Title:
Iterative Generation of Feature-Based CAD Models Using an LLM With Domain-Specific Constraints

Authors:
Ammon I. Hepworth, ammon.hepworth@gmail.com, Southern Virginia University
John M. Gauch, jgauch@uark.edu, University of Arkansas

Keywords:
AI CAD, LLM CAD, text to parametric CAD

DOI: 10.14733/cadconfP.2026.7-13

Introduction:
Creating detailed 3D computer-aided design (CAD) models from text-based descriptions is an exciting notion. It not only has potential for much faster model creation, but it also democratizes CAD model building, allowing less experienced builders to bring their ideas to life without having to go through the tedious steps of creating a model from the ground up in a complex CAD system.

A substantial amount of research has explored the generation of 3D geometry from natural language using deep learning. However, most of these approaches use voxels, point clouds, and meshes, which are not parametric CAD models and are not suitable to be used for manufacturing [6, 9, 14, 18]. More recent approaches generate boundary representation (B-Rep) to directly create manufacturable solid models, including SolidGen [2] and BrepGen [16]. However, these approaches do not generate editable, feature-based parametric models and are thus less useful for engineering and manufacturing companies that require iterative design.

Starting with sketches [11], and then extrusions, researchers have developed custom deep neural networks capable of generatively outputting a sequence of CAD operations to directly build native solid CAD models, including the oft cited DeepCAD [15]. Since then, other researchers created neural networks which also generate CAD models from a sequence of CAD commands which are trained on the DeepCAD dataset including Text2CAD [5], and CAD-GPT [13].

Recent research has shown that general large language models (LLMs), like ChatGPT, can generate models for feature-based parametric mechanical CAD systems by producing a script using the application programming interface (API) to build the CAD model programmatically [4, 7, 8, 12, 17]. Fine tuning and using a domain specific language can improve model generation [3, 12], but these approaches still only produce simple CAD models with basic features. Recently, a few startup companies, including Adamcad, NeoCAD, and Zoo, have emerged that generate parametric, manufacturable from text prompts, but are in very early stages [1, 10, 19].

Although general LLMs can produce working code in many domains, generating scripts with a CAD API to produce parametric CAD models is more difficult. CAD models generated by script need to be written in a particular way to build valid geometry, and even scripts that are syntactically correct may cause execution errors.

The research to date has also focused on building a complete model from a single text prompt, as opposed to an iterative approach that uses multiple text prompts to improve and refine models with a feedback loop. An iterative approach is more challenging than previous approaches due to the need for the user to inspect intermediate results and give further instruction after each iteration. The LLM needs to be aware of previous prompts and resulting code to make changes based on new prompts.

This paper presents two complementary, but independent contributions: 1) a method of using rules, or domain-specific constraints (DSCs), to guide a general LLM on how to generate scripts to build

models for feature-based parametric mechanical CAD systems; and 2) a framework for an iterative text-based system that produces parametric CAD models, along with an architecture that implements it using a browser-based user interface.

Method:

Domain Specific Constraints

Using a general LLM (e.g. ChatGPT or Gemeni) to produce a script that generates a CAD model for a feature-based parametric CAD system (e.g. SolidWorks, OnShape, FreeCAD) from a prompt is simple, but results are highly limited. As shown in the results section, only the most basic scripts can be run without errors, even though the code is syntactically correct. This is because CAD APIs are semantically fragile, requiring very specific feature ordering and valid topological references. The CAD API documentation is often limited, only discussing syntax, but not semantics. Errors often arise during script execution, so they are difficult to anticipate from code alone. Because LLMs are trained on API documentation and script examples, not failure modes, they make mistakes because they lack awareness of how to robustly build a model using the API. Complex models have more features and are thus more likely to fail during execution time.

To overcome some of these challenges a set of rules, called domain-specific constraints (DSCs), are provided to the LLM, in addition to the user prompt, as additional context to guide a general LLM on how to generate scripts in the best way. This effectively provides the necessary additional information for the LLM to effectively generate the script, which is missing in the API documentation and code examples that it likely has already been trained on. The DSCs may include constraints related to general modeling, specific features, primitives, sketching, topology references, unions/subtractions, saving/exporting geometry, and model completeness.

