



Iterative Diagnosis-Driven Augmented Generation (IDDAG) for Programmatic 3D CAD

Thomas Paviot¹, Virginie Fortineau¹ and Samir Lamouri^{2,3}

¹*meeDIA, 15 rue Glais Bizoïn, 35000 Rennes, France*

²*Arts & Métiers, 151 bd de l'Hôpital, 75013 Paris, France*

³*LAMIH CNRS, Université Polytechnique Hauts-de-France, Campus Mont Houy, 59313 Valenciennes Cedex 9, France*

Keywords: CAD, 3D, LLM, Context Augmentation, BRep.

Abstract: This paper presents a novel approach for automated generation of 3D CAD models using Large Language Models (LLMs) within Model-Based Systems Engineering workflows. We introduce Iterative Diagnosis-Driven Augmented Generation (IDDAG), a methodology combining programmatic geometry creation with systematic diagnostic feedback. The approach leverages a dedicated API for exact Boundary Representation (B-Rep) geometry generation, augmented by a closed-loop architecture that provides iterative refinement based on syntactic, runtime, and geometric analysis. Unlike existing methods requiring extensive training datasets or producing approximate geometries, our solution generates topologically valid, parameterized models while maintaining traceability to engineering requirements. Results demonstrate progressive geometric refinement across iterations, with the diagnostic feedback mechanism effectively identifying and correcting topological inconsistencies.

1 INTRODUCTION

In the context of Industry 4.0's digitalization of industrial value chains, the integration between Model-Based System Engineering (MBSE) and Computer-Aided Design (CAD) becomes essential to facilitate the digitization of physical product development processes (Meussen, 2021).


3D CAD models are particularly critical as they represent the exact geometric definition that will be used throughout the product lifecycle, at each stage of the digital chain (visualization, simulation, automated manufacturing, etc.). Integrating 3D representation within an MBSE approach therefore imposes stringent requirements: models must comply with structural and semantic consistency constraints with other system views (functional, logical, physical architectures); their construction must follow a traceable process, where each design choice can be justified against initial requirements; their maintenance over time requires precise and unambiguous parameterization enabling adaptation to evolving needs. Cre-


ating these 3D models requires significant technical expertise, substantial time investment, and the use of complex professional CAD software - making these costly processes.


Concurrently, the development of Large Language Models (LLMs) reveals remarkable opportunities in the Industry 4.0 domain (Bourdin et al., 2024).

This paper thus aims to address two research questions: is it possible, using LLMs, to automate all or part of the process of creating 3D models from a set of requirements? If so, how can we ensure the explainability of the provided solution to enable validation of choices and therefore design traceability? Industrial 3D CAD processes utilize Boundary Representation (B-Rep), as it offers mathematically exact geometry representation. While recent AI advances have enabled notable progress in processing approximate geometries (particularly through meshes) with applications in entertainment, visualization, and 3D printing, research concerning exact representation remains less numerous and advanced.

As a consequence, the paper is structured as follows: Section 2 presents the state of the art regarding LLM utilization for B-Rep generation and its current limitations. Section 3 introduces our two scientific contributions: a programmatic methodology coupled

^a <https://orcid.org/0000-0002-3380-8845>

^b <https://orcid.org/0000-0001-7043-4849>

^c <https://orcid.org/0000-0003-3868-9280>

with an iterative context enrichment mechanism for solution quality improvement. Section 4 details the experimental validation protocol and results. Finally, Section 5 discusses the contributions relative to existing approaches and identifies future research directions.

2 RELATED WORK

2.1 B-Rep Model

The BRep model describes the mathematically exact geometry of a component through non-volumetric elements that compose its surface. These elements consist of faces, edges, and vertices arranged in an adjacency graph. A face corresponds to a surface patch, an edge to an intersection curve segment between at least two surfaces, and a vertex to an intersection point between two distinct curves (Mortenson, 1997).

2.2 Large Language Models

A Large Language Model (LLM) is an artificial intelligence system based on a transformer architecture, trained on massive volumes of textual data (ranging from hundreds of billions to several trillion tokens) for self-supervised learning purposes. These models are capable of Natural Language Processing (NLP) and can perform various complex linguistic and cognitive tasks such as translation, reasoning, programming, and problem-solving, either in few-shot mode (with few examples) or zero-shot mode (without examples). Their learning capacity emerges from attention mechanisms allowing them to capture long-term dependencies in textual sequences and acquire sophisticated contextual representations of language (Naveed et al., 2024).

