LLM-supported 3D Modeling Tool for Radio Radiance Field Reconstruction

Chengling Xu*, Huiwen Zhang*, Haijian Sun† and Feng Ye*
*Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI, USA
†School of Electrical and Computer Engineering University of Georgia, Athens, GA, USA
Emails: *{cxu338, hzhang2279, feng.ye}@wisc.edu, †hsun@uga.edu

***Abstract*—Accurate channel estimation is essential for massive multiple-input multiple-output (MIMO) technologies in next-generation wireless communications. Recently, the radio radiance field (RRF) has emerged as a promising approach for wireless channel modeling, offering a comprehensive spatial representation of channels based on environmental geometry. State-of-the-art RRF reconstruction methods, such as RF-3DGS, can render channel parameters, including gain, angle of arrival, angle of departure, and delay, within milliseconds. However, creating the required 3D environment typically demands precise measurements and advanced computer vision techniques, limiting accessibility. This paper introduces a locally deployable tool that simplifies 3D environment creation for RRF reconstruction. The system combines finetuned language models, generative 3D modeling frameworks, and Blender integration to enable intuitive, chat-based scene design. Specifically, T5-mini is fine-tuned for parsing user commands, while all-MiniLM-L6-v2 supports semantic retrieval from a local object library. For model generation, LLaMA-Mesh provides fast mesh creation, and Shap-E delivers high-quality outputs. A custom Blender export plugin ensures compatibility with the RF-3DGS pipeline. We demonstrate the tool by constructing 3D models of the NIST lobby and the UW–Madison wireless lab, followed by corresponding RRF reconstructions. This approach significantly reduces modeling complexity, enhancing the usability of RRF for wireless research and spectrum planning.**

I. Introduction

Current and next-generation wireless communication systems rely heavily on multiple-input multiple-output (MIMO) and massive MIMO technologies, including beamforming, delay-alignment modulation, and reconfigurable intelligent surfaces. Accurate channel information is essential for optimizing these technologies. Recently, the radio radiance field (RRF) has emerged as a promising approach for spatial representation of wireless channels [1–3]. Unlike traditional channel estimation and modeling techniques, an RRF characterizes how radio waves propagate from every point in a 3D space toward the entire spherical domain. This representation captures delay, angle of arrival, angle of departure, and polarization in addition to conventional channel gain. However, reconstructing an RRF, such as with RF-3DGS [3], requires geometric information about a bounded environment, typically obtained through physical measurements and advanced computer vision techniques. Existing 3D reconstruction tools (e.g., Blender) are not directly compatible with current RRF pipelines, creating a significant barrier for researchers.

This project is partially supported by the U.S. National Science Foundation under Grants ECCS-2336234 and CNS-2236449.

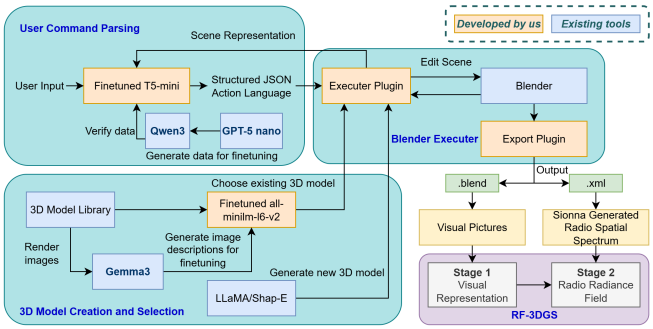


Fig. 1: Overview of the LLM-supported 3D modeling tool.

