusions, or workarounds, and post-task interviews to explore their experiences in more depth.

More advanced usability evaluation methods, such as controlled laboratory studies or cognitive workload assessments, could have provided deeper insights into user performance [79]. However, these methods often require specialized equipment, controlled environments, or long testing sessions, which are beyond the practical scope of this study. Instead, we prioritized methods that provided rich, in-depth, and

actionable feedback within real-world constraints and use cases, ensuring that the findings directly inform the design process.

Our usability tests involved both experienced and inexperienced users of the software, as well as individuals with no prior exposure. This was important because interaction design should not only optimize experiences for existing users but also lower entry barriers for new users [63]. While longitudinal studies could have offered insights into learning curves and long-term adoption, time constraints made this impractical, so our evaluation focused on immediate usability and emerging behavior rather than long-term user adaptation.

4.1.2 Define Phase

In the Define phase, we analyzed the collected data to refine our understanding of usability challenges and user needs. This involved thematic coding to identify recurring patterns in survey responses and usability test findings. This approach was chosen because thematic coding provided a structured but adaptable way to interpret qualitative data, allowing us to translate user feedback into clear and relevant insights [80]. These insights then informed our design and were important for ensuring that the resulting prototypes address actual pain points, and align with user behavior and needs, rather than being based on technical possibilities or novelty [63].

While participatory design workshops could have helped refine our design further, we chose to rely on the identified codes and issues instead. This decision was made to keep the process focused and simple, especially given time and resource constraints. By basing our design work and prototypes on direct observations and user feedback, we were able to capture a range of perspectives nonetheless.

Personas is another method we considered during this phase. Introduced by Cooper [81] and later developed further by Pruitt and Grudin, [82], personas are fictional characters based primarily on qualitative data gathered in the discover phase. They are used to represent groups of users or stakeholders and typically include goals, workflows, needs, and pain points.

We initially planned to use personas, or were at least open to the idea, but the discover phase quickly showed just how broad and diverse the user base was. It included everyone from software engineers and function architects to 3D artists and analysts. In practice, most participants had differing goals and use cases for Director, making it difficult to generalize them into one or even a few personas. There was also the risk of how designing for a few personas might lead to solutions only being optimized for the workflows represented in those personas. Instead, we chose to simply focus on the most prominent pain points and recurring issues that appeared across user groups.

4.1.3 Develop Phase

During the Develop phase, we began creating and refining prototypes to test and validate our design concepts. Prototyping is often essential in interaction design as it enables iterative testing and adjustment before committing to full-scale implementation [83]. Rather than starting with wireframes or paper sketches, we chose to move directly into more functional prototypes, allowing us to quickly assess the feasibility of our ideas in the context of the actual Director software. This approach ensured our early design work was grounded in real-world constraints and technical considerations, providing more immediate insights into user needs and possible solutions.

Considered prototyping methods included the Wizard of Oz technique, which involves faking prototype functionality to gather user feedback without investing extensive development time [79]. However, we quickly encountered a core issue: what kind of AI behavior should we simulate? A best-case scenario? Worst-case? Something in between? At this stage, we had no clear idea of how the AI would actually perform or how it would respond to user prompts. Instead, we chose to build simple yet functional prototypes to test feasibility and gather user feedback using a real LLM, which was especially important given the unpredictable nature of LLMs in a niche, proprietary system like Director.

Additionally, while a more complete participatory co-design method can be useful for aligning solutions with real-world use cases [84], it was not the primary approach in this project. Mainly due to limited user availability, we adopted a lighter form of co-design, relying on short, targeted feedback sessions throughout development. User feedback was crucial, but doing it this way allowed us to incorporate user input effectively, aligning the design with real-world workflows without overcomplicating the early stages by placing too much burden on participants.

4.1.4 Deliver Phase

Finally, in the Deliver phase, we evaluated our prototype to understand how DirectorAI impacted the user experience and in which ways it supported scenario creation. This evaluation focused on gathering qualitative insights through user studies, where methods such as think-aloud protocols and post-task interviews were used to capture the diverse ways in which participants interact with the assistant. Rather than simply measuring performance improvements, our goal was to explore how DirectorAI influenced user workflows, reduced cognitive load, and supported creative exploration, providing an understanding of the design's practical impact.

By following this structured process, we ensured that the developed solution, DirectorAI, was grounded in user research and iterative refinement. At each stage, we tried to balance methodological depth with practical constraints, making deliberate trade-offs based on which methods best fit our time constraints, user access, and the goals of the project.

4.2 Technical Methods

This section explains technical methodologies relevant to adapting LLMs for specific contexts. Understanding these techniques provides important context for the design choices made and the specific methods used in this thesis, particularly the project's focus on prompt engineering for DirectorAI.

4.2.1 Fine-tuning

Fine-tuning refers to the process of adapting a pre-trained LLM to a specific task or domain by further training it on a targeted dataset. This approach adjusts the model's parameters to enhance performance for applications such as text classification, question answering, or scientific text generation [35]. Typically, fine-tuning employs supervised learning, where labeled data is used to minimize a task-specific loss function. For example, fine-tuning BERT on domain-specific corpora significantly improves its accuracy in tasks like natural language inference [35]. The process requires high-quality labeled datasets and substantial computational resources, which can be a limitation in certain contexts [85].

Instruction Fine-Tuning

GPT models can be fine-tuned using structured prompt-response datasets, aligning model outputs with desired behaviors. This process is often enhanced by reinforcement learning from human feedback to promote safety, usefulness, and coherence [86].

Parameter-Efficient Adaptation

Approaches like Low-Rank Adaptation (LoRA) update only a subset of model parameters, allowing resource-efficient customization for domain-specific deployments [87].

4.2.2 Prompt Engineering

Prompt engineering involves designing input prompts to guide a pre-trained LLM to produce desired outputs without modifying its parameters. This technique leverages the model's existing knowledge, making it resource-efficient for tasks such as text generation or sentiment analysis [36]. Strategies such as few-shot learning, where prompts include a few task examples, or zero-shot learning, where only task instructions are provided, enhance model performance [36]. Prompt engineering is particularly valuable in scenarios requiring rapid adaptation to new tasks with minimal data, though its effectiveness depends on the model's generalization capabilities [88].

Zero-Shot and Few-Shot Learning

Zero-shot learning (ZSL) enables LLMs to perform tasks without prior task-specific training or examples, relying solely on a descriptive prompt [36]. For instance, instructing a model to "Classify the sentiment of this review as positive or negative" without providing examples tests its ability to generalize from pre-trained knowledge. ZSL is particularly useful for rapid task adaptation, but it can fail in domains requiring specialized knowledge or nuanced reasoning [39].

Few-shot learning (FSL), on the contrary, includes a small number of task examples within the prompt to guide the model [36]. For example, a prompt might include: "Example: 'Great product!' → Positive. 'Terrible service.' → Negative. Now classify: 'Amazing experience!"'. FSL often outperforms ZSL by providing context that aligns the model's output with the desired format or style [88]. One-shot learning, a special case of FSL with a single example, serves as an intermediate approach. Both ZSL and FSL are forms of in-context learning, where the model learns from the prompt context without weight updates [89].

The effectiveness of ZSL and FSL depends on prompt clarity, the model's pre-training data, and task complexity. Recent studies suggest that larger models, such as GPT-3 or its successors, exhibit stronger zero-shot and few-shot capabilities due to their extensive training corpora [90]. However, performance can be degraded for tasks that require deep reasoning or domain-specific expertise, necessitating advanced techniques such as prompt chaining or fine-tuning [88].

Prompt Chaining

Prompt chaining is a technique that decomposes complex tasks into a sequence of smaller, interdependent prompts, where the output of one prompt serves as input for the next [90]. This approach mitigates the limitations of single-prompt interactions by structuring tasks into manageable steps, improving accuracy and coherence. For example, to generate a business plan, a chain might include:

1. "List the key sections of a business plan."
2. "Write an executive summary for a company with the following description: [input]."
3. "Draft a market analysis based on the executive summary: [output from step 2]."

Prompt chaining is particularly effective for multi-step reasoning tasks, such as planning, code debugging, or report generation [90].

Related to prompt chaining is the concept of *chain-of-thought (CoT)* prompting, which encourages the model to articulate intermediate reasoning steps within a single prompt [90]. For instance, a CoT prompt might state: "Solve this math problem and explain each step." While CoT focuses on reasoning within one prompt, prompt chaining distributes reasoning across multiple prompts, making it suitable for tasks requiring iterative refinement or modular outputs [90]. A more advanced variant,

tree-of-thought (ToT) prompting, explores multiple reasoning paths in parallel and selects the optimal one, often implemented through chained prompts [91].

Prompt chaining requires careful design to ensure alignment between steps and to prevent error propagation. Automated workflows, where outputs are programmatically fed into subsequent prompts, can streamline this process, particularly in API-based applications [90].

Meta-Prompting

Meta-prompting involves crafting prompts that instruct the model to design, evaluate, or optimize prompts before addressing a task [92]. This higher-level approach leverages the model's self-reflective capabilities to improve prompt quality. For example, a meta-prompt might state: "Write an optimized prompt to elicit a detailed summary of a scientific article." Alternatively, it could ask: "Review this prompt for clarity and suggest improvements: [insert prompt]." Meta-prompting is particularly valuable for users with limited prompt engineering expertise or when tackling novel tasks [92].

A related concept is *self-consistency*, where the model generates multiple responses to a prompt and selects the most consistent or highest-quality output **Wang2023**. Meta-prompting can orchestrate self-consistency by instructing the model to compare outputs and refine its approach. Another related technique, *reflexive prompting*, asks the model to reflect on its reasoning process or prompt effectiveness, e.g., "Why did this prompt yield a vague response, and how can it be improved?" [92].

Meta-prompting can be combined with prompt chaining to create dynamic workflows. For instance, a meta-prompt might generate an initial prompt, which is then used in a chained sequence, with subsequent meta-prompts refining the process based on intermediate results [90]. However, meta-prompting requires precise phrasing to avoid confusion, as the model must balance the meta-task (e.g., prompt design) with the actual task.

4.2.3 Other Relevant Concepts

Several additional concepts improve prompt engineering, particularly for complex or reasoning-heavy tasks:

- **Self-Consistency**: As mentioned, self-consistency involves generating multiple outputs and selecting the best one, often improving performance on tasks such as mathematical reasoning or factual question answering **Wang2023**. This technique can be integrated into prompt chaining or meta-prompting workflows to ensure robust outputs.
- **Automated Prompt Engineering**: Frameworks like DSPy [93] programmatically optimize prompts by iterating over prompt variations and evaluating performance metrics. Unlike meta-prompting, which relies on the model's natural language capabilities, automated prompt engineering uses computational optimization, making it suitable for large-scale applications.

- **Temperature and Top-k Sampling**: Model parameters like temperature (controlling response randomness) and top-k sampling (limiting token selection to the k most likely options) indirectly influence prompt engineering outcomes [39]. For instance, lower temperature values (e.g., 0.5) produce deterministic outputs, while higher values (e.g., 1.0) increase creativity. Prompt engineers must account for these parameters when designing prompts, especially for tasks requiring specific tones or styles.
- **Agentic Workflows**: Advanced prompt engineering can emulate agent-like behavior, where the model autonomously decides subsequent steps based on prior outputs [90]. For example, a meta-prompt might instruct the model to "Generate a plan, execute each step, and adjust based on results." Such workflows combine prompt chaining, meta-prompting, and self-consistency to create dynamic, goal-oriented interactions.

4.2.4 Retrieval-Augmented Generation (RAG)

RAG is a hybrid framework that combines information retrieval with natural language generation to enhance LLMs. Introduced by Lewis et al. [94], RAG integrates parametric knowledge (encoded in model weights) with non-parametric knowledge (retrieved from external sources) to improve factual accuracy and contextual relevance.

In the RAG pipeline, a query initiates the retrieval of relevant documents from an external knowledge base, typically a vector-indexed corpus. Retrieval uses dense embedding models such as BERT [35] or Dense Passage Retrieval (DPR) [95] to identify semantically similar content. The retrieved information is then appended to the query and passed to a generative model, enabling outputs grounded in both learned and external knowledge.

This architecture mitigates key limitations of standard LLMs, particularly hallucinations and static knowledge, by conditioning responses on verifiable sources [96]. It is especially effective for domain-specific or private queries, where fine-tuning would be impractical [40].

Nevertheless, RAG presents challenges such as retrieval noise, increased inference latency, and the need to balance contributions from parametric and non-parametric knowledge [97]. Active research explores solutions including domain-specific retriever tuning, structured data retrieval, and improved fusion mechanisms [98]. RAG's modular design also opens avenues for scalable updates and theoretical advances in memory, reasoning, and contextual language modeling.

4.3 Time Plan

The following Figure 4.2 describes the time plan for approximate date of the project.

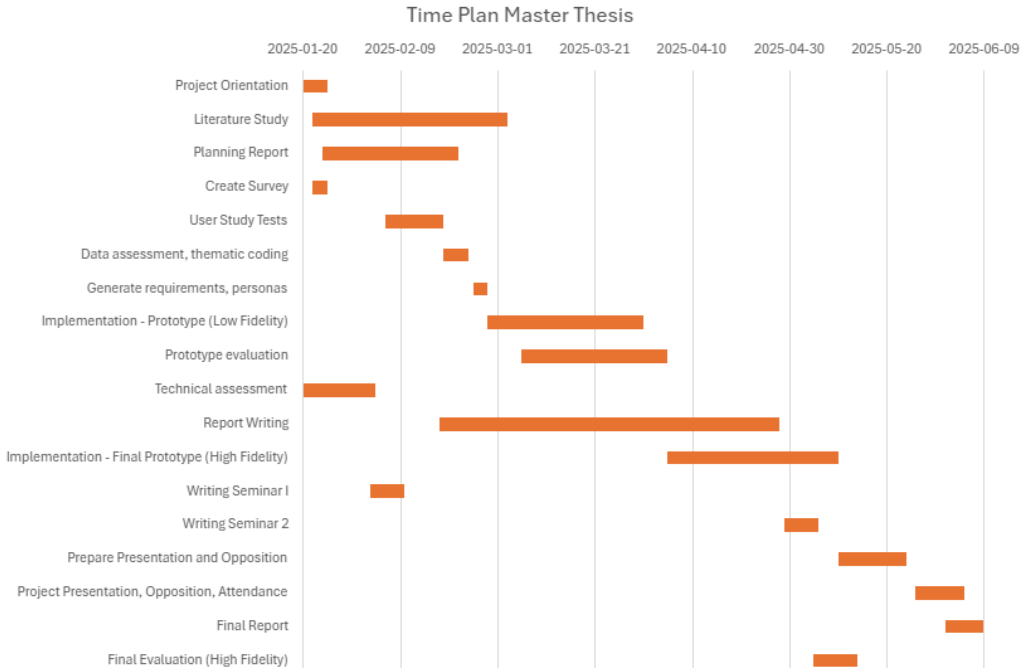

Figure 4.2: Time plan of the thesis as a GANTT-chart.

5

Process

This chapter outlines how we designed and built DirectorAI, using a user-centered approach to tackle the usability issues present in Director. It covers the initial scoping and problem discovery phases, the iterative design of the prototype, and the technical challenges involved in integrating AI into a complex, domain-specific system.

5.1 Preliminary Scoping and Feasibility

Before conducting formal user studies, we began with an initial exploration of the technical landscape surrounding Director. This phase took place early on, while we were still getting to know the tools and gathering participants for our studies. It played an important role in shaping the project's direction and helped ground the rest of our process in what was technically feasible and realistic within the time frame and scope of the project.

Our approach involved reviewing the available source code for Director, identifying which parts of the software we could work with and how, and mapping out the structure of the data we had access to. We also considered how Director connected to ProSim, which is the broader simulation platform it interfaces with. Early on, we determined that we did not have access to the full ProSim environment, which ruled out changes that would require modifying the underlying simulation software, such as camera control or environmental interactions. While these suggestions might arise later as potential improvements, it was important to understand that they would fall outside the bounds of what we could realistically address.

This early scoping also gave us a clearer understanding of the kind of AI functionality we might be able to implement. Initially, we considered possibilities like training a custom model to support scenario creation. However, we quickly realized that this would require access to a large, labeled dataset of simulation scenarios, which did not exist. Creating such a dataset would have been a major undertaking in itself and was not feasible given our resources. As a result, we began focusing more on how we could apply existing LLMs to augment the scenario creation process without custom model training.

Ultimately, this stage helped us set clear boundaries for the project. It allowed us to better interpret user feedback later in the process, as we could distinguish between

desirable but impractical features and problems that could actually be addressed in a meaningful way. It also informed the way we designed our user studies, allowing us to focus on pain points within the Director interface and scenario creation workflow that we knew we could feasibly explore and improve. In this sense, the preliminary scoping phase became a short but important part of the design process, shaping both the direction of our research and the tools we would later design and prototype.

5.2 Problem Discovery

To gain a preliminary understanding of how Director and ProSim are used, we conducted a short survey targeting Volvo Cars employees with either experience in these tools or an interest in their development. The goal was to identify usage patterns, common tasks, and any early frustrations or wishes users had. This was important to ensure that the design work would be grounded in real-world usage.

The survey included questions open-ended responses, such as:

- "What do you use ProSim for?"
- "Which features do you use most?"
- "Have you used Director, and if so, for what?"
- "What features do you like about Director?"
- "What challenges or feature requests do you have?"

A final yes/no question was included asking if the respondent would like to participate in future user studies on the matter.

While the response count was limited (nine respondents), and the answers were mixed and often vague, a few clear themes emerged:

- ProSim was used for a wide range of purposes, from visualizing scenarios and FSM behavior to running driver studies and showcasing car functionality.
- Of the few respondents who had used Director, several mentioned frustrations with configuring and timing actions, adjusting signals, and managing scenario sequences.

Although the survey did not reveal strong patterns, it hinted at a general sense of complexity and inconsistency in how the tools were used. This initial impression was further supported by early feedback and observational data from experienced users and developers, which indicated that users frequently encountered friction during scenario creation, particularly when trying to locate signals and understand their functionality.

To assess the scope and details of these issues, we conducted an initial structured user study with both experienced and potential users of ProSim and/or Director. Respondents of our survey who had answered yes to the question about participating in a user study were recruited to partake. The overarching goal was to investigate

the potential for AI-powered enhancements based on users' pain points and overall experiences.

The user studies, described in more detail in Appendix A, were conducted with eight participants. Each session lasted approximately 30 minutes. First, participants received a brief tutorial on the functionality of both ProSim and Director to ensure a shared baseline of understanding, acknowledging that prior experience and proficiency with the software varied. Participants were then given time to explore Director independently to become more familiar with its interface and features.

During the study, participants were encouraged to think aloud by verbalizing their thoughts and decision-making processes to help us gather as much insight as possible into their behavior, challenges, and perceptions.

The main task required participants to create a short scenario involving:

- Changing camera angles and positions
- Opening and closing vehicle doors
- Activating the car horn for a specified duration

This scenario was chosen because it involved common, easy-to-understand signals that also varied in type and parameters. Minimal guidance was provided to ensure a fair and unbiased representation of each participant's experience.

