The Pennsylvania State University

The J. Jeffrey and Ann Marie Fox Graduate School

EMBEDDING INTELLIGENCE IN 3D MODELING WORKFLOWS: AN APPROACH FOR ADAPTING LANGUAGE MODELS TO ASSIST USERS IN 3D MODELING USING NATURAL LANGUAGE

Final Project for the class CMPSC - 497

by

Deebak Tamilmani

1 Overview

The progression of human civilization is inherently linked to the evolution of tools. Over the past century, architectural design has evolved alongside the development of computational tools and digital modeling environments that mediate creative production. 3D modeling remains an integral part of today's design process. However, the steep learning curve of 3D modeling applications, due to their complex user interfaces and extensive command sets, presents a significant barrier. In academic settings, students often dedicate considerable time to learning these tools, limiting their ability to focus on design exploration and development.

This research positions itself within this critical context and investigates how Large Language Models (LLMs) can be adapted to reimagine human-computer interaction in 3D modeling environments. It challenges conventional modeling methods by embedding an LLM within Rhinoceros 3D, enabling users to communicate their design intent using natural language through both text and voice inputs.

The key adaptation method for this purpose is fine-tuning the model on Rhino-specific data to generate executable Python scripts for complex modeling operations.

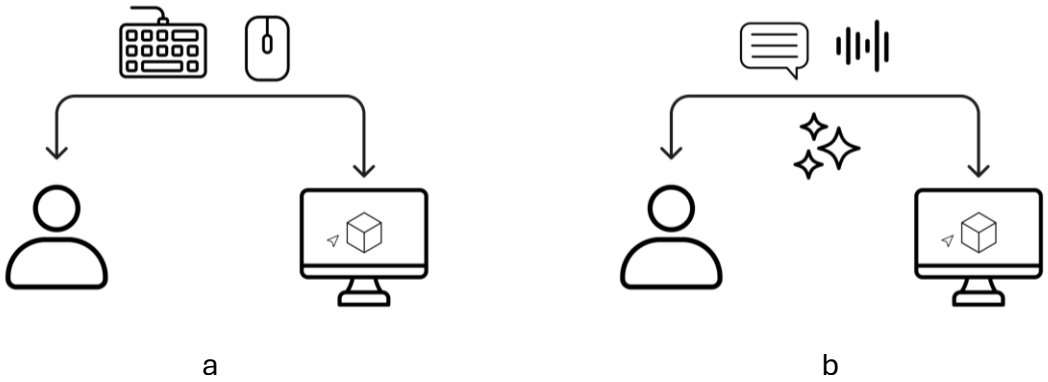

Figure 1-1: a. Traditional 3D workflow (using keyboard, mouse, icons) b. Proposed 3D workflow using Large Language Models (LLMs)

2 Goals and Objectives

This research aims to develop an LLM-based system that redefines how users interact with 3D modeling software by enabling natural language communication. By challenging conventional interface paradigms centered around command-line inputs, menus, and icons, the project seeks to make modeling tools more accessible, intuitive, and inclusive for diverse users, regardless of their technical expertise or physical abilities.

To accomplish this goal, the research will develop a framework for adapting general-purpose language models to 3D modeling workflows in Rhinoceros through instruction fine-tuning on Rhino-specific data.

3 Research Plan

3.1 Adapting Language Models for 3D Modeling Tasks

Pretrained language models, such as GPT, LLaMA, and CodeLLaMA, are optimized for general-purpose language generation and code synthesis tasks. However, these models are not inherently aligned with domain-specific requirements such as generating executable commands or scripts within a 3D modeling environment like Rhinoceros.

To adapt language models for 3D modeling tasks, this research employs instruction fine-tuning on a Rhino-specific dataset, consisting of modeling tasks paired with corresponding Python script responses. Fine-tuning enhances the model's ability to generate complex scripts for modeling tasks.