To address this challenge, we propose an intuitive tool that leverages multiple locally deployed large language models (LLMs) to generate 3D environments compatible with RF-3DGS. Our system enables users to create and manipulate complex 3D models through a simple chat interface, with support for sharing and reusing environments within the research community. While users can directly employ a general-purpose LLM within our framework, we focus on a local deployment in is work for better privacy, latency, and resource efficiency. As illustrated in Fig. 1, the tool consists of three main components: natural language parsing, 3D model creation/selection, and output interfacing. First, natural language commands are converted into a structured, machine-readable format for 3D applications. This is achieved by combining a general-purpose LLM for intent understanding with a finetuned T5-mini model [4] that translates commands into structured JSON actions. Second, a custom Blender executor plugin applies these actions to edit the scene. Users can create new 3D objects using local generative models such as LLaMA-Mesh [5] or Shap-E [6], or search an existing library via a finetuned all-minilm-l6-v2 model [7]. This design promotes flexibility and collaboration. Finally, a customized export plugin ensures seamless integration with RF-3DGS by supporting both visual and RF model reconstructions.

We demonstrate the proposed tool by constructing two complex indoor environments: the National Institute of Standards and Technology (NIST) lobby [8] and the wireless lab at the University of Wisconsin-Madison (UW-Madison). Natural language instructions specifying room structure, object di-

mensions, materials, and fine-grained scene manipulations are provided to the chatbot. The resulting environments are then used for RRF reconstruction with RF-3DGS. Experimental results show that scenes generated through our system are highly comparable to those built using traditional manual methods, highlighting its potential for practical applications. The remainder of this paper is organized as follows: Section II-A describes the natural language parsing approach using LLMs; Section III details the 3D modeling process with local generative AI and library integration; Section IV introduces the customized Blender plugin for RRF pipeline compatibility; and Section VI concludes the paper.

II. A Local LLM for User Command Parsing

While users can directly employ a general-purpose LLM within our framework for natural language parsing, it introduces heavy computational and storage overhead. To address this problem and inadequate ability of smaller models without adaptation, we present an LLM distillation approach of finetuning a small sequence-to-sequence [9] model with synthetic data generated by LLM.

A. Direct Parsing with LLM

The primary objective of parsing is to translate a user's natural language command into a JSON array of action objects, where each object represents a discrete operation in the 3D scene. This structured format ensures deterministic interpretation and execution. For example, given the command "Add a nightstand, then put a lamp on it", the second action depends on the first, so actions must be executed sequentially in array order. Each action object includes an action_type field, which must match one of 16 predefined values (e.g., create_object_relative, move_object_offset, change_object_material). Additional fields depend on the action type: **Creation Actions**("create_object_absolute", "create_object_relative") require fields like "object_type" (e.g., "chair"), quantity, and a "local_id" (e.g., "1"). The local_id enables subsequent actions to reference newly created objects using the # prefix. **Modification Actions** (move, rotate, resize, delete, change material) require an "object_name" to identify the target, which may refer to an existing scene object or a local ID. Parameters specify details such as position (x, y, z), rotation angle, size, material, or scale factor. **Room Setup Action** ("setup_room") initializes the basic room structure, assumed to be a cube, and requires room size parameters This schema is designed to comprehensively support essential operations for constructing 3D scenes in wireless communication research. More actions can be added straightforwardly if deemed necessary.

Achieving accurate parsing with a general-purpose LLM depends on a rigorous prompt, dynamically constructed for each user command. The prompt includes:

- Role definition: A directive specifying the model's role as an AI assistant for 3D applications and its task of converting user input into a JSON list of actions.
- Scene Context: A list of existing objects, valid materials, directions, and room elements. Object names (e.g., "chair.001") are descriptive to aid interpretation.
- Schema Definition: Explicit JSON output rules to enforce structural correctness.
- Few-Shot Examples: High-quality examples illustrating multi-step commands, relative positioning, and object references.
- Input Output Format: The user command appears at the end of the prompt, and the output must be enclosed in a JSON code block.

This structured approach constrains the LLM's creativity and ensures syntactically and semantically valid JSON outputs.

