PLLM: Pseudo-Labeling Large Language Models for CAD Program Synthesis

Anonymous CVPR submission

Paper ID *****

Abstract

Recovering Computer-Aided Design (CAD) programs from 3D geometries is a widely studied problem. With the recent advancements in large language models (LLMs), several works have explored leveraging their strong symbolic reasoning capabilities for CAD program synthesis. However, existing methods that train LLMs to generate CAD programs rely on supervised learning, whereas ground-truth CAD program datasets are often unavailable in practice. We introduce PLLM : Pseudo-Labeling Large Language Models for CAD Program Synthesis, an unsupervised self-training framework that fine-tunes LLMs for CAD program generation without requiring paired supervision. Our method takes as input a pre-trained LLM capable of generating CAD programs and a 3D shape dataset. The model iteratively refines the pre-trained LLM's performance on the new dataset, achieving improved program synthesis quality without access to ground-truth CAD programs.

1. Introduction

Computer-Aided Design (CAD) is the industry standard for 3D modeling in engineering and manufacturing. Designers typically construct models through a sequence of parametric operations, which, when executed, produce boundary representations (B-reps) of 3D geometry. The inverse problem of recovering a CAD program from a given shape is also extensively studied. Recovering the program enables semantic editing, programmatic modification, and compact representation of 3D models.

Previous approaches address this inverse problem by training lightweight neural networks to predict CAD operations and their corresponding parameters [1, 5, 11, 52, 55]. More recently, large language models (LLMs) have been explored for this reverse-engineering task due to their strong symbolic reasoning abilities and rapid progress in program synthesis [27, 30, 34, 43, 44]. However, existing methods all rely on supervised learning that requires ground-truth CAD programs. This reliance introduces two major challenges: (1) when applying a model trained on one dataset to another without ground-truth programs, fine-tuning becomes difficult due to the absence of supervision; and (2) the existence of multiple CAD programming languages makes it challenging for a model trained on one grammar to generalize to another.

In this work, we introduce a new framework to address this problem. Formally, our system takes as input a pre-trained LLM $p(z \mid x, \mathcal{L})$ that generates a CAD program z in language $\mathcal{L}$ from a shape x, where x is sampled from a distribution $\mathcal{S}$. Given another distribution of shapes $\mathcal{S}^*$, our goal is to fine-tune the pre-trained model to adapt it to the new domain. The main challenge is that the model may perform poorly on $\mathcal{S}^*$ because it is not well adapted to this distribution. Moreover, $\mathcal{S}^*$ may lack ground-truth CAD programs or include programs not expressed in $\mathcal{L}$, making direct supervised fine-tuning infeasible.

Our key observation is that the pre-trained LLM inherently possesses the ability to generate programs for shapes from the new domain in its original language $\mathcal{L}$. However, the generated results may be suboptimal. To address this, our method samples programs from the pre-trained model, executes them, and compares the outputs with the target inputs. This process enables the model to learn from its best-performing results, where the best programs serve as pseudo-labels that progressively improve the model through iterative self-training. Specifically, we use CAD-Recode [34], which is trained on the DeepCAD dataset [49] and outputs programs in the CadQuery language, as our pre-trained LLM. We then fine-tune it on the ABC dataset [25], a widely used benchmark that does not include ground-truth CAD programs.

In summary, we propose a novel method to fine-tune existing LLMs for improved CAD program synthesis for new domain in the absence of ground-truth supervision. Our key contributions are as follows:

- We introduce PLLM, a self-training framework that fine-tunes pre-trained LLMs on unlabeled 3D datasets by jointly leveraging search and distillation to discover high-quality pseudo programs.
- We develop a method to sample output programs from LLMs and apply programmatic edits to generate diverse

variations, enriching supervision and improving model robustness across training iterations.
- We validate our method by fine-tuning CAD-Recode (pre-trained on DeepCAD) on the ABC dataset—showing improvements in geometric fidelity.

2. Related Works

2.1. Self Training

Our work primarily belongs to the broader category of unsupervised and weakly-supervised learning [7, 46]. For these families of tasks, many approaches resort to general-purpose policy gradient reinforcement learning [32, 38, 39, 46, 57].However, as CAD programs are generally not differentiable, reinforcement learning methods are not applicable. Instead, we adopt a self-training approach, which has been widely used to improve model performance in weakly-supervised settings [31, 35, 53]. Recent advances further show that self-training and data-augmentation-based methods can enhance neural models across various domains [18, 22, 58].

In the domain of visual program synthesis, self-training has emerged as an effective strategy for learning in the absence of ground-truth program supervision [15, 19, 20]. Our program synthesis method can be seen as execution-guided [8, 12–14], where the training process its guided by the predicted programs' execution results rather than explicit supervision. Notably, PLAD [21] introduces a bootstrapped learning framework that leverages a pre-trained program generator to produce candidate programs for unseen shapes, which are then used to iteratively fine-tune the model. Our approach follows this paradigm, in which we treat our pre-train LLM as the the model, and CAD program synthesis as the task.

2.2. Learning to Recover CAD Programs

Our work also relates to the larger goal of reverse CAD engineering from diverse input modalities, such as voxel grids [26, 36, 40], point clouds [10, 17, 28, 28, 37, 41, 48, 49], and boundary representations [52]. Early approaches relied on heuristic algorithms or lightweight neural networks, whereas recent works have begun to explore large language models for this task [2–4, 16, 29, 33, 45, 50, 54, 56] given their strong symbolic reasoning abilities. Our method falls within this family of approaches.