Appropriate constraints are developed through analysis of the fragile and ambiguous aspects of the specific CAD API and through testing various prompts that generate models of differing complexity. As models fail, more rules are added to enable the DSCs to guide the LLM to reliably create CAD models. Prompting the LLM for a set of constraints based on documentation ambiguity is a good start; however, the LLM is not likely to produce an effective DSC without testing with models of various descriptions.

Iterative text-based CAD modeling

Design is iterative. Therefore, an effective text-based modeling system requires a means to iterate on CAD models during design. The method presented here uses a simple user interface which includes a live CAD model viewer to display updates, as well as a field to input text. The viewer includes a means to zoom, rotate, and pan the geometry, enabling the user to effectively analyze the model after each iteration.

In addition to the user interface, an execution engine takes the prompts from the user and builds a more complete prompt to send the LLM using the predefined DSCs, the previous prompts in the design session, the code generated to produce the current model, and the current prompt entered by the user. This aggregated prompt is sent to the LLM via API call and a script to build the model is returned. This code is executed by calling the CAD system in headless mode with the script as input. This generates a triangulated model to be sent to the user interface for visualization in the viewer. The script execution also generates a native parametric model which the user can download to edit it in the native CAD platform. These steps are repeated with each additional prompt in the design session to update the interactive view and associated model.

Implementation:

The methods described above were implemented using FreeCAD as the parametric CAD system, its associated API for scripting, and OpenAI GPT-5.2 as the LLM to generate scripts.

Domain Specific Constraints

The DSCs for this implementation were developed specifically for GPT-5.2 to generate FreeCAD API scripts to create CAD models. The initial DSCs were created by prompting GPT-5.2 for the constraints it should include, based on where it is expected to see failures. Then, the DSCs were tested on various

prompts leading to failed script execution in FreeCAD. The DSCs were revised based on how to avoid the specific error that was generated. In some cases, the DSC became over constrained, requiring a revision. GPT-5.2 was often helpful in making suggestions on what to revise as well. Iterations continued until the DSC was determined "good enough" based on incremental improvements. It is by no means all-inclusive but does represent a significant improvement in the generation of specific types of models as shown in the results section.

The DSCs begin with instructions on how the code builds a parametric PartDesign model with an editable feature tree, including clarification that if a rule is violated, the LLM must correct the output before responding. The specific constraints related to geometry creation are then spelled out in different sections. Note that they are called rules in this context (instead of constraints) to avoid confusion with geometric constraints. The first set are general environment rules which provide guidelines on how to build a model generally. Here is an example of how this section is formatted:

```
General environment rules:
    - Use PartDesign and Sketcher only.
    - Create exactly one PartDesign::Body.
    ...
```

The subsequent constraints follow a similar pattern and include allowed features, primitive preference rules, sketch rules, feature direction rules, hole/cut rules, and feature sequencing rules. Next, are rules to ensure robust export of an STL file (to be used for visualization) and a native FreeCAD (.FCStd) file (to be used for native CAD file editing, if desired by the user). The final DSCs are output requirements to ensure only code is generated and that the LLM generates the entire model as described. Each follows a similar structure to the general environment rules example show above.

Iterative text-based CAD modeling

Software was written to implement the DSC method and iterative text-based CAD modeling methods into a usable web application that effectively generates FreeCAD models based on user text prompts. The user interface is shown in Fig. 1.

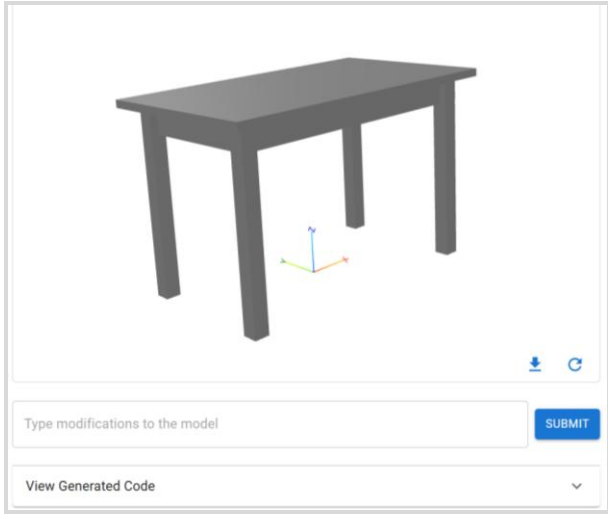

Fig. 1: Web UI of the text-based iterative design application.