B. Distill a Local Model With Synthetic Data

Decoder-only LLMs excel at generating fluent text but often introduce syntax errors and hallucinated fields when producing structured outputs like JSON. Moreover, large models incur significant computational overhead and latency, making them unsuitable for real-time interactive applications. To address these challenges, we adopt a model distillation strategy: First, use a powerful LLM to generate a synthetic, high-quality dataset of natural language commands and corresponding JSON outputs; Second, finetune a smaller, efficient model, specifically Google's T5 [10], on this dataset. Common decoder-only LLMs excel at creative tasks like generating fluent text by sampling the next word from a probability distribution, but their creativity is a liability for structured data with strict grammar like JSON. It often introduces syntax errors and hallucinated fields. Although a large language model is a powerful tool for natural language parsing, it introduces significant computational overhead and latency, making it unsuitable for a real-time and interactive application. To address this, we adopted a model distillation approach. We used a powerful LLM to generate a synthetic high-quality dataset, which will then be used to finetune a much smaller and more efficient model, specifically Google's T5 [10]. This finetuned model can perform the parsing task with much lower computational cost and latency. T5's encoder-decoder architecture is well-suited for tasks requiring strong alignment between input and output, such as translating natural language into structured JSON. This design ensures deterministic, stable results. We also apply beam search during decoding to improve accuracy. Synthetic data generation offers several advantages over manual annotation: it is faster, covers edge cases, and allows controlled complexity. By leveraging a capable LLM for data generation, we ensure comprehensive coverage and minimal human effort.

To ensure high quality dataset for model distillation, the creation pipeline consists of two main stages: data generation and multi-stage validation. In the first stage, a generator LLM is prompted with detailed instructions, schema definitions, and parameters that control sample diversity. These parameters specify the number of samples, the complexity of the scene (ranging from none to many objects), and the complexity of user commands (simple, medium, or complex). To further

enhance diversity, a random required action is added to each prompt. The generator then produces samples that include a natural language command, a list of existing objects in the scene, and the corresponding structured JSON actions. The second stage ensures the quality and correctness of the generated data through a rigorous validation process. Initially, each sample is checked for valid JSON syntax. Next, a rule-based validator corrects common errors such as inconsistent local ID sequencing and verifies compliance with schema rules, including action types, field names, and data types (e.g., ensuring quantity is an integer). Duplicate samples are then removed to maintain dataset uniqueness. Finally, semantic validation is performed using a reasoning-capable LLM (Qwen3-8B), which evaluates whether the JSON output logically interprets the input command according to the defined rules. This step catches subtle semantic errors and hallucinations that rule-based checks might miss. Only samples that pass all validation stages are included in the final dataset for finetuning, ensuring high-quality training data for the distilled model.

III. 3D Model Creation and Selection

Our tool allows users to either create a new 3D model or select one from an existing library. Model creation is powered by locally deployed Shap-E [6] and LLaMA-Mesh [5], which provide complementary generation capabilities. Shap-E, a diffusion-based model, produces high-quality 3D models from text prompts but operates relatively slowly, making it ideal when visual fidelity is the priority. In contrast, LLaMA-Mesh represents 3D meshes as text tokens using a large language model approach, enabling much faster generation at the cost of lower detail, which makes it better suited for interactive applications. Users can choose Shap-E for superior quality or LLaMA-Mesh for rapid content creation, depending on their requirements for accuracy versus latency.

Alternatively, users can retrieve 3D objects from a local library through a semantic search system that matches descriptive language (e.g., "a vase with a wide base") rather than requiring specific identifiers (e.g., "vase_0501"). This system ensures that retrieved objects align with the user's intended visual characteristics and operates in two stages: data preparation and real-time inference. During data preparation, we adopted ModelNet-40 [11] as the base asset library. For each object, two images were rendered from different viewpoints, and a multi-modal LLM (Gemma3:12B) generated three textual descriptions per image. After removing duplicates, each object had up to six distinct descriptions, forming a rich semantic representation. To overcome the limitations of standard embedding models, which cluster objects by category rather than instance-level differences, we finetuned all-MiniLM-L6-v2 using a contrastive learning approach with triplet loss. This training strategy ensures that descriptions of the same object remain close in embedding space while those of different objects are pushed apart. After finetuning, all descriptions were converted into vector embeddings and indexed using FAISS [12] for efficient similarity search. During inference, the user's descriptive phrase is embedded and matched against the stored vectors to retrieve the most semantically similar object from the library.