However, existing methods [23, 24, 27, 34, 43, 51] rely on datasets containing paired ground-truth CAD programs and shapes. In practice, however, high-quality CAD datasets such as [6, 25, 42, 47] are limited, and many of them lack ground-truth programs. We adopt CAD-Recode [34] as the pre-trained LLM for our method, and fine-tunes on it.

3. Method

In this section, we formally describe the PLLM framework, which takes input of the following components:

- **(1) Pre-trained LLM:** A model $p(z \mid x, \mathcal{L})$ capable of generating CAD programs z from input shapes x using the language $\mathcal{L}$, where x is drawn from a source distribution $\mathcal{S}$.
- **(2) Training dataset:** A new dataset of shapes $\mathcal{S}^*$, representing a target distribution that differs from $\mathcal{S}$.
- **(3) Black-box executor:** An executor $\mathcal{E}$ that can execute generated programs z to produce corresponding 3D geometries.

The objective of the PLLM framework is to fine-tune the pre-trained model on the new distribution $\mathcal{S}^*$ to obtain an updated model p'. For an input shape $x^* \in \mathcal{S}^*$, the execution $\mathcal{E}(z^*)$, where $z^* \sim p'(z \mid x^*, \mathcal{L})$, should yield a shape that achieves a higher reward (in our system, a lower Chamfer Distance) when compared to the execution of the original model's output on the same input x^*.

We illustrate the overall PLLM procedure in Figure 1 To fine-tune $p(z \mid x, \mathcal{L})$ toward the target distribution $\mathcal{S}^*$, PLLM iteratively performs four key steps. First, the pre-trained model $p(z \mid x, \mathcal{L})$ is used to sample multiple candidate programs for each input shape $x^* \in \mathcal{S}^*$ (section 3.1). Second, for each input shape, the best sampled program is identified based on the Chamfer Distance between its execution $\mathcal{E}(z)$ and the target shape x^* (section 3.1). Third, programmatic edits are applied to the selected programs to generate additional variants, enabling the model to observe a broader range of valid programs (section 3.2). Finally, the LLM is fine-tuned on these edited programs and their corresponding executions (section 3.3).

Through successive iterations, these steps bootstrap one another, forming a virtuous cycle: improvements in $p(z \mid x, \mathcal{L})$ lead to higher-quality (X, Z) pairs that better reflect the target distribution $\mathcal{S}^*$, and training on these improved pairs further refines the model toward $\mathcal{S}^*$.

3.1. Program Sampling

Given an input shape x^*, the pre-trained LLM $p(z \mid x^*, \mathcal{L})$ generates a set of $k = 10$ candidate programs $\{z_i\}_{i=1}^{k}$ through stochastic sampling (detailed in Appendix 6.2). Since the generation process is non-deterministic, multiple samples encourage exploration of diverse program candidates, allowing the selection of the best among them. Across iterations, this diversity enables the model to refine its output distribution toward higher-quality and more accurate programs. This process is analogous to self-distillation, where the model iteratively learns from its own best generations.

To select the best program, we compute the Chamfer Distance between each execution $\mathcal{E}(z_i)$ and the target shape

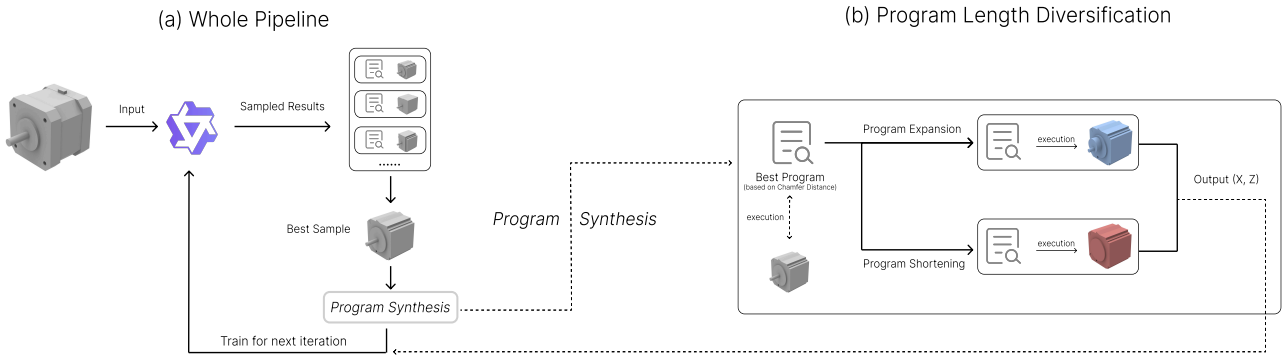


Figure 1. We show the overall pipeline in (a). At each iteration, the model first takes an input shape and samples multiple candidate programs. The selection algorithm then identifies the best program–shape pairs, which are used for training in the next iteration. (b) illustrates the details of the program length diversification process, where we perform both program expansion and shortening to create additional variants. The edited programs serve as labels Z, and their corresponding executions are treated as inputs X to form the new training dataset.

x^*, choosing the candidate with the lowest value as the optimal program z^*. If multiple candidates yield nearly identical reconstructions (diff Chamfer Distance $< 1 \times 10^{-4}$), preference is given to shorter programs to promote concise and efficient geometric representations.