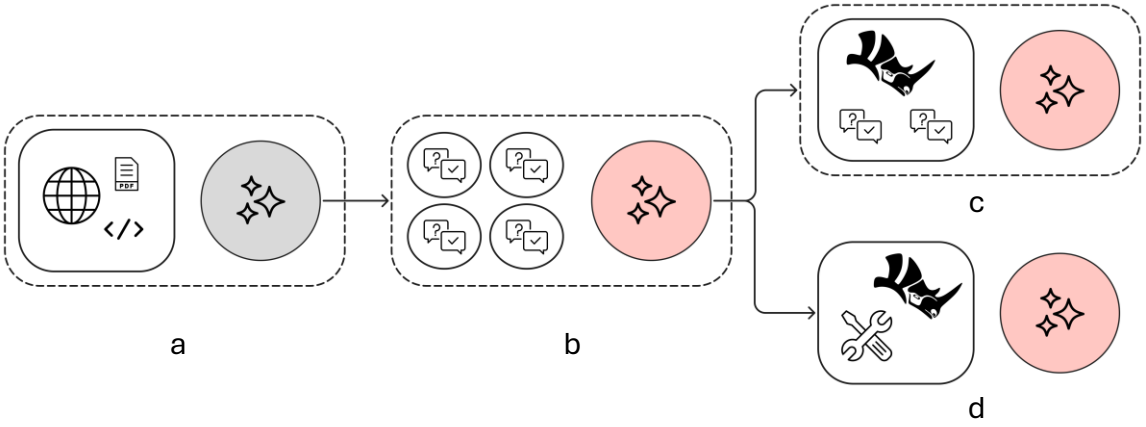

Figure 3-1: Illustrates the general training process of language models (a. Pretraining b. Fine-tuning), and the two primary adaptation strategies adopted in this research for 3D modeling tasks (c. Instruction fine-tuning on a Rhino-specific dataset d. Tool-calling using exposed Rhino functions).

3.2 Challenges in Fine-tuning and Mitigation Strategy

Fine-tuning large-scale language models presents significant computational challenges, particularly when working with models containing billions of parameters. Moreover, adapting these models to domain-specific tasks increases the risk of catastrophic forgetting, wherein the model loses its general language understanding capabilities learned during pretraining.

To address these challenges, this research employs parameter-efficient fine-tuning (PEFT) techniques, specifically Low-Rank Adaptation (LoRA). LoRA introduces a small set of trainable low-rank matrices into the model architecture, enabling domain adaptation with reduced computational overhead while retaining the model's general language capabilities.

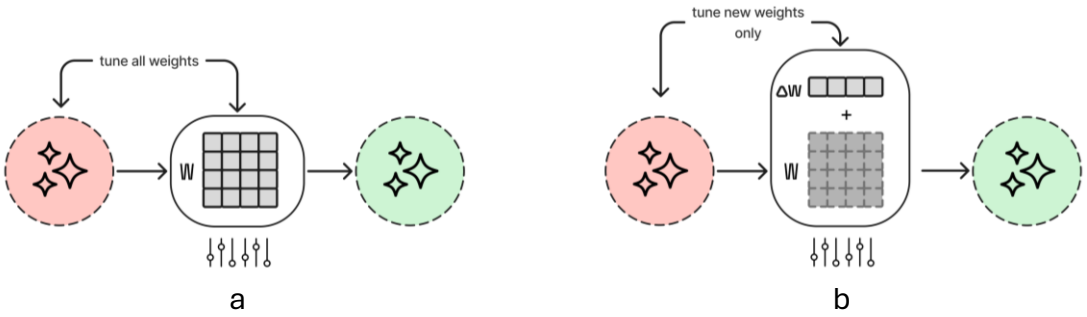

Figure 3-2: a. Full Finetuning (Computationally expensive, leads to catastrophic forgetting). b. Low Rank Adaptation (LoRA) finetuning

4. Dataset Construction for Instruction Fine-Tuning

4.1 Extracting Rhino-Specific Data

Rhino API documentation for Python scripting serves as the primary source of structured command data, including command names, parameters, descriptions, and example scripts. A Python program is developed to automate the extraction of this content using the BeautifulSoup library, outputting a structured JSON object containing command name, command description, parameters, and example script fields.

4.2 Synthesizing User Queries

To simulate real-world user interaction, synthetic queries are generated for each command. This process utilizes GPT-4o via OpenAI's API to generate four distinct types of user queries:

1. A beginner-level query describing the modeling goal without command knowledge
2. An intermediate query reflecting conceptual understanding but lacking technical specifics
3. An expert query referencing command names and parameters
4. A contextual query embedding the command within a specific design scenario or workflow

The synthetic user query, combined with the system prompt, forms the instruction component of the instruction-response pair.