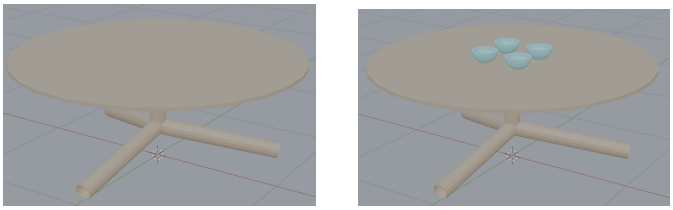
IV. Custom Blender Executor Plugin

A. The Executer Plugin

To connect the backend Blender, an executor plugin is created as a server, which listens to requests and executing the expected actions. The transferred data is in JSON format through HTTP requests. This plugin has two main functions: getting scene status and modifying the scene.

The "get_scene" API returns all necessary scene information. This data includes room dimensions and details of scene objects. Each object is represented by not only standard attributes like location and rotation but also crucial geometric properties such as its geometric center, bounding box coordinates, and overall size for better spatial relation understanding and object placement. As illustrated in Fig. 2a, given a command *"Create a wood table in the center of the room"*, the system confirms the acquisition of the asset, and transmits the object to Blender.

The second function is to execute modifications to the scene. It receives action requests in series. The abstract parameters have been resolved into concrete operations. For instance, the description for object creation is replaced with a specific filename. Besides, the reference objects using local ID is replaced with actual names. Then relative placements are further calculated into absolute position coordinates. Then the plugin executes each action sequentially by calling the appropriate Blender API function. As shown in Fig. 2b, given a follow-up command *"Create 4 bowls on the table,"* the system interprets the instruction and generates the corresponding 3D elements within the established scene.



(a) Instruction 1: Create a table in the center of the room

(b) Instruction2: Put 4 bowls on the table

Fig. 2: A demonstration of 3D objects creation using the chat-based interface.

B. The Blender Export Plugin

To integrate 3D environments created within Blender into the RF-3DGS pipeline, the output files need to be compatible with an open-source ray tracing simulation tool, Sionna [13]. An existing solution is the Blender plugin called *mitsuba-blender* [14]. However, this plugin is not available after build of Blender version 4.3 and is not stable on earlier builds either. To address this issue, a custom Blender export plugin was developed. It converts all mesh objects from the Blender scene into individual Polygon File Format (PLY) files and generates a single Extensible Markup Language (XML) file

that describes the scene structure and material properties of each object. The plugin only chooses visible mesh objects by using the condition `obj.type == 'MESH' and not obj.hide_viewport` to export. It also automatically deals with the Sionna built-in material prefix "itu_" and material name suffix (e.g., ".001") due to object copy. Besides, it applies the common colors of each material for better visual effects. The main XML file is then written. It begins with a root `<scene>` tag and everything will be in the root tag. Following the header, each material is defined in a `<bsdf>` (Bidirectional Scattering Distribution Function) tag. These BSDF definitions represent the material properties like type and color. In this implementation, each material is defined as a simple "twosided" BSDF with the pre-defined diffuse RGB color that is expressed by 0-1 decimals. After defining all materials, the plugin proceeds to export each filtered mesh object as a PLY file in the created "meshes" folder. A standard PLY file needs the information of the vertices' positions, normal, and indices for faces to represent a mesh. To avoid using extra Python library, we manually constructed the file with the object's mesh data from Blender's bmesh module. For each mesh object, a corresponding `<shape>` entry in the XML file is needed. This tag defines the relative filename and the material information. The ID and name attributes of the shape are derived from the Blender object's name. Finally, after processing all relevant objects, the XML file records all necessary information along with the specified corresponding mesh files. This results in a folder containing of scene that is ready for loading into the RF-3DGS environment.