3.2. Program Length Diversification

The pre-trained model may not capture the range of programs required by the new distribution $\mathcal{S}^*$, limiting its ability to represent shapes of varying complexity (see detailed discussion in Section 5.3). To address this, we synthetically expand (algorithm detailed in Appendix 6.4) or shorten (algorithm detailed in Appendix 6.5) the selected programs to create more variety. This broader length distribution enables the model to generalize across different structural complexities and thus adapt to inputs with a larger complexity variance. This process is shown in Figure 1(b).

3.3. Training Data Pairs

We perform LoRA fine-tuning on the LLM using both the extended and shortened programs as Z, paired with their corresponding executions as X (additional training details are provided in Appendix 6.3). A key advantage of this design is that in each (X, Z) pair, the shape X is the exact execution result of program Z, ensuring consistent supervision during fine-tuning. Moreover, incorporating both extended and shortened programs introduces greater variation in program lengths, which enhances the model's capacity to generalize across different levels of program complexity. This strategy maintains training stability while enriching the model's capacity to produce a wider variety of program lengths through iterative updates. We present additional experiments using alternative data pair configurations in Section 5.4.

4. Implementation

In our implementation, we use CAD-Recode [34] as the pre-trained model, which was trained on the DeepCAD dataset [49]. We use the ABC dataset [25] as the new domain $\mathcal{S}^*$, and CadQuery together with its interpreter [9] as the execution environment.

4.1. CAD-Recode

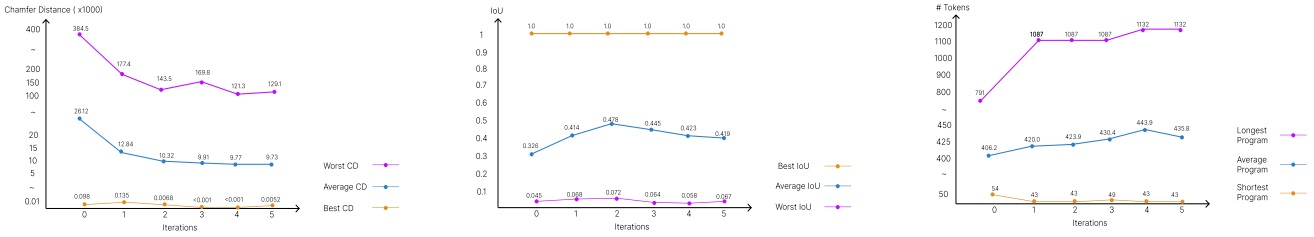
CAD-Recode [34] is originally trained on the DeepCAD dataset [49], containing only sketch–extrude CAD programs (see Appendix 6.1 architecture for details). However, the ABC dataset [25] requires more types of operations than that. So our goal is to approximate the shapes in ABC-dataset using only sketch–extrude operations instead of reconstruct the exact same shapes.

Another limitation of CAD-Recode is that it caps its output program length at 768 tokens, which is insufficient for capturing the fine geometric details of many shapes in the ABC dataset. We extend the maximum program length to 1200 tokens and apply our program diversification strategy to expose the model to longer samples during training, enabling it to gradually generate more detailed and complex programs.

4.2. Computational Cost

We use the first 15 batches from the ABC dataset, sampling 5,000 shapes from each batch, for total 75,000 shapes. However, CAD-Recode is able to produce executable programs for only 71,784 shapes, all experiments are conducted on this subset.

We use a system with four NVIDIA L40S GPUs (each with 48 GB of memory) and an AMD EPYC 7R13 CPU (24 cores, 48 threads, 2.45 GHz). Running 6 self-training



(a) Chamfer Distance across Iterations (b) IoU across Iterations (c) Program Length across Iterations

Figure 2. We compare quantitative results across iterations: (a) Chamfer Distance, (b) IoU, and (c) Program Length.

iterations takes 150 hours in total (around 25 hours per iteration). In each iteration, about 12 hours are spent on sampling programs from the dataset, 10 hours on program selection (execution, Chamfer distance computation, and length diversification), and 2 hours fine-tuning the language model for four epochs.

5. Results and Evaluations

We take shapes from the ABC dataset as input and sample point clouds from them. These point clouds are then processed through our PLLM pipeline to generate outputs. We present qualitative results by comparing our outputs with those produced by CAD-Recode (Figure 4), as well as results across different training iterations (Figure 5). We also provide quantitative evaluations of Chamfer Distance, Intersection over Union (IoU), and program length in Figure 2 and Sections 5.1, 5.2, and 5.3.

5.1. Chamfer Distance Across Iterations

We report the best, average, and worst Chamfer Distances across iterations in Figure 2(a). Each distance is computed after normalizing the predicted and input shapes to a unit bounding box (1^3) and scaling by 10^3. The best and worst scores correspond to the mean of the top 10 and bottom 10 shapes per iteration, respectively, while the average reflects the mean over all shapes. The Chamfer Distance generally decreases over the first four iterations, after which improvements plateau or slightly regress, likely due to the limited CAD operations supported by our base model, CAD-Recode (see Section 4.1).

5.2. IoU Across Iterations

Another interesting metric to consider is the IoU across iterations (Figure 2(b)), which is not directly optimized in our framework. We do not intentionally select programs with high IoU, as our objective focuses on minimizing the Chamfer Distance (CD). While IoU measures volumetric overlap, CD evaluates surface alignment between the generated and target shapes. In our results, we observe that IoU increases during the first two iterations but decreases in later ones. This behavior arises because IoU is not explicitly used as a reward signal—thus, as the model focuses more on lowering CD, it may overfit surface alignment without necessarily improving volumetric consistency.