The user interface is simple. It includes a text input field with a 3D viewer which shows the updates of the model after each prompt. It allows users to pan, zoom, and rotate the model. The application also includes an output of the code that was generated for the user to interrogate if more details on the construction of the model are desired. A download button is also there to allow the user to download the generated FreeCAD part so they can do further refinements in the native CAD system, as needed. The prompt used to create the model shown in Fig. 1 was simply: "table".

Iterations to the model are made by entering a description of how to update the model in the field which states: "Type modifications to the model". For example, if a user wanted to modify the table created in Fig. 1 to be round, they could enter "make the table round". The model would then update to having a circular top instead of a rectangular one. A user could then enter "make the legs 30% shorter" to make the legs shorter. Iterations to the model can continue to be made until the model is satisfactory to the user.

The architecture of the application consists of the following components show in Fig. 2:

- **Web UI:** user interface for prompt input and model/code visualization
- **App server:** the interface service between the Web UI and the rest of the backend; constructs the API calls using the DSCs
- **Execution service:** runs the script in headless FreeCAD and produces the STL and FCStd files
- **LLM:** the Open AI API service to execute prompts and produce scripts
- **Model store:** stores the STL and FCStd files

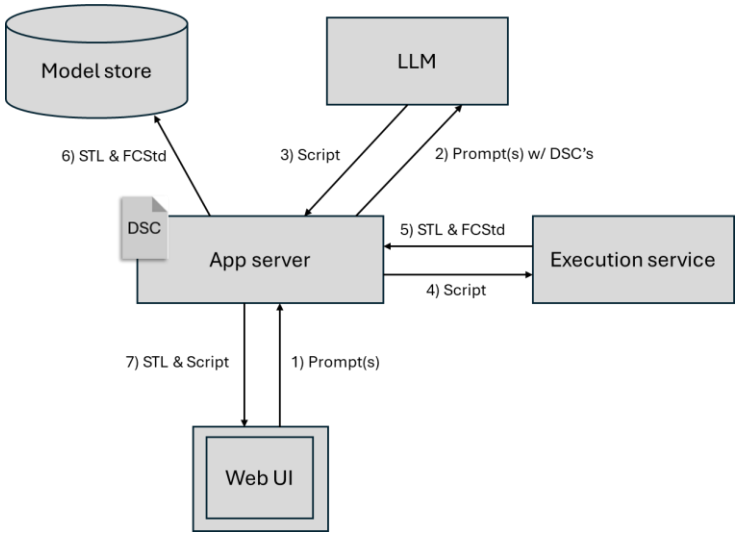

Fig. 2: Architecture of the application.

The workflow to generate a visual model from a prompt works as follows (also shown in Fig. 2):

1. The user inputs a prompt describing the object they want to build and is sent to the app server on submission.
2. The new prompt is constructed by combining the DSCs with the user prompt, previous prompts, and current model script and is sent to the LLM as an Open AI API call.
3. The LLM sends back the generated script to the app server.
4. The app server sends the script to the execution service to generate the model.
5. Both the STL and FCStd files are sent back to the app server.
6. The app server saves the STL and FCStd files in the model store.
7. The STL and script files are sent to the Web UI for visualization.

When the user clicks the download button, the app server pulls the FCStd file from the model store and sends it to the Web UI for download.

Results:
The software described in the implementation was used to test and quantify the improvement in reliability of LLM model generation using DCSs. Two tests were run with a series of different model descriptions used as inputs. The first test compared the results of models generated with and without the DSCs for primitive geometry described with a single word (e.g. cone, cube, etc.). For the no-DSCs run to give the appropriate outputs, basic additional context (beyond the prompt) was included to the LLM to tell it to generate a FreeCAD 1.0 Python code that builds a parametric PartDesign model with an editable feature tree, to respond with just code, and to produce STL and FCStd files.

The results of the first test are in Tab. 1 and show that only 50% of the no-DSCs generated models were successfully created, while 83% of the DSCs-generated models were created successfully. An additional note is that the failures in the no-DSCs versions produced execution errors, while the single failure in the DSL model created the torus, but did so incorrectly (it was cut in half). The visual results of these generated primitives (with DCSs) are shown in Fig. 3, including the incorrectly created torus. This test clearly shows a significant improvement in model generation using DSCs, even in the most basic models.

