Exploratory Analysis of Generative AI in Automated CAD Design of Mechanical Parts

Shubham Yadav[1*], Sumit Kumar[1], and A.V. Muley[2]

[1] Department of Mechanical Engineering, Netaji Subhas University of Technology (formerly NSIT), New Delhi, India
[2] Faculty of Department of Mechanical Engineering, Netaji Subhas University of Technology (formerly NSIT), New Delhi, India
*shubham35325@gmail.com

Abstract. Modern manufacturing basically relies on CAD, yet building precise 3D models is still manual grind process which usually demands a lot of expertise. While Generative AI is beginning to address this through Text-to-3D, current models often prove insufficient for practical engineering. They tend to produce static meshes (like PLY or OBJ files) that look fine visually but lack detail and parametric history required for iterative design. This paper introduces CADgen an automated framework that leverages Large Language Models (LLMs) specifically speaking Google Gemini 2.5 Flash to generate editable, parametric OpenSCAD scripts from natural language prompts. We evaluate the system at range of complexity levels starting with simple structural primitives and moving up to mathematically complex mechanical parts. The results were mixed. While the model pretty much achieves a near 100% success rate for basic structural elements, it started to struggle with complex assemblies. That said, we identified that keeping a human-in-feedback-loop architecture significantly mitigated errors. It suggests that while LLMs may not yet be ready for fully autonomous operation, they demonstrate clear viability as a collaborative framework for engineering design. This study highlights the transformative role of LLMs in design workflows.

Keywords: Generative AI, Computer-Aided Design (CAD), OpenSCAD, Large Language Models (LLMs), 3D Modeling.

1 Introduction

The product development lifecycle, from automotive chassis design to aerospace components, relies heavily on Computer-Aided Design (CAD) [1]. These digital blueprints serve as the source of truth for simulation (FEA/CFD) and manufacturing (CAM). However, the interface for creating these models has remained largely unchanged for decades; a manual, visually intensive process requiring significant human expertise. This creates a bottleneck where design iteration is limited by the speed at which engineers can manipulate complex GUI-based software. The recent advancements in Generative AI offers potential solutions. The use of artificial intelligence (AI) has expanded rapidly across numerous domains, including finance [2], automotive systems [3], healthcare, cybersecurity [4], and beyond. This growth

V. Kumar et al. (eds.), *Proceedings of the International Conference on Advanced Design, Manufacturing, and Sustainable Energy Systems (ICADMSES 2026)*, Atlantis Highlights in Engineering 45,
https://doi.org/10.2991/978-94-6239-750-7_76

has transformed how we approach automation and decision-making. Among the most significant developments in AI is the emergence of generative AI (GenAI), which emphasizes the creation of new content and solutions rather than solely analyzing existing data. GenAI has unlocked unprecedented opportunities, ranging from the generation of realistic text and imagery to the automation of complex design workflows. Models capable of generating images (Midjourney [5], Nano-Banana) and code (ChatGPT, Claude,[6] GitHub Copilot [7]) have revolutionized their respective fields. However, the application of AI to mechanical design faces a unique data representation challenge: the conflict between Meshes and Boundary Representations (B-Reps). This paper addresses this gap by proposing a Code-as-Policy approach. Instead of predicting geometry directly (which often results in non-manufacturable meshes), we employ an LLM to generate OpenSCAD script [8] code that defines geometry through Constructive Solid Geometry (CSG) [9]. This ensures that the output is not only visual but also mathematically precise and fully editable.

2 Related Work

2.1 AI in Design: Optimization vs Creation

Historically, AI in design has been synonymous with Topology Optimization (TO). Algorithms iteratively remove material from a design space to optimize for weight and stress [10]. While this is effective, TO produces organic, lattice-like structures that are difficult to edit or manufacture using traditional subtractive methods.

2.2 The Representation Gap

Recent deep learning models like Shap-E and Point-E [11] have demonstrated Text-to-3D capabilities. However, these models output point clouds or polygonal meshes. In mechanical engineering, a mesh is often considered dead geometry because it lacks a feature tree and engineers require parametric models where dimensions (e.g. angle, length, radius, etc) can be modified after generation.

2.3 Programmatic CAD

The use of code to define geometry (e.g. OpenSCAD) bridges this gap. Definition of shape in CSG modeling technique, aligns naturally with the capabilities of LLMs trained on code repositories. Our work builds upon this hypothesis, testing the efficacy of the latest multimodal models (Gemini 2.5) in this domain.