5.3. Program Length Distance Across Iterations

We analyze how average, longest, and shortest program lengths evolve across iterations in Figure 2(c). Initially, average length increases, allowing finer shape generation. The baseline model, CAD-Recode, is limited to 768 tokens. When this cap is raised to 1200 tokens at iteration 0, program length grows slightly. From iteration 2 onward, as longer programs are added through expansion (see Section 3.2), the maximum length rises markedly, improving the model's capacity to represent detailed geometries.

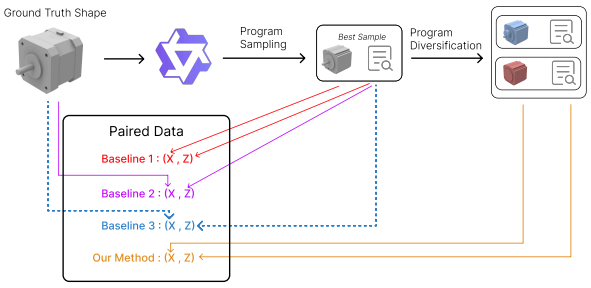


Figure 3. Overview of different baseline strategies compared in our study. The figure illustrates how each baseline constructs its (X, Z) training pairs. Baseline 1 uses the generated program and its execution; Baseline 2 uses the input shape and its best generated program; and Baseline 3 samples within each batch, selecting only the top 20% of high-performing pairs. Our proposed method further introduces program expansion and shortening to generate paired data (X, Z) that better align with the target distribution.

5.4. Experiments with Different Pseudo Label Pairs

To iteratively fine-tune the model for improved performance, we require our paired dataset to satisfy four key criteria:

1. The program represents the top-performing outputs,

Table 1. Comparison of different pseudo-label and program pairing strategies evaluated at the final iteration. Our proposed method, which uses paired synthetic programs and their executions for training, achieves the lowest Chamfer Distance and demonstrates the most consistent performance improvement across iterations.

Sampling Method	Final Average CD
Our Method	9.73
CAD-Recode	26.12
Baseline 1 (best sample, its execution)	28.24
Baseline 2 (best sample, input shape)	10.28
Baseline 3 (In Batch Sampling)	22.84

ensuring that the model shifts its distribution toward higher-quality generations.

2. The program Z, which serves as the label, can be executed to produce the shape X, providing unambiguous supervision.
3. The shape X distribution is close to the target distribution
4. The dataset introduces additional programmatic information that enhances the model's reasoning and generative ability.

Criterion (1) is automatically satisfied by the sampling stage (Section 3.1), which consistently selects the best program among all generated samples. Our method introduced in Section 3.2, which expands and shortens programs and uses the resulting diversified programs together with their executions for training, automatically satisfies criteria (2) and (4), while criterion (3) is only partially addressed.

In practice, it is impossible to satisfy all four criteria simultaneously; only paired ground-truth programs and shapes can fully meet them. For pseudo-labeling methods, certain trade-offs are inevitable. In this subsection, we discuss alternative approaches (Figure 3) that fulfill different subsets of these criteria. The results of these methods, evaluated at the final iteration, are presented in Table 1, where our proposed method achieves the best overall performance.

5.4.1. Baseline 1: (best sample, its execution) pair

The first baseline method (red line in Figure 3) trains the model using pairs of generated programs as Z and their corresponding executions as X. However, this approach actually degrades performance, as the model repeatedly observes shapes that lie outside the target distribution paired with their generated programs, preventing it from making meaningful improvements.

5.4.2. Baseline 2: (best sample, input shape) pair

The second baseline method (purple line in Figure 3) trains the model using pairs of generated programs as Z and the corresponding input shapes as X. In essence, this approach performs a self-guided search within the model, allowing it to train on its own best-available results at each iteration. This method achieves noticeable improvements; however, it compromises criterion (2), since the input shape and the program are not perfectly matched.

5.4.3. Baseline 3: In Batch Sampling

The final baseline method extends from Baseline 2 by performing sampling within each batch (blue dashed line in Figure 3). Instead of using all data for the next iteration, we select only the top 20% of samples based on performance. Thus, while the next iteration is still trained using *(best sample, input shape)* pairs, lower-quality samples are excluded, representing an improvement over the previous baseline.

However, in our experiments, we observed that this approach primarily enhances the model's performance on the best shapes. As the top-performing samples continue to improve across iterations, the remaining 80% of shapes receive no updates, resulting in little to no improvement for the lower-quality cases.

6. Conclusion

We presented PLLM, a self-training framework for unsupervised fine-tuning of large language models in CAD program synthesis. By iteratively generating, selecting, and refining pseudo-labeled CAD programs, PLLM enables model improvement without requiring paired shape–program datasets. Our approach combines knowledge distillation and search-based pseudo-labeling to bridge the gap between pre-trained CAD models and unlabeled shape data. Empirical evaluations show that PLLM outperforms the baseline CAD-Recode model in both geometric reconstruction quality and program diversity, achieving lower Chamfer Distances across iterations while maintaining valid and interpretable CAD code.

A major drawback of the pseudo-labeling approach is its computational cost. The process involves multiple iterations, each consisting of sampling, selection, and training stages. In each iteration, beyond model training, the program sampling and selection steps also require non-trivial time. This overhead reflects the inherent cost of operating without ground-truth programs.

