Large Language Models for Creating Customized Virtual Environments

Riccardo Caprile, Matteo Martini, Marianna Pizzo, Fabio Solari and Manuela Chessa

University of Genoa, Italy
ORCID (Matteo Martini): https://orcid.org/0009-0006-3929-5055, ORCID (Marianna Pizzo): https://orcid.org/0009-0004-8653-4018, ORCID (Fabio Solari): https://orcid.org/0000-0002-8111-0409, ORCID (Manuela Chessa): https://orcid.org/0000-0003-3098-5894

Abstract. In many fields of application, such as exergames for cognitive or physical training based on virtual reality, one of the main challenges is creating a wide range of simulated scenarios to avoid the onset of boredom and habituation. However, creating many simulated situations can be a time-consuming and tedious task. Here, we propose a framework for generating different virtual reality scenarios described in natural language. Using a conversational agent, the framework will allow unskilled individuals to add virtual scenarios and simulations to existing software, such as cognitive exergames. We will analyze the use of generative language models and their application in creating virtual environments.

1 Introduction

Large Language Models (LLMs) trained for code generation can be used to transform a natural language request for a virtual world into reality. In recent years, Artificial Intelligence (AI) models have achieved remarkable levels of reliability when responding to inquiries. Therefore, we intend to exploit these models and provide users with a framework that can overcome the problem of creating different immersive environments for various situations with minimal effort. Our goal is to enable the construction of virtual environments (VEs) using natural language, making the framework accessible to individuals outside the realm of computer science. For instance, a physiotherapist could request a VE tuned to the needs of a patient with an unusual injury. Students in a laboratory could inspect, analyse, and become familiar with materials or machines that they might not have access to because of budget constraints, safety concerns, or physical limitations. Furthermore, serious games could be customized considering the age, preferences, and attitudes of the users. The main objective of this paper is to utilize LLMs to generate custom VEs tailored to user requests. Specifically, our framework offers the possibility of generating VEs from a predefined library of environments or choosing virtual assets and positioning them in 3D positions decided by the users. The framework could be used both by developers, who can then access and modify the generated code, or by non-skilled users, who can directly be immersed in the generated environments without additional coding. The remainder of the paper is the following: In Section 2, we revise state of the art about Large Language Models, their use as conversational agents, and prompt engineering. Then, we review the recent interest in the use of LLMs with Virtual Reality (VR) to implement social agents and as a tool to create VE. In Section 3 we provide an overview of the system we have developed, its main software components and the different usage modalities. In Section 4, we present the results of a preliminary evaluation, focused on quantifying the computational time and the errors of our system, considering the different available LLMs. Finally, in Section 5 we draw some conclusions and we discuss the further developments and applications of the proposed work.

2 State of the art

2.1 Large Language Models

A large language model is a deep learning algorithm that can perform multiple Natural Language Processing (NLP) tasks, such as: text generation, code generation and information retrieval. LLMs and Generative Artificial Intelligence are obviously linked to each other because LLMs are a type of generative AI built to generate text-based content. Basically, LLMs are designed to understand and generate text, like a human being. They base their knowledge on the vast amount of data available used for their training. They have the ability to infer from context and generate coherent and relevant responses that are comprehensible to humans. These kinds of models are based on a transformer architecture, which is particularly well-suited for handling sequential data such as text input [10].

2.2 Generative Pre-Trained Transformer (GPT)

Specifically, GPT stands for Generative Pre-Trained Transformer, which is a large language model released by OpenAI in 2020 [1] that belongs to a family of AI models. The latest GPT model is GPT-4, which belongs to the fourth generation, although various versions of GPT-3 are still widely used and available. We will use these models in the testing phase of our virtual environment experiments. Several models are available in the GPT-3 family [1] (see Table 1).

The parameters in GPT-3, like any neural network, consist of the weight and the biases of the layers. The sizes are determined by the number of layers. Therefore, versions with more layers have a higher number of parameters.