4.3 Generating Python Script Responses

For each synthetic query, a corresponding Python script is generated to serve as the response component. This process uses GPT-4o with a structured system prompt, an example code template, and task descriptions to guide the model in generating accurate, executable Python scripts aligned with the Rhino API.

4.4 Compiling Instruction-Response Pairs

The final dataset is formatted in a conversational structure consistent with OpenAI's fine-tuning data specifications. This formatting enables the dataset to be directly utilized for supervised instruction fine-tuning. The dataset will be uploaded to the Hugging Face Hub for version control and subsequent access during model training.

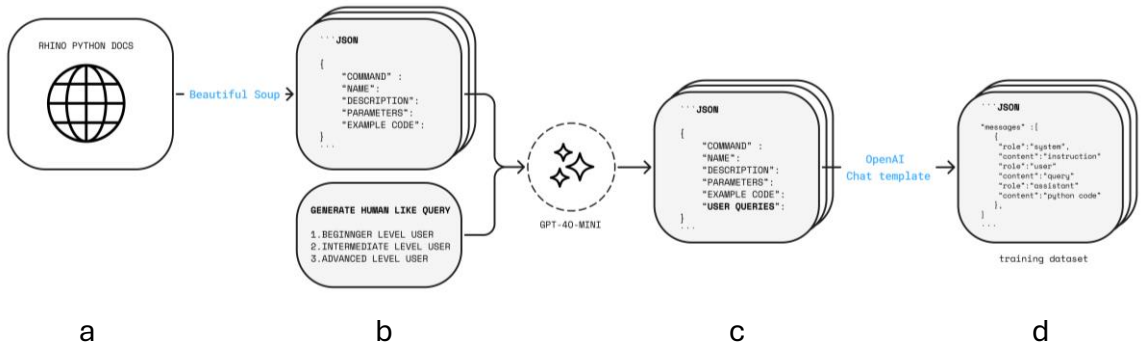

Figure 3-3: a. Extracting raw data from Rhino API documentation; b. Structuring data into JSON format; c. Generating synthetic user queries and Python scripts using GPT-4o Mini; d. Compiling the dataset in OpenAI chat template format for instruction fine-tuning.

5. Supervised Fine-Tuning (SFT)

5.1 Model Selection

Large-scale language models containing over 100 billion parameters, while highly capable, present significant computational challenges for domain-specific fine-tuning and inference tasks. To balance model performance with computational efficiency, this research focuses on smaller-scale models ranging from 1 billion to 11 billion parameters, which can be hosted locally and fine-tuned with limited hardware resources.

This study employs the Gemma 3 27B model as the base model for supervised fine-tuning, selected for its open-source availability, instruction-tuned architecture, and compatibility with parameter-efficient fine-tuning techniques.

5.2 Fine-Tuning with Low-Rank Adaptation (LoRA)

Supervised fine-tuning will be conducted using the Hugging Face Transformers and PEFT libraries. The instruction-response dataset will be loaded into the training pipeline, with instruction text masked during training to calculate loss exclusively on the response generation.

LoRA introduces additional trainable low-rank matrices into specific layers of the model, enabling task-specific adaptation while preserving the general knowledge of the pretrained model. During inference, these low-rank adaptations are combined with the base model weights to generate output without modifying the full set of original parameters.

This approach significantly reduces training time, mitigates catastrophic forgetting, and facilitates efficient fine-tuning across different tasks. The number of training epochs will be determined based on validation performance and available computational resources.