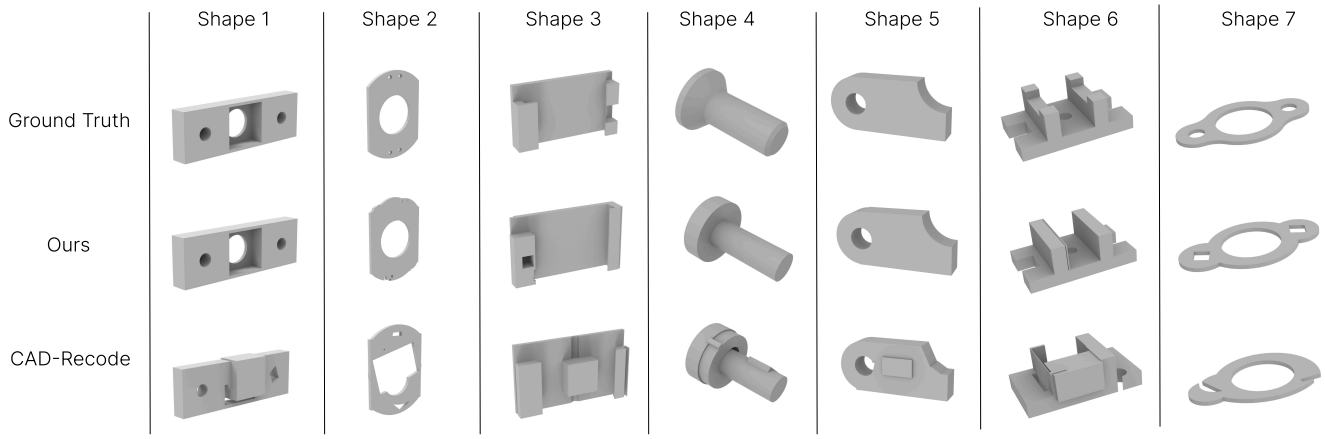


Figure 4. Comparison between our results and those produced by CAD-Recode, which correspond to the outputs from the first iteration of our framework

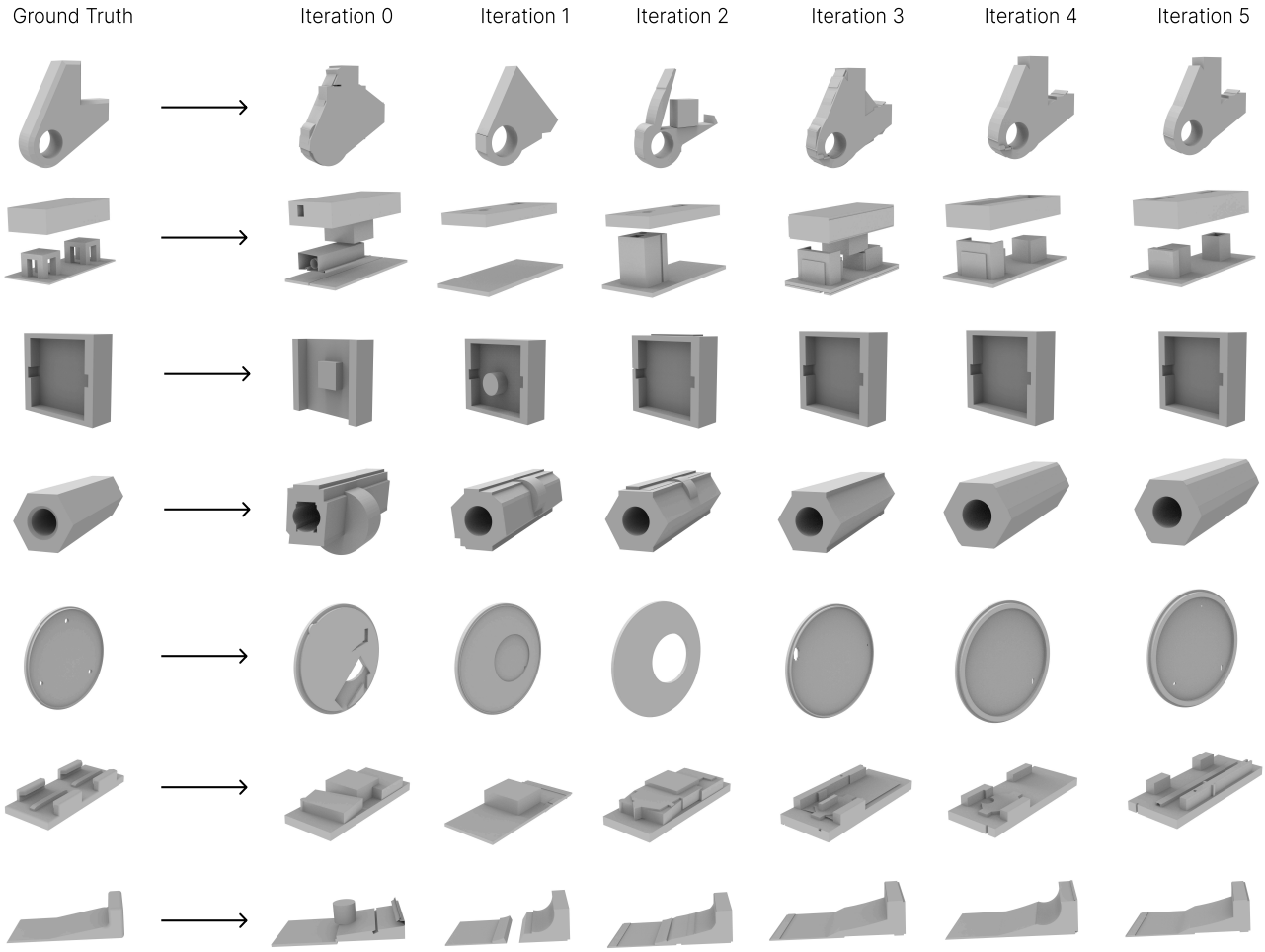


Figure 5. Results across different iterations, showing that the generated shapes gradually improve in quality as training progresses