Primarily qualitative data was documented during the user studies. Such information included observations of hesitation, confusion, and general behavior. This qualitative data provided nuanced insights into usability issues. In addition, after the tasks were completed, participants were interviewed about their experiences, focusing on what aspects of the software they found satisfying or frustrating.

From these interviews and our observations, three major categories of user frustration emerged:

- Difficulty finding signals
- Lack of information on how signals and their parameters work
- Limited feedback and unclear information about how the program functions overall

Some of the identified issues stemmed from the underlying implementation of signals, elements that could not be directly addressed within Director alone. This prompted a careful consideration of which problems could be realistically solved at the level of Director and which would require changes to the broader software ecosystem.

5.3 Problem Definition

Following our user studies, we analyzed both observational and interview data to identify the most pressing usability issues in Director. Our insights were primarily derived from qualitative methods, particularly think-aloud protocols and post-task

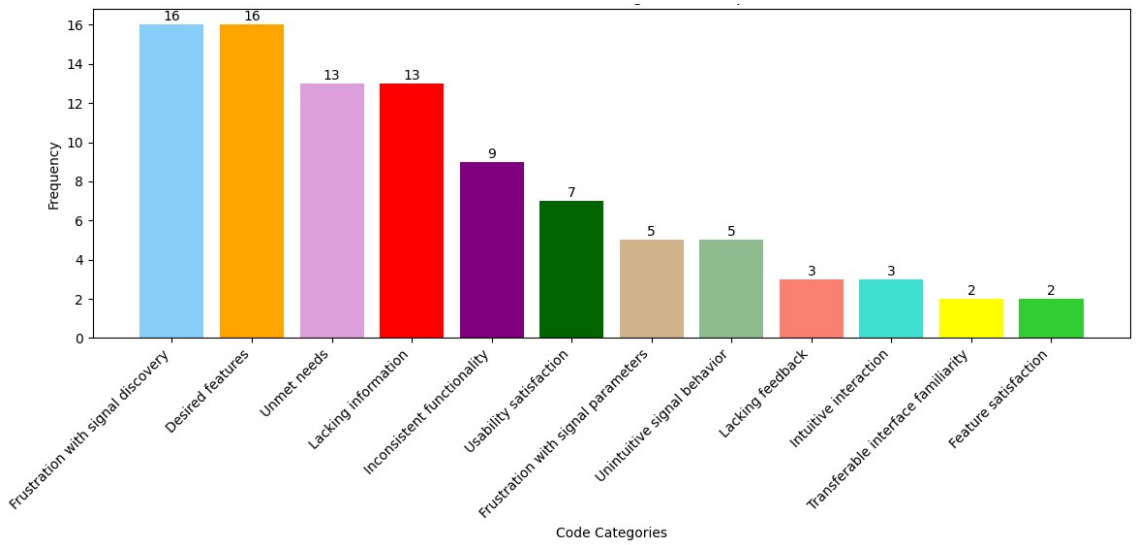

Figure 5.1: Distribution of code categories in interview responses.

interviews. Thematic coding was applied to the interview transcripts, allowing us to identify recurring patterns, frustrations, and pain points experienced by users.

5.3.1 Data Analysis

One of the most frequently observed issues was the difficulty participants faced in locating the correct signals when creating scenarios. This challenge was evident both in observed behavior and participant feedback. Participants often attempted to search for signals using natural language terms such as driver's door or trunk, only to receive no results, as Director requires queries to match system-specific signal names like Door FL Open or tailgate. The absence of synonym support or semantic flexibility created friction, particularly for non-expert or occasional users unfamiliar with the tool's naming conventions. Even experienced users reported that the need to recall or uncover exact terminology interrupted their creative flow.

The full list of identified usability themes, along with the frequency of their occurrence across sessions, is presented in Figure 5.1. This visualization offers an overview of the most prominent barriers to effective interaction, helping to contextualize the relative impact of this and other issues.

These issues were compounded by inconsistencies in how signals were structured. Some actions were represented by discrete signals (e.g., Door Open and Door Close), while others, like the horn, used a single signal (hornSoundOn) with parameters to control behavior. Users frequently searched for signals that did not exist, such as hornSoundOff, not realizing the action was handled differently. This lack of consistency increased the cognitive load and hindered intuitive interaction.

A closely related concern was the configuration and interpretability of signal parameters. Although less commonly raised in interviews, this issue became apparent during observations and through our own experience using the tool. Parameters sometimes used technical labels (e.g., a parameter named boolean), which confused

non-technical users. Others, such as color configuration for lights, used ambiguous input formats: RGB values required floats (0-1), but users often entered integers (0-255), leading to unexpected behavior. Without clearer information or feedback, even simple parameter adjustments became a point of friction.

Something worth noting is that codes like *'Frustration with signal parameters'* appear in the lower half of the identified codes, yet parameter configuration became a major focus during the design and implementation of DirectorAI. There are a few reasons for this. One is that some user complaints or requests related to features that were simply not feasible for us to implement. For example, a *'Desired feature'* was the ability to "fly" in ProSim, whereas the simulation currently only supports walking. Implementing these types of features was beyond our scope, especially since we did not have access to ProSims source code. Another reason signal parameters became a focus is that codes like *'Lacking information'* often referred to missing or inadequate documentation and metadata related to parameters, which made them difficult for users to understand or use effectively.

Table 5.1: Selected Participant Quotes

Participant	**Quotes**
P1	"There is no guide for how the signals are different from one another. A lot of signals have unclear info." "The camera doesn't work as expected. And it's unclear what the difference between the camera and ego character is."
P2	"It lacks info for what signals do and what the parameters mean." "Some signals do just one thing but some do multiple things."
P3	"Improve the way to find signals, [it needs to] match with what I actually want to do."
P4	"You need that instant feedback. By the time I finally have a scenario ready, the requirements can have changed." "This type of work must be more intuitive. Like a car is."
P5	"[Other teams] always need to ask experts for help, because the tool is too technical for them."
P6	"There's so much potential here, but you need to make it faster." "It should be as simple as possible, so more people can try." "Getting started is hard, so many people skip it."

Beyond these specific issues, participants also expressed a broader sense of unfulfilled potential. Several interviewees envisioned using Director for fast-paced ideation or live scenario building during team discussions but found the program too rigid and slow to support such workflows. Others noted that while testing is valuable in Director and ProSim, it is often skipped due to the complexity and time investment

required to build even simple scenarios. Selected participant quotes that influenced our understanding of Director's workflow issues and the following work on DirectorAI can be seen in table 5.1.

These findings led us to a clear problem framing: Director, while powerful, is limited by its reliance on system-specific syntax, technical terminology, and rigid workflows. These constraints make the tool difficult to access for new, occasional, or non-technical users, and slow even for experienced ones. This challenge is not unique to Director, but indicative of a broader issue in many complex software tools, that being the gap between user intent and system operation.

5.3.2 How AI Could Help

At this point, we began exploring how AI, specifically natural language processing (NLP), might help bridge this gap. The idea was planted early in the project, as we reflected on how similar issues appear in other domains: software development, data analysis, or engineers working with circuit design [99]–[102]. Users may know what they want to do, but not necessarily how to express it in system-specific terms, or may greatly benefit from the automation provided by AI. AI interfaces have increasingly shown promise in translating natural human requests into structured commands, enabling smoother interaction with complex systems.

This potential aligned closely with the challenges we uncovered in Director. AI could help users find signals even if they use imprecise or conversational terms, for example interpreting driver's door as Door FL Open, or mapping boot to tailgate. Language models can infer synonyms, correct misspellings, and interpret user intent, especially when trained or prompted with domain-specific context. Moreover, AI can act as a knowledge proxy: we can encode expert knowledge into prompts, descriptions, or examples, allowing the AI to answer questions and provide guidance without requiring users to rely on internal documentation or interrupting workflows to ask a colleague.

To further explore the feasibility of our approach and gain a deeper understanding of where the current issues stem from, we consulted with Directors developers. These stakeholders provided valuable insights into why certain usability issues persist in the current software. When asked why documentation improvements had not been addressed, the developers pointed to the sheer scale of the system and how Director has approximately 900 unique signals. Updating and standardizing all of them, along with their metadata and parameters, would be extremely time-consuming. Due to resource limitations and other focus areas, this has not been a priority.

Regarding the technical language used in Director, the developers explained that it aligns with their workflows and how the software has historically been used. For example, a value like *"HornSoundOn"* with a boolean parameter controlled by true/false makes perfect sense to a software engineer. As a result, updating the structure or terminology of signals may improve clarity for some users, but risks inconveniencing others who rely on or have adapted to the existing conventions.

These conversations reinforced our belief that AI could offer a viable alternative,

not by replacing Director's structure, but by sitting between the user and Director's functionality. Rather than requiring extensive documentation rewrites, restructuring Director's backend, or disrupting existing workflows, AI could provide a semantic bridge where it interprets the user's requests, selects relevant signals, and assists in parameter configuration. In summary, our problem definition rests on three main points:

- **Discoverability**: Users struggle to find the right signals using natural language or intuitive phrasing.
- **Interpretability**: Signal parameters are difficult to understand or configure correctly without technical knowledge.
- **Speed and accessibility**: Scenario creation is slower and more effortful than it needs to be, especially for less technical users.

These challenges pointed us toward a solution that leverages AI to interpret user intent, embed expert knowledge, and reduce cognitive load when using Director by making the tool more accessible without requiring a fundamental redesign.

5.4 Design and Implementation

This section presents the design process and implementation details of our AI assistant for Director. The goal of this tool is to improve usability in the scripting of 3D simulation scenarios by allowing users to interact with Director via natural language. We begin by introducing terminology used throughout this section, followed by a breakdown of the assistant's architecture, interaction modes, and AI prompt strategies. Lastly, we outline the specific components developed during this project.

5.4.1 Terminology

To ensure clarity, we define several terms used throughout this section:

- **Signal**: A command issued from the Director program to ProSim, where the 3D simulation occurs. Each signal controls a specific aspect of the simulation, such as opening and closing car doors, adjusting the camera, or modifying weather conditions. Signals are added to Director's timeline as actions, each with a defined start time and associated parameters. When the timeline is played, these actions are triggered at the specified times, sending the corresponding signal and its values to the simulation.
- **AI Mode**: A selectable interaction context that defines the behavior of the AI assistant. Each mode corresponds to a distinct use case (e.g., camera positioning or signal editing) and determines how user prompts are interpreted and processed by the assistant and LLM.
- **Generator**: A modular processing unit responsible for two tasks: (1) composing and sending a task-specific system prompt to the LLM, and (2) handling

the returned data in a structured way to produce meaningful changes in the simulation environment.

- **Classifier**: An AI module that dictates which AI generator is most appropriate for the given input.
- **Pipeline**: A chained sequence of generators and classifiers that transforms a user prompt into application-specific output. Pipelines allow for multi-step processing, such as refining AI outputs through further generators or user interactions.

5.4.2 Early Attempts: Simple Prompting

Our initial approach was simple: we added a chat window to the existing interface where the user could enter their prompt, which we would later pass to the AI. The user's prompt, along with the entire list of available signals and their descriptions, was sent to a GPT-4o-mini model via an API request. The model was instructed to recommend the most appropriate signal based on the prompt. Despite its simplicity, this worked surprisingly well in many cases. The model could often infer what the user wanted and suggest one or several relevant signals.

However, several limitations quickly became apparent. First, signal descriptions alone were often not detailed enough to make an informed decision. Two signals might appear similar and have similar descriptions, for example 'Door Open FL' and 'Ext Door Handle R1 L Ui,' but behave quite differently. Of these two signals, the first one directly opens the door, while the other simulates a user pulling the handle, which may fail if the door is locked. Secondly, even when the correct signal was found, users still had to manually configure its parameters, sometimes without knowing what the parameters meant or how they worked. Third, some signals required contextual or spatial understanding not conveyed in their metadata. For example, moving the camera to a specific location requires understanding not only XYZ coordinates but how those coordinates map to meaningful positions in the 3D space.

5.4.3 Evolving the Design: Introducing AI Modes

To address these challenges, we began breaking our system into smaller, purpose-built AI agents, what we called modes. Each mode handled a specific task and was equipped with special instructions and knowledge about that task. For example, a "Camera Mode" included preset positions and contextual knowledge about how the camera behaves in the simulation space. Similarly, a "Human Mode" knew how to place a human character near interactable elements without causing logical inconsistencies (like trying to walk through the vehicle). We also had more general modes:

- Suggest Mode, which provided signal recommendations and let users decide which to use.

- Generate Mode, similar to suggest mode, but would select a signal for the user.
- Parameter Mode, would suggest a signal and then also fill out its parameters.

These modes improved both accuracy and usability, but created new challenges. Users had to know which mode was appropriate in each case. Swapping between modes became tedious, and accidentally entering a prompt in the wrong mode would cause incorrect generation, which would manually have to be fixed. While some expert users might prefer manual control, novices often made mistakes or found the process frustrating.

5.4.4 The Full Pipeline: A Structured AI Workflow

To streamline the experience, we transitioned to a more intelligent and fully automated pipeline architecture. Rather than relying on the user to select a mode or re-enter prompts for each signal, we designed a multi-step system that broke down the task, interpreted it, and generated all necessary signals, parameters, and timings automatically. At this point, we also began moving away from specific AI modes, which users had to select, and repurposed the underlying functionality to be an AI generator. Much like modes, these generators would have specific instructions depending on their role, and could handle returned data in an appropriate way. The pipeline can be seen in Figure 5.2, and consists of the following steps:

1. Sub-Prompt Generation

The user inputs a natural language request, such as "Move the camera to the driver's door and open it." Since each action in the simulation typically maps to a single signal, we first break this prompt into manageable sub-prompts:

1. "Move the camera to the driver's door"
2. "Open the driver's door"

Each sub-prompt must be sufficiently specific to guide signal selection and parameter generation, while remaining concise as to not confuse future steps with unnecessary information.

2. Task Classification

Each sub-prompt is then routed through an AI task classifier, which determines the most appropriate AI generator to handle it. For example:

- Prompts involving visual positioning are routed to the Camera Generator.
- Prompts involving the 3D human avatar are routed to the Human Generator.
- Prompts about seating use the Seat Generator.
- Everything else is sent to the Signal Generator, a more generalized AI equipped to handle diverse cases.

3. Specialized Generators

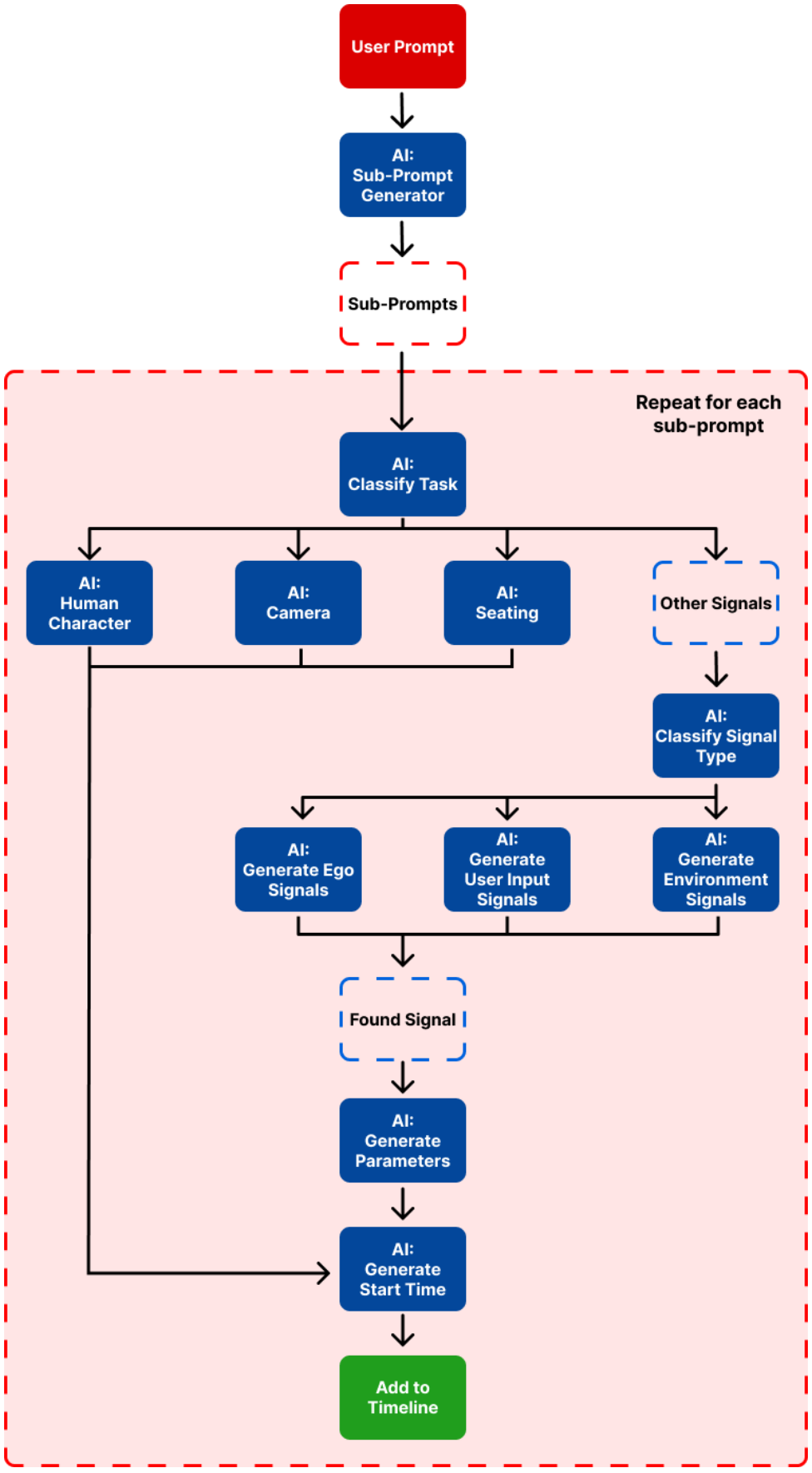

Figure 5.2: The structured pipeline showing how a user's prompt is processed by the system.

Each generator is equipped with curated knowledge and contextual instructions, what we will refer to as contextual scaffolding. For instance, the Camera Generator knows that X-axis movement corresponds to lateral shifts in the car's forward direction, and is given known reference positions like driver seat or behind the vehicle, and overall car position coordinates. The Human Generator understands basic car geometry, interaction ranges, and safety constraints. Each generator is tightly scoped, which improves reliability and output quality. An example visualization of the camera generator can be seen in figure 5.3.

Within the generalized Signal Generator, we introduce another level of classification where it determines whether the signal falls into one of three categories: ego signals (direct vehicle manipulation), user inputs (actions like pulling a handle), or environment/scenario controls (e.g., road conditions). This enables more targeted generation, especially for ambiguous actions with multiple valid interpretations. For instance, it addresses the common issue of distinguishing between pulling the door handle and opening the door. The user does not need to understand the technical differences between these actions, but the classifier can automatically determine the correct interpretation based on context, ensuring the appropriate signal is generated.