Regarding coding tasks, (Liang et al., 2024) reveal that LLMs demonstrate their greatest efficiency when tasks are structured with clear context, for example targeted modifications of existing code, data visualization tasks, or documented API implementation. Efficiency is maximized when source code is available as context, objectives are well-defined, and the user possesses sufficient technical expertise to formulate precise queries.

2.3 LLMs for B-Rep Geometry Generation

In the field of automatic BRep geometry generation, scientific literature works can be classified into two

categories: a) those that directly produce BRep geometry; b) those that produce an intermediate artifact that drives geometry creation.

B-Rep Geometry Generation. This approach is chosen, for example, by (Xu et al., 2024) (Zhang et al., 2023) (Zhang et al., 2024). The algorithm generates a graph representing BRep topology and geometry. These approaches use neural networks, but not directly LLMs. A preliminary learning phase is necessary, involving datasets of several hundred or thousand examples. Then validation tests are performed on data not included in the initial dataset. These works face issues of precision, explainability, and lack of formal guarantees on the production of topologically valid CAD models (risks of generating non-manifold geometries, or surfaces with self-intersections).

Indirect Generation. (Wu et al., 2023) introduce CAD-MLLM, a system using multimodal LLMs (text, image) to generate CAD models from various inputs (text, images, point clouds). In this case, geometry is not directly generated by the LLM: the output is a geometry description in the form of a "command sequence," which drives geometry construction (a method that falls within the framework of Parametric Macro Approaches introduced by (Mun et al., 2003)). Each sequence element is identified by a mnemonic associated with a token in the fine-tuning phase of the chosen LLM. According to the authors, CAD-MLLM presents the following limitations: the description in the form of basic operation sequences uses an ad hoc model composed of a small number of operations in linear form. It is work centered on exact geometry reconstruction, but not generation from requirements. (Yuan et al., 2024) Zhang et al. propose a very similar approach and report the same limitations.

2.4 Open Issues

We identify three significant barriers in the presented works:

- all works rely on a learning phase based on a large dataset, which is a costly operation,
- produced geometries are limited to simple cases,
- it is not possible to guarantee the topological validity of generated shapes.

Finally, we note that in these works, geometry is never related to the notions of function or requirement so important in MBSE. In the remainder of this paper, we propose a method to address the preceding barriers.

3 PROGRAMMATIC APPROACH FOR BRep GENERATION

We therefore orient our work toward an approach that:

- generates parameterized geometry programmatically (see section 3.1), enabling for instance the creation of component catalogs with complex geometries, or inclusion in simulation optimization loops (e.g., FEM),
- generates exact, topologically consistent geometry,
- does not require LLM fine-tuning or learning, nor datasets. We prefer context augmentation.

3.1 Programmatic Approach

Principle. Following from the previous section, we propose a path that falls within the "indirect generation" category of geometry, different from those proposed in section 2: a "programmatic" approach. Rather than asking the LLM to generate a command sequence in an ad hoc model, we propose asking it to generate a computer program to which form creation is delegated. Two approaches can be considered:

- design a Domain Specific Language for 3D form programming;
- use a common programming language (JavaScript, Java, C, etc.) and a domain-specific 3D development library.

The design and implementation of a DSL present two challenges: with its specific grammar, it requires LLM fine-tuning, and thus the creation of a sufficiently large dataset to ensure proper learning. Additionally, a compiler or interpreter for this DSL must be developed. To overcome these difficulties, we choose the Python programming language in this work.

Example and Limitations. To illustrate this approach, here is a basic ChatGPT prompt to generate a 3D nut using Python and the free and open-source 3D CAD library pythonocc (Paviot, 2022) :

```
Write a python script, based on the pythonocc library, to generate the parameterized 3D geometry of a standard H-M10 nut.
```

The generated program is included in Appendix 1. We observe that:

- the program is difficult for humans to read, except for library specialists;

- the program, when executed, generates an error: the *BRepBuilderAPI_MakePrism* class does not exist in the library, it is an LLM hallucination.

This programmatic approach shows its limitations on a simple example: ChatGPT has only limited knowledge of the utilized library and proposes a non-existent class.

Dedicated API. To gain explainability, we seek to obtain a more concise and clearer program. We propose relying on a dedicated API based on:

- a taxonomy of elementary operations for creation, transformation, and measurement of basic shapes
- a naming convention for classes and functions methods that is closest to natural language. This proximity to natural language improves program readability for humans;

Furthermore, we propose passing textual API documentation to the LLM. This contextual element should enable the LLM to generate a program conforming to the provided specification, addressing the problem raised in the previous section. The documentation must be concise, to limit prompt length, and exhaustive to cover all proposed functionalities.