References

[1] Sk Aziz Ali, Mohammad Sadil Khan, and Didier Stricker. Brep boundary and junction detection for cad reverse engineering. In *IEEE International Conference on Computing and Machine Intelligence (ICMI)*, 2024. 1

[2] Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew C. Gombolay. Generating cad code with vision-language models for 3d designs. *arXiv preprint arXiv:2410.05340*, 2024. arXiv:2410.05340. 2

[3] Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew C. Gombolay. Generating cad code with vision-language models for 3d designs. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025. Preprint available via IEEE Xplore (Document 10890248).

[4] Akshay Badagabettu, Sai Sravan Yarlagadda, and Amir Barati Farimani. Query2cad: Generating cad models using natural language queries. *CoRR*, abs/2406.00144, 2024. 2

[5] Pal Benko and J *et al.* Faigl. Algorithms for reverse engineering boundary representation (b-rep) solid models. Technical report, Berkeley EECS / UC Berkeley, 2001. 1

[6] Pratyush Bharadwaj, Paul Willberg, Faizan Ahmad, Adeel Ahmad, et al. Simjeb: Simulated joint engineering benchmark. https://simjeb.github.io, 2023. Accessed: 2025-07-12. 2

[7] Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018. 2

[8] Xinyun Chen, Chang Liu, and Dawn Song. Execution-guided neural program synthesis. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019. 2

[9] CadQuery developers. Cadquery — a python parametric cad scripting framework. 3

[10] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia*, 37(6), 2018. 2

[11] Elona Dupont, Kseniya Cherenkova, Anis Kacem, Sk Aziz Ali, Ilya Arzhannikov, Gleb Gusev, and Djamila Aouada. Cadops-net: Jointly learning cad operation types and steps from boundary-representations. In *arXiv preprint arXiv:2208.10555*, 2022. 1

[12] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 2

[13] Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[14] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. *arXiv preprint arXiv:2006.08381*, 2020. 2

[15] Aditya Ganeshan, R. Kenny Jones, and Daniel Ritchie. Improving unsupervised visual program inference with code rewriting families. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. 2

[16] Yandong Guan, Xilin Wang, Xingxi Ming, Jing Zhang, Dong Xu, and Qian Yu. Cad-coder: Text-to-cad generation with chain-of-thought and geometric reward. *arXiv preprint arXiv:2505.19713*, 2025. arXiv:2505.19713. 2

[17] Haoxiang Guo, Shilin Liu, Hao Pan, Yang Liu, Xin Tong, and Baining Guo. Complexgen: Cad reconstruction by b-rep chain complex generation. *ACM Transactions on Graphics (SIGGRAPH)*, 41(4), 2022. 2

[18] Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. Revisiting self-training for neural sequence generation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. 2

[19] Robert Jones, Daniel Ritchie, and Armando Solar-Lezama. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Transactions on Graphics (TOG)*, 42(4):1–13, 2023. 2

[20] Robert Jones, Shoubhik Bhat, Daniel Ritchie, and Armando Solar-Lezama. Learning to edit visual programs with self-supervision. *arXiv preprint arXiv:2406.02383*, 2024. 2

[21] R. Kenny Jones, Homer Walke, and Daniel Ritchie. Plad: Learning to infer shape programs with pseudo-labels and approximate distributions. *CVPR*, 2022. Revised version v4, 22 Mar 2022. 2

[22] Jacob Kahn, Ann Lee, and Awni Hannun. Self-training for end-to-end speech recognition. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7084–7088. IEEE, 2020. 2

[23] Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. *CVPR*, 2024. 2

[24] Muhammad Tayyab Khan, Lequn Chen, Ye Han Ng, Wenhe Feng, Nicholas Yew Jin Tan, and Seung Ki Moon. Leveraging vision-language models for manufacturing feature recognition in cad designs. *arXiv preprint arXiv:2411.02810*, 2024. 2

[25] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 2, 3

[26] Joseph George Lambourne, Karl Willis, Pradeep Kumar Jayaraman, Longfei Zhang, Aditya Sanghi, and Kamal Rahimi Malekshan. Reconstructing editable prismatic cad from rounded voxel models. In *SIGGRAPH Asia Conference Papers*, 2022. 2

[27] Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. Cad-llama: Leveraging large language models for computer-aided design parametric 3d model generation. *arXiv preprint arXiv:2505.04481*, 2025. 1, 2

[28] Yujia Liu, Anton Obukhov, Jan Dirk Wegner, and Konrad Schindler. Point2cad: Reverse engineering cad models from 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1540–1550, 2024. 2