3 Methodology

3.1 System Architecture

We developed CADgen, a monolithic Python-based framework designed to facilitate an automated design loop. The General architecture consists of three layers:

- Input Layer: Captures natural language engineering prompts.

- Inference Layer: Utilizes the Google Gemini 2.5 Flash API to interpret the prompt. A strict system instruction constrains or limits the model to act as an OpenSCAD Expert, ensuring output is valid syntax.
- Compilation Layer: Passes the generated script to the OpenSCAD compiler in headless mode to render the 3D geometry and generate preview images.

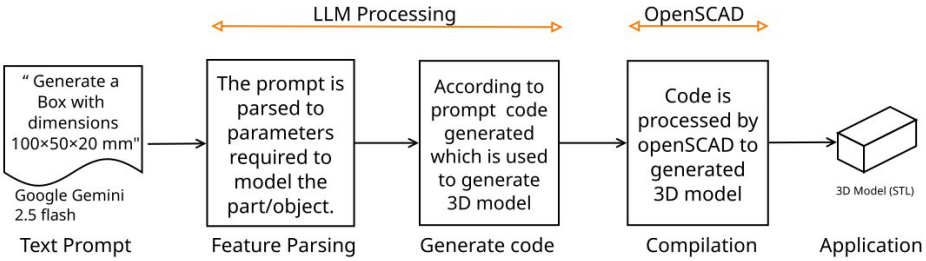

Fig. 1. Common system Workflow.

3.2 Prompt Engineering

To mitigate hallucinations(syntax errors or non-manifold geometry), we utilized a constrained prompting strategy.

- Output Constraints and Parametric Enforcement: The system prompt explicitly enforces the use of parametric variables at the top of the script (e.g. radius = 10;) rather than hard coding values. This ensures the output retains the editability advantage of CAD over meshes.
- System Level Instructions (Persona Adoption): To condition the model prior to user input, explicitly instructing it to adopt the persona of an "Expert OpenSCAD Programmer and Mechanical Engineer." This significantly reduces the likelihood of generating conversational filler or non-functional text.
- Few-Shot Prompting for Complex Logic: Zero-Shot prompting [12] often failed for complex components requiring mathematical relationships, such as involute gears. The provided example acts like a template that helps the model focus towards the correct syntax, enabling it to adapt its broader reasoning skills to the specific rules of geometric scripting without requiring specialized fine-tuning.

3.3 Taxonomy of Generative Frameworks

If you look at the current landscape of LLM-driven code generation, the literature review points towards three main Architecture. The real difference is the validation loop basically, is there a feedback mechanism, and if so, what does it actually look like?

For this study, we didn't try to tackle all three at once. Instead, we focused specifically on the first two architectures. We wanted to get a clear read on how these

simpler setups perform to establish a solid baseline, rather than jumping straight to fully autonomous systems before we understand the fundamentals.

Architecture I: Open-Loop Generation (Direct Inference) This represents the baseline Zero-Shot architecture. It operates as a unidirectional pipeline where the user provides a natural language specification, and the system synthesizes OpenSCAD script in a single pass.

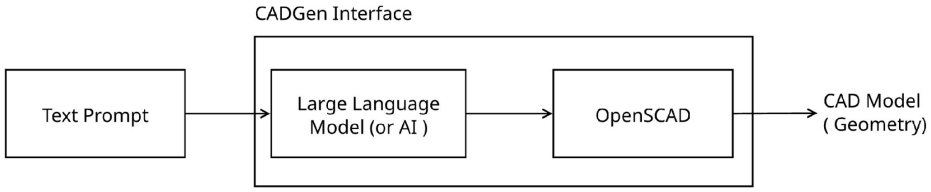

Fig. 2. Architecture I (Open-Loop)

Characteristics: This architecture lacks an error-correction layer. If the generated code contains syntax errors or geometric hallucinations, the process terminates as a failure. It serves as the basis for evaluating the raw reasoning capabilities of the Gemini 2.5 model.

Architecture II: Human-in-the-Loop (Iterative Refinement) This iterative architecture incorporates the human engineer as a critical validation node. Following the initial generation, when the script compiles and the output visually inspected. Upon encountering errors (e.g. syntax correction or dimensional inaccuracies), the user provides feedback via a refined prompt.