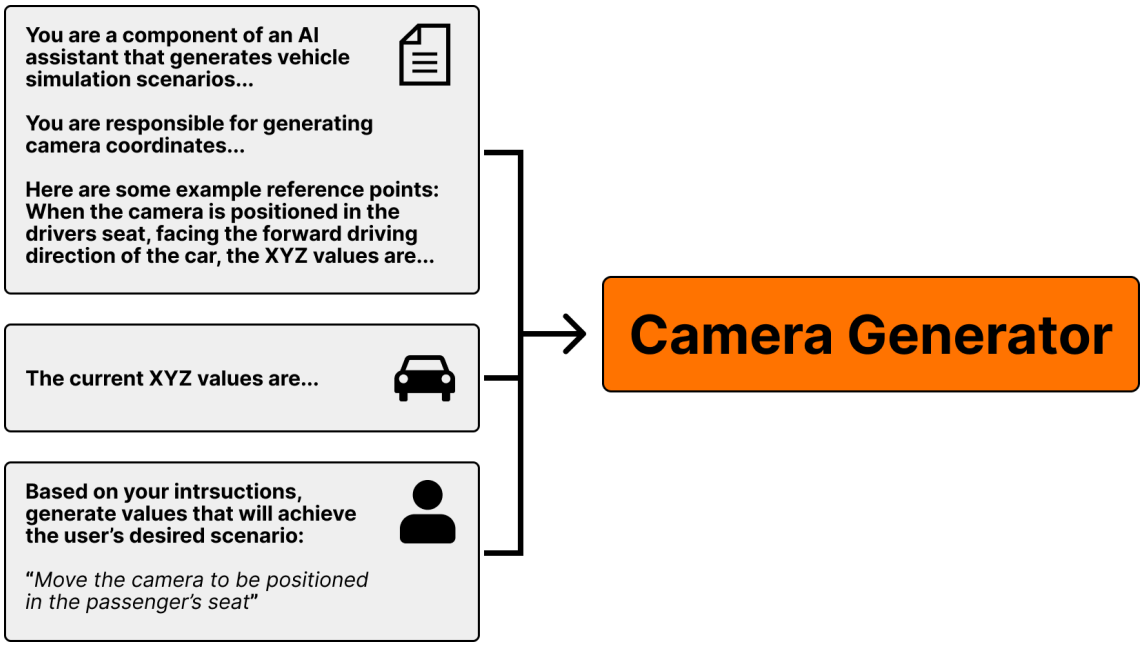

Figure 5.3: A visualized example instance of the camera generator, showcasing what information it receives.

4. Parameter Generation

Once a signal is selected, a dedicated component generates appropriate parameter values based on the sub-prompt, signal description, data types, and constraints. While some specialized generators (such as the Camera or Human Generator) handle parameters internally, this step is crucial for more abstract or unfamiliar signals.

5. Timing Generation

The simulation timeline is stored in JSON format, with a lot of irrelevant metadata for this generator. Thus, the timeline is first formatted to a more readable format, including only the most crucial data for the generator to complete its task. This generator is necessary in order to maintain logical sequence and coherence, the formatted timeline is fed to the Start Time generator that inspects the current timeline, referencing previously inserted signals, and determining when the new action should occur. This generator knows that, for instance, moving certain parts of the car takes time, and if a new signal is to be inserted, it must consider the animation time of previous actions. A visualized example instance can be seen in figure 5.4.

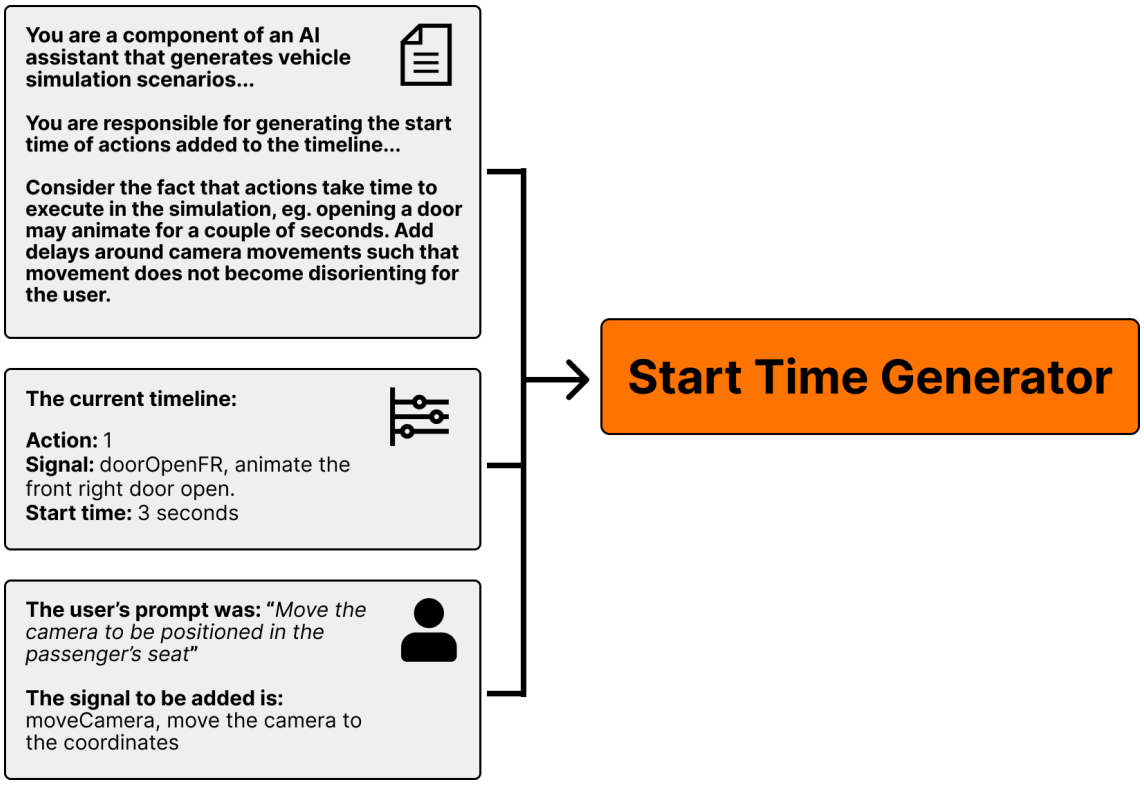

Figure 5.4: A visualized example instance of the start time generator. Showcasing what information it receives and what it must consider to generate appropriate timings.

Once the signal, parameters, and start time are generated, the system automatically adds the signal to the simulation timeline.

5.4.5 Why a Pipeline?

We were initially inspired by Polak and Morgan's [103] ChatExtract method, which utilizes a similar pipeline architecture. ChatExtract operates by employing a series of specialized prompts, each of which builds upon and validates previous outputs. These outputs are continuously fed downstream, promoting the generation of a final, high-quality result.

In our case, this structured pipeline approach, while technically more complex, offers several advantages over a single monolithic prompt. It allows each step to operate with focused context and optimized instructions. For example, the Signal Generator can be told "A previous step has already determined this is an ego signal." Such modularization improves reliability, reduces ambiguity, and enables targeted improvements to individual components.

From an interaction design perspective, the pipeline offers an intelligent, conversational experience that adapts to user intent without overwhelming them with decisions. It shifts cognitive load from the user to the system, embodying the principle of progressive disclosure, where users receive only what they need, when they need it [104].

5.4.6 Technical Considerations and Trade-offs

Implementing the pipeline came with several trade-offs. While it significantly improved both accuracy and task completion rates, it also introduced higher latency and operational costs. In the early suggest mode, a single API call cost around 0.024 SEK. But a full pipeline request, especially one involving multiple signals each with parameters to be filled out, could easily reach 1 SEK or more. This increase was not just due to a greater number of calls, but also because each individual call became more expensive.

Originally, many tasks used the cheaper GPT-4o-mini model. However, most API calls were eventually migrated to the larger, more capable GPT-4o. This shift was driven by two key factors. First, the larger model consistently delivered more accurate and reliable results, which mattered a lot in a workflow. Second, the GPT-4o-mini version we had access to did not support a crucial feature: structured outputs.

Structured output supports the definition of a JSON schema that the model must follow, ensuring that the response includes exactly the fields needed, in the correct format. For example, when editing a camera signal, we could guarantee that all required parameters, such as position and rotation, were always returned. This eliminated a whole category of edge cases, like missing X-values or ambiguous partial responses, and drastically simplified post-processing.

Despite the increased cost, the switch proved worthwhile. The boost in result quality and the reduced development overhead from consistent outputs all made the trade-off more than acceptable.

Another notable limitation was the manual effort required to configure each gen-

erator in the pipeline. These were not generic plug-and-play components as each generated output needed a carefully tuned prompt and contextual scaffolding to perform reliably. This meant that every generator had to be handcrafted and iteratively refined.

For example, the camera generator's prompts underwent multiple revisions. Small changes such as rephrasing how position or orientation should be described could significantly impact the consistency of the output. These changes were validated through targeted testing, where the generator was asked to produce a variety of camera positions and angles to see how well it followed instructions.

User feedback also played a crucial role in uncovering edge cases and unusual phrasing that we had not anticipated. These interactions helped us refine prompts further and improve robustness across a wider range of inputs.

This level of hand-tuning was essential to achieve a reliable pipeline, but it came at the cost of flexibility and scalability. Each component required dedicated design attention, and chaining them together added latency to the assistant, an inevitable trade-off for the improved structure and interpretability that the modular approach provided.

5.4.7 Why GPT-4?

The language model used throughout the assistant was GPT-4o and GPT-4o-mini, accessed through the internal Azure infrastructure of Volvo Cars. This choice was primarily dictated by organizational constraints. Due to the sensitivity of the signal data and strict internal approval processes, we were restricted to using pre-approved infrastructure that supported GPT models. Although there are other competitive models available (e.g., Claude, Gemini, LLaMA), our project did not aim to benchmark model performance, as its focus was on interaction design and system usability. Moreover, recent evaluations such as the LMSYS Arena Leaderboard [105] and the MMLU benchmark [106] suggest that state-of-the-art language models perform similarly across a wide range of tasks, with GPT-4 consistently ranking near or at the top. This supports the assumption that the pipelines and interaction flows designed in our project would remain broadly applicable in comparable high-performing models. Furthermore, GPT-4's support for structured outputs made it particularly well-suited to the requirements of our assistant.

5.4.8 From Generation to Correction: Introducing the Edit Pipeline

While the initial generation pipeline marked a major step forward in automating scenario creation, it did not take long before a new challenge made itself clear. Even with carefully structured generators and solid output, the AI wasn't perfect. Inevitably, things would go slightly wrong, such as incorrect coordinates, misunderstood user intent, or simply suboptimal signal values. And when that happened, the tool hit a hard limit.

Up until this point, the assistant could generate useful results, but it had no understanding of failure, let alone any way to fix it. If the AI misunderstood the user, or placed a camera a few meters too far to the left, the only real option was for the user to start over, manually adjust the signal in the Director UI, or try to prompt the AI again and hope for a better outcome. None of these options solved the core issue we were trying to address: reducing the amount of technical micromanagement users had to do in order to get a scenario just right.

This shortcoming became especially obvious during early user testing. One participant gave the following feedback:

> "If the camera generated slightly incorrect coordinates, I just want to be able to say: a bit more to the left."

That kind of feedback directly inspired the next major feature of our assistant: the Edit Pipeline. We took everything we had learned from the creation pipeline, its structure, its ability to route tasks to specialized modules, the contextual scaffolding, and applied it to a new set of tasks: fixing mistakes.

5.4.9 Design of the Edit Pipeline

Much like the generation pipeline, the edit pipeline is a multi-stage process. But instead of starting from scratch, it works with existing timeline content. Its job is to interpret a user's correction and figure out how to apply that change to the current scenario.

The first step in the edit pipeline takes in two things:

1. The user's correction or complaint ("Move that camera back", "I wanted the door to stay closed", "This light's color is wrong").
2. The current state of the scenario timeline (similarly to how the Start Time generator reads the timeline).

Using this context, the first step's goal is to diagnose the issue. What, exactly, is causing the mismatch between the user's intention and the current scenario? Is there a signal that needs to be removed or replaced? Are the parameters slightly off? Is the timing misaligned?

Once the issue is located, the pipeline proceeds in a similar fashion as the original pipeline: routing the task to a specialized generator based on the type of problem and the signal involved.

For example:

- If the user says, "No, I wanted yellow headlights, not green", the assistant identifies the signal controlling the headlights, determines that the correct signal type is being used (so no need to replace it), and passes it to the Parameter Generator with an added note such as: "User wants yellow lights, but these parameters currently generate green. Adjust accordingly."

- If the issue is more structural, for instance, if the AI added a door-opening signal when the user wanted the door to stay shut, it could decide to remove the signal entirely.
- If the object and parameters are fine, but the issue is with when the event occurs, the task can be routed to the Start Time generator, with an instruction like: "This signal should happen 3 seconds later."

This internal routing allows the edit pipeline to behave intelligently even when the user input is vague or informal. The user does not need to be concerned with the details about why something is wrong, and they do not need to have any idea how to fix it. They just need to express what is wrong, and the AI can do the rest. Since many of the signals did not have any description of how they worked, the Edit mode helped in correcting and learning from mistakes.

5.4.10 Trade-offs and Implementation Notes

As with the original pipeline, the modular design of the edit pipeline came with both benefits and drawbacks.

Benefits:

- Context-awareness: Because it always receives both the user input and the current state of the timeline, the edit pipeline can make highly targeted fixes.
- Flexibility: Users don't have to remember commands or be precise in their wording. For instance, users don't have to know that it is the X-axis that has incorrect values, as corrections like move that back a bit can still result in valid edits.
- Efficiency: It reduces the need to regenerate entire sequences or go into Director manually for minor tweaks.

Challenges:

- Much like all the generators related to creating content, implementation for the generators for fixing issues required significant tuning to make sure they received the right context.
- The assistant could occasionally misdiagnose the issue, especially if user input was ambiguous.

5.4.11 Enabling Safe Experimentation

During testing, we observed cases where users would write a prompt, let DirectorAI generate content, and then feel dissatisfied with the results. At that point, they faced a dilemma: to either use the edit pipeline and try prompting the AI to undo the changes, or manually revert the scenario themselves. The latter was tedious, reintroduced the very friction the AI was meant to reduce, and required users to remember what the scenario looked like before the AI's changes. In other words, the AI's actions were effectively destructive, leaving no easy way to go back.

To address this, we introduced a 'Revert' function. Each time the AI generated content, the corresponding chat message included a Revert button. Clicking it restored the scenario to the state the timeline was in when that specific AI response was returned. Essentially, each set of AI-generated actions was saved, allowing users to jump back and forth between states. This transformed DirectorAI's behavior from destructive to non-destructive, enabling users to experiment freely with prompts without risking their current progress if the outcome wasn't satisfactory.

5.4.12 Switching Between Pipelines: Manual vs. Automatic

In our initial design, the creation and editing pipelines were meant to remain separate, but we planned to let the AI automatically determine which pipeline a user input should go through. For example, a prompt like Move the camera forward a bit would be routed to editing, while Honk the horn would go to the creation pipeline since this is seen as a new signal action. However, this automatic routing turned out to be somewhat unreliable. The AI often misclassified user intent, which led to confusing results, especially in borderline cases where intent was ambiguous.

Because of this inconsistency, we decided to require users to manually choose whether they were in creation or editing mode. While not ideal from a usability perspective, it gave users clarity and control, and avoided misinterpretations that would undermine trust in the assistant.

Another limitation that emerged, regardless of routing method, is that the assistant's current functionality cannot handle prompts that mix creation and editing. For instance, a user saying Open the door, but also have the camera move inside the car instead of right outside the door includes both an instruction to generate new content and to modify existing content. Since each pipeline operates independently and assumes a single mode of intent, mixed prompts like these are not supported at this time.

This kind of interaction could be supported by further preprocessing user input to split it into corresponding task types, or merging the two pipelines into a more flexible, unified process. However, this introduces complexity and reopens challenges we had previously worked around such as the AI's inconsistency in distinguishing between creation and editing, which is why this functionality was made a manual choice in the first place.

5.4.13 The Dual Pipeline Architecture

Together, the creation and editing pipelines formed the core of our AI-augmented approach to working with the simulation timeline. The creation pipeline focused on structured creation, breaking down the task of scripting scenes into discrete, specialized steps. This modular design allowed each generator to operate with clear context and intent, producing more accurate results than a single, generalized prompt could have.

The editing pipeline built on these same principles but shifted the focus from cre-

ation to correction. It enabled users to revise existing AI-generated content through natural language, identifying what part of the timeline was causing confusion or dissatisfaction, and routing the fix through specialized generators, whether that meant adjusting a parameter, changing the timing, or removing a signal entirely. This preserved the benefits of automation while acknowledging the inevitability of imperfect generation.

Both pipelines required a significant degree of contextual scaffolding, manual prompt design, iterative testing, and thoughtful coordination between components. But together, they transformed the assistant from a tool that merely generated suggestions into one that could also respond meaningfully to user feedback. This dual capability, creation and revision, was central to making the AI assistant not just a one-off generator, but a collaborative partner throughout the scenario scripting process.

5.4.14 Improving Explainability

As we continued testing, another issue quickly became clear: explainability. When new users tried the assistant, things usually went well at first, but as soon as something broke or behaved unexpectedly, it was often difficult to figure out why. Some errors were obvious; maybe we had parsed data incorrectly or something crashed due to a bug. These kinds of issues were straightforward to identify and fix.

However, when the problem stemmed from how the AI interpreted a prompt, misunderstood the timeline, or followed unclear instructions, things got murkier. Even for us, who had developed the assistant and knew it inside out, understanding why the AI behaved a certain way could be surprisingly difficult. This suggests that for typical users, making sense of such behavior would likely be even more challenging.

Therefore, towards the end of development, we introduced a simple but powerful feature: motivation statements. We added a requirement to most AI generators that they must explain why they generated what they did. This meant that each generator, whether for the camera, human avatar, or other signals, had to provide a short rationale based on the user's prompt. What part of the input led to the decisions it made? Why was a specific camera position chosen, or why were particular parameters set?

This change made a big difference. From a development perspective, it helped us debug AI behavior. When something looked off, we could immediately see how the AI had interpreted the prompt and whether it had misunderstood the user or been given ambiguous input. It also provided valuable insight into how prompts could be rewritten or improved.

From a user perspective, it brought a new level of transparency and trust. When the assistant generated something unexpected, users were no longer left guessing. They could see exactly what the AI thought they meant, and why it acted accordingly. It didn't make the assistant perfect, but it made it more understandable, and that alone improved the experience.

5.5 Evaluation

Following the completion of the design and development phase, we conducted a final set of user studies to evaluate the prototype in a more structured and formal manner; this study can be seen in detail in Appendix B. While iterative evaluations and feedback had been a continuous part of the development process, this session was intended to assess DirectorAI as a more complete experience and to gain deeper insight into how users interacted with and perceived the AI-assisted scenario creation process. This included questions of usability, perceived utility, trust, and user expectations.