Weights are the most important parameters through which the model learns by exploiting the training data. For example, if a model sees the word “Dog” followed by the word “bark”, it will assign a

[1] https://openai.com

Model Name	Parameters	Number of layers	API Name
GPT-3 Small	125M	12	n/a
GPT-3 Medium	350M	24	ada
GPT-3 Large	760M	24	n/a
GPT-3 XL	1.3B	24	babbage
GPT-3 2.7B	2.7B	32	n/a
GPT-3 6.7B	6.7B	32	curie
GPT-3 13B	13.0B	40	n/a
GPT-3 75 or GPT-3	175.0B	96	davinci

Table 1. OpenAI GPT-3 models

higher weight to this pair of words. Next time, the model will be able to predict more likely the word "bark" after "dog". Bias is a parameter used as an adjusting factor to fine-tune the predictions of an entire layer, making them more accurate. Basically, it is employed to adjust the output values and influences how easily a node can be activated. Most of the models mentioned above are accessible through APIs.

Recently, OpenAI released GPT-4, a multimodal large language model, successor of GPT-3 and GPT-3.5. The model is trained on a large amount of multimodal data, including text and images from different sources and datasets. After training, the model is aligned with manually labeled datasets containing verifiable facts and desired behaviors.

2.3 Chat-GPT and Prompt Engineering

ChatGPT [2] is a chatbot developed and released by OpenAI in November 2022. As the name suggests, it is based on the LLM GPT, previously described. It gives the user the possibility to have a conversation of the desired length, style, and language. ChatGPT is available for everybody in two versions, one built with GPT-3.5 and the other with GPT-4.

The main objective of ChatGPT is to engage in humanlike conversations, trying to answer users' questions as precisely as possible.

Prompt engineering is an artificial intelligence engineering technique that comprehends the process of refining LLMs with specific prompts and recommended output, and, at the same time, the process of refining input to get the best output from an LLM.

2.4 Large Language Models and Virtual Reality

Among the different uses of LLMs, recently, there has been a growing interest in their use within VR. For example, some approaches exploit the use of LLM-based AI agents for human-agent interaction in VR [11]. In other works, VR and LLM architectures are used together to enhance the learning process across diverse educational contexts, ranging from school to industrial settings [5]. Generative AI and LLMs are also useful for establishing an iterative prompting mechanism that provides professional design prompts to generate precise visual schemes. In [12]. the authors propose a framework where Generative AI cooperates with Mixed Reality technology to form an interactive and immersive environment for enabling full participation in the design process. Finally, LLMs have been recently exploited to solve specific tasks in VR, such as text entry [2].

Other works address the problem of VR content generation using LLMs. In [14], the authors describe a preliminary approach to generate complex VR scenes based solely on natural language. Other works propose methods focused on generating 3D scenes, which maybe applicable to VR scenarios, but limited to generating static indoor scenes without dynamics [6, 13].

[2] https://chat.openai.com

In [9], the developers created a VR game with non-deterministic game mechanics called Codex VR Pong, powered by OpenAI's text generative models, which were integrated with the Unity game engine. This is a sort of Ping Pong game where the players can transform both the paddles and the ball into any 3D object, and these transformed objects interact in semantically sensible ways. For instance, a ball transformed into an egg colliding with a pan results in a fried egg. To implement Codex VR Pong, the developers used an integration of Codex for Unity, which is a descendant of GPT-3. Codex's training data contains both natural language and billions of lines of source code from publicly available sources.

In [4], the authors aim to create a system based on Unity that bridges the gap between static content and dynamic behavior generation in VR environments by using LLMs for the generation of run-time code, using the C# Compiler Roslyn[3]. The authors aimed to exploit LLMs because they can enhance software developers' productivity by automating various tasks, such as implementing new features or translating natural language program descriptions into an actual program. On the other hand, we focused on non-skilled users who are not developers, with the objective of eliminating boring and redundant tasks to create an environment.

The main difference of [4] with respect to what we are proposing is that they exploited large language models for the code generation of objects' behavior. They required the user to ask the AI to create a certain behavior (e.g., to make a circular movement) for a selected object already inside the Unity scene. Instead, we wanted to let the users to create their own environment and, in a "game modality", positioning models wherever they pleased.

Secondly, in [4], the authors use only primitives, such as cubes, spheres and cones. Our aim is to offer users a "real world" experience by providing them with objects that can be found in the real world, ensuring a faithful representation, and giving them a VR experience that simulates what they see every day.