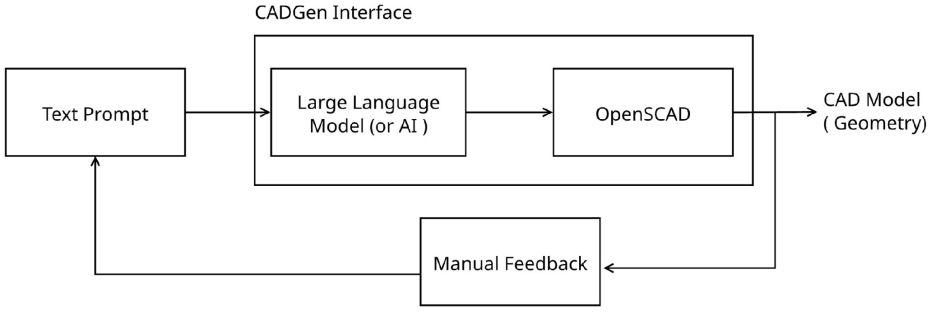

Fig. 3. Architecture II (Human-in-the-Loop)

Characteristics: The feedback cycle allows for the correction of complex spatial logic that the model cannot intrinsically verify. This framework is the primary focus of this project, demonstrating the efficacy of AI as a co-pilot rather than an autopilot.

Architecture III: Autonomous Agents (Closed-Loop Feedback) The theoretical frontier of generative design lies in the Autonomous Agentic Framework. In this architecture, the model functions as an independent agent capable of parsing compiler logs (Standard Error) and self-correcting syntax without human intervention.

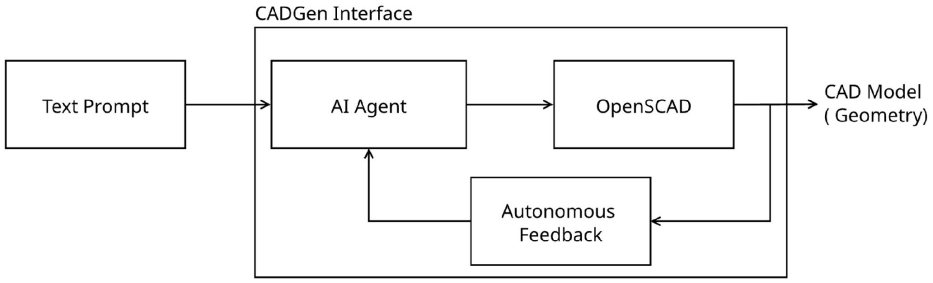

Fig. 4. Architecture III (Autonomous Agents)

Characteristics: While recognized as the ultimate goal for industrial automation, the development of a fully autonomous agentic framework is outside the scope of this paper. This research focuses on establishing the baseline efficacy of Architecture I and II to provide the foundational data necessary for future autonomous systems.

4 Experimental Setup and Assessment

To evaluate the effectiveness of LLMs in generating OpenSCAD scripts, we designed a structured set of 14 test cases with increasing levels of complexity. These experiments assess how well an LLM can generate functional CAD scripts that correctly execute within OpenSCAD environment, ensuring syntactic accuracy, geometric validity, and parametric adaptability. The evaluation follows a complexity scale from 1 (simple shape creation) to 14 (advanced complex shapes).

4.1 Environment Setup

The framework was implemented on an Ubuntu 24.04 LTS as base workstation using Visual Studio Code as the primary development environment for CADGen Interface. The core application was written in Python 3.12.3, utilizing the Google Gemini 2.5 Flash API for code generation and OpenSCAD in headless CLI mode for geometric compilation. The subprocess library managed the automated pipeline between the AI model and the compiler(here OpenSCAD). To ensure reproducibility and track iterations, the codebase was version-controlled using Git and hosted on GitHub. Model temperature was set to a low value (0.2) to prioritize deterministic, syntactically correct output.

4.2 Performance Analysis

To quantitatively evaluate the efficacy of the CADgen framework, 14 distinct test cases were executed, ranging from geometric primitives to complex mathematical components. The performance metrics recorded include Generation Latency (Time in seconds), Iterative Attempts required for a valid model, and Failure Modes. The summary of these experiments is presented in Table 1.

Table 1. Summary of Iterations, Execution Times, and Outcomes Across Test Cases

Test	Shape/Task	Iterations	Time(s)	Error Type	Outcome
1	Basic Cube	1	5.152s	None	Success
2	Pyramid(pentagonal)	1	18.26s	None	Success
3	Simple Beam	1	5.4s	None	Success
4	I-Beam	1	7.8s	None	Success
5	T-Beam	1	9.8s	None	Success
6	Channel(C)-Beam	1	10.6s	None	Success
7	Woodruff key(4×13mm)	2	27.2s	None	Success
8	Trusses(cubic)	2	61.5s	Extended Bar	Success
9	Trusses*	2	19.3s	Minor Syntax Error	Success, 2nd
10	Thread	2	48.08s	Incomplete Geometry	Failure
11	Thread*	2	23.9s	Minor Syntax Error	Success, 2nd
12	Gear	3	317.96s	Null shape, Error	Failure
13	Gear*	2	19.04s	Minor Syntax Err	Success, 2nd
14	Helical Torsion Spring	3	123.08s	Incomplete Geometry	Failure

Note: * Indicates that the prompt includes an example code with correct syntax. Execution time represents the sum of total completion time for all iterations.