The evaluation setup mirrored the structure of our earlier discover study, with participants being invited to complete a scenario-based task. However, unlike in the earlier session, where the scenario was constructed completely manually, this session asked participants to use DirectorAI to accomplish the same kind of goal. Participants were instructed that the assistant allowed them to describe their desired scenario using natural language, and that the AI would attempt to construct it on their behalf. They were introduced to the two modes of interaction available in the tool: create, which is used for adding new elements, and edit, which is used to modify or correct existing content.

Deliberately, we did not provide extensive instruction on the AI's inner workings or limitations. This decision was motivated by a desire to observe participants' expectations as naturally as possible: what they assumed the AI could do, how they explored its capabilities, and what assumptions they brought to the interaction. We considered this an opportunity to study not only DirectorAI's performance, but also the emerging ways users approached, interpreted, and interacted with the system.

It is worth noting that some participants diverged from the predefined scenario or attempted tasks that deviated from the given instructions. We interpreted this behavior not as noise, but as indicative of a curious mindset and a test of the assistant's flexibility. These instances provided valuable insight into users' curiosity and willingness to explore the limits of what the AI could do.

The following sections present the main findings from our evaluation of DirectorAI, organized by topic. Each subsection includes our interpretation of participant feedback, supported by participant quotes. To distinguish these participants from those in the exploratory study, they are instead labeled with 'U'.

5.5.1 Participant Background and Preconceptions

We began the session by asking participants about their prior experience with AI tools. This was intended to establish a sense of their familiarity with the domain and whether they entered the session with preconceived expectations of how AI should behave. Most participants had experience with tools such as ChatGPT or GitHub Copilot, and several participants referenced the need to "learn the language" of AI assistants in order to use them effectively. This notion, that AI tools are powerful but require adaptation, recurred throughout the evaluation.

5.5.2 General Impressions

Participants generally responded positively to the integration of AI into the scenario creation process. Several participants commented that it helped reduce the time and effort required to produce a working scenario, particularly in terms of automating low-level tasks such as placing signals, configuring parameters, or setting timings. One participant described the experience as faster and better, and mentioned that they preferred to try the result immediately rather than reading the feedback messages provided by the AI. Another participant praised the ability to experiment freely, using the tool's ability to revert AI-generated changes as a way to explore alternative outcomes without risking current progress.

> "Configuring [parameters] is much easier than before. This saves time." (U1)

> "It helped me [find] signals, I can adjust [timings] myself." (U2)

> "It was better and faster. I don't want to read [AI motivations] though, I would just test and then maybe read if it doesn't work." (U1)

> "I like the revert, it means I can try and experiment." (U2)

A common theme across several participants was the desire to use DirectorAI either by constructing one part of the scenario at a time, or by describing the entire scenario in one input. The variation in preference suggests a need to support multiple authoring styles. DirectorAI supported both these input methods thanks to the sub-prompt generator step of the pipeline.

> "I like how it can do [multiple steps] at once, but I might do it one at a time." (U3)

> "I can explain [the scenario] more roughly and still get something to work with." (U5)

> "If i had a scenario [description] from before, I might just paste it [in the chat]. Otherwise I probably would've done [prompting] step by step, because that is how I usually work." (U2)

5.5.3 Trust and Effectiveness

When asked whether the AI-generated scenarios worked as expected, most participants agreed that they did overall work. While some minor adjustments were sometimes required, often related to edge cases such as ambiguous signals or specific camera viewing angles, the tool was generally seen as producing usable and meaningful outputs. Trust in the DirectorAI's output appeared to be shaped by participants' ability to understand and, when needed, correct its behavior. One participant noted that they felt in control, even when the assistant made unexpected decisions, while another commented that the tool "ran away in a good way," meaning it expanded on the input in ways they found valuable.

> "There is a good balance between [user] control and how much the AI decides." (U1)

> "The camera was put too close to the door. But it was [by the door]." (U5)

> "It's like ChatGPT, I check and fix the output." (U5)

> "It ran away in a good way. It took the prompt and made it better." (U3)

However, limitations were also identified. Some participants felt that DirectorAI could benefit from clearer, more visible feedback, especially during moments when the assistant was processing commands. One participant mentioned that they were unsure whether the assistant was "thinking" or frozen, suggesting the need for a more obvious loading or progress indicator to help manage user expectations and prevent confusion. Additionally, participants noted that the AI sometimes took instructions too literally or failed to grasp context, requiring users to manually intervene or rephrase prompts.

> "Its hard to know if it's thinking or stuck." (U5)

> "It can be a bit literal. Like when it put the camera too close [to the tailgate]." (U1)

Despite these issues, there was general consensus that DirectorAI was particularly valuable for getting started with scenario creation. Participants appreciated its ability to quickly generate a first draft of a scenario that could then be manually refined. This iterative dynamic, where the assistant handles the initial heavy lifting and the user steps in for detailed adjustments, emerged as a central pattern in how participants worked with the assistant. Rather than expecting perfection, users seemed comfortable treating the AI as a collaborator, especially when the system's output was transparent, reversible, and easily editable.

> "It helped me with finding signals and placing them. I don't mind fine-tuning them myself." (U2)

> "I just want to create stuff and test it, I can fix the details later." (U3)

> "It can create a rough draft, then I can adjust [the output] myself." (U4)

> "It gives me a lot as a foundation, then you refine yourself. I can create now, then adjust later." (U5)

5.5.4 Interaction Framework

Overall, the interaction framework was seen as intuitive. The ability to interact with the AI via chat, combined with real-time timeline updates and an option to undo changes, contributed to a sense of fluidity. Participants appreciated not having to navigate complex timelines or configuration menus manually. For some people, the AI acted almost as a collaborative partner: one participant imagined a future

version where voice input would allow them to just talk to it, or even have it act as an intelligent assistant during design meetings.

> "I don't have to search and scroll through these large menus anymore." (U1)

However, not all aspects of control were equally successful. Some participants experienced confusion or frustration when attempting to create a scenario with DirectorAI, particularly when the assistant had to understand temporal sequences or interactions based on ambiguous input. One participant also mentioned the difficulty that may come with debugging and understanding DirectorAI's output before it can be fixed. Other participants wanted the ability to ask the assistant how to make a change, rather than only asking it to perform the change, pointing toward a blended interaction framework that supports both automation and educational functionality.

> "It's not perfect with the timings, at least when I don't specify." (U4)

> "The timings are too quick." (U3)

> "When it does make mistakes, I kind of have to catch up [mentally] to understand what it has done." (U4)

> "I want to ask for instructions, like how to change and edit the camera manually." (U2)

5.6 Post-Evaluation Adjustments

After the feedback from the user studies, we polished and added several features to DirectorAI. When it comes to the UI, we moved DirectorAI to the left-hand side to be able to have both the tabs 'Property' and DirectorAI fully open at the same time. We also added an animated loading icon to make it visible for the user to know when the generation is done. The coloring scheme was streamlined to that of Director, making it more consistent with its mostly orange, black, and white colors.

When it comes to the functionality of the program, we added two checkboxes, 'Reasonings' and 'Preset' which the user could choose to include or not. From the feedback, it was evident that having a long text of reasonings was most often not needed, unless the user was curious or wanted to know why DirectorAI had produced something undesired. Preset was included as an option to have the existing generators of camera, human, and seat, or to exclude those generators and let the AI instead choose freely among all signals, those which are less common. With preset unselected, we added enums for each set of parameters, and based on the enums, the corresponding instructions were fetched and used by the GPT model. Having preset unselected makes it more important for the user to write well-defined and accurate prompts with the risk of getting the wrong type of signals and scenario. Thus, this would be better for more experienced users who know what types of signals exist and how to distinctively describe scenarios.

As previously noted, initial development used the GPT-4o and GPT-4o mini models. Towards the project's conclusion, new GPT-4.1 models were released [107]. These newer models demonstrated comparable or even superior performance, while also offering significant cost reductions. Thus, the final version of DirectorAI utilizes these newer models instead.

6

Results

The final version of the DirectorAI prototype allows users to describe simulation scenarios using natural language, which the AI then interprets and generates within the Director environment. While the assistant remains open to further improvements, this version represents the culmination of our iterative design and prototyping process. In this section, we present an overview of the prototype's final feature set and discuss how its development and evaluation inform our research questions, with particular focus on the design patterns and limitations that emerged throughout the project.

6.1 Overview of DirectorAI

The finished prototype was developed as an integrated, chat-based assistant inside the existing Director interface. Users can describe what they want to have happen in a scenario using natural language, and the assistant automatically creates or modifies the scenario timeline accordingly by selecting signals, adjusting parameters, and timing events to match the user's intent. Its purpose is to help users create and edit simulation scenarios more efficiently through natural language interaction. While there was a major focus on reducing the cognitive load of manual signal searches and parameter configuration, the assistant also aims to streamline the entire scenario-building workflow, from signal discovery to configuring parameters, timing events, adjusting camera positions, and controlling human character behavior.

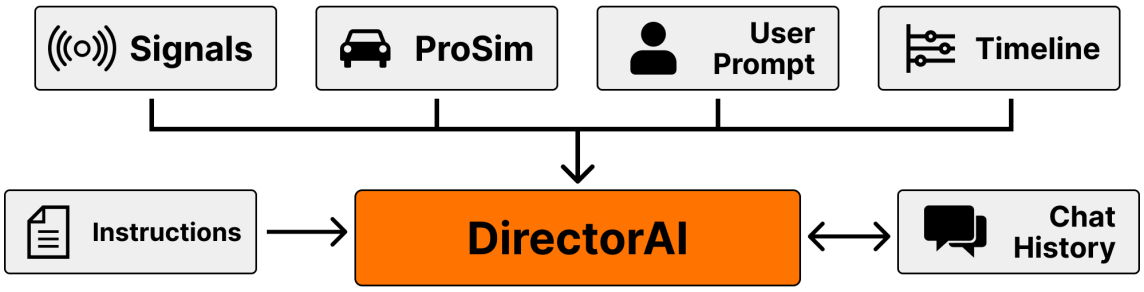

Figure 6.1: A simplified overview of the various information sources DirectorAI has access to.

Figure 6.1 presents a simplified overview of the different information sources DirectorAI can access and work with during the process of generating and editing simula-

tion scenarios. While the different generators or classifiers in the pipelines use only relevant information for their specific task, the overall system has the potential to leverage:

- The user's natural language prompt, which becomes the primary instruction all future steps base their output on. This prompt may be internally broken down or augmented by components in the pipeline (e.g., broken into sub-prompts by the sub-prompt generator).
- A repository of Director's signals that can be sent to the ProSim simulation. This set of signals includes information about each signal's topic, description, and metadata. The metadata primarily concerns the signal's parameters and includes the description, title, and value type for each parameter. This information is important for tasks like identifying appropriate signals based on the user's intent and enabling correct parameter configuration.
- Pre-defined contextual scaffolding, which includes task-specific instructions and curated domain knowledge provided to the varying specialized generators. For instance, a camera generator would have access to spatial presets (like "driver seat view") and rules for coordinate systems, while a human character generator would understand constraints for character placement.
- Dynamic data from the ProSim environment, fetched as needed to inform real-time decisions. An example is retrieving current camera coordinates for relative adjustments.
- The current state of Director's scenario timeline, which is important for contextual understanding when editing existing actions or determining appropriate timings and sequences for newly generated actions.
- The chat history, which provides conversational context, helping DirectorAI to understand follow-up requests or maintain coherence over multiple interactions between the user and AI.

This selective use of information allows DirectorAI to manage complexity and apply focused contextual knowledge to different aspects and tasks relating to scenario creation in Director.

The assistant appears as a chat window in the same space as the manual signal search ('Signal'). Both 'DirectorAI' and 'Signal' can be toggled to show or hide them. This placement was based on user feedback, aligning with the goal of replacing traditional search-based workflows with a simpler natural-language, AI-driven approach. Technically, the prototype was fully integrated into Director's existing Flutter-based codebase, allowing it to interact directly with core functionalities such as timeline control and signal data models.

The chat window includes several key features:

- **Mode Toggle:** Users can switch between Create, Edit, Suggest, and Chat modes. The Edit feature was introduced because relying on the AI to infer user intent could lead to inconsistencies, either due to lack of understanding

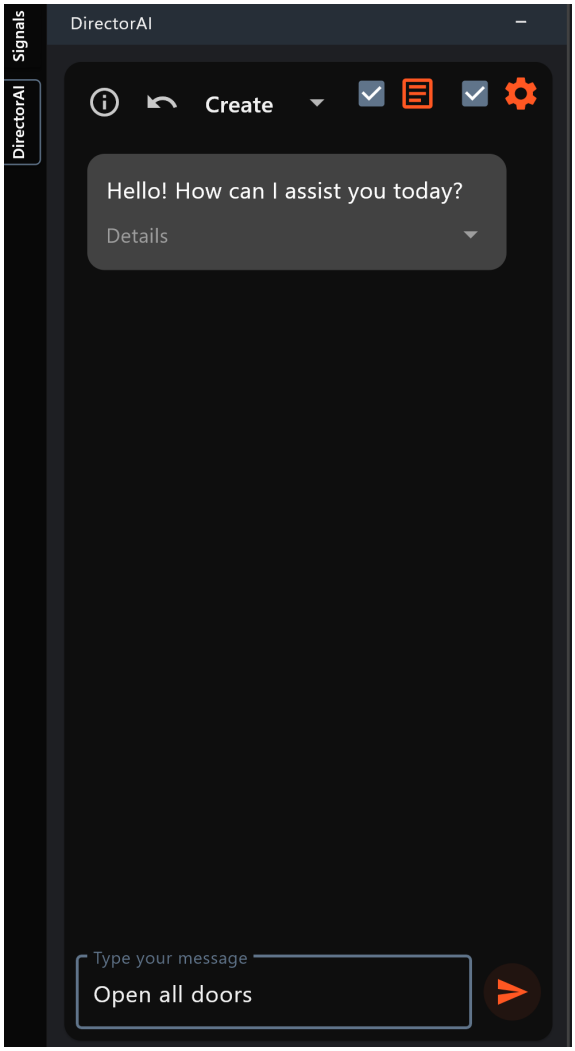

Figure 6.2: Section of the new Director interface, showcasing the DirectorAI chat.

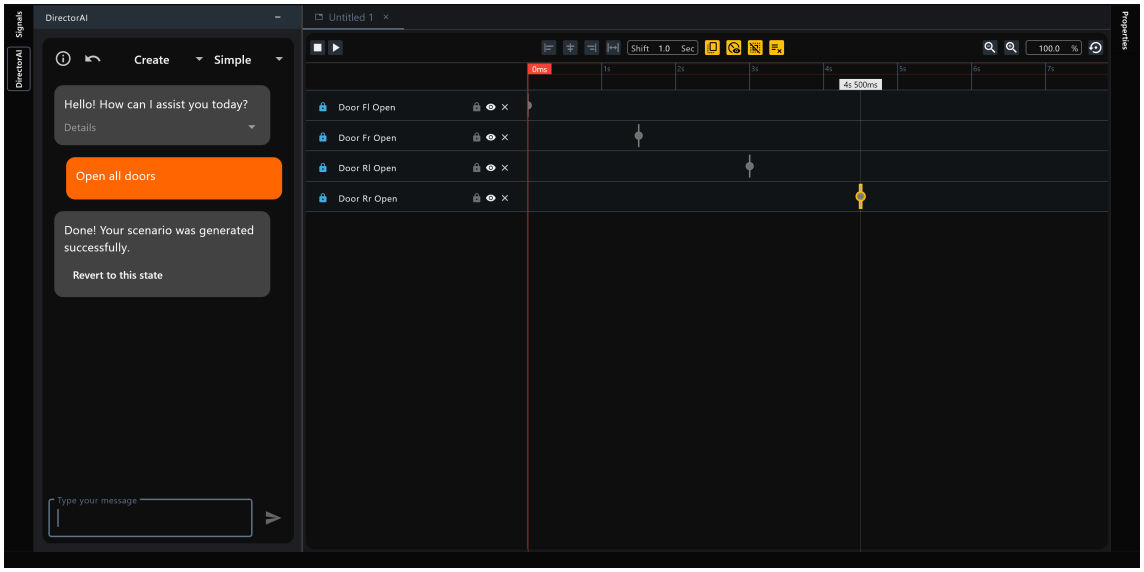

Figure 6.3: DirectorAI used to generate a scenario that opens all car doors.

from the AI or ambiguity and lack of information from the user's prompt. By manually selecting a mode, users could explicitly signal whether they intended to create new content or modify existing elements. However, as discussed further in the user evaluation section, participants often found mode-switching unintuitive and prone to errors, particularly in scenarios where the boundary between "creation" and "editing" was ambiguous. To accommodate this request, we also have a 'Chat' mode that tries to classify whether to create or edit a scenario.

The 'Suggest' mode gives the user suggestions of signals to use for a scenario, where the user can choose which of the signals to add to the timeline with checkboxes and an 'Add' button. This is helpful in scenarios where the user does not know the names signals, or when the user wants to have groups of signals to choose from, for instance signals related to windows, doors, and seats.

- **Revert Functionality:** Each AI-generated message includes a "Revert" button. Pressing it restores the simulation state to what it was at the time of that specific AI action, enabling users to quickly undo changes without restarting their work. Although not heavily used during evaluations, the presence of a safe revert option helped users feel more confident when experimenting with different commands and phrasing strategies. The revert button is also present at the top left, above the chat contents, in the form of a backwards arrow, where the user can toggle back and forwards between the previous and present creation or edit.
- **Info Panel:** A collapsible information panel offers a brief explanation of the AI assistant's capabilities and limitations. This was added to help manage user expectations and encourage more effective use of the tool.
- **Reasoning and Preset Selection:** Users can choose between including reasonings and preset signals, which are shown as a text field icon and a gear icon respectively, with a hoverable text of the icon names. 'Reasonings' are further explanations to how the AI generated its results, and 'Preset' is a set of preset signals that have been manually curated to use the most common signals when it comes to the camera, human movement and sitting, since these otherwise were often misinterpreted if the user was not specific enough.

  The options to include the reasonings, were added after the last user study since there were many who did not show interest in reading the reasonings, and it affected the speed as well as the cost of the generation to some extent since there were more API calls. When it comes to 'Preset', there were several occasions when users wanted to access signals which had to do with e.g. the camera, but were redirected to the signal 'moveCamera' when they wanted to access 'resetCamera'.

Visually, DirectorAI's chat interface follows a standard conversational layout similar to popular AI tools such as GitHub Copilot. The familiar format was deliberately chosen to minimize the learning curve and leverage users' existing mental models

of conversational AI interactions. The integration was also mindful of Director's already crowded interface: by replacing the manual signal search area rather than adding additional windows, the design maintains a streamlined, focused workspace without introducing new clutter. Example screenshots of the final prototype are provided in Figure 6.2 and Figure 6.3.