3 Materials and Methods

3.1 Software components

The framework is composed of the following software modules:

- Unity 3D: it is one of the most used game engines. It has been developed by Unity Technologies and released in 2005. It is a cross-platform solution available for Windows, macOS and Linux; in addition to them, it supports the development of a large set of other platforms for which it is possible to build applications like iOS, Android and Virtual Reality platforms.
- OpenAI API: it provides programmatic access to the OpenAI models and services. Specifically, we used a package that integrates OpenAI API inside Unity, the OpenAI-DotNet [4] written in C#, the programming language used inside Unity scripting, which is installable through the Unity package manager. This package offers multiple functions and features. For example, you can list the various models available in the API, also checkout model endpoint compatibility to understand which models work with which endpoints. Then, you can retrieve information about a certain model such as: the owner and permissions. The most important feature for us which this package offers is the Chat feature. Basically, as ChatGPT, if you provide an input to the model, you receive a text output that answers the question you previously asked.

[3] https://assetstore.unity.com/packages/tools/integration/roslyn-c-runtime-compiler-142753

[4] https://github.com/RageAgainstThePixel/OpenAI-DotNet

- Roslyn C# Runtime Compiler: it is a compiler that allows runtime loading of assemblies and C# scripts. It allows us to compile C# scripts while the application is running in order to build environments whenever a script is provided by the AI.
- Meta XR SDK: it gives us the ability to create immersive applications, including advanced rendering, social and community building. There are many packages of the Meta XR SDK, but we have used two of them: Core SDK and Interaction SDK.
- Vocal Commands (Hugging Face API): it has been used to allow the users to use vocal commands to ask which environment they desire or which object they want inside their customized world.

3.2 System Design

3.2.1 Developer Mode and Pre-built VEs

This modality has been built for the user familiar with programming because it includes some technical aspects that may not be comprehensible to not-skilled users. The first window the user sees is the list of models and pre-built environments implemented inside the framework (see Fig. 1).

Figure 1. List of available pre-built virtual environments and available models.

The "Pre-Built Environments" are pre-implemented environments that simulate some real-world situations, and they are built with models available in the "Available Models" list, downloaded from the Unity Asset Store. Additionally, there is the possibility of selecting the desired number of specific models to include inside the environment as long as they belong to the same category. At the moment, the models/objects are divided into 5 different categories:

- Furniture : Bed, Chair, Table, Drawer, Desk, Shower, Sink
- Nature : Wood, Rock, Oak, Mushroom, Flower, Bush, Pine
- Cars : Cops, Sedan, Sport, Suv, Taxi
- City : Barrel, Bench, Bin, Dumpster, Hydrant, Mailbox
- Industry : Cable, Car, Garbage, Plank, Tank, Tubes

Moving on with the framework's windows, there is the "Script Request" window, where the user can ask for the desired Pre-Built Environment or for the specific models (see Fig. 2).

In the window on the left, the users can write the environment they wish by choosing it from the pre-built environments; for example, "Please build an office for me, thanks." After typing the sentence, they have to click the button "Generate Script," and the LLM will create an appropriate C# script for the generation of a customized environment. In Figure 2, there are two more buttons: "Start Recording" and "Stop Recording". If the users click "Start Recording," they can make a request just by using their own voice, thanks to a microphone installed in the VR headset, e.g., in the Meta Quest 2. The recorded sentence will be inserted inside the window "Script Request"; then, as before, the users have to click the button "Generate Script" to complete the operation and generate the environment.

Figure 2. Developer Mode user interface: on the left is the panel where to write the request; in the middle, the buttons to start and stop with vocal commands; on the right is the generated script.

In case the C# script generated contains issues, such as a wrong syntax or other compiling errors, a message will appear in the "Script Generation Details" window, and another request will be sent to the LLM until a satisfactory result is provided and executed.

3.2.2 Developer Mode and Free VEs

In this modality, the user can create a customized virtual reality environment, by deciding which models to include and their position in the VE (see Fig. 3).