Test Case	***No DSCs***	***DSCs***
Cone	Pass	Pass
Cube	Fail	Pass
Cylinder	Fail	Pass
Ellipsoid	Pass	Pass
Sphere	Pass	Pass
Torus	Fail	Fail
Percent Passed	**50%**	**83%**

Tab. 1: DSC vs No-DSC primitive model generation test.

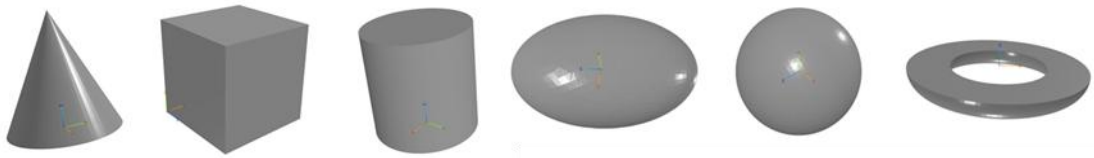

Fig. 3: Primitive models generated with DSCs.

The second test consisted of ten models, which had prompts that included detailed descriptions of each model. Each description had three distinct sections (separated by semi-colons) that provided additional details. For example, the prompt of the electronics enclosure base is as follows: “Open-top rectangular enclosure with outer dimensions 120 × 80 × 30 mm and uniform wall thickness 3 mm; four cylindrical bosses on the interior floor, each 10 mm outer diameter, positioned 10 mm from each interior corner; the bosses have a concentric 4 mm through-hole.” The model generated from this prompt is shown in Fig. 4.

Tab. 2 shows the results of the second test, which generated more complex models. The percentages shown in the table are defined as the percentage of the model that was created, based on how many of the three sections of the prompt the script produced. If the features were interpreted as correct via visual inspection, the feature set was deemed valid.

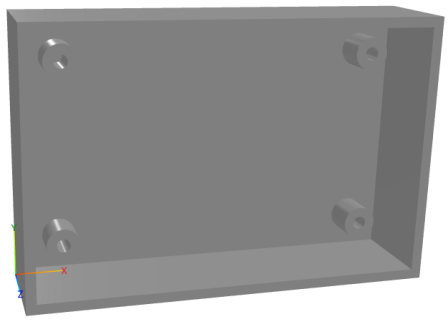

Fig. 4: Electronics enclosure base test.

Test Case	*No DSCs*	*DSCs*	*Iterative DSCs*
L-Bracket	0%	67%	67%
Stepped Shaft With Keyway	0%	67%	67%
Electronics Enclosure Base	0%	100%	100%
Mounting Plate With Counterbores	0%	100%	100%
Hollow Cylinder With Radial Holes	0%	100%	100%
Simple Spur Gear	0%	100%	67%
Rectangular Block With Slot and Cross-Holes	0%	67%	100%
Cylindrical Hub With Bore and Bolt Circle	0%	67%	67%
Rectangular Frame With Cutout and Corner Holes	0%	100%	100%
Base Block With Pocket and Vertical Hole	0%	100%	100%
Average	**0%**	**87%**	**87%**

Tab. 2: Results of the complex model test.

In the tests shown in Tab. 2 using GTP-5.2 without DCSs did not generate any script that produced a valid model. Meanwhile, both the other methods that used DSCs were able to generate most of the models correctly. Overall, the tests shown in Tab. 1 and 2 indicate that DSCs not only improve model generation reliability but enable parametric model generation for all but the most basic CAD models.

Conclusion:

This paper shows that using appropriate domain-specific constraints in the generation of scripts for parametric CAD models using an LLM significantly improves model generation reliability. By providing rules that express modeling semantics, the LLM can generate CAD modeling scripts that produce valid geometry without execution errors. The results suggest that using DSCs with a general LLM is an effective alternative to fine-tuning or domain-specific languages to improve model generation reliability. The iterative, text-based modeling approach presented enables users to build models incrementally. In the tests presented, model generation reliability appears to be mostly driven by domain-specific constraints rather than iterative modeling. However, the visual inspection interface implemented in the iterative modeling approach enables the user to incrementally identify and correct errors by modifying prompts. This has the potential to improve model reliability in more open-ended design situations that are not defined from the start. More research is needed to quantify the benefits of the iterative approach in model generation reliability. The current implementation presented is limited to FreeCAD with GPT-5.2 to create models with limited feature sets. Future research should focus on generalization across more CAD platforms as well as generating models with more complex feature sets. Automation of domain-specific constraints using an LLM feedback loop may be a viable approach to accomplish that task.