V. Implementation and Evaluation Results

To evaluate the efficacy and practical application of our 3D scene generation tool, visual and RRF models of the NIST lobby and the wireless lab at UW-Madison are generated. Note that the NIST lobby has been physically measured with point cloud data available, while the wireless lab at UW-Madison has no such data.

A. Performance Evaluation on User Command Parsing

To identify the most effective generator LLM, we evaluated ten models, including variants of Qwen3 [15], Gemma3 [16], Deepseek (DS)-R1 [17], Llama 3.1 [18], and Phi-4 [19]. Each model was tasked with generating five samples per trial across 100 trials. Performance was assessed using five key metrics decribed below:

- **JSON validity rate**: The percentage of outputs that were syntactically valid JSON lists.
- **Format validity rate**: Among valid JSON outputs, the percentage that passed rule-based format validation.
- **Unique sample rate**: The proportion of non-duplicate samples within the valid set, indicating diversity.
- **Meaning matched rate**: Among unique samples, the percentage passing semantic validation by an LLM validator, ensuring that intended actions matched actual outputs.
- **Overall usability rate**: The percentage of all generated samples suitable for finetuning the target model.

TABLE I: Synthetic data generation performance (%) comparison among different LLMs (Since Qwen3 can switch between reasoning and non-reasoning mode, '*' indicates reasoning mode is on.)

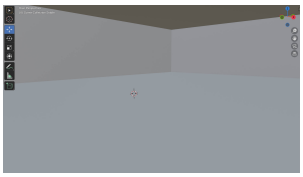
LLM	JSON	Format	Unique	Meaning	Overall
Qwen3 (0.6B)*	90.0	37.6	60.3	46.2	9.4
Qwen3 (0.6B)	92.0	46.3	63.6	50.2	13.6
Qwen3 (8B)*	79.0	18.0	84.8	93.0	11.2
Qwen3 (8B)	96.0	82.1	63.8	82.0	41.2
Qwen3 (14B)*	83.0	50.1	88.8	89.9	33.2
Qwen3 (14B)	78.0	71.8	90.1	83.2	42.0
Gemma3 (1B)	64.0	96.9	43.2	78.4	21.0
Gemma3 (12B)	97.0	82.3	55.6	83.5	37.1
DS-R1 (1.5B)	28.0	11.4	100.0	12.5	0.4
DS-R1 (8B)	0.0	0.0	100.0	0.0	0.0
DS-R1 (14B)	84.0	50.0	93.2	63.3	24.8
Llama3.1 (8B)	78.0	44.9	100.0	37.1	13.0
Phi4 (14B)	81.0	78.0	90.2	80.4	**45.8**
GPT-5(nano)	96.0	90.4	100.0	83.9	**72.8**

As shown in Table I, some models struggled to consistently produce valid JSON, while others delivered high-quality outputs. In general, larger models achieved better performance.

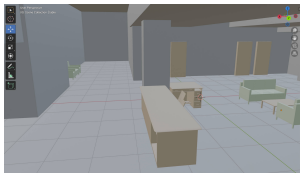
Based on empirical results, we selected GPT-5 (nano) as the generator model for synthetic data creation. Using this dataset, we finetuned several models from the T5 family to evaluate their ability to translate natural language commands into structured JSON outputs. This comparison allowed us to identify the most efficient and accurate model for local deployment. Accuracy was chosen as the primary evaluation metric during finetuning. It is computed through strict field-by-field matching, requiring the predicted output to exactly match the ground truth. However, this metric slightly underestimates practical performance for two reasons. First, it treats minor variations in capitalization or formatting as mismatches, even when they are semantically equivalent (e.g., "tv_stand", "TV stand", and "tv-stand"). Second, multiple valid interpretations of a command are possible. For example, "create a wood chair" could be executed in one step (create a chair with wood material) or two steps (create a chair, then change its material

TABLE II: Finetuned model performance comparison among different base models ('*' indicates reasoning mode.)