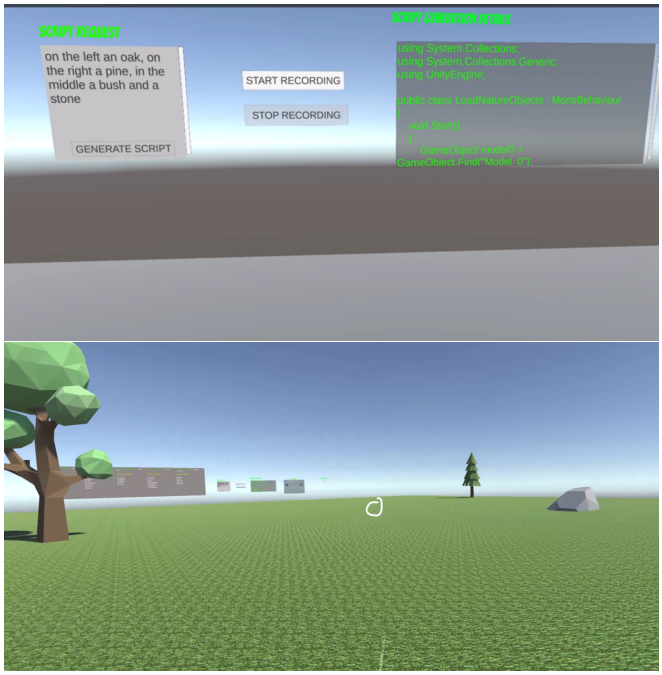

Figure 3. (Top) Example of VE generation using the "Free VE" modality. (Bottom) The generated VE: the oak is positioned in the left part of the environment, the bush in the middle (the model in the picture is circled because it is too small to be clearly visualized), the pine is on the right, and the stone is randomly positioned. The pavement's texture has been changed into grass because we have selected models belonging to the same macro category, which is nature.

In this example, we ask the system for the placements of 4 models (specifically an oak, a pine, a bush, and a stone) in the environment.

There are 4 different positions: left, right, middle/center, and random. Basically, the model will be positioned in the

left/right/middle(center) available part of the pavement of the environment; otherwise, if the user does not specify the position, the model will be placed randomly in the world. In this modality, there are two strict rules that must be followed.

1) The position of the object in the Script Request window must be written before the model's name, for example: "Please, I would like a pine on the right, an oak on the left, and a stone in the middle."

2) The user can select models belonging to the same macro category, only. An acceptable example is "position a pine, an oak, another pine and a stone". An unacceptable example is "place a pine, a mushroom, a chair, a dumpster."

In the example in Figure 3, the user wants to create an environment with an oak on the left, a pine on the right, and in the middle, a bush and a stone (no position inserted). The resulting VE is shown in Fig. 3 (bottom).

It is worth noting that the "nature" category is the one that provides better results in terms of visual appearance, also when many models are requested, considering that the real-world nature environment comprehends hundreds of trees, plants, and rocks positioned randomly (see Fig. 4 for an example of VE generated placing 15 objects).

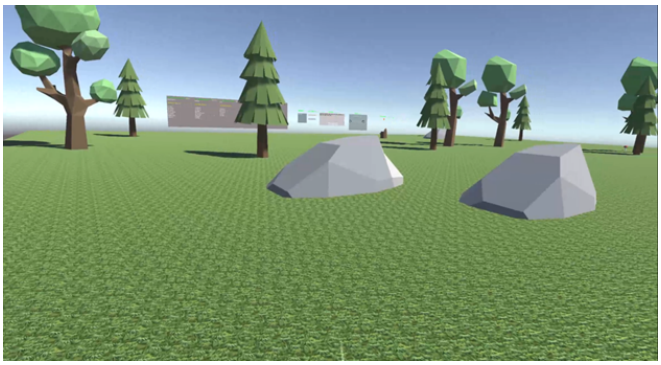

Figure 4. Example of VE generated placing 15 objects.

3.2.3 User Mode

Our framework could also be used in "User Mode". Basically, it is very similar to the "Developer Mode" one, but it gets rid of all the technical information that a non-programmer user could not understand; for instance, the window with all the C# code information is substituted with a simpler one.

Again, the user can choose among some pre-built environments that should simulate real-world rooms or habitats. We have implemented 7 different environments. Every pre-built environment is made of models that are available also on their own when the user does not want to use environments already implemented by us.

Figure 5 shows an example of usage where the user asks for the generation of an apartment with the "Pre-built VEs" modality.

Similarly, in "User Mode," one could generate environments with the "Free Ves" modality, as seen before.

4 Results

We tested our system in the "Developer Mode and Free VEs" modality considering the following:

- 3 different LLMs: GPT-3.5-turbo, GPT-3.5-turbo16k and GPT-4.