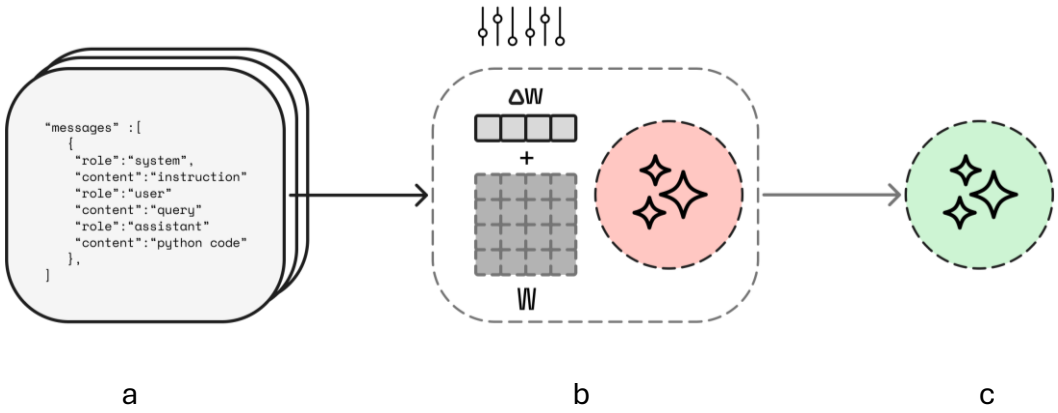

Figure 3-4: a. Instruction-response dataset for fine-tuning; b. Fine-tuning process using LoRA; c. Fine-tuned LLaMA 3B-Instruct model.

6. Evaluation metric and experiments

The analysis of the learning curve shows a clear progression in the fine-tuning process across two distinct epochs. During Epoch 1 (steps 1-43), the model exhibits rapid initial learning as it adapts to the domain-specific 3D modeling tasks. Starting with a relatively high loss of 1.19, it achieves a 62.2% reduction to reach 0.45 by the end of the first epoch. This phase demonstrates higher variability (standard deviation of 0.22) as the model makes substantial weight adjustments to accommodate the new task domain.

Epoch 2 (steps 44-86) begins with a notable anomalous spike (0.97) at step 44, which could indicate either a learning rate adjustment or a particularly challenging batch of training data. However, the model quickly recovers and continues its downward trajectory with more consistent, incremental improvements. The second epoch shows significantly lower variability (standard deviation of 0.09), suggesting the model is refining its parameters more

subtly as it approaches convergence. By the end of Epoch 2, the loss reaches 0.37, representing a 61.9% reduction within the epoch itself.

The overall training process achieves a 69.0% reduction in loss from the starting point (1.19) to the final step (0.37), with the lowest loss value of 0.32 occurring at step 82. This substantial improvement demonstrates the effectiveness of the parameter-efficient fine-tuning approach using Low-Rank Adaptation (LoRA) for adapting large language models to the specialized domain of 3D modeling tasks in Rhinoceros. The stability exhibited in the later steps suggests the model has successfully adapted to generate Python scripts for 3D modeling operations while maintaining its general language capabilities.