3.2 Context, Metadata, Requirements, and Knowledge

Context The structure of the prompt passed to the LLM is therefore:

```
<instructions>[...]</instructions>
<api_documentation>[...]</api_documentation>
```

For example:

```
<instructions>
Generate a Python script to create the 3D
geometry of an H-M10 nut, using the library
whose documentation is given below
</instructions>
<api_documentation>
3D Modeling Library
=====
This API allows easy creation and manipulation
of 3D geometric objects.
[...]
</api_documentation>
```

Requirements and Knowledge. Requirements are part of the prompt passed to the LLM, enclosed in `<requirements></requirements>` tags. Specifications are of three types:

- functional requirements (maximum dimensions, maximum mass)

- technical requirements (technology used, materials, etc.)
- various additional requirements (expected level of detail, aesthetics, etc.)

Requirements are passed as text, which can represent structured (JSON, XML), semi-structured, or unstructured information (natural language) depending on the context. Similarly, domain-specific knowledge is passed between `<knowledge></knowledge>` tags. For example, this could be an excerpt from the standard specifying H-M10 nut dimensions. Thus, the final structure of the prompt passed to the LLM looks like (replace the `#component` token with any component name, for example H-M10 nut):

```
<instructions>
Write a Python program to generate
the parameterized 3D geometry of a #component:
- using the API whose documentation is given
below
- using #component metadata
- using the given knowledge elements
- respecting all listed requirements
</instructions>
<metadata>[...]</metadata>
<api_documentation>[...]</api_documentation>
<knowledge>[...]</knowledge>
<requirements>[...]</requirements>
```

The Python program generated this way offers better human readability and creates the following graphical 3D geometry (see 1):

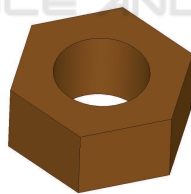


Figure 1: Fixed 3D nut generation.

4 DIAGNOSIS-DRIVEN AUGMENTED GENERATION

Despite the precautions taken previously (dedicated API, concise documentation in context), output quality cannot be guaranteed: the generated program may still present syntactic errors, the geometry creation may prove mathematically unfeasible, or the geometry may be topologically incorrect. To address these deficiencies and enhance reliability, we propose augmenting the LLM with reflective capabilities by providing a set of information constructed a posteriori, based on the LLM output. This involves enriching the

LLM prompt context with a diagnostic report comprising information on:

- syntactic compliance of the generated program, utilizing static code analysis tools
- program execution success and geometry creation, derived from Python console output
- analysis of the produced geometry (topological coherence, inertial elements, bounding volume, etc.), performed by a dedicated software component

The generated diagnosis thus comprises three distinct sections:

```
<diagnosis>
  <static_analysis>[...]</static_analysis>
  <runtime_analysis>[...]</runtime_analysis>
  <geometric_analysis>[...]</geometric_analysis>
</diagnosis>
```

Once this diagnostic report is generated, it is added to the context and returned to the LLM for potential modification requests. This process iterates until a satisfactory response is obtained, with the user determining whether to initiate a new generation cycle. Thus, during successive iterations, the prompt passed to the LLM the second time (after the first iteration) follows this schematic XML model:

```
<input>
  <instructions>[...]</instructions>
  <metadata>[...]</metadata>
  <api_documentation>[...]</api_documentation>
  <knowledge>[...]</knowledge>
  <requirements>[...]</requirements>
</input>
<output>[... first LLM response ...]</output>
<diagnosis>
  <static_analysis>[...]</static_analysis>
  <runtime_analysis>[...]</runtime_analysis>
  <geometric_analysis>[...]</geometric_analysis>
</diagnosis>
<input>
  <instructions>Given the diagnostic report,
  update the 3D generation code.
</instructions>
</input>
```

After the second iteration, the following is added to the previous prompt:

```
<output>[... second LLM response ...]</output>
<diagnosis>
  <static_analysis>[...]</static_analysis>
  <runtime_analysis>[...]</runtime_analysis>
  <geometric_analysis>[...]</geometric_analysis>
</diagnosis>
<input>
  <instructions>Given the second diagnostic
  report, update the 3D generation code.
</instructions>
</input>
```

and so forth. This workflow process can be illustrated by the synoptic diagram in Figure 2, which adopts closed-loop control system semantics.

5 EXPERIMENTS

The experimental validation was conducted using Python 3.10 and the latest pythonocc version. The Large Language Model used was Claude Haiku, selected for its balance of performance and response speed. Two distinct test cases were implemented to evaluate the IDDAG methodology: a standardized mechanical component (ball bearing) and an aerodynamic surface (airplane wing). For each experiment, we analyze the geometric evolution across successive iterations and examine convergence characteristics.