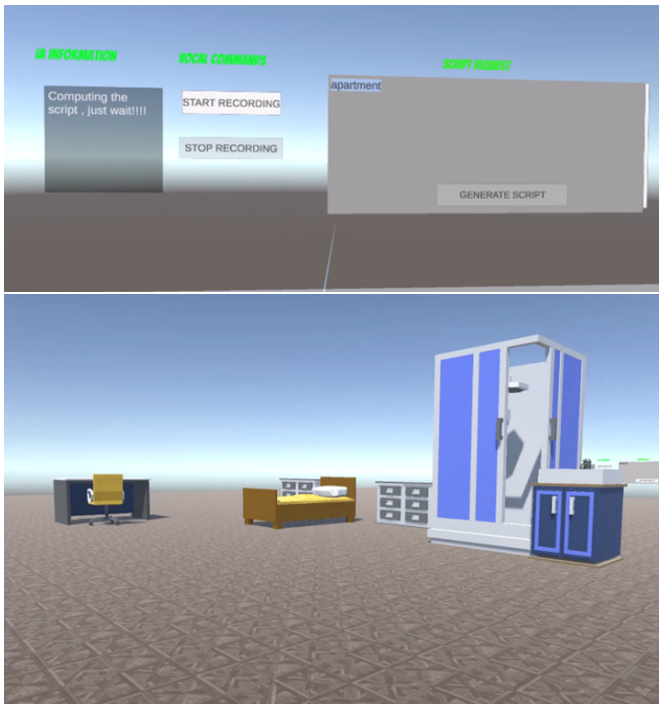

Figure 5. Usage Example of the User Mode "Pre-built VEs", to generate the VE representing an apartment.

- creation of environments composed of 5, 15 and 30 models

For each combination, we have run the system 14 times, and we have computed:

- The average number of attempts (reflecting the number of wrongly generated scripts) to create the VE;
- the average time in seconds to create the VE and the standard deviation.

Table 2 shows the obtained results. Results for GPT-3.5-turbo and GPT-3.5-turbo16k are very similar, the second one having a smaller number of errors, thus requiring a smaller number of requests to create the VE. It must be said that GPT-3.5-turbo16k is trained with a higher quantity of data, and the price for tokens is higher, too, but its behavior is almost the same compared to GPT-3.5-turbo. It is worth noting that GPT-4 shows higher accuracy in terms of required attempts (though this accuracy does not seem related to the number of objects to be paced), but with an average time to create the VE that is noticeably higher (3 times higher when 5 models are placed in the VE, and 4 times higher when 30 models are placed). This result is worth further analyzing because GPT-4 should have larger capabilities and broader knowledge, and it must be taken into account that the price is 4.5 times higher than GPT-3.5-turbo.

5 Conclusions and Discussion

In this work, we have presented a framework capable of creating custom immersive VR environments with the support of LLMs. The system was built in Unity, C#, and with the support of different APIs for the connection to the LLM and the runtime compiler. We have implemented two different working modalities for different end-users: "User Mode" and "Developer Mode". For both modalities, there are two different "game modes" called: "Pre-built VEs" and "Free VEs".

	5 models			15 models			30 models		
	#tries	avg (s)	std (s)	#tries	avg (s)	std (s)	#tries	avg (s)	std (s)
GPT-3.5-turbo	1.9	8.86	1.28	4.6	13.96	2.34	4.6	17.46	2.85
GPT-3.5-turbo16k	1.9	8.45	1.54	2.6	15.40	3.74	2.8	21.60	7.10
GPT-4	2.7	24.51	4.12	1.3	43.70	9.53	2.3	76.58	11.85

Table 2. The average number of attempts (i.e., the average number of wrongly generated scripts), the average time in seconds, and standard deviation necessary to create VE composed of 5, 15, and 30 models with the 3 considered LLMs.

With the first one, the user can choose between a list of different environments that simulate real-world rooms or environments; with the second one, the users can choose how many models they want to place inside a custom VE and the position of each 3D object. Then, we tested the framework with 3 different LLMs: GPT-3.5-turbo, GPT-3.5-turbo-16k, and GPT-4, considering the creation of VEs composed of a different number of objects (5, 15, 30). Results highlight that the systems still needs several retries before being able to create a correct script for the generation of the desired VE. Indeed, an average of around 2 retries is necessary to generate a VE with 5 models with all the LLM modes. Though our system is not designed to run in real-time and time is not a strict constraint, it is worth noting that GPT-4 needs more than 1 minute to create a VE with 30 models. This could be indeed tedious, also taking into consideration the cost per token to use the APIs.