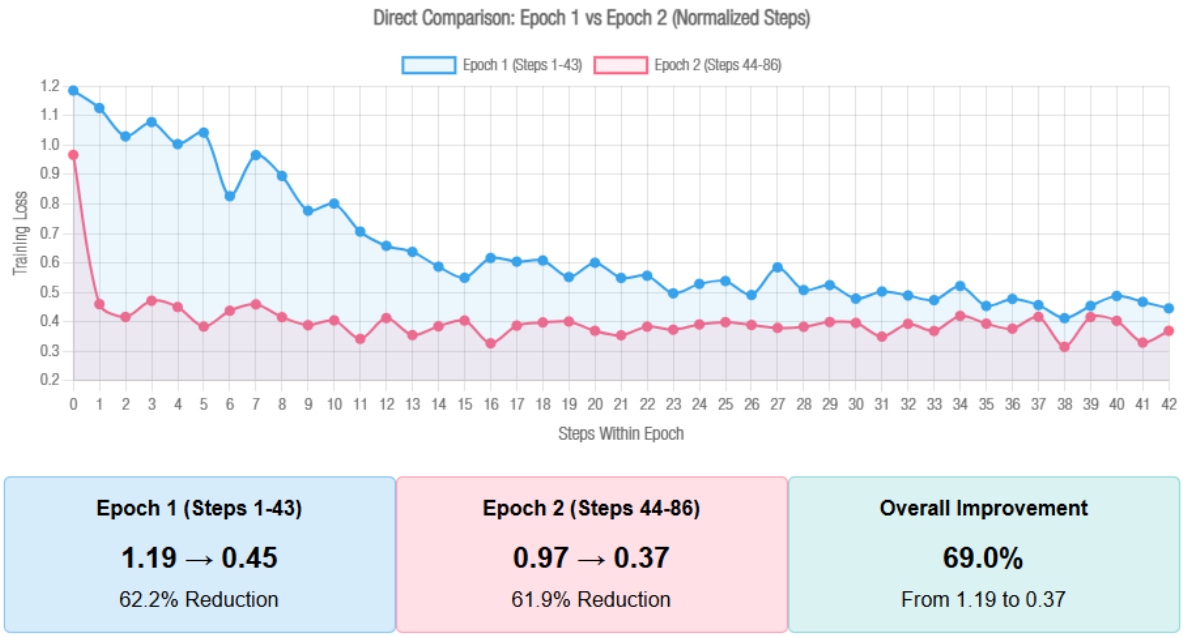

Figure 3-5: Learning Curve on the finetuning data

7. Discussion and Future Work

The analysis of the learning curve shows that the fine-tuned model has successfully learned to interpret natural language instructions for 3D modeling operations. The model demonstrates competency in parsing user inputs, decomposing complex modeling tasks into logical steps, creating pseudocode representations, and identifying appropriate function signatures within the Rhinoceros Python API. The significant reduction in training loss across both epochs (69.0% overall) indicates effective adaptation to the domain-specific task.

However, the model still exhibits limitations in its function generation capabilities. Specifically, it occasionally produces functions that do not exist within the RhinoPython library ecosystem, creating theoretically sound but practically non-executable code. This discrepancy highlights a gap between the model's general code generation abilities and its domain-specific knowledge of the Rhinoceros Python API constraints.

To address this limitation, implementing Direct Preference Optimization (DPO) as the next refinement stage could substantially improve performance. DPO represents an advancement over traditional reinforcement learning from human feedback methods by directly optimizing the preference gap between desired and undesired outputs without requiring an explicit reward model. The implementation would involve:

1. Generating paired samples of responses (correct RhinoPython functions vs. invalid functions) for identical user queries
2. Formulating a preference dataset where valid RhinoPython API implementations are preferred over invalid ones

3. Optimizing the model to maximize the likelihood gap between preferred and non-preferred responses

This approach would effectively penalize the model for generating non-existent functions while reinforcing the use of valid RhinoPython API calls.

8 References

[1] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems 33* (pp. 1877-1901). Curran Associates, Inc.

[2] Cai, T. (2024). Large language models as tool makers. *Artificial Intelligence*, 325, 103481. https://doi.org/10.1016/j.artint.2023.103481

[3] Chaillou, S. (2022). Artificial intelligence and architecture: From research to practice. *Architectural Design*, 92(1), 76-83. https://doi.org/10.1002/ad.2785

[4] Du, C., Nousias, S., & Borrmann, A. (2024). Towards a copilot in BIM authoring tool using large language model based agent for intelligent human-machine interaction. *Automation in Construction*, 156, 105124. https://doi.org/10.1016/j.autcon.2023.105124

[5] Jiang, C., Zheng, Z., Liang, X., Lin, J. R., Ma, Z., & Lu, X. Z. (2024). A new interaction paradigm for architectural design driven by large language model. *Automation in Construction*, 158, 105216. https://doi.org/10.1016/j.autcon.2023.105216

[6] Li, C., Zhang, T., Du, X., Zhang, Y., & Xie, H. (2024). Generative AI for architectural design: A literature review. *Buildings*, 14(2), 361. https://doi.org/10.3390/buildings14020361

[7] Qin, Y. (2023). Tool learning with foundation models. *Journal of Artificial Intelligence Research*, 78, 1449-1459. https://doi.org/10.1613/jair.1.14564

[8] Talkhounche, S. N. (2024). A natural language interface for modeling in Rhino 3D using large language models. *Computer-Aided Design and Applications*, 21(4), 723-741. https://doi.org/10.14733/cadaps.2024.723-741

[9] Zhang, Y., & Wallace, B. (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv preprint arXiv:1510.03820

[10] Bat El Hizmi, Shkolnik, A., Austern, G., & Sterman, Y. (2024). LLMto3D - Generating Parametric Objects from Text Prompts. https://doi.org/10.13140/RG.2.2.25424.83206

[11] Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., & Finn, C. (2023). Direct preference optimization: Your language model is secretly a reward model. Advances in Neural Information Processing Systems, 36, 53728-53741

[12] Parthasarathy, V. B., Zafar, A., Khan, A., & Shahid, A. (2024). The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. arXiv preprint arXiv:2408.13296