Ball Bearing. The first experiment focused on generating a 6203-series radial ball bearing, chosen for its well-defined geometric constraints and standardized dimensions. This is the only requirement.

Table 1 illustrates the geometric evolution across five iterations. Initial generation ($n=1$) produced a simplified cylindrical representation lacking internal components. Subsequent iterations progressively refined the model: $n=2$: Addition of basic rolling element placeholders, $n=3$: Implementation of correct race profiles, $n=4$: Refinement of ball geometry and spacing $n=5$: Final optimization of contact surfaces and clearances.

The diagnostic feedback loop identified and corrected multiple issues, including topological inconsistencies in race-to-ball interfaces ($n=2 \rightarrow n=3$) and geometric validity constraints for rolling element distribution ($n=3 \rightarrow n=4$). 1

Airplane Wing. The second experiment involved generating an aircraft wing section using the NACA 0012 airfoil profile, chosen for its well-documented geometry and extensive validation data. The specification included a 1000mm span, 200mm chord length, and precise adherence to the NACA 0012 coordinate system. The evolution shown in Table 2 demonstrates significant challenges in initial geometry creation: $n=1$: Failed mathematical generation of intrados and extrados curves (not visualized), $n=2$: Successfully generated planar surface but incorrect extrusion direction, $n=3$: Corrected extrusion vector orientation, resulting in topologically and geometrically valid 3D form.

Discussion. The experimental results demonstrate iterative geometric refinement across successive generations, though convergence is not consistently achieved for all test cases. The ball bearing example exhibited systematic improvement in geometric accuracy and topological validity over 5 iterations, progressing from a simplified cylindrical representation to a complete model with proper race profiles and ball spacing. In contrast, the aircraft wing case highlighted potential limitations, requiring only 3 iterations but encountering initial mathematical generation failures before achieving a valid 3D form. These preliminary findings, while promising, warrant further validation across a broader range of mechanical components to comprehensively assess the methodology's robustness and generalizability. The results also suggest that targeted human intervention, through manual augmentation of diagnostic feedback, may enhance convergence by redirecting suboptimal initial design choices. Additional experimentation is needed to evaluate the approach's scalability to more complex geometries and to establish quantitative metrics for convergence behavior.

6 CONCLUSION AND PERSPECTIVES

This paper presented two scientific contributions enabling 3D geometry generation using Large Language Models:

- A programmatic approach using a dedicated API for geometry creation
- A closed-loop architecture for context augmentation, termed Iterative Diagnosis-Driven Augmented Generation (IDDAG)

This solution offers key advantages: without requiring extensive training or fine-tuning, it produces human-readable, explainable, and parameterized models that generate topologically valid geometries. The 3D artifact generation maintains traceability to requirements and refines through iterations. The proposed methodology has been implemented and validated on elementary test cases. Short term research directions include formal analysis of convergence properties and optimization of iteration counts, more complex geometries and extension to complex assemblies with multiple interacting components. The diagnostic-driven augmentation approach shows potential for transposition to other design domains where systematic feedback can be generated. Mid to long-term research priorities include establishing formal convergence guarantees, evaluating sys-

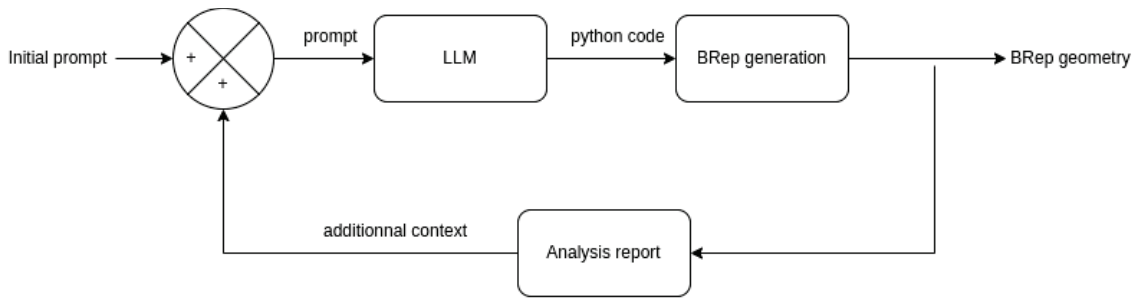


Figure 2: Closed-loop architecture of the Iterative Diagnosis-Driven Augmented Generation (IDDAG) system.

Table 1: Ball Bearing: Geometric Evolution Across Iterations.