The project can obviously be improved from many points of view, and we are going to work on them in the near future to solve the system's issues, make it more stable and less faulty, and add new features. To make the system more stable, avoiding any kind of error made by the AI, we have to generate every type of possible C# faulty script, list all the errors made by the LLM, and then add new acceptability checks and filters in the code; we must say that the number of different errors is still quite high, and so this is a task which is going to be time-consuming and expensive (because of the costs of the API).

Then, the system would benefit a lot from the usage of a 3D asset database such as Sketchfab [5], from which we could download any 3D models required by the user just for that session, without the need to download each 3D models and importing in the Unity project offline. To implement this feature, we should describe a new request text to send to the OpenAI API. At this point, the system could be used to generate a variety of new VEs.

Another improvement for the system could be allowing the users to choose which LLM they wish to use. In addition to this, we would like to insert other LLMs, not only the ones related to OpenAI but also Gemini developed by Google [6], Llama 3 developed by Meta [7], and NeMo developed by NVIDIA [8].

Finally, future work will be the use of the framework presented in this paper to create VEs blended with real-world ones, substituting real objects with virtual ones, e.g., exploiting the system described in [8]. In such a case, a possible input to the system would be "Substitute the sofa with a chaise longue, the lamp with a beach umbrella, and put a fresh drink on the table", so the living room or the office could become a personalized VE, e.g., a beach, where passive haptics could be exploited, too [3, 7].

Acknowledgements

This work has been supported by Ministero dell'Università e della Ricerca (Italian Ministry of University and Research), PNC - Piano Nazionale Complementare, FIT4MEDROB (PNC0000007) "Fit4MedRob- Fit for Medical Robotics" Project - Mission 2

[5] https://sketchfab.com
[6] https://gemini.google.com/?hl=it
[7] https://llama.meta.com/llama3/
[8] https://www.nvidia.com/it-it/ai-data-science/products/nemo/

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] L. Chen, Y. Cai, R. Wang, S. Ding, Y. Tang, P. Hansen, and L. Sun. Supporting text entry in virtual reality with large language models. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 524–534. IEEE, 2024.

[3] L. Gerini, F. Solari, and M. Chessa. A cup of coffee in mixed reality: analysis of movements' smoothness from real to virtual. In *2022 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 566–569. IEEE, 2022.

[4] D. Giunchi, N. Numan, E. Gatti, and A. Steed. Dreamcodevr: Towards democratizing behavior design in virtual reality with speech-driven programming. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 579–589. IEEE, 2024.

[5] J. Izquierdo-Domenech, J. Linares-Pellicer, and I. Ferri-Molla. Virtual reality and language models, a new frontier in learning. 2024.

[6] R. Ma, A. G. Patil, M. Fisher, M. Li, S. Pirk, B.-S. Hua, S.-K. Yeung, X. Tong, L. Guibas, and H. Zhang. Language-driven synthesis of 3d scenes from scene databases. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018.

[7] M. Martini, F. Solari, and M. Chessa. Obstacle avoidance and interaction in extended reality: An approach based on 3d object detection. In *International Conference on Image Analysis and Processing*, pages 111–122. Springer, 2023.

[8] M. Pizzo, E. Viola, F. Solari, and M. Chessa. Evaluation of 3d reconstruction techniques for the blending of real and virtual environments. In *2024 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 360–367. IEEE, 2024.

[9] J. Roberts, A. Banburski-Fahey, and J. Lanier. Steps towards prompt-based creation of virtual worlds. *arXiv preprint arXiv:2211.05875*, 2022.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[11] H. Wan, J. Zhang, A. A. Suria, B. Yao, D. Wang, Y. Coady, and M. Prpa. Building llm-based ai agents in social virtual reality. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2024.

[12] S. Xu, Y. Wei, P. Zheng, J. Zhang, and C. Yu. Llm enabled generative collaborative design in a mixed reality environment. *Journal of Manufacturing Systems*, 74:703–715, 2024.

[13] H. Yi, C.-H. P. Huang, S. Tripathi, L. Hering, J. Thies, and M. J. Black. Mime: Human-aware 3d scene generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12976, 2023.

[14] Z. Yin, Y. Wang, T. Papatheodorou, and P. Hui. Text2vrscene: Exploring the framework of automated text-driven generation system for vr experience. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 701–711. IEEE, 2024.