4.3 Success Rate and Complexity Correlation

The data reveals a strong inverse correlation between geometric complexity and zero-shot success rates.

- Tier I: Structural Primitives (Rows 1-7): The model achieved a 100% Zero-Shot Success Rate for basic structural elements (Cubes, Beams, Channels). The average generation latency for these components was remarkably low (~9.4 seconds), indicating that the model possesses a robust internal representation of fundamental CSG operations like cube() and union().
- Tier II: Complex Assemblies (Rows 8-14): As complexity increased to include mathematical curves (Gears, Springs) or spatial arrays (Trusses), the zero-shot failure rate increased significantly. Components like the Helical Torsion Spring (Row 14) failed completely due to Incorrect Geometry, here the model struggles with 3D path extrusion without explicit mathematical guidance.

4.4 Latency Analysis and Hallucination Loops

A critical observation regarding computational cost is the disparity in generation time between successful and failed attempts.

- Successful Primitives: Converged rapidly (5s – 15s).
- Failed Complex Parts: The Gear (Row 12) represents a notable outlier, consuming 317.96 seconds before failing. This excessive latency suggests a Hallucination Loop, where the model attempts to generate overly verbose or

recursive code to solve a problem it does not fundamentally understand, eventually hitting token limits or timing out with a Null shape.

4.5 Efficacy of Iterative Refinement (Few-Shot Prompting)

The experiments marked with an asterisk (*) demonstrate the profound impact of the Human-in-the-Loop (Architecture II) workflow.

The Gear Case Study: The initial zero-shot attempt for the Gear failed after 317s with 3 iterations. However, in the second attempt (Gear*, Row-13)- where the model was provided with a syntax example - the generation time dropped drastically to 18.04s, and the output was successful.

Significance: This 94% reduction in latency and conversion from failure to success validates that providing a syntactic anchor (Few-Shot prompting) is far more computationally efficient than allowing the model to reason from scratch in a zero-shot environment.

5 Result

The experimental results validate the efficacy of the Text-to-Script methodology over traditional Text-to-Mesh approaches. While mesh-based generators produce static, non-editable geometry, our framework yields OpenSCAD scripts that retain the full feature history. Our performance analysis reveals a distinct contrast in model capability:

- Mastery over Primitives: The model achieved a 100% success rate with low latency <11s for standard structural components (Beams, cube and pyramid), demonstrating a robust internal representation of Constructive Solid Geometry (CSG) logic.
- Complexity Threshold: For complex mathematical components (e.g., Gears, Springs), the model struggled in zero-shot scenarios, often succumbing to geometric hallucinations or excessive generation latency (up to 317s).
- In-Context Learning: Crucially, we demonstrated that Few-Shot Prompting-providing a single syntactic example-served as a powerful retrieval cue. This technique reduced generation time by 94% (in the Gear case study) and converted failure states into success, validating the Human-in-the-Loop (Architecture II) workflow as the currently optimal approach for AI-assisted design.

6 Discussion

We define a new failure mode specific to this domain: Geometric Hallucination. Unlike syntax errors (which stop compilation), these are valid scripts that produce physically impossible geometry (e.g., floating parts or non-manifold intersections). These errors suggest that LLMs treat CAD generation as a linguistic task rather than a spatial one.

The choice of OpenSCAD proved critical. By outputting code, the AI's errors were transparent and debuggable. If a generated beam was too short, the user could simply edit the length variable in the script. This confirms that a Human-in-the-Loop workflow is currently the most viable path for AI-assisted engineering. Despite its successes, the current I and II architecture have to struggle with:

- Spatial Disorientation: In the absence of visual feedback, the model occasionally loses track of the global coordinate system, leading to floating or unnecessary parts in multi-component assemblies (as observed in the Truss extension experiment).
- Library Hallucination: The model attempts to call functions from external libraries (e.g., BOSL2) using outdated or incorrect syntax unless explicitly provided with documentation in the context window.
- Geometric Failures: Certain classes of geometry involving complex path extrusions, such as the Helical Torsion Spring, remained unsolvable within the current prompt constraints, resulting in invalid geometry.