References:

[1] AdamCAD. https://www.adamcad.com.

[2] Jayaraman, P. K.; Lambourne, J. G.; Desai, N.; Willis, K. D. D.; Sanghi, A.; Morris, N. J. W.: SolidGen: An autoregressive model for direct B-rep synthesis, Transactions on Machine Learning Research, 2023.

[3] Jones, B. T.; Zhang, Z.; Hähnlein, F.; Ahmad, M.; Kim, V.; Schulz, A.: A solver-aided hierarchical language for LLM-driven CAD design, Computer Graphics Forum, Proceedings of Eurographics, 2025. https://doi.org/10.1111/cgf.70250

[4] Kapsalis, T.: CADgpt: Harnessing natural language processing for 3D modelling to enhance computer-aided design workflows, arXiv preprint arXiv:2401.05476, 2024.

[5] Khan, M. S.; Sinha, S.; Sheikh, T. U.; Stricker, D.; Ali, S. A.; Afzal, M. Z.: Text2CAD: Generating sequential CAD models from beginner-to-expert level text prompts, Advances in Neural Information Processing Systems (NeurIPS), 2024.

[6] Klokov, R.; Boyer, E.; Verbeek, J.: Discrete point flow networks for efficient point cloud generation, arXiv preprint arXiv:2007.10170, 2020. https://doi.org/10.1007/978-3-030-58592-1_41

[7] Li, X.; Sun, Y.; Sha, Z.: LLM4CAD: Multi-modal large language models for three-dimensional computer-aided design generation, Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE), 2024. https://doi.org/10.1115/DETC2024-143740

[8] Makatura, L.; Foshey, M.; Wang, B.; Hähnlein, F.; Ma, P.; Deng, B.; Tjandrasuwita, M.; Spielberg, A.; Owens, C. E.; Chen, P. Y.; Zhao, A.; Zhu, A.; Norton, W.; Gu, E.; Jacob, J.; Li, Y.; Schulz, A.; Matusik, W.: How can large language models help humans in design and manufacturing?, Harvard Data Science Review, 2023. https://doi.org/10.21428/e4baedd9.745b62fa

[9] Nash, C.; Ganin, Y.; Eslami, S. M. A.; Battaglia, P.: PolyGen: An autoregressive generative model of 3D meshes, Proceedings of the International Conference on Machine Learning (ICML), 2020.

[10] NeoCAD. https://www.neocadsrl.com.

[11] Seff, A.; Ovadia, Y.; Zhou, W.; Adams, R. P.: SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design, arXiv preprint arXiv:2007.08506, 2020.

[12] Sun, Y.; Li, X.; Sha, Z.: Large language models for computer-aided design fine tuned: Dataset and experiments, Journal of Mechanical Design, 147(4), 2025. https://doi.org/10.1115/1.4067713

[13] Wang, S.; Chen, C.; Le, X.; Xu, Q.; Xu, L.; Zhang, Y.; Yang, J.: CAD-GPT: Synthesising CAD construction sequence with spatial reasoning-enhanced multimodal LLMs, arXiv preprint arXiv:2412.19663, 2024. https://doi.org/10.1609/aaai.v39i8.32849

[14] Wu, J.; Zhang, C.; Xue, T.; Freeman, W. T.; Tenenbaum, J. B.: Learning a probabilistic latent space of object shapes via 3D generative adversarial modeling, Advances in Neural Information Processing Systems (NeurIPS), 2016.

[15] Wu, R.; Xiao, C.; Zheng, C.: DeepCAD: A deep generative network for computer-aided design models, Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2021. https://doi.org/10.1109/ICCV48922.2021.00670

[16] Xu, X.; Lambourne, J. G.; Jayaraman, P. K.; Wang, Z.; Willis, K. D. D.; Furukawa, Y.: BrepGen: A B-rep generative diffusion model with structured latent geometry, ACM Transactions on Graphics, Proceedings of SIGGRAPH, 2024. https://doi.org/10.1145/3658129

[17] Yuan, Z.; Lan, H.; Zou, Q.: 3D-PreMise: Can large language models generate 3D shapes with sharp features and parametric control?, arXiv preprint arXiv:2401.06437, 2024.

[18] Zhou, L.; et al.: 3D shape generation and completion through point voxel diffusion, Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2021. https://doi.org/10.1109/ICCV48922.2021.00577

[19] Zoo.dev. https://zoo.dev.