6.2 Findings on Supporting Research Question 1

> *What limitations emerge when using general-purpose language models as assistants in complex domain-specific environments without domain-specific training data?*

Our findings reveal two primary categories of limitations when deploying a general-purpose language model in complex domain-specific environments such as Director and ProSim. First, the model struggles in scenarios where domain-specific context such as spatial relationships, parameter constraints, or documentation is missing or insufficient. Second, even when signals are correctly identified, the model risks interpreting user input too literally or fails to infer intended meanings, resulting in technically valid but semantically misaligned outputs. The following sections elaborate on these challenges and the strategies we used to mitigate some of them.

6.2.1 Limitations of LLMs in the Absence of Domain-Specific Context

A main limitation identified during the prototyping and user testing phases was the LLM's dependency on clearly defined and semantically rich signal documentation. When interacting with well-labeled and intuitively structured signals, such as a boolean parameter for the horn, accompanied by a description, such as controls whether the horn is on or off, DirectorAI performed reliably. The clarity and simplicity of the signal allowed the model to infer appropriate behavior without extensive domain-specific guidance.

However, performance deteriorated when the assistant lacked spatial or contextual information. For instance, in attempting to simulate an NFC card being placed near the car's door handle, the model failed to generate functional output. This failure occurred due to insufficient signal documentation, a lack of spatial awareness, and missing information regarding the 3D position of the door handle. Director and ProSim do not expose this contextual data in a format the AI can easily interpret, making such tasks infeasible without additional customization to the AI system.

To address these shortcomings, we developed a modular system of custom prompt generators. These generators provide domain-specific contextual scaffolding to the AI in the form of system prompts, supplying the model with additional context and information for how to best configure a specific signal. For example, camera positioning tasks were particularly error-prone until we embedded predefined spatial presets (e.g., driver seat, behind car, next to door) directly into the prompt. This method was similarly applied to seating arrangements and human character move-

ment, enabling the model to interact with spatial elements it would not otherwise understand.

While effective, this approach introduces a new form of overhead: crafting, refining, and testing each generator requires iterative design and integration effort. Each new behavior or signal that lacks self-contained documentation may necessitate a custom generator. In this sense, the method is not easily scalable in terms of implementation effort. However, it is scalable in terms of user benefit, since once a generator is constructed, it can be reused across user interactions, enabling non-expert users to access functionality that would otherwise require in-depth system knowledge. This positions the AI not as a plug-and-play solution, but as a build-once, serve-many tool that relies on strong infrastructural support to compensate for its general-purpose limitations.

Later, to generalize AI instructions so that they applied beyond a few selected signals, every set of parameters was given an added instruction enum. This enum bypassed the 'Preset' selection of different generator modes. If the enum contained an instruction string, the GPT-model was provided with the corresponding instruction. This meant that not only was, for example, the camera mode given an instruction, but every signal containing the same set of parameters as the camera signal also received the same instruction. Since the level of detail for description varied for each signal and the same parameter name could have different functionality, this was done for only a few signals that were commonly used and had a unique set of parameters. Even this generalization was not enough, at least not within the time and scope of this project. This was because there were approximately 900 signals and the signal parameters were not consistent across the signal list, and information on how some signals worked was lacking. In practice, this provided the same principles and functionality as the previously mentioned contextual scaffolding from generators, but this method allowed for greater flexibility concerning the signals and parameters to which it could be applied.

This highlights the fact that the LLM's effectiveness and accuracy are limited by how extensive their provided context is. When domain-specific, spatial, or behavioral context is lacking, the assistant's shortcomings become increasingly apparent.

This resembles the challenges faced by human users where poor or inconsistent signal documentation and meta-data act as a barrier to usability, whether the user is human or AI. However, the AI offers a route to overcome this since it can be provided instructions once, and then apply that knowledge across interactions. In contrast, a human user must individually learn each signal and its parameters.

6.2.2 Literal Interpretations and Semantic Misalignment

Another observed limitation involved the model's tendency toward literal or overly narrow interpretations of user instructions, particularly when requests lacked detail or assumed contextual knowledge. During user testing, a participant requested that the camera be moved in front of the driver's door. In one instance, the AI placed the camera correctly in front of the door, but too close to the vehicle. This resulted in

the camera clipping through the door when opened in the following step. This was a technically correct solution that failed to account for practical usability or spatial reasoning.

In another session, DirectorAI placed the camera in the correct position spatially but oriented it away from the car, highlighting a misunderstanding of user intent. These issues could likely have emerged from the assistant's literal interpretation of language, lack of deeper spatial understanding, or inability to understand cause and effect in certain cases.

More broadly, DirectorAI often defaulted to the most plausible guess or estimations when essential parameters such as timing, spatial relationships, or object references were omitted. In some cases, this produced results that were structurally correct, but did not align with what the user expected or intended.

These findings highlight a limitation of general-purpose LLMs when used in simulated 3D systems; they often struggle with situated reasoning, the ability to interpret instructions that depend on spatial relationships, timing, or implicit user expectations. When context is incomplete or ambiguous, the model may still produce technically valid output, but fails to reflect the users actual intent or the scenarios practical constraints.

6.3 Findings on Supporting Research Question 2

> *What constitutes "effective assistance" in the context of AI-supported domain-specific workflows?*

Our findings indicate that what made the DirectorAI helpful wasnt just technical accuracy, but how well it fit into users workflows and supported their progress. Participants often felt it was most useful when it helped them skip tedious setup steps, especially when translating their broad, high-level ideas into Director's specific implementation, all while the user remained in control.

A recurring theme was the value of momentum: DirectorAI was most helpful when it accelerated users past implementation bottlenecks and kept them in a creative or problem-solving flow. Participants, including experienced users, appreciated that they did not have to recall or look up exact signal names, which are numerous and inconsistently labeled. Even for experts, remembering the correct spelling or structure of a signal could be a point of friction and the AI's ability to infer intent from natural language allowed them to bypass that cognitive overhead.

Many users described DirectorAI as a tool for generating a first draft, a rough approximation of an intended scenario that could then be refined. In this sense, effectiveness is less about perfect automation and more about productive delegation where the assistant takes the first pass, and the user then tailors the result. Allowing for reversibility meant participants were more willing to try AI-generated changes when they knew they could easily undo them. This created a low-risk environment for experimentation.

The interaction framework also played a central role. Participants valued the ability to shift fluidly between conversational input and direct manipulation. This flexibility allowed users to delegate some actions to the assistant while reserving others for manual adjustment, reinforcing the sense of a collaborative, rather than automated, workflow.

Importantly, these strengths stand in contrast to other possible solutions, such as standardizing all signal names or rewriting documentation. While improving documentation might reduce friction for human users, it presents significant scalability challenges since the simulation environment contains nearly 900 signals, and changing their names or structures would risk disrupting existing ProSim and FSM workflows. Renaming the entire set of signals might benefit one stakeholder, but disrupt another whose workflow is based on current patterns and naming conventions. In contrast, teaching the AI to adapt to the current system, rather than redesigning the system itself, allowed for a more immediately viable and less disruptive path to effective assistance.

Taken together, these findings suggest that effective AI assistance in this domain is characterized by:

- Reducing cognitive load while also fostering user agency.
- Accelerating tasks without enforcing rigid structures or workflows.
- Integrating into existing workflows without requiring them to change.

6.4 Findings on the Main Research Question

> *What design patterns enable general-purpose language models to function as effective assistants in complex, domain-specific software environments without domain-specific training?*

Our findings suggest that the performance of general-purpose LLMs in domain-specific software systems relies more on how their interactions are designed than on changes to the models themselves. Rather than relying on model fine-tuning, carefully structuring the input and context can significantly improve their effectiveness in specialized tasks. Through iterative prototyping and user evaluation, several design patterns emerged that enabled the LLM to function as a useful assistant without the need to train or fine-tune a custom model. These findings emphasize the importance of interaction design in shaping the utility and usability of AI-assisted systems.

6.4.1 Supporting Varied Prompting Strategies

Participants demonstrated diverse mental models and workflows when interacting with DirectorAI, especially when it came to how they structured their prompts and the level of detail they provided. This behavior was observed in our user studies as some participants entered the entire scenario into DirectorAI's chat at once, while others proceeded step by step. We also noted that some participants switched

between these approaches within the same session. One participant noted, "I like how it can do [multiple steps] at once, but I might do it one at a time." This suggests that designing for both strategies and allowing users to switch between them fluidly is critical.

Our sub-prompt generator accompanied these different mental models well by allowing the assistant to handle both high-level descriptions and low-level, step-by-step refinement. This adaptability appeared to support a range of user strategies, with participants fluidly switching between approaches. Observed usage patterns suggested that this flexibility contributed to a sense of responsiveness and alignment with users' natural workflows, rather than imposing a rigid or restrictive method of interaction.

6.4.2 Support for Non-Destructive, Editable Workflows

The ability to experiment without risk through the revert function increased participants' willingness to engage and iterate with the assistant. Additionally, because DirectorAI's output was integrated directly into the existing timeline and remained fully editable, participants were less concerned with whether the results were perfect. Participants noted that they were more open to trying AI-generated suggestions precisely because they knew they could revise or discard them without negative consequences.

During our evaluation of DirectorAI, we observed how participants tried multiple ways of phrasing the instructions to move the camera, then testing the results, and reverting if they were unhappy with the outcome. One participant noted, "I like the revert, it means I can try and experiment," highlighting how the non-destructive workflow encouraged creative exploration. Another participant mentioned that they appreciated how they could still manually edit the generated camera signal, showing the importance of AI-generated content remaining editable.

This highlights an important interaction principle: AI assistants, like DirectorAI, should be implemented in a manner that retains user control and enables experimentation. In our prototype, the presence of easily reversible actions encouraged users to explore unfamiliar features, iterate more freely, and ultimately arrive at more satisfying outcomes. Rather than needing to trust the AI completely, users could treat it as a first pass or exploratory tool.

6.4.3 Assistants That Execute and Educate

DirectorAI was most effective when it reduced interface complexity and friction, particularly by allowing users to ask for actions that would otherwise require navigating through menus or memorizing signal naming conventions and configurations. However, participants also expressed a desire to learn from the assistant by asking, for example, "How would I do this manually?" or "Where is that setting in the UI?"

During DirectorAI's evaluation, we observed a case where the assistant made a mistake when configuring the signal controlling the car horn. The participant noted

that they knew how to fix it manually but worried that, had they not known, they might have gotten stuck. They went on to explain how this would be a great opportunity to ask the assistant how to perform such fixes manually.

Our prototype did not support this explanatory functionality. This reflects an opportunity where effective AI assistants should not only execute user commands but also provide instructional guidance upon request. Participants described an ideal assistant as both a shortcut and a tutor that could be capable of teaching underlying mechanisms in addition to automating them.

Designing for this dual role could enhance DirectorAI's value, particularly for novice onboarding. This is because while more experienced users may be more capable of auditing the assistant's output and refining minor errors, newer users may not know how to do this. Novices, unfamiliar with the output structure, parameters configuration, and how the timeline works, may not even know where to look for mistakes. Thus, if a user could, for example, ask "How do I change the animation time of the camera," the assistant could guide them to that section of the UI, and the user could double-check that the AI generated such timings correctly, where the user could refine if not.

6.4.4 Feedback and System Visibility

Given the generative nature of LLMs, participants often expressed uncertainty about what to expect from the assistant. In this context, visible feedback, including system status indicators and concise explanations, played an essential role in maintaining trust and clarity.

Some participants ignored lengthy AI responses and preferred to test outputs directly, but still valued feedback when things did not go as expected. One participant said that the provided AI motivations were "too long," and that they would only read them if they could not figure out errors through testing. Subtle cues such as loading indicators or error explanations helped reduce user frustration and uncertainty. These were implemented after one participant commented that they didn't know whether the assistant was "thinking or stuck." They mentioned preferring a clear response, such as "sorry I can't do that," rather than no feedback at all when the assistant encountered an issue. These findings suggest that interpretable and non-intrusive feedback is essential for aligning user expectations and preventing misinterpretation of both AI output and situations where no output is returned.

Importantly, feedback needs to be both present and optional, with feedback being concise by default, but expandable when users seek deeper understanding. This is since the extensiveness of the feedback must strike a balance, as some users stated it was too much and they were not interested in reading it because of its length. Other users mentioned it could be useful if the AI kept making mistakes, and they could investigate why, where such a case would require a more detailed explanation of the AI's reasoning.

6.4.5 First Drafts as a Value Proposition

A recurring theme, also reinforced in our findings related to our second supporting research question, was that participants did not expect the AI to produce perfect results. Instead, they appreciated its ability to generate useful starting points: draft camera setups, initial signal configurations, or high-level scenario templates.

Across several testing sessions and user studies, participants mentioned how they appreciated that DirectorAI at least gave them something to work with. "It helped me with finding signals and placing them. I don't mind fine-tuning them myself," and "It can create a rough draft, then I can adjust [the output] myself," were representative sentiments.

This workflow, where DirectorAI contributes the heavy lifting or boilerplate, and the user provides the finer adjustments, was repeatedly described as time-saving and motivating. Users reported feeling more efficient, particularly in repetitive or complex scenario creation. This exemplifies the value of treating DirectorAI not as an autonomous decision-maker, but as a collaborative partner, an assistant that helps users quickly generate a functional template for scenarios in Director, while still leaving room for adjustments and manual control over details like timing, spatial layout, and certain signal parameters.

6.4.6 Providing Domain Context Through System Prompts

Even without any fine-tuning or custom training, DirectorAI showed that general-purpose LLMs can perform effectively when provided with well-designed contextual scaffolding. This effectiveness was achieved through task-specific system prompts that included examples, constraints, instructions, and signal schemas to guide the AI's behavior across the various tasks it was designed for.

In DirectorAI, such contextual scaffolding was implemented through our system of generators and pipelines. Each generator included task-specific instructions, constraints, examples, reference points, and other considerations important for performing a given task in Director. The pipeline structure ensured that only relevant and necessary information was provided to the LLM at the appropriate time. Testing and evaluation sessions with participants helped us construct and refine these generators by uncovering edge cases and user expectations. We observed that as generators were refined and expanded, users increasingly remarked on the assistant's accuracy for those tasks. However, when users attempted tasks that lacked dedicated generators, results were more mixed. The general-purpose generator could often still configure signals correctly, but struggled more when tasks required deeper spatial understanding of the 3D simulation environment.

These findings suggest that supporting domain-specific output from a general-purpose language model does not always require fine-tuning or custom training. In our case, carefully crafted system prompts with contextual information were often enough to guide the AI toward usable results within Director. While this approach has limitations, it demonstrates that structured prompt design can meaningfully extend the applicability of LLMs in specialized software like Director.

6.4.7 Summary

Taken together, these findings suggest that enabling LLMs to operate effectively in domain-specific software environments is less a challenge of model development and more a challenge of system and interaction design. The most impactful choices were not about improving the model itself, but about designing the conversation and supporting structures around it by focusing on prompt architecture, interface affordances, and the user's capacity to guide, interpret, and adapt DirectorAI's contributions.

Rather than right out replacing user expertise, the most successful patterns framed DirectorAI as a collaborator. The assistant could be queried conversationally, overridden easily, and potentially learned from. By creating DirectorAI with transparency, user control, and safety nets in mind, we enabled participants to meaningfully integrate it into their workflows regardless of their prior domain knowledge.

7

Discussion

In this chapter, we discuss the findings from DirectorAI's development and evaluation. We offer an interpretation of the results, highlighting the role of interaction design in adapting general-purpose language models for complex, domain-specific software. The findings are contextualized through related literature, followed by reflections on the process, implications, limitations, future work, and ethical considerations.

7.1 Interpreting Results

An important insight from the development and evaluation of DirectorAI is that interaction design and contextual scaffolding have the potential to outweigh the importance of model tuning when deploying general-purpose LLMs in complex, domain-specific environments. This insight moves away from the assumption that better performance comes primarily from refining the model itself. Instead, our findings suggest that development efforts may yield more value by focusing on the interaction ecosystem around the model, including how users are guided, how ambiguity is managed, and how explanations or feedback help users understand system behavior.

The DirectorAI prototype successfully lowered barriers to entry for new or less confident users. Participants emphasized that they no longer needed to recall complex signal names or navigate dense menus, "I don't need to remember all the signal names now," as one participant put it. By offloading this memorization and enabling conversational exploration, the assistant made Director feel more approachable. However, this success also brought to light new layers of complexity in human-AI interaction.

One example is the revert feature. While not universally used across sessions, when it was used, it proved to be an enabler of exploratory behavior. Rather than committing to a single course of action, users would test out different AI-generated options and then backtrack if the result wasn't satisfactory. The mere presence of a revert feature appeared to foster confidence as it created a low-risk space for trying unfamiliar things. This underscores a subtle but important shift in how we think about usability: not as efficiency or first-time success, but as support for safe, iterative experimentation. This also aligns with participants' framing of the AI as producing first drafts rather than definitive outputs, allowing for scenarios that could be reviewed, modified, or rejected.

This observation aligns well with one of Nielsens usability heuristics (H3) [68], which emphasizes user control and freedom. Specifically, this heuristic emphasizes the importance of allowing users to undo actions. DirectorAI shows how important such concepts become when designing AI assistants, where output is less predictable and iterative refinement is essential.

When it comes to SRQ1, which concerns the limitations of using an LLM, DirectorAI often interprets prompts literally and misses important contextual nuance. For instance, when one user requested, "open the tailgate," the AI performed the task using a particular method without checking which of the multiple valid approaches the user intended (e.g., triggering the tailgate signal directly versus simulating a button press). The user later noted, "Oh yeah, I didn't tell the AI how to open the tailgate," reflecting a gap between user intention and system execution.

This moment of misalignment highlights a broader issue as DirectorAI was not designed to ask for clarification from the user. A clarification prompt, such as "Do you want to open it directly, or simulate a button press?" could have helped resolve the issue and reinforced user trust, even though it raises the complexity of DirectorAI. The absence of such prompts meant that the assistant sometimes made assumptions, which could lead to confusion or misinterpretation. In this domain, where different actions can have subtly different effects, these small mismatches can have large usability consequences.

Regarding SRQ2, which explores DirectorAI's role in providing effective assistance, we found that while the assistant eased certain cognitive burdens (such as memorization and syntax), it did less to actively enhance user awareness of how the system works or what options were available. In complex systems like Director, effective assistance goes beyond executing a command; it requires managing uncertainty and helping users navigate a space of possibilities.

Turning to the Main Research Question, this project identified several design patterns that contributed to the usability and perceived helpfulness of DirectorAI, namely flexible input formats, non-destructive workflows, transparency, contextual scaffolding, and the point of AI outputs being editable drafts. Each of these patterns supported a more fluid and forgiving interaction style. However, their success was in part limited by the assistant's shortcomings. When the AI was unable to clarify ambiguous user intent or lacked understanding of domain-specific nuances, these interaction patterns weren't always enough to prevent breakdowns in usability.