7 Conclusion

This research presents CADgen, a CLI Interface and framework designed to bridge the semantic gap between Natural Language Processing (NLP) and Mechanical Engineering. By treating Computer-Aided Design (CAD) not as a visual task but as a code-generation task, we successfully leveraged the reasoning capabilities of Large Language Models (specifically Google Gemini 2.5 Flash) to generate precise, parametric 3D models.

We plan to extend this work in future by integrating VLMs [13] for visual feedback, developing autonomous agents for closed-loop error correction, and fine-tuning on domain-specific datasets to enhance the model's understanding of mechanical constraints.

8 Data and Code Availability

To facilitate reproducibility and further research in automated mechanical design, the complete source code for the CADgen framework, along with the dataset of experimental prompts, generated OpenSCAD scripts, and performance logs, has been made open-source.

The repository is hosted on GitHub and can be accessed at: https://github.com/Asvera/Text2CAD [14]

The repository contains:

- Source Code: The main.py inference engine and requirements.txt
- Experimental Data: Raw .scad files for all 14 test cases (Beams, Gears, Trusses).

- Prompt Library: The exact System Instructions and prompts used to generate the results presented in this paper.

References

1. A. Saxena and B. Sahay, Computer aided engineering design. Springer, (2005).
2. L. Cao, "Ai in finance: challenges, techniques, and opportunities," ACM Computing Surveys (CSUR), vol. 55, no. 3, pp. 1–38, (2022).
3. K. Rana and N. Khatri, "Automotive intelligence: Unleashing the potential of ai beyond advance driver assisting system, a comprehensive review," Computers and Electrical Engineering, vol. 117, p. 109237, (2024).
4. Z. Zhang, H. Ning, F. Shi, F. Farha, Y. Xu, J. Xu, F. Zhang, and K.-K. R. Choo, "Artificial intelligence in cyber security: research advances, challenges, and opportunities," Artificial Intelligence Review, vol. 55, no. 2, pp. 1029–1053, (2022).
5. Wikipedia contributors, "Midjourney — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Midjourney&oldid=1326142425, last accessed 2025/12/16.
6. Sobo, A., Mubarak, A., Baimagambetov, A., & Polatidis, N. Evaluating LLMs for Code Generation in HRI: A Comparative Study of ChatGPT, Gemini, and Claude. Applied Artificial Intelligence, 39(1). https://doi.org/10.1080/08839514.2024.2439610, (2025).
7. Wikipedia contributors, "Github copilot — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=GitHub_Copilot&oldid=1324850150, last accessed 2025/12/16.
8. Wikipedia contributors, "Openscad — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=OpenSCAD&oldid=1325663822, last accessed 2025/12/16.
9. Wikipedia contributors, "Constructive solid geometry — Wikipedia, the free encyclopedia."https://en.wikipedia.org/w/index.php?title=Constructive_solid_geometry&oldid=1316137672, last accessed 2025/12/16.
10. E. Ogunnowo, E. Ogu, P. Egbumokei, I. Dienagha, and W. Digitemie, "Conceptual model for topology optimization in mechanical engineering to enhance structural efficiency and material utilization," Iconic Research and Engineering Journals, vol. 7, no. 12, pp. 2456–8880, (2024).
11. H. Jun and A. Nichol, "Shap-e: Generating conditional 3d implicit functions," arXiv preprint arXiv:2305.02463, (2023).
12. A. Tam, "What are zero-shot few-shot prompting."https://machinelearningmastery.com/what-are-zero-shot-prompting-and-few-shot-prompting/, last accessed 2025/12/16.
13. F. Bordes, R. Y. Pang, A. Ajay, A. C. Li, A. Bardes, S. Petryk, O. Mañas, Z. Lin, A. Mahmoud, B. Jayaraman, M. Ibrahim, M. Hall, Y. Xiong, J. Lebensold, C. Ross, S. Jayakumar, C. Guo, D. Bouchacourt, H. Al-Tahan, K. Padthe, V. Sharma, H. Xu, X. E. Tan, M. Richards, S. Lavoie, P. Astolfi, R. A. Hemmat, J. Chen, K. Tirumala, R. Assouel, M. Moayeri, A. Talattof, K. Chaudhuri, Z. Liu, X. Chen, Q. Garrido, K. Ullrich, A. Agrawal, K. Saenko, A. Celikyilmaz, and V. Chandra, "An introduction to vision-language modeling," (2024).
14. S. Yadav(Asvera), "Cadgen: Automated 3d cad model generation using llms." https://github.com/Asvera/Text2CAD, (2025).