[29] Liane Makatura, Michael Foshey, Bohan Wang, Felix Hähnlein, Pingchuan Ma, Bolei Deng, Megan Tjandrasuwita, Andrew Spielberg, Crystal Elaine Owens, Peter Yichen Chen, Allan Zhao, Amy Zhu, Wil J. Norton, Edward Gu, Joshua Jacob, Yifei Li, Adriana Schulz, and Wojciech Matusik. How can large language models help humans in design and manufacturing? *arXiv preprint arXiv:2307.14377*, 2023. arXiv:2307.14377. 2

[30] Dimitrios Mallis, Ahmet Serdar Karadeniz, Sebastian Cavada, Danila Rukhovich, Niki Foteinopoulou, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-assistant: Tool-augmented vllms as generic cad task solvers. *ICCV*, 2025. arXiv:2412.13810. 1

[31] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159, New York City, USA, 2006. Association for Computational Linguistics. 2

[32] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016. 2

[33] Felix Ocker, Stefan Menzel, Ahmed Sadik, and Thiago Rios. From idea to cad: A language model-driven multi-agent system for collaborative design. *arXiv preprint arXiv:2503.04417*, 2025. 2

[34] Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-recode: Reverse engineering cad code from point clouds. *ICCV*, 2025. 1, 2, 3

[35] H. Scudder. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3):363–371, 1965. 2

[36] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[37] Gopal Sharma, Difan Liu, Evangelos Kalogerakis, Subhransu Maji, Siddhartha Chaudhuri, and Radomír Měch. Parsenet: A parametric surface fitting network for 3d point clouds. In *Proc. European Conference on Computer Vision (ECCV)*, 2020. 2

[38] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, 2014. 2

[39] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063, 2000. 2

[40] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. *arXiv*, 2019. Presented at ICLR 2019. 2

[41] Mikaela Angelina Uy, Yen yu Chang, Minhyuk Sung, Purvi Goel, Joseph Lambourne, Tolga Birdal, and Leonidas Guibas. Point2cyl: Reverse engineering 3d objects from point clouds to extrusion cylinders. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[42] Aayush Vardhan, Rishabh Sahay, Abhishek Pandey, et al. Mcm: A mechanical components dataset for geometric deep learning. *arXiv preprint arXiv:2306.09053*, 2023. 2

[43] Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. Text-to-cad generation through infusing visual feedback in large language models. *ICML*, 2025. 1, 2

[44] Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. Cad-gpt: Synthesising cad construction sequence with spatial reasoning-enhanced multimodal llms. *arXiv preprint arXiv:2412.19663*, 2024. 1

[45] Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. Cad-gpt: Synthesising cad construction sequence with spatial reasoning-enhanced multimodal llms. *AAAI*, 2025. Accepted at AAAI 2025 (Vol. 39, No. 8, pp. 7880–7888). 2

[46] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. 2

[47] Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics (TOG)*, 40(4):1–21, 2021. 2

[48] Q. Wu, K. Xu, and J. Wang. Constructing 3d csg models from 3d raw point clouds. *Computer Graphics Forum*, 37 (5), 2018. 2

[49] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. *ICCV*, 2021. 1, 2, 3

[50] Sifan Wu, Amir Khasahmadi, Mor Katz, Pradeep Kumar Jayaraman, Yewen Pu, Karl Willis, and Bang Liu. Cad-llm: Large language model for cad generation. In *NeurIPS Workshop on Machine Learning for Creativity and Design*, 2023. Workshop, NeurIPS 2023, New Orleans, LA. 2

[51] Jingwei Xu, Zibo Zhao, Chenyu Wang, Wen Liu, Yi Ma, and Shenghua Gao. Cad-mllm: Unifying multimodality-conditioned cad generation with mllm. *arXiv preprint arXiv:2411.04954*, 2025. 2

[52] Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl D. D. Willis, and Daniel Ritchie. Inferring CAD modeling sequences using zone graphs. *arXiv preprint arXiv:2104.03900*, 2021. Submitted 30 Mar 2021; revised 20 Apr 2021. 1, 2

[53] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, Massachusetts, USA, 1995. Association for Computational Linguistics. 2

[54] Licheng Zhang, Bach Le, Naveed Akhtar, Siew-Kei Lam, and Tuan Ngo. Large language models for computer-aided design: A survey. *arXiv preprint arXiv:2505.08137*, 2025. arXiv:2505.08137. 2

[55] Shengdi Zhou, Tianyi Tang, and Bin Zhou. Cadparser: A learning approach of sequence modeling for b-rep cad. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, 2023. 1

[56] Haotian Zhu, Guyue Zhang, Zekun Hao, Zipeng Gao, Hengyang Zhao, Yifei Ren, Qingyang Wu, Xuan Luo, Jiahao Zhang, Masha Shugrina, and Xinchen Yan. Text2cad: A large-scale benchmark for language-driven cad modeling. *arXiv preprint arXiv:2409.17106*, 2024. 2

[57] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, pages 1433–1438, 2008. 2

[58] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin D. Cubuk, and Quoc V. Le. Rethinking pre-training and self-training. *arXiv preprint arXiv:2006.06882*, 2020. 2

PLLM: Pseudo-Labeling Large Language Models for CAD Program Synthesis

Supplementary Material

6.1. CAD-Recode

CAD-Recode addresses the task of CAD reverse engineering by mapping a 3D input point cloud to executable CAD code. The overall pipeline comprises two primary components: (i) a point-cloud encoder ("point projector") which downsamples the input point cloud, applies positional encoding and a shallow feed-forward network, and produces a sequence of feature embeddings; and (ii) a language-model decoder, which is a small-scale pretrained large-language model (e.g., Qwen2-1.5B) adapted via a lightweight projection layer that accepts the point-cloud embeddings and generates CAD code (in Python, using the CadQuery library) as output.