More importantly, many of the successful patterns emerged directly from user involvement. Users played a central role not just in evaluating the assistant, but in shaping its design. Through iterative testing, we discovered that small prompt adjustments, such as including relevant parameters in system messages, or providing examples of signal use could dramatically improve outcomes. Users also surfaced domain-specific edge cases that would have been difficult to predict in advance. For example, users highlighted when multiple valid methods existed for achieving the same outcome (e.g., simulating a button press vs. directly triggering a signal). They also uncovered cases like ambiguous parameters, such as supplying an integer where a floating point value was expected for RGB parameters. More broadly, user pref-

erences varied widely as some prioritized functionality over aesthetics, while others emphasized visual polish for presentations. This diversity underscored the importance of designing an assistant flexible enough to accommodate different workflows and expectations.

In this way, users did not solely validate DirectorAI; they helped design the scaffolding around it. Given the open-ended nature of generative models, there are countless ways to frame, prompt, or interpret a response. Without user feedback, it is nearly impossible to anticipate which interaction paths are intuitive and which are misleading. By observing real usage and iterating based on user feedback, we were able to align DirectorAI's functionality more closely with practical workflows.

Ultimately, this highlights how essential a UCD approach was in this project. DirectorAI's successful features emerged not from assumptions about what users wanted, but from direct observation and iteration based on real workflows, feedback, and user needs. For future design and development of assistants in complex domain-specific software, user input is not just valuable for refinement; instead, it is foundational for designing systems that feel intuitive, trustworthy, and genuinely useful.

Finally, the promising results and positive user feedback for DirectorAI have not only validated the design approach but have also led to discussions within Volvo Cars regarding its potential for integration into their production workflows, demonstrating the real-world applicability of this research.

7.2 Relating to literature

Our findings relate to the guidelines by Amershi et al. [18] for human-AI interaction, particularly those concerning system transparency. In our study, these ideas became especially relevant when users issued underspecified commands. For example, in the tailgate case, the assistant technically fulfilled the request but left the user unsure of what exactly had been done. This illustrates that when users are operating in open-ended and unfamiliar domains like Director, it may not be enough to simply show what the AI has done after the fact. Instead, designers might also consider when and how to offer active clarification, such as surfacing possible interpretations or prompting for more detail. While Amershi et al.'s guidelines provide a strong foundation, our experience suggests that in tools like DirectorAI, additional strategies may be needed to handle ambiguous user intent and the high variability in domain knowledge.

Varma et al. [19] present a compelling case that general-purpose LLMs can function effectively in domain-specific tools. Our findings are consistent with this, as participants were able to create scenarios in Director using a conversational interface backed by a general-purpose LLM. However, we also observed how much invisible design work was necessary to enable this functionality. The assistant relied heavily on contextual scaffolding in the form of carefully engineered prompts and data schemas to function reliably. This was not a simple plug-and-play integration as much of the AI's usefulness depended on how the interaction was framed, how Director itself was explained to the AI, and how the AI handled edge cases like obscure or

complex signals. These findings are particularly relevant when working with niche and proprietary software, since the success of integrating LLMs relies less on model performance and more on the design around it.

In terms of cognitive load, our results partially align with Sweller's [66] theory of cognitive load. Many users reported a reduced need to remember signal titles or parameter configurations, or setting up the camera, and this appeared to ease the initial barrier to scripting scenarios. At the same time, abstraction introduced new types of effort. When the assistant produced an incorrect or unclear output, users often had to shift mental modes from relying on the AI to figuring out what went wrong and how to fix it. One participant noted that this shift created a kind of catch-up effect where, unlike manually authored scenarios, where every step is explicit and known, AI-generated content first had to be reverse-engineered before it could be edited. This extra layer of mental overhead sometimes made troubleshooting more effortful, even if DirectorAI had saved time earlier in the process.

Several other findings also tie into existing concerns in the literature, particularly around user understanding and trust. For instance, Rismani et al. [21] emphasize the importance of supporting users in forming accurate mental models of an AI system. In our study, breakdowns in this regard were not always immediately disruptive but still had meaningful implications. For example, how DirectorAI makes silent assumptions when multiple valid solutions exist. Users may either think the assistant knows best, or may not even know there are multiple options to begin with.

A recurring pattern was that the assistant often made assumptions about user intent, such as selecting a specific signal method without prompting, that were not always surfaced to the user. These assumptions could lead to plausible but incorrect or unintended outputs. While users could notice and correct these issues, the correction process relied on their initiative and prior knowledge, and in some cases, the underlying error was subtle enough to go unnoticed. This dynamic aligns with Khurana et al.'s [22] concerns about overtrust, not in the sense that users blindly accepted AI suggestions, but in that the assistant's confident presentation of results may have discouraged closer inspection.

These findings underscore that the risks identified by prior work ([21], [22]) are not hypothetical. Even when users are attentive, the lack of explicit clarification or feedback from the AI increases the likelihood of unnoticed errors. We attempted to mitigate this risk during development, for instance by instructing the AI to select different signal methods based on prompt phrasing (e.g., using the handle when the user says try to open the door versus directly toggling the door when the prompt is open the door). However, this strategy relied on interpreting the user's prompt to determine what they wanted, instead of directly asking them. As a result, the AI made hidden assumptions instead of showing users different options. This illustrates that anticipating ambiguity is not enough and that future iterations may benefit from surfacing alternative options, offering clarification prompts, or exposing the assistant's rationale more transparently.

Finally, our results offer a nuanced view of how users relate to generative systems in complex tools. Schoenegger et al. [23] describe AI as a "decision aid," which

aligns with how users treated DirectorAI's suggestions. Many saw the assistant as a helpful starting point where it was useful for generating initial drafts or exploring ideas. However, some participants expressed a desire for more than just output as they wanted the assistant to guide them through how tasks could be performed manually. While DirectorAI was not designed to offer this kind of support, this feedback suggests that combining automated generation with educational features could make assistants like DirectorAI even more useful. Even with this limitation, DirectorAI functioned as a decision aid by reducing friction in scenario creation and offering a low-effort way to begin creating simulation scenarios.

Taken together, these findings support many of the ideas in prior work while also highlighting the nuanced demands of applying LLMs in technical, open-ended domains. While LLMs can simplify certain tasks, they also introduce new design challenges around clarification, error recovery, education, and user control, particularly when added to tools like Director, which were not originally designed for conversational interaction. In such cases, the AI may struggle to surface needed context, clarify intent, or provide meaningful feedback unless the system is designed and adapted to support such kinds of interactions.

7.3 Project Framing and Evolution

Early in the project, there were discussions with company stakeholders around the potential of using AI to improve Director's usability. While not formally required, this implicitly nudged us toward exploring AI as part of the solution space. This introduced an inherent tension: we were following a user-centered design process (the Double Diamond), but we had effectively committed to a solution from the start.

In practice, this tension proved productive. Rather than undermining the discovery-oriented nature of the process, it helped refine our research focus, from simply solving a predefined usability problem to investigating how general-purpose AI might enable new interaction patterns in a complex technical context.

Grounding the project in a user-centered design process ensured the AI's role was not defined by technical novelty alone, but by its usefulness to real users. Importantly, this wasn't about training a model, but about designing the interaction around it. User feedback informed how prompts were structured, what affordances were surfaced, and how DirectorAI responded to edge cases, resulting in an iterative process that shaped the assistant's final behavior.

This framing also clarified the project's contribution, that LLMs can be valuable in specialized domains without retraining, provided that design supports the interaction between AI and domain-specific requirements. The ambiguity of our starting point ultimately aligned with the nature of design research itself: exploratory, iterative, and responsive to emerging insights.

7.4 Reflections on the Process

Although our design process followed a familiar Double Diamond structure with discovery, definition, development, and delivery phases, much of the work resulting from this process did not surface in the interface design. Instead, much of the effort took place in configuring interactions between users and the language model by crafting prompts, assistant functionality, and iteratively responding to user feedback.

This design work required not just technical implementation skills, but also an awareness of how phrasing affects model responses, and a deep understanding of users' expectations and struggles within the domain. The design focus moved away from interface layouts and visuals, and instead centered on prompt writing, context structuring, and logic that dictates how DirectorAI interprets and responds to users. As the project progressed, maintaining consistency and extensibility became significant challenges, especially as the number of supported tasks grew. Supporting roughly 900 signals in Director, for example, would require scalable approaches to prompt creation. While some attempts were made to generalize prompt patterns across similar signals, these often relied on high-quality documentation and meta-data in Director, something that was not always available or consistent.

When planning our user studies, we initially planned on capturing quantitative data, such as time to task completion. However, once we began conducting studies, it became clear that this approach had limitations. One challenge was defining a clear endpoint for each task. For instance, if the task was to "move the camera to the rear of the car," determining when this task was genuinely complete proved ambiguous. Technically, the task might be considered complete as soon as the camera was positioned behind the car. However, many participants chose to refine their work further, testing the camera placement and transition in addition to making small adjustments to achieve a more aesthetically pleasing or context-appropriate view. Other participants took the task even further, adding animations or experimenting with different angles, effectively transforming a simple instruction into a more complex, personalized creative process.

Additionally, technical limitations in Director itself complicated the data collection. For example, the software occasionally failed to save coordinates correctly, meaning that even when a participant had technically performed a task correctly, the results might appear incorrect through no fault of their own. This introduced noise into our data, further reducing the reliability of strict timing metrics.

Crucially, this more open-ended observation approach also revealed unexpected but valuable insights. For instance, one participant chose to move the 3D human character instead of the camera (which had the same effect, since the camera is by default attached to the human character), explaining that this was more aligned with how they would approach the task in a real-world context. This approach influenced our later design decisions, inspiring the creation of a dedicated generator for the human character. It is possible we would have missed this valuable input if we had enforced stricter task definitions.

In hindsight, this decision to prioritize rich qualitative insights over rigid quantitative

measurements enhanced our understanding of the varied strategies users employed, likely proving more informative for the overall design process.

7.5 Design and Engineering Implications

Implications for Interaction Designers: A central implication is the expanded role of designers in shaping AI behavior, not just through prompts but through the overall interaction environment. This includes designing for clarification, fallback mechanisms, and scaffolds that help users manage ambiguity. In our case, one key oversight became apparent: while DirectorAI could execute tasks, it was not able to teach users how to do them manually. We had focused on enabling productivity through automation rather than supporting learning. In hindsight, we might have explored additional AI features that helped users build a deeper understanding of Director itself. This could further help retain user agency as users could learn how to edit and double-check AI output.

Another missed opportunity was the lack of explicit clarification. When user input was vague or underspecified, the AI often guessed rather than asking follow-up questions. Designing those interactions where the AI seeks clarity rather than assuming would likely have improved both trust and output quality. Together, these reflections point to important considerations in AI-design, not all assistance is educational, and not all educational moments can or should be handled passively. Designers must consider when the AI should act, when it should explain, and when it should ask.

Engineering of Director: The project demonstrates that AI can address real usability issues, even in environments with poor documentation and inconsistent data structures. However, it also revealed the limitations of treating AI as a patch rather than addressing deeper systemic problems. Improving signal metadata, documentation standards, and tool consistency would not only reduce the burden on AI systems, but also improve usability more broadly. While DirectorAI works around many of these limitations, its latency and reliance on paid APIs, along with extensive development effort currently make it better suited for exploration than large-scale deployment. Long-term impact could be enhanced through parallel infrastructure improvements within Director.

7.6 Implications for Creative Software

The current DirectorAI prototype is highly unlikely to be directly applicable to software outside of Director and ProSim. This is because it was designed specifically around Director's signals and corresponding data structures. Additionally, DirectorAI is implemented to interface directly with Director's existing timeline control functionality, which is written in Dart. Finally, because Director is a proprietary and confidential tool used internally at Volvo Cars, it is not publicly available. DirectorAI depends on internal code and APIs from Director and ProSim to function, which in practice limits its openness and reusability outside of these systems.

That said, the purpose of this work was never to create a broadly portable assistant. While the DirectorAI prototype itself is tightly coupled to software at Volvo Cars, the insights and design patterns explored in this thesis are applicable more broadly. Many creative and technical tools, such as 3D modeling environments like Blender, or timeline-based applications in video editing, animation, or simulation, face similar challenges of complex interfaces and workflows, steep learning curves, and domain-specific terminology. In such contexts, features like contextual scaffolding could help users automate repetitive setup tasks; non-destructive workflows and revert options could encourage experimentation; and educational dialogue could support user learning within the current context and workflow. For example, in software like Blender, an assistant could help users script object behaviors or camera movements using natural language, while still allowing manual adjustments in the timeline editor.

7.7 Limitations

This section outlines limitations of the project, including constraints related to study scope, prototype maturity, and the generalizability of findings.

Study Scope and Participants

Our study involved a small sample size with eight participants in the discovery phase and five in the final evaluation, all from within Volvo Cars. While this limited the generalizability of our findings, it enabled a deep, qualitative understanding of user needs within the actual context of use. The absence of a formal control group also means that we cannot reliably quantify efficiency gains. Additionally, our participant group, while diverse in expertise, may not represent the full spectrum of potential future users.

Prototype Maturity

DirectorAI had several critical limitations that impacted both performance and user experience. Most notably, it lacked clarification mechanisms: when users issued vague commands, the AI often guessed instead of asking for clarification. This omission was a conceptual oversight only fully recognized after the evaluation. DirectorAI also required manual mode switching to compensate for unreliable intent detection, and could not handle mixed intents. These issues reflect deeper challenges in AI interaction design that we did not fully resolve. Furthermore, the abstraction layer that was intended to simplify outputs sometimes obscured important details, especially during troubleshooting. Finally, the AI pipeline's latency, cost, and reliance on manual prompt tuning raise scalability concerns for long-term deployment.

Scope of Design Improvements

This project focused on adding new features to Director, rather than redesigning existing functionality. While this allowed us to work within real-world constraints, it also meant we could only address a subset of usability issues. Our improvements

were constrained by what data and features Director exposed, and by Volvo Cars' policies around proprietary software and generative AI.

Long-term Adoption Uncertainty

The observed enthusiasm during testing may have been influenced by novelty effects. It remains uncertain whether experienced users would continue to rely on DirectorAI over time, or whether novice users might outgrow the assistant as their expertise with the software increases. We were unable to assess long-term engagement or retention within the scope of this study.

Generalizability of Findings

Given the formative approach of this thesis, the insights produced throughout the work informed how we progressed with DirectorAI's development, rather than aiming for statistical generalizability. Hence, DirectorAI's success was heavily shaped by the specific pain points of Volvo Cars' Director tool, such as inconsistent signal naming and lacking documentation, in addition to the qualitative data collected from Volvo Cars employees.

We believe that many of the patterns identified as a result of this work are likely applicable in other niche, proprietary, domain-specific software like Director. However, since DirectorAI was closely integrated with Director, the actual implementation such as code, specific generators, and how it interfaces with the existing Director functions is highly unlikely to be transferrable to other software without major adaptations.

7.8 Future Work

This project suggests several paths for future research and development in AI-assisted design tools, particularly in the context of complex technical software.

LLM Training with DirectorAI

With the addition of DirectorAI and as the usage increases, it would be possible to collect prompt data and timeline scenarios to be used for training DirectorAI. For instance, DirectorAI would have data on which signals are the most common and the timings of signal actions in scenarios, and perhaps most importantly how a scenario should be mapped with user prompts in a more general sense.

DirectorAI could also get feedback in the form of what edits were made after DirectorAI's scenario generations, thus anticipating future generations with that knowledge. Thereby, DirectorAI would improve its accuracy when it comes to timings, choice of signals, and action parameters. There could also be thumbs up/down buttons to know whether a user is content with a scenario, which is a common function in other generative AI chat systems, such as ChatGPT and GitHub Copilot.

Improving Performance with More Context

A limiting factor of the current DirectorAI is the lack of context from ProSim and the lack of documentation in Director. At the moment, the only information that the DirectorAI gets from ProSim is the human character positional coordinates. The rest of the context for DirectorAI is manually written instructions for the most common signals that lack context, such as the camera signal 'moveCamera', as well as basic descriptions of most of the signals.

By having more data from ProSim throughout a simulation session, the problem of manually needing to specify instructions for the AI when it comes to spatial context, including car model dimensions, interactable objects, and environment objects would disappear. Another issue is that DirectorAI does not know the state of the car, such as if a door is open, a light is on, the speed of the car, if someone is sitting etc., or other simulation-related information including time of day, weather, and pedestrians.

Designing Clarification Interactions

A key shortcoming of our prototype was its inability to ask for clarification when faced with ambiguous input. Future work should explore interaction strategies that allow AI assistants to disambiguate intent through targeted follow-up questions (e.g., "Did you mean A or B?"). These mechanisms are especially important in technical domains where minor misunderstandings can have significant consequences. An alternative to this could be to add suggestions to similar or closely related signals after some signals are generated, so that the user has the option to faster adjust the content in the timeline with better fitting signals or to simply add onto the existing scenario.

AI as Educator, Not Just Executor

A recurring challenge was DirectorAI's inability to help users learn Director. Future prototypes could support why or how questions, enabling users to better understand both the domain and the assistant's reasoning. Features like visual previews, traceable command generation, or step-by-step breakdowns could support this educational role.

Improved Intent Recognition

DirectorAI could not reliably detect mixed or layered intents, forcing users into either a create or edit workflow. Future assistants should aim for more flexible dialogue handling, allowing users to switch or blend intents more naturally.

Longitudinal and Comparative Evaluation

This study focused on short-term interactions and on testing aimed at understanding how users initially explore and make sense of the system. Future work should include longitudinal studies to evaluate how user reliance, learning, and tool integration

evolve over time. Extensive comparative studies between the AI tool and manual workflows, or between generic LLMs and fine-tuned models, would offer insight into trade-offs between flexibility, accuracy, and scalability.

Ethical and Organizational Implications

Finally, deeper exploration is needed into the ethical dimensions of integrating AI into expert workflows such as scenario creation in Director. Concerns include declining user skill, overreliance on AI-generated output, and blurred lines of accountability when errors occur. These risks are particularly relevant in domains involving safety-critical simulations or collaborative decision-making.

7.9 Ethics

While the ethical concerns surrounding our project may not appear as pressing as those in domains such as autonomous driving, it remains important to critically assess potential risks and how we addressed them. The aim is not to exaggerate risks but to demonstrate a reflective and responsible approach to ethical awareness and mitigation, thus ensuring our work remains relevant, thoughtful, and trustworthy.

Transparency

Transparency is a common concern in AI systems, particularly around how outputs are generated [18]. In our case, although DirectorAI does not make real-world decisions with direct safety consequences, it is still essential for users to understand how their natural language inputs are translated into simulation scenarios. Without this clarity, trust and usability could be compromised, especially when the AI's interpretations are unexpected or opaque.

In an attempt to address this, we developed motivation statements which are descriptive outputs that explain how the AI interpreted a user prompt and mapped it to specific actions, signals, or parameters. This feature emerged from iterative design and user feedback, which made clear the need for interpretability, particularly in cases of ambiguity or failure. These statements aim to preserve user confidence and help them evaluate the AI's reasoning, contributing to a more transparent and trustworthy experience.