$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$

Table 2: Aircraft Wing: Geometric Evolution Across Iterations.

$n = 2$	$n = 3$

tem scalability for complex assemblies, and developing robust integration protocols with established CAD workflows.

REFERENCES

- Bourdin, M., Neumann, A., Paviot, T., Pellerin, R., and Lamouri, S. (2024). Exploring the applications of natural language processing and language models for production, planning, and control activities of SMEs in industry 4.0: a systematic literature review. *Journal of Intelligent Manufacturing*.
- Liang, J. T., Yang, C., and Myers, B. A. (2024). A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*.
- Meussen, B. (2021). On the use of model based systems engineering and CAD for the design of physical products. *Proceedings of the Design Society*, 1:2317–2326.
- Mortenson, M. E. (1997). *Geometric modeling*. John Wiley & Sons, Inc.
- Mun, D., Han, S., Kim, J., and Oh, Y. (2003). A set of standard modeling commands for the history-based parametric approach. *Computer-Aided Design*, 35(13):1171–1179.
- Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., and Mian, A. (2024). A comprehensive overview of large language models.
- Paviot, T. (2022). pythonocc.
- Wu, S., Khasahmadi, A., Katz, M., Jayaraman, P. K., Pu, Y., Willis, K., and Liu, B. (2023). CAD-LLM: Large language model for CAD generation. In *NeurIPS 2023 Workshop on Machine Learning for Creativity and Design*.
- Xu, X., Lambourne, J., Jayaraman, P., Wang, Z., Willis, K., and Furukawa, Y. (2024). BrepGen: A b-rep generative diffusion model with structured latent geometry. *ACM Transactions on Graphics*, 43(4).
- Yuan, Z., Shi, J., and Huang, Y. (2024). OpenECAD: An efficient visual language model for editable 3D-CAD design. *Computers & Graphics*, 124.
- Zhang, C., Piquié, R., Polette, A., Carasi, G., De Char-

nace, H., and Pernot, J.-P. (2023). Automatic 3D CAD models reconstruction from 2D orthographic drawings. *Computers & Graphics*, 114:179–189.

Zhang, C., Polette, A., Pinqu  , R., Carasi, G., De Char-nace, H., and Pernot, J.-P. (2024). eCAD-Net: Edit-able parametric CAD models reconstruction from dumb B-Rep models using deep neural networks. *Computer-Aided Design*, 178.

```

29 Shape ()
30 # Subtract cylinder from hexagonal
   prism to form the nut body
31 nut = BRepAlgoAPI_Cut(hex_prism,
   cylinder).Shape ()

```

APPENDIX

```

1 from OCC.Core.BRepBuilderAPI import
   BRepBuilderAPI_MakePolygon,
   BRepBuilderAPI_MakeFace,
   BRepBuilderAPI_MakePrism
2 from OCC.Core.BRepPrimAPI import
   BRepPrimAPI_MakeCylinder
3 from OCC.Core.BRepAlgoAPI import
   BRepAlgoAPI_Cut
4 from OCC.Core.gp import gp_Pnt,
   gp_Vec, gp_Ax2, gp_Dir
5 from OCC.Core.BRepOffsetAPI import
   BRepOffsetAPI_MakeThickSolid
6
7 # Nut specifications
8 hex_radius = 8.0 # Distance from
   center to a flat, in mm
9 nut_thickness = 8.0 # Thickness of
   the nut, in mm
10 thread_diameter = 10.0 # Diameter
   of the inner thread, in mm
11 thread_clearance = 1.0 # Additional
   clearance for threading
12
13 # Create hexagonal base
14 hexagon = BRepBuilderAPI_MakePolygon
   ()
15 for i in range(6):
16     angle = i * (2 *
17         3.141592653589793 / 6)
18     x = hex_radius * math.cos(angle)
19     y = hex_radius * math.sin(angle)
20     hexagon.Add(gp_Pnt(x, y, 0))
21 hexagon.Close()
22 hex_face = BRepBuilderAPI_MakeFace (
23     hexagon.Wire()).Face()
24
25 # Extrude to create the 3D hexagonal
   prism
26 extrusion_vec = gp_Vec(0, 0,
27     nut_thickness)
28 hex_prism = BRepBuilderAPI_MakePrism
   (hex_face, extrusion_vec).Shape()
29
30 # Create cylindrical hole for
   threading
31 cylinder = BRepPrimAPI_MakeCylinder (
32     gp_Ax2(gp_Pnt(0, 0, 0), gp_Dir(0,
33     0, 1)), thread_diameter / 2 +
34     thread_clearance, nut_thickness).

```