Training is done end-to-end on a large synthetic dataset of over one million program–shape pairs: each pair comprises a point cloud sampled from executing a ground-truth CAD script and the corresponding Python source code that produced it. Teacher-forcing is used during training to minimise token-level negative log-likelihood. At inference time multiple candidate programs are decoded; among these the one whose execution yields a point-cloud representation most closely matching the input (measured via Chamfer Distance) is selected as the final output. We show the pipeline of CAD-Recode in Figure 6.

The CAD code is expressed in the CadQuery Python scripting language, allowing interpretable, modular, and directly executable CAD representations rather than opaque numeric vectors. The dataset is procedurally generated to cover a broad variety of sketch-and-extrude operations, providing a scalable and controlled training supply.

6.2. Program Sampling

Given an input shape, we sample *10 candidate programs* from the LLM using stochastic decoding to encourage diversity while maintaining structural consistency. Specifically, we apply nucleus sampling with `top_p = 0.8` and `top_k = 30`, and set the temperature to `1.2` to introduce moderate randomness in token generation. This setup ensures that sampled programs differ in operation order, parameterization, or minor geometric variations, yet remain semantically close to the input shape. In other words, the generated candidates are diverse but not divergent—they explore multiple plausible reconstructions without deviating excessively from the shape's geometry or intended design semantics.

6.3. LoRA Fine-Tuning

We fine-tune the pretrained *CAD-Recode* model using Low-Rank Adaptation (LoRA) to specialize it for longer and more complex program generation conditioned on 3D point clouds. The original CAD-Recode architecture supports a maximum token length of 768. To encourage the model to produce longer and more expressive programs, we extend this limit to 1200 tokens, effectively expanding the language capacity of the decoder while maintaining the same point cloud resolution.

Our fine-tuning strategy preserves the model's ability to output syntactically valid and executable `CadQuery` code. To achieve this, we apply LoRA updates only to the *middle transformer layers* (layers 4–8), which primarily govern high-level reasoning and compositional planning, while keeping the bottom layers (responsible for tokenization, geometric grounding, and syntax formation) frozen. This design allows the model to adapt its semantic understanding of CAD programs without disrupting the stable syntax-generation capability of the pretrained backbone. The LoRA configuration uses rank $r = 8$, $\alpha = 32$, and dropout $p = 0.1$, applied to both the self-attention and MLP projections within the selected layers.

6.4. Program Expansion

In CadQuery, a *workspace* corresponds to a local coordinate frame used for sketching and feature operations (e.g., `extrude`, `cut`, `union`). And each workspace encapsulates a self-contained sequence of modeling steps that contribute to the final solid geometry.

The base CAD-Recode output typically instantiates one or two workspaces. We iteratively expand the program by either (i) spawning a new workspace (creating a new Workplane with its own procedurally generated sketch and feature operations), or (ii) appending additional operations to an existing workspace. We cap the total number of workspaces at $W_{\max} = 5$ to encourage modular but compact program structure. Each iteration adds either 1 workspace with 2 CAD operations, or max 5 new operations but without new workspace created. This ensures the program length grows gradually across iterations while remaining syntactically valid and executable.

6.5. Program Shortening

We shorten CadQuery programs by removing all top-level boolean calls `union`, `cut`, and `intersect` from the expression, and leave the remainder intact. The procedure is a single left-to-right pass over the expression that tracks (i)

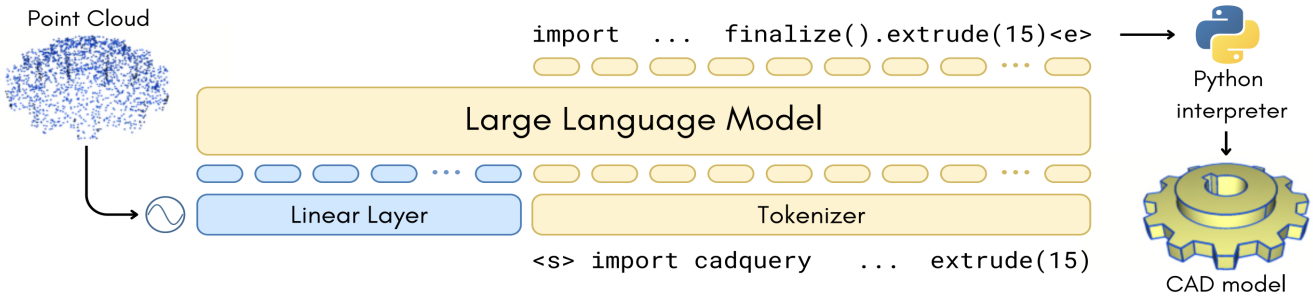


Figure 6. We show the pipeline of CAD-Recode, image from the orginal work.

the current parenthesis depth and (ii) whether the cursor is inside a quoted string (with escape handling). Whenever the cursor is not inside a string and the depth is zero, we test for one of the boolean operator prefixes; upon a match, we parse and skip the entire balanced-call payload (its matching closing parenthesis), correctly handling nested parentheses and quoted substrings. After collecting all matched call intervals, we rebuild the expression by dropping those ranges and keeping everything else unchanged. This approach guarantees that only top-level boolean edits are removed while nested calls and string literals are preserved.