User Empowerment and Control

AI tools risk undermining users' sense of agency by automating tasks they would otherwise perform manually [108]. In DirectorAI, we aimed to improve efficiency through automation while maintaining user control, which is especially important when the AI might misinterpret an input or fall short of a user's intention.

We addressed this by designing DirectorAI as a supportive assistant rather than a decision-maker. Features that embody this idea include:

- The Edit Pipeline, allowing users to iteratively adjust any AI-generated content via natural language.
- A Revert feature, enabling users to undo specific AI actions without losing their progress.
- Retaining manual editability of AI-generated outputs within Director's timeline interface.

These mechanisms preserve user authority and support an iterative workflow, ensuring that users remain in control and can build upon, rather than be constrained by, the AI's suggestions.

Bias in AI Models

Bias in large language models is a well-established issue, often rooted in their training data [109]. DirectorAI used several variants of GPT-4, which may carry such inherent biases. Although these biases are less likely to manifest in vehicle simulation scenarios than in socially sensitive domains, they remain a concern worth considering.

More relevant to our project was the potential for design-induced bias, such as through our prompt structures, generator logic, or the contextual data we supplied. For example, if the AI consistently generates similar road and environment conditions, or if signal abstractions disproportionately emphasize certain vehicle functions, the outputs could become skewed.

To mitigate these risks, we:

- Carefully iterated on system prompts and scaffolded context throughout development.
- Tested DirectorAI across varying scenarios and edge cases to surface potential inconsistencies.
- Actively gathered user feedback during evaluations to identify misinterpretations or emergent patterns of bias.

While the impacts of such biases in this domain may be subtle, these practices helped ensure the assistant remained as fair and balanced as possible.

Implications for Vehicle Testing

DirectorAI is designed for use in simulation environments related to vehicle testing. This brings ethical attention to the potential consequences of AI-generated inaccuracies. While errors in simulation are unlikely to directly affect real-world vehicle safety, they could still influence assumptions, waste development time, or mislead users during testing workflows.

That said, minimizing such risks was not the central focus of this project. From the outset, we recognized that the AI would not produce perfect results and designed

with that limitation in mind. Over time, our focus shifted more towards treating DirectorAI's role as that of a first-draft generator rather than a definitive scenario generator. Most users did not accept AI-generated content uncritically as they reviewed, edited, and refined outputs to fit their intentions, as discussed earlier in the thesis.

To support this, we:

- Conducted focused evaluations in controlled test scenarios to assess output consistency.
- Included users throughout the design process to uncover and resolve weaknesses.
- Emphasized transparency through features like motivation statements and editable outputs, enabling users to critically assess and adjust AI-generated content.

By designing with human-in-the-loop oversight in mind, we reduced the risk of misleading outputs being accepted at face value and encouraged a collaborative approach to scenario creation. This approach was further supported by the Director's integration with ProSim, which allowed users to immediately visualize and assess the impact of AI-generated scenarios in a 3D simulation environment. This immediate feedback loop made it easier for users to catch errors, understand outcomes, and iteratively refine the AI's suggestions with greater confidence.

Ethical Conduct of Usability Studies

All usability testing adhered to established ethical guidelines ([110], [111]). Participants provided informed consent and were clearly briefed on the study's purpose, the nature and use of any recordings, their anonymity, and their right to withdraw at any time without consequence. Personally identifiable data was handled securely, anonymized, and audio recordings were made only with explicit permission.

Emerging Ethical Considerations

While the focus of this project was on usability and technical feasibility, integrating AI assistants like DirectorAI into workflows like those seen in Director introduces ethical considerations. One concern is the potential decline in user skill, where reliance on AI could reduce familiarity with underlying systems over time. Notably, some participants expressed a desire for the assistant to be capable of a more educational role, suggesting opportunities for the AI to support skill development rather than replace it.

Further concerns relate to accountability, particularly if AI-generated scenarios lead to flawed simulation outcomes, and to the long-term sustainability of DirectorAI, considering its dependence on a proprietary, closed-source, and resource-intensive LLM. While these concerns were beyond the scope of this study, they highlight the important ethical aspects that may exist in assistants like DirectorAI.

8

Conclusion

This thesis set out to explore what design patterns and considerations contribute to enabling the effective use of a general-purpose LLM within a complex, domain-specific tool like Director. We successfully designed and implemented the functional AI assistant DirectorAI, which demonstrated the ability to provide such assistance. The main goals were to lower barriers for users, reduce cognitive load, and enable faster scenario creation, ultimately improving the usability of this powerful but complex tool. Through iterative development and evaluation, we found that DirectorAI achieved these goals. This thesis has explored the following research questions:

Main Research Question:

> *What design patterns enable general-purpose language models to function as effective assistants in complex, domain-specific software environments without domain-specific training?*

Supporting Research Question 1:

> *What limitations emerge when using general-purpose language models as assistants in complex domain-specific environments without domain-specific training data?*

Supporting Research Question 2:

> *What constitutes "effective assistance" in the context of AI-supported domain-specific workflows?*

The contributions of this work consist of the following design patterns, which proved central to DirectorAI's success:

- **Contextual Scaffolding:** The contextual scaffolding of DirectorAI, referring here to our system of generators and pipelines, embeds relevant details about how various aspects of Director work. This structure allows the assistant, and its underlying LLM, to receive instructions, information, and task-specific context precisely when it becomes relevant. Not only does this significantly improve DirectorAI's accuracy, but it also reduces cognitive load for the user, as the learning and memory previously required to perform a task manually are now offloaded to the assistant's embedded modules.

- **Interaction Flexibility:** Supporting diverse user workflows is another essential consideration. Users approach DirectorAI with varying strategies, from describing entire scenarios at once to iteratively refining small details. An effective assistant needs to support these differing interaction styles, allowing users to switch seamlessly between high-level scenario descriptions and precise refinement.
- **Non-Destructive and Iterative Workflows:** A beneficial design pattern comes from the implementation of non-destructive workflows, implemented in DirectorAI with the 'Revert' feature. This approach not only fosters user trust by mitigating the impact of inevitable mistakes from the assistant, but also encourages experimentation and exploration as this creates a risk-free environment to do so. Additionally, enabling users to refine DirectorAI's generated content through further natural language prompts is useful for aligning outputs with user intent.
- **Transparency and Explainability:** Providing transparency into DirectorAI's interpretations and understanding of user prompts and system instructions through motivation statements is useful for two main reasons. First, it helps users understand how their prompts are interpreted by the assistant and what leads to the generated output. Second, for developers, it serves as a useful method for debugging and refining system-level instructions provided to the assistant's LLM.

While these design patterns demonstrated success in making DirectorAI an effective assistant, our research also showed areas where the assistant fell short, pointing towards necessary future improvements to further enhance the efficacy and helpfulness of DirectorAI:

- **An Assistant that Educates:** A missed opportunity is users' desire for DirectorAI to also function as an educator, not solely executing tasks but being capable of explaining how those tasks could be performed manually. The inclusion of such features could further empower users to learn the underlying functionality of Director.
- **Managing Ambiguity through Clarification:** A limitation of DirectorAI is its tendency to make silent assumptions when prompted with ambiguous user requests. For example, the assistant often selects a single interpretation when multiple options exist, missing an opportunity to clarify and refine its responses and output. (e.g., "There are multiple ways to open the tailgate, would you prefer method A or method B?").

The work on DirectorAI and its evaluation has demonstrated that effective assistance extends beyond task automation. Users in this study valued the assistant not only for its ability to execute tasks but for generating useful "first drafts," where the assistant handles the initial, often cumbersome setup of a scenario that typically requires in-depth knowledge of Director's signals and features. By offloading these high-friction steps to DirectorAI, users were able to focus more on the creative and iterative aspects of scenario creation. They did not mind refining the output

and instead described DirectorAI as a collaborative partner that helped maintain workflow momentum.

The primary contribution of this thesis is demonstrating how a general-purpose LLM can, with suitable design patterns, be integrated to function effectively in a domain-specific and technically complex tool like Director, without the need for domain-specific model retraining or fine-tuning. However, this comes with challenges as DirectorAI relied on a system of carefully designed generators to provide context, and the assistant occasionally struggled with ambiguous user input. Still, the assistant proved capable of lowering the threshold for interacting with Director and enabling faster, more accessible scenario creation. The success in this work lies in turning a general-purpose LLM into a practical and helpful assistant.

In a broader sense, DirectorAI highlights that integrating AI into professional tools requires more than just model performance. It also demands careful consideration of interaction design and domain-specific workflows, considerations that benefit greatly from a user-centered design perspective. What makes DirectorAI work is not only what it can do, but how we frame its responses and output, design feedback loops, and support different user strategies. This suggests that future AI assistants, particularly those in complex workflows, must be seen not just as technical components, but as interactional systems that necessitate just as much design thinking as they do engineering.

Ultimately, DirectorAI illustrates how general-purpose models can support expert users in domain-specific software when paired with thoughtful design and proper contextual scaffolding. DirectorAI does not aim to replace user expertise, but to extend and enhance it. This work offers a step toward creating assistants that fit into professional workflows in domain-specific, proprietary, and complex systems. Not through raw model performance alone, but through usable, understandable, and supportive design patterns.

Bibliography

[1] K. S. Kalyan, "A survey of gpt-3 family large language models including chatgpt and gpt-4," *arXiv preprint arXiv:2310.12321*, Oct. 2023, Accessed: 2025-04-29. DOI: 10.48550/arXiv.2310.12321. [Online]. Available: https://arxiv.org/abs/2310.12321.

[2] Unity Technologies, *Unity - real-time development platform*, Version 2024.2, accessed December 16, 2024, 2024. [Online]. Available: https://unity.com.

[3] Paessler, *IT Explained: MQTT*, Accessed: 2025-05-07. [Online]. Available: https://www.paessler.com/it-explained/mqtt.

[4] A. Khanna, B. Pandey, K. Vashishta, K. Kalia, B. Pradeepkumar, and T. Das, "A study of todays ai through chatbots and rediscovery of machine intelligence," *International Journal of u-and e-Service, Science and Technology*, vol. 8, no. 7, pp. 277–284, 2015.

[5] E. Adamopoulou and L. Moussiades, "Chatbots: History, technology, and applications," *Machine Learning with Applications*, vol. 2, p. 100 006, 2020, ISSN: 2666-8270. DOI: https://doi.org/10.1016/j.mlwa.2020.100006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827020300062.

[6] M. Adam, M. Wessel, and A. Benlian, "Ai-based chatbots in customer service and their effects on user compliance," *Electronic Markets*, vol. 31, no. 2, pp. 427–445, 2021.

[7] D. M. Akdemir and Z. A. Bulut, "Business and customer-based chatbot activities: The role of customer satisfaction in online purchase intention and intention to reuse chatbots," *Journal of Theoretical and Applied Electronic Commerce Research*, vol. 19, no. 4, pp. 2961–2979, 2024.

[8] E. Kasneci, K. SeSSler, S. Küchemann, *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102 274, 2023.

[9] J. Li, "Security implications of ai chatbots in health care," *Journal of medical Internet research*, vol. 25, e47551, 2023.

[10] H. D. Wube, S. Z. Esubalew, F. F. Weldesellasie, and T. G. Debelee, "Text-based chatbot in financial sector: A systematic literature review," *Data Sci. Financ. Econ*, vol. 2, no. 3, pp. 232–259, 2022.

[11] dhakalnirajan, *Blender Open MCP*, Accessed on June 3, 2025, 2025. [Online]. Available: https://glama.ai/mcp/servers/@dhakalnirajan/blender-open-mcp.

[12] DeepMotion, Inc., *Text to 3D Animation*, Accessed on June 4, 2025. [Online]. Available: https://www.deepmotion.com/doc/saymotion.

[13] A. Inc., Accessed on June 3, 2025. [Online]. Available: https://www.adobe.com/products/firefly.html.

[14] Runway ML, Inc., Accessed on June 4, 2025. [Online]. Available: https://runwayml.com/.

[15] Synthesia Ltd., *Turn text to video, in minutes.* Accessed on June 4, 2025. [Online]. Available: https://www.synthesia.io/.

[16] Descript Inc., *Make any video you want to make*, Accessed on June 3, 2025. [Online]. Available: https://www.descript.com/.

[17] H. Zhao, J. Sun, Z. Yang, *et al.*, "From Shots to Stories: LLM-Assisted Video Editing with Unified Language Representations," *arXiv preprint arXiv:2505.12237*, 2025, Accessed on June 4, 2025. [Online]. Available: https://arxiv.org/abs/2505.12237.

[18] S. Amershi, D. Weld, M. Vorvoreanu, *et al.*, "Guidelines for human-ai interaction," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19, Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–13, ISBN: 9781450359702. DOI: 10.1145/3290605.3300233. [Online]. Available: https://doi.org/10.1145/3290605.3300233.

[19] N. M. Kailash Varma, A. Aryan, P. Dhanush, R. Manikanta, N. Chandhu, and G. D. Arora, "Developing an ai-based library assistant: Enhancing book retrieval with natural language processing and machine learning," in *2025 2nd International Conference on Computational Intelligence, Communication Technology and Networking (CICTN)*, 2025, pp. 136–140. DOI: 10.1109/CICTN64563.2025.10932399.

[20] V. Chen, A. Zhu, S. Zhao, H. Mozannar, D. Sontag, and A. Talwalkar, "Need help? designing proactive ai assistants for programming," in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, ser. CHI '25, Association for Computing Machinery, 2025, ISBN: 9798400713941. DOI: 10.1145/3706598.3714002. [Online]. Available: https://doi.org/10.1145/3706598.3714002.

[21] S. Rismani, S. L. Blodgett, A. Olteanu, Q. V. Liao, and A. Moon, "How different mental models of ai-based writing assistants impact writers interactions with them," in *Proceedings of the Third Workshop on Intelligent and Interactive Writing Assistants*, ser. In2Writing '24, Honolulu, HI, USA: Association for Computing Machinery, 2024, pp. 34–37, ISBN: 9798400710315. DOI: 10.1145/3690712.3690722. [Online]. Available: https://doi.org/10.1145/3690712.3690722.

[22] A. Khurana, H. Subramonyam, and P. K. Chilana, "Why and when llm-based assistants can go wrong: Investigating the effectiveness of prompt-based interactions for software help-seeking," in *Proceedings of the 29th International Conference on Intelligent User Interfaces*, ser. IUI '24, Greenville, SC, USA: Association for Computing Machinery, 2024, pp. 288–303, ISBN: 9798400705083. DOI: 10.1145/3640543.3645200. [Online]. Available: https://doi.org/10.1145/3640543.3645200.

[23] P. Schoenegger, P. S. Park, E. Karger, S. Trott, and P. E. Tetlock, "Ai-augmented predictions: Llm assistants improve human forecasting accuracy," *ACM Trans. Interact. Intell. Syst.*, vol. 15, no. 1, Feb. 2025, ISSN: 2160-6455. DOI: 10.1145/3707649. [Online]. Available: https://doi.org/10.1145/3707649.

[24] J. Smith *et al.*, "Ai-driven innovations in software engineering: A review of current practices and future directions," *Applied Sciences*, vol. 15, no. 3, p. 1344, 2025. [Online]. Available: https://www.mdpi.com/2076-3417/15/3/1344.

[25] N. F. Davar, M. A. A. Dewan, and X. Zhang, "Ai chatbots in education: Challenges and opportunities," *Information*, vol. 16, no. 3, p. 235, 2025.

[26] L. Petracek, *Ai chatbots for mental health: Opportunities and limitations*, Accessed: 2025-04-29, 2024. [Online]. Available: https://www.psychologytoday.com/us/blog/the-psyche-pulse/202407/ai-chatbots-for-mental-health-opportunities-and-limitations.

[27] F. Tortella, A. Palese, A. Turolla, *et al.*, "Knowledge and use, perceptions of benefits and limitations of artificial intelligence chatbots among italian physiotherapy students: A cross-sectional national study," *BMC Medical Education*, vol. 25, no. 1, p. 572, 2025.

[28] J. Yang, Y.-L. Chen, L. Y. Por, and C. S. Ku, "A systematic literature review of information security in chatbots," *Applied Sciences*, vol. 13, no. 11, p. 6355, 2023.

[29] S. Coghlan, K. Leins, S. Sheldrick, M. Cheong, P. Gooding, and S. D'Alfonso, "To chat or bot to chat: Ethical issues with using chatbots in mental health," *Digital health*, vol. 9, p. 20 552 076 231 183 542, 2023.

[30] R. T. Williams, "The ethical implications of using generative chatbots in higher education," in *Frontiers in Education*, Frontiers Media SA, vol. 8, 2024, p. 1 331 607.

[31] P. Jiang, C. Sonne, W. Li, F. You, and S. You, "Preventing the immense increase in the life-cycle energy and carbon footprints of llm-powered intelligent chatbots," *Engineering*, vol. 40, pp. 202–210, 2024.

[32] OpenAI, *Introducing ChatGPT*, https://openai.com/blog/chatgpt, Accessed: 2025-04-28, Nov. 2022.

[33] Candle Digital, *Creating scenario-based learning simulations with ChatGPT*, https://www.candle.digital/blog/2023/05/04/creating-scenario-based-learning-simulations-with-chat-gpt, Accessed: 2025-04-28, May 2023.

[34] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1810.04805.

[36] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[37] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pp. 610–623, 2021.

[38] R. Bommasani, D. A. Hudson, E. Adeli, *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[39] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[40] D. Jurafsky and J. H. Martin, "Question answering, information retrieval, and rag," in *Speech and Language Processing (3rd ed. draft)*, Accessed: December 16, 2024, Draft hosted online by the authors, 2024, ch. 14. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/.

[41] D. Jurafsky and J. H. Martin, "Introduction," in *Speech and Language Processing (3rd ed. draft)*, Accessed: December 16, 2024, Draft hosted online by the authors, 2024, ch. 14. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/.

[42] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[43] . W. Tunstall Von Werra, *Natural language processing with transformers*, 1st. O'Reilly Media, Inc., 2022.

[44] Dart, *Dart: A client-optimized language for fast apps on any platform*, Accessed: April 30, 2025, 2025. [Online]. Available: https://dart.dev/.

[45] Dart, *Dart overview*, Accessed: April 30, 2025, 2025. [Online]. Available: https://dart.dev/overview.

[46] E. Windmill, *Flutter in action*. Simon and Schuster, 2020.

[47] Flutter, *Flutter: Build apps for any screen*, Accessed: April 30, 2025, 2025. [Online]. Available: https://flutter.dev/.

[48] Flutter, *Flutter architectural overview*, Accessed: April 30, 2025, 2025. [Online]. Available: https://docs.flutter.dev/resources/architectural-overview.

[49] Microsoft Azure, *What is azure openai service?* Accessed: April 30, 2025, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/openai/overview.

[50] E. Boyd, *General availability of azure openai service expands access to large, advanced ai models with added enterprise benefits*, Accessed: April 30, 2025, 2023. [Online]. Available: https://azure.microsoft.com/en-us/blog/general-availability-of-azure-openai-service-expands-access-to-large-advanced-ai-models-with-added-enterprise-benefits/.