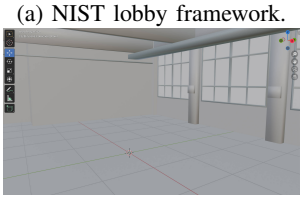
Model	#Parameters	JSON(%)	Accuracy(%)
T5 (tiny)	15.6M	73.42	45.99
T5 (mini)	31.2M	100.0	85.91
T5 (small)	60.5M	100.0	86.34
T5 (base)	222.9M	100.0	**87.53**
Qwen3*	0.6B	99.37	44.34
Qwen3	0.6B	99.79	39.55
Qwen3*	8B	99.16	65.27
Qwen3	8B	100.0	67.03
Qwen3*	14B	98.95	68.53
Qwen3	14B	100.0	**72.31**
Gemma3	1B	99.79	39.79
Gemma3	12B	100.0	70.10
DS-R1	1.5B	68.14	16.49
DS-R1	8B	0.00	0.00
DS-R1	14B	97.89	67.62
Llama3.1	8B	83.76	49.14
Phi4	14B	95.99	71.47



(a) NIST lobby framework.



(b) Furnished NIST lobby.



(c) UW-Madison lab framework.

(d) Furnished UW-Madison lab.

Fig. 3: Demo of the 3D sene gereration.

to wood). Although the latter is redundant, both sequences produce correct results. The results in Table II show that finetuned T5 models, despite their smaller size, significantly outperform general-purpose LLMs on this specialized parsing task. While T5-base achieved the highest accuracy, improvements plateaued as model size increased. T5-mini, with only 31.2M parameters, reached an accuracy of 85.91%, and doubling the size to T5-small yielded only a marginal 0.43% gain. Given our goal of building an efficient, locally deployable tool, T5-mini offers the best trade-off between accuracy and computational cost. Therefore, we select the finetuned T5-mini as the core language model for our application.

B. Demonstration of 3D Scene Generation and RRF

Using the 3D point cloud of the NIST lobby from our previous work [3], we leveraged known room dimensions and precise furniture locations to guide scene construction. We began by creating six cubes to define the outer boundaries of the room. Once the outer shell was established, the chat-based interface was used to refine the structure by adding walls and pillars. Next, we populated the scene with furniture and other objects, specifying accurate positions and dimensions to replicate the real-world layout. Material properties were also incorporated to enable realistic radio wave propagation during RRF reconstruction. The starting frame and the furnished 3D models are shown in Figs. 3a and 3b, respectively. The process for building the wireless lab at UW–Madison followed a similar approach, though without a reference point cloud. In this case, we generated a custom whiteboard model instead of using one from the built-in library and ensured that windows were accurately represented for improved RRF reconstruction fidelity. The starting frame and the completed 3D models are depicted in Figs. 3c and 3d, respectively.

With the constructed scenes, we implemented RRF reconstruction using the RF-3DGS pipeline. For details on the NIST lobby reconstruction, please refer to [3]. Here, we briefly outline the process for the UW-Madison lab. In Stage 1 of RF-3DGS, the standard approach was applied, with a manually designed camera trajectory to avoid blind spots. Fig. 4 shows a top-down cross-sectional view of the lab in

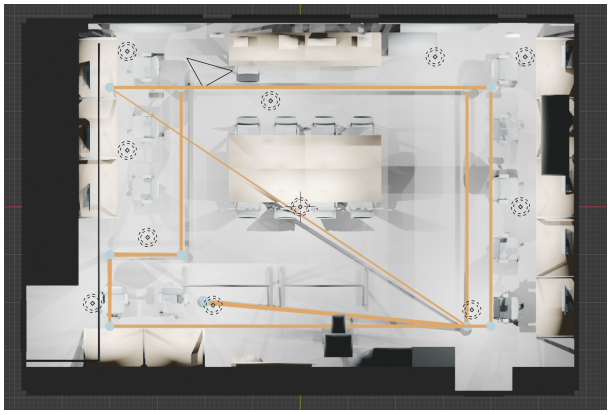


Fig. 4: The top-down cross-sectional view of the UW-Madison lab scene in Blender and the configured camera trajectory (hidden ceiling and tubes).

Blender, where the orange line represents the camera path and the light blue points indicate stop positions used to define the trajectory. In Stage 2, we employed Blender's BVHTree module from the mathutils library [20] to detect co-located Tx and receivers Rx. A BVHTree (Bounding Volume Hierarchy Tree) was constructed from all mesh objects in the scene, enabling efficient spatial queries. Before placing a Tx or Rx, the algorithm checks whether the candidate location intersects any geometry using ray-casting and point-containment tests. This automated validation ensures that all Tx and Rx nodes are positioned in free space, preserving physical realism and preventing implausible placements. Fig. 5 and Fig. 6 illustrate the final visual and RRF reconstructions for the UW–Madison lab and NIST lobby, respectively.

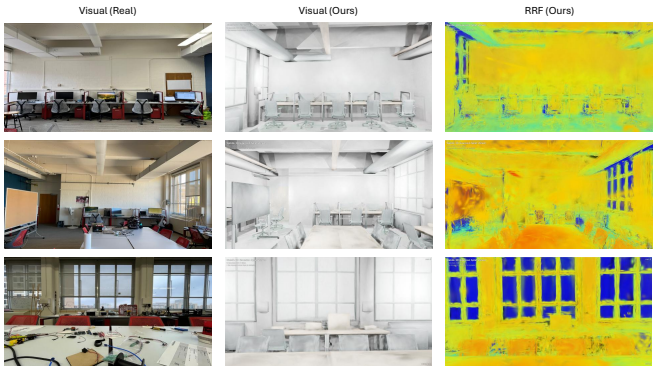


Fig. 5: Demonstration of visual and RRF reconstruction of the wireless lab at UW-Madison by our method.

Please note that the colors in the rendered models are intentionally kept plain, as they do not affect the final RRF reconstruction. The reconstructed object structures and their corresponding RRF representations closely resemble those of the actual environments. As shown in Table III, both the Structural Similarity Index Measure (SSIM) and the Learned Perceptual Image Patch Similarity (LPIPS) scores are comparable to the benchmark RF-3DGS approach and outperform

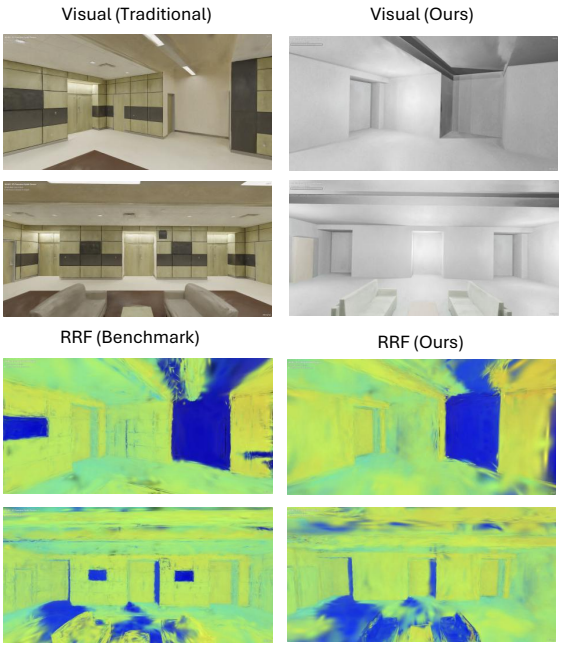


Fig. 6: Comparison of visual and RRF reconstruction of the NIST lobby between traditional and our methods.

TABLE III: Comparison of RRF reconstruction between the proposed method and benchmarks [3].

Metric	Ours	RF-3DGS	NeRF^2	CGAN
SSIM	0.66	0.50	0.46	-
LPIPS	0.57	0.40	0.69	0.87

alternative methods. While our proposed tool significantly streamlines the process of 3D scene generation, the current implementation is limited to a predefined set of operations (create, update, delete, etc.) and does not fully leverage the capabilities of backend AI models for generalized user inputs. Future versions will integrate more advanced LLMs, enhanced task-processing prompts, and stricter output constraints to support a broader range of natural language commands and enable more flexible scene manipulation.