[51] Microsoft Azure, *Azure openai service models*, Accessed: April 30, 2025, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models.

[52] Microsoft Azure, *Azure openai service rest api reference*, Accessed: April 30, 2025, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/openai/reference.

[53] Microsoft Azure, *Quickstart: Get started using chat completions with azure openai service*, Accessed: April 30, 2025, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/openai/chatgpt-quickstart.

[54] Microsoft Azure, *Azure openai service quotas and limits*, Accessed: April 30, 2025, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/openai/quotas-limits.

[55] Microsoft Azure, *Using your data with azure openai service*, Accessed: April 30, 2025, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/use-your-data.

[56] Unity Technologies, *Unity: Real-time development platform*, Accessed: April 30, 2025, 2025. [Online]. Available: https://unity.com/.

[57] J. Haas, "A history of the unity game engine," *Worcester Polytechnic Institute Digital Commons*, 2014. [Online]. Available: https://digitalcommons.wpi.edu/iqp-all/243.

[58] J. Gregory, *Game Engine Architecture*. CRC Press, 2018, ISBN: 9781138035454.

[59] C.-W. Yang, T.-H. Lee, C.-L. Huang, and K.-S. Hsu, "Unity 3d production and environmental perception vehicle simulation platform," in *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*, 2016, pp. 452–455. DOI: 10.1109/ICAMSE.2016.7840349.

[60] J. A. and, "Speculative design: Crafting the speculation," *Digital Creativity*, vol. 24, no. 1, pp. 11–35, 2013. DOI: 10.1080/14626268.2013.767276. eprint: https://doi.org/10.1080/14626268.2013.767276. [Online]. Available: https://doi.org/10.1080/14626268.2013.767276.

[61] D. Schuler and A. Namioka, Eds., *Participatory Design: Principles and Practices*, 1st. CRC Press, 1993. DOI: 10.1201/9780203744338. [Online]. Available: https://doi.org/10.1201/9780203744338.

[62] K. Höök, A. Ståhl, M. Jonsson, J. Mercurio, A. Karlsson, and E.-C. B. Johnson, "Cover storysomaesthetic design," *Interactions*, vol. 22, no. 4, pp. 26–33, Jun. 2015, ISSN: 1072-5520. DOI: 10.1145/2770888. [Online]. Available: https://doi.org/10.1145/2770888.

[63] D. A. Norman, *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books, 2013.

[64] D. A. Norman and S. W. Draper, Eds., *User Centered System Design: New Perspectives on Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986.

[65] J. D. Gould and C. Lewis, "Designing for usability: Key principles and what designers think," *Commun. ACM*, vol. 28, no. 3, pp. 300–311, Mar. 1985, ISSN: 0001-0782. DOI: 10.1145/3166.3170. [Online]. Available: https://doi.org/10.1145/3166.3170.

[66] J. Sweller, P. Ayres, and S. Kalyuga, *Cognitive Load Theory*. Springer Science & Business Media, 2011.

[67] A. Cooper, R. Reimann, D. Cronin, and C. Noessel, *About Face: The Essentials of Interaction Design*, 4th. Wiley Publishing, 2014, ISBN: 1118766571.

[68] J. Nielsen, *10 usability heuristics for user interface design*, Originally published 1994. Accessed: 2025-06-04, 2024-01-30. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/.

[69] A. M. Arash Mirabdolah Maryam Alaeifard, "User-centered design in hci: Enhancing usability and interaction in complex systems," *International Journal of Advanced Human Computer Interaction*, vol. 1, no. 1, pp. 16–33, Dec. 2023. [Online]. Available: https://www.ijahci.com/index.php/ijahci/article/view/16.

[70] D. Council, *What is the framework for innovation? design councils evolved double diamond*, Accessed: 2025-01-30, 2019. [Online]. Available: https://www.designcouncil.org.uk/our-resources/the-double-diamond/.

[71] T. Brown *et al.*, "Design thinking," *Harvard business review*, vol. 86, no. 6, p. 84, 2008.

[72] B. Lawson, *How Designers Think*, 4th. Oxford, UK: Architectural Press, 2005.

[73] A. Williams, "User-centered design, activity-centered design, and goal-directed design: A review of three methods for designing web applications," in *Proceedings of the 27th ACM International Conference on Design of Communication*, ser. SIGDOC '09, Bloomington, Indiana, USA: Association for Computing Machinery, 2009, pp. 1–8, ISBN: 9781605585598. DOI: 10.1145/1621995.1621997. [Online]. Available: https://doi.org/10.1145/1621995.1621997.

[74] J. Gothelf and J. Seiden, *Lean UX*, English. Sebastopol, CA: O'Reilly Media, 2021, ISBN: 9781098116309.

[75] M. Hammersley and P. Atkinson, *Ethnography: Principles in Practice*, 3rd. London, UK: Routledge, 2007.

[76] J. Blomberg, J. Giacomi, A. Mosher, and P. Swenton-Wall, "Ethnographic field methods and their relation to design," *Participatory Design: Principles and Practices*, pp. 123–155, 1993.

[77] J. Lazar, J. H. Feng, and H. Hochheiser, *Research Methods in Human-Computer Interaction*, 2nd. Cambridge, MA: Morgan Kaufmann, 2017.

[78] J. Rubin and D. Chisnell, *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*, 2nd. Indianapolis, IN: Wiley, 2008.

[79] J. Preece, Y. Rogers, and H. Sharp, *Interaction Design: Beyond Human-Computer Interaction (4th ed.)* Wiley, 2015.

[80] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006. DOI: 10.1191/1478088706qp063oa.

[81] A. Cooper, "The inmates are running the asylum," in *Software-Ergonomie '99: Design von Informationswelten*, U. Arend, E. Eberleh, and K. Pitschke, Eds. Wiesbaden: Vieweg+Teubner Verlag, 1999, pp. 17–17, ISBN: 978-3-322-99786-9. DOI: 10.1007/978-3-322-99786-9_1. [Online]. Available: https://doi.org/10.1007/978-3-322-99786-9_1.

[82] J. Pruitt and J. Grudin, "Personas: Practice and theory," in *Proceedings of the 2003 Conference on Designing for User Experiences*, ser. DUX '03, San Francisco, California: Association for Computing Machinery, 2003, pp. 1–15,

ISBN: 1581137281. DOI: 10.1145/997078.997089. [Online]. Available: https://doi.org/10.1145/997078.997089.

[83] Y.-K. Lim, E. Stolterman, and J. Tenenberg, "The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas," *ACM Transactions on Computer-Human Interaction*, vol. 15, no. 2, 7:1–7:27, 2008. DOI: 10.1145/1375761.1375762.

[84] E. B.-N. Sanders and P. J. Stappers, "Co-creation and the new landscapes of design," *CoDesign*, vol. 4, no. 1, pp. 5–18, 2008. DOI: 10.1080/15710880701875068.

[85] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 328–339, 2018.

[86] L. Ouyang, J. Wu, X. Jiang, *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[87] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models.," *ICLR*, vol. 1, no. 2, p. 3, 2022.

[88] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *arXiv preprint arXiv:2107.13586*, 2021.

[89] S. M. Xie, A. Raghunathan, P. Liang, and T. Ma, *An explanation of in-context learning as implicit bayesian inference*, 2022. arXiv: 2111.02080 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2111.02080.

[90] J. Wei, X. Wang, D. Schuurmans, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b3Paper-Conference.pdf.

[91] S. Yao, D. Yu, J. Zhao, *et al.*, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

[92] L. Reynolds and K. McDonell, "Prompt programming for large language models: Beyond the few-shot paradigm," *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–7, 2021. DOI: 10.1145/3411763.3441766.

[93] O. Khattab, A. Singhvi, P. Ma, *et al.*, "Dspy: Compiling declarative language model calls into self-improving pipelines," *arXiv preprint arXiv:2310.03714*, 2023. [Online]. Available: https://arxiv.org/abs/2310.03714.

[94] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[95] Y. Liu, K. Hashimoto, Y. Zhou, S. Yavuz, C. Xiong, and P. S. Yu, "Dense hierarchical retrieval for open-domain question answering," *arXiv preprint arXiv:2110.15439*, 2021.

[96] Z. Ji, N. Lee, R. Frieske, *et al.*, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[97] A. Asai, S. Min, Z. Zhong, and D. Chen, "Retrieval-based language models and applications," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 6: Tutorial Abstracts)*, 2023, pp. 41–46.

[98] G. Izacard, P. Lewis, M. Lomeli, *et al.*, "Few-shot learning with retrieval augmented language models," *arXiv preprint arXiv:2208.03299*, vol. 1, no. 2, p. 4, 2022.

[99] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, *The impact of ai on developer productivity: Evidence from github copilot*, 2023. arXiv: 2302.06590 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2302.06590.

[100] P. Liu, R. Luo, C. Jiang, T. Gao, and Y. Li, "Ai-assisted bug detection in open-source software," in *2024 11th International Conference on Dependable Systems and Their Applications (DSA)*, 2024, pp. 428–429. DOI: 10.1109/DSA63982.2024.00065.

[101] K. Gu, R. Shang, T. Althoff, C. Wang, and S. M. Drucker, *How do analysts understand and verify ai-assisted data analyses?* 2024. arXiv: 2309.10947 [cs.HC]. [Online]. Available: https://arxiv.org/abs/2309.10947.

[102] P. Manrique-Merchán, G. Liñán-Cembrano, and J. de La Rosa, "Aisad: A user-friedly ai-assisted matlab tool for the high-level design of modulators," in *2024 39th Conference on Design of Circuits and Integrated Systems (DCIS)*, 2024, pp. 1–6. DOI: 10.1109/DCIS62603.2024.10769147.

[103] M. P. Polak and D. Morgan, "Extracting accurate materials data from research papers with conversational language models and prompt engineering," *Nature Communications*, vol. 15, no. 1, p. 1569, Feb. 2024, ISSN: 2041-1723. DOI: 10.1038/s41467-024-45914-8. [Online]. Available: https://doi.org/10.1038/s41467-024-45914-8.

[104] F. Spillers, *Progressive disclosure*, https://www.interaction-design.org/literature/book/the-glossary-of-human-computer-interaction/progressive-disclosure, Interaction Design Foundation - IxDF, Jul. 2015.

[105] LMArena, *Leaderboard*, Accessed: 2025-04-11, 2024. [Online]. Available: https://lmarena.ai/leaderboard.

[106] Papers with Code, *Multi-task language understanding on mmlu*, Accessed: 2025-04-11, 2024. [Online]. Available: https://paperswithcode.com/sota/multi-task-language-understanding-on-mmlu.

[107] OpenAI, *Introducing GPT-4.1 in the API*, Accessed: 2025-05-07, Apr. 2025. [Online]. Available: https://openai.com/index/gpt-4-1/.

[108] M. Raees, I. Meijerink, I. Lykourentzou, V.-J. Khan, and K. Papangelis, *From explainable to interactive ai: A literature review on current trends in human-ai interaction*, 2024. arXiv: 2405.15051 [cs.HC]. [Online]. Available: https://arxiv.org/abs/2405.15051.

[109] D. Roselli, J. Matthews, and N. Talagala, "Managing bias in ai," in *Companion Proceedings of The 2019 World Wide Web Conference*, ser. WWW '19, San Francisco, USA: Association for Computing Machinery, 2019, pp. 539–

544, ISBN: 9781450366755. DOI: 10.1145/3308560.3317590. [Online]. Available: https://doi.org/10.1145/3308560.3317590.

[110] Association for Computing Machinery Council, *Acm code of ethics and professional conduct*, https://www.acm.org/code-of-ethics, Adopted by ACM Council June 22, 2018. Accessed: May 6, 2025, Jun. 2018.

[111] Sveriges riksdag, *Lag (2003:460) om etikprövning av forskning som avser människor*, https://www.riksdagen.se/sv/dokument-och-lagar/dokument/svensk-forfattningssamling/lag-2003460-om-etikprovning-av-forskning_sfs-2003-460/, SFS 2003:460. Accessed: May 6, 2025, 2003.

A

User Study Protocol 1: Exploratory Study

A.1 Participant Onboarding & Consent Process

Before the study begins, participants must be properly informed about their rights and the purpose of the study.

Consent & Ethical Considerations

Participants will be provided with the following information:

- The purpose of the study
- What participation involves
- Their right to withdraw at any time
- Assurance that their responses will be anonymized
- How their data will be used
- Whether screen and voice recording will take place, acquiring separate consent if so

Reason: Ensures compliance with research ethics and allows participants to feel comfortable sharing honest feedback.

If participants decline recording, researchers will take extra notes to capture their feedback.

A.2 User Background

To better contextualize results, we will gather relevant background information, including:

- Experience with the software
  - "Have you used Product Simulator before? How often?"
  - "Have you used Director before? How often?"

  - "Experience with 3D software, eg. games, 3d modelling etc. How often?"
- Domain expertise
  - "What is your main area of expertise?"

A.3 Study Introduction: Context & Objectives

Explain the Purpose of the Study

Participants will be informed:

> "We are evaluating how users interact with Director to identify usability challenges and improve its design. This is not a test of your abilities but an assessment of the software itself. Our goal is to identify the most difficult and tedious parts of using the software to begin designing and prototyping improvements."

A.4 Initial Training & Exploration Phase

Introduce Key Functions Without Over-Explaining

Instead of a full walkthrough, highlight essential areas:

- "Here is where you can search for signals."
- "This is the timeline where you add signals."
- "To add a new key/node, you can do this."
- "To edit a node's parameters, you do this."
- "This is how you work with the camera, getting coords, resetting position"
- Use "Scenario Move Relative Human Green" instead, same principles

Allow Participants to Explore the Interface

Users navigate the software without a formal task.

Encourage them to verbally describe their thoughts.

A.5 Introduce the Task

Participants are given a scenario to create:

- Move the camera to be looking at the tailgate
- Open the tailgate
- Close the tailgate

- Move camera to driver's door
- Open the driver's door
- Honk for 3 seconds

Researchers Clarify

Researchers provide clarification but avoid giving direct solutions.

They can ask questions, but researchers only clarify, not guide.

Example: Instead of "Click here to do that," ask, "What do you expect would happen if you clicked that?"

Participants complete tasks independently

Users work as they would in real-world conditions.

Emphasize: "You are now working on your own, just as you would in a real-world scenario."

Qualitative Data Collected:

- Observations of hesitation, confusion, backtracking
- Notes on interface struggles

A.6 Post-Test Interview

This is a semi-structured interview. Follow-up questions are allowed, and questions can be skipped if the participant's answers cover multiple questions at a time.

A.6.1 General Impressions

These questions give an overarching view of the participant's experience with the software.

Questions:

- "How did you feel about using the software overall?"
  Reason: Provides an open-ended starting point for participants to freely share their opinions and set the tone for more detailed questions.
- "Was anything particularly enjoyable or frustrating during the tasks?"
  Reason: Encourages participants to highlight specific positive or negative experiences.
- "Was there any point during the tasks when you felt confused or unsure about what to do next?"
  Reason: Pinpoints moments where the interface or workflow caused hesitation.

- "What signals were the most difficult to get working? Why?"
- "Did the software behave as you expected?"
  Reason: Helps identify mismatches between user expectations and actual functionality, which can highlight usability issues.

A.6.2 Task-Specific Feedback

Focuses on understanding how users interacted with specific tasks and any difficulties they faced.

Questions:

- "Did you feel you had all the tools and information you needed to complete the tasks?"
  Reason: Highlights gaps in functionality or clarity that may hinder task completion.
- "If you made a mistake during any task, what do you think caused it?"
  Reason: Helps uncover the root causes of errors, whether due to design flaws or user misconceptions.

A.6.3 Interface and Usability

Explores how intuitive, efficient, and user-friendly the interface is.

Questions:

- "How intuitive did you find the software's interface? Were any buttons, menus, or features difficult to locate?"
  Reason: Identifies navigation and layout issues.
- "What did you want to achieve/do but couldn't? Why?"
  Reason: Highlights unmet user needs or missing features.
- Did you feel confident in your ability to complete the tasks?
  Reason: Assesses whether the software empowers users or leaves them uncertain.

A.6.4 Efficiency and Workflow

Evaluates whether the software supports productivity and avoids unnecessary complexity.

Questions:

- "Did you feel the software allowed you to complete tasks efficiently?"
  Reason: Measures how well the software supports users in achieving their goals quickly and effectively.
- "Were there any steps or features that felt unnecessary or repetitive?"
  Reason: Identifies inefficiencies in the workflow or redundant features.

- "Did you feel there was a better way to achieve the same results?"
  Reason: Encourages participants to suggest improvements or alternative approaches.
- Anything else. . . ?

B

User Study Protocol 2: Prototype Evaluation

B.1 Participant Onboarding & Consent Process

Before the study begins, participants must be properly informed about their rights and the purpose of the study.

Consent Form & Ethical Considerations

Participants will be provided with the following information:

- The purpose of the study
- What participation involves
- Their right to withdraw at any time
- Assurance that their responses will be anonymized
- How their data will be used
- Whether screen and voice recording will take place, acquiring separate consent if so

Purpose of this specific study:

1. Did we manage to solve some of the identified problems, and exactly how did we manage to do that?
2. What did we fail at solving, and why did we fail at that?
3. What new issues did we unexpectedly introduce, how did that happen?

B.2 User Background

- Experience with the software
  - "Have you used Product Simulator before? How often?"
  - "Have you used Director before? How often?"

  - "Experience with 3D software, eg. games, 3d modelling etc. How often?"
- Domain expertise
  - "What is your main area of expertise?"
- Preconceptions of AI
  - "What are your previous experiences and opinions on AI tools?"

B.3 Introduction to DirectorAI

Briefly how it works. Don't overexplain.
Basic: modes, can create signal, parameters, timings, and edit.
Inform the participant that we are equally interested in the successes of DirectorAI, as well as its failures. Tell them to not worry about being harsh or critical about the prototype, as identifying flaws is important for its future success.

B.4 Scenario

Participants are given a scenario to create:

- Move the camera to be looking at the tailgate
- Open the tailgate
- Close the tailgate
- Move camera to driver's door
- Open the driver's door
- Honk for 3 seconds

Qualitative Data Collected:

- Observations of hesitation, confusion, backtracking
- Notes on interface struggles

B.5 Interview

This is a semi-structured interview. Follow-up questions are allowed, and questions can be skipped if the participant's answers cover multiple questions at a time.

- "How did you feel about using DirectorAI?"
  Reason: Provides an open-ended starting point for participants to freely share their opinions and set the tone for more detailed questions.
- Effectiveness: "Did the AI-generated scenario work as expected? Were any adjustments needed?"

- "Was DirectorAI an effective alternative to manually creating the scenario?"
  Reason: Effectiveness compared to before.
- "How easy was it to interact with the AI? Were there moments of confusion?"
  Reason: Usability
- "Did you feel like you had control over the AI-generated output? Did you trust the results?"
  Reason: Trust & Control
- "What did the AI struggle with? What new issues arose that weren't present before?"
  Reason: Failures & Unexpected Issues
- "What did you not like about the AI tool?"