VI. Conclusion

This paper presented a locally deployable 3D modeling tool for RRF reconstruction. The tool enabled intuitive, chat-based instructions for creating and manipulating 3D objects and environments fully compatible with the RF-3DGS pipeline. Specifically, the backend integrated finetuned T5-mini and all-MiniLM-L6-v2 models alongside open-source generative frameworks such as LLaMA-Mesh and Shap-E. A chat interface facilitated user-friendly interactions, while a custom Blender executor plugin ensured seamless scene editing. Additionally, an export plugin was developed to produce 3D models compatible with RRF reconstruction requirements. The tool was demonstrated by generating visual and RRF models of the NIST lobby and the wireless lab at UW-Madison. These case studies highlighted the system's ability to simplify complex scene creation while maintaining compatibility with advanced wireless simulation workflows. Overall, the proposed tool provided an intuitive and efficient approach for implementing RRF, laying a strong foundation for future wireless research and spectrum planning.

References

[1] X. Zhao, Z. An, Q. Pan, and L. Yang, "Nerf2: Neural radio-frequency radiance fields," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, ser. ACM MobiCom '23. ACM, Oct. 2023, p. 1–15. [Online]. Available: http://dx.doi.org/10.1145/3570361.3592527

[2] G. Wu, Z. Lyu, J. Zhang, and J. Xu, "Embracing radiance field rendering in 6g: Over-the-air training and inference with 3-d contents," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 4275–4292, 2024.

[3] L. Zhang, H. Sun, S. Berweger, C. Gentile, and R. Q. Hu, "Rf-3dgs: Wireless channel modeling with radio radiance field and 3d gaussian splatting," 2025. [Online]. Available: https://arxiv.org/abs/2411.19420

[4] Y. Tay, M. Dehghani, J. Rao, W. Fedus, S. Abnar, H. W. Chung, S. Narang, D. Yogatama, A. Vaswani, and D. Metzler, "Scale efficiently: Insights from pre-training and fine-tuning transformers," *arXiv preprint arXiv:2109.10686*, 2021.

[5] Z. Wang, J. Lorraine, Y. Wang, H. Su, J. Zhu, S. Fidler, and X. Zeng, "LLaMA-Mesh: Unifying 3D Mesh Generation with Language Models," Nov. 2024, arXiv:2411.09595 [cs]. [Online]. Available: http://arxiv.org/abs/2411.09595

[6] H. Jun and A. Nichol, "Shap-e: Generating conditional 3d implicit functions," *arXiv preprint arXiv:2305.02463*, 2023.

[7] S. Transformers, "all-minilm-l6-v2," 2021. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[8] C. Gentile, J. Senic, A. Bodi, S. Berweger, R. Caromi, and N. Golmie, "Context-aware channel sounder for ai-assisted radio-frequency channel modeling," in *2024 18th European Conference on Antennas and Propagation (EuCAP)*, 2024, pp. 1–5.

[9] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[10] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html

[11] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[12] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," 2024.

[13] J. Hoydis, S. Cammerer, F. Ait Aoudia, M. Nimier-David, L. Maggi, G. Marcus, A. Vem, and A. Keller, "Sionna," 2022, https://nvlabs.github.io/sionna/.

[14] Mitsuba Renderer Team, "mitsuba-blender: Mitsuba integration add-on for blender," https://github.com/mitsuba-renderer/mitsuba-blender, 2024, version 0.4.0, accessed April 30, 2025.

[15] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[16] G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, M. Rivière *et al.*, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.

[17] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[18] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[19] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann *et al.*, "Phi-4 technical report," *arXiv preprint arXiv:2412.08905*, 2024.

[20] Blender Foundation, "Blender bvhtree module," 2024, available from https://docs.blender.org/api/current/mathutils.bvhtree.html.