TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 68, Issue Special I, May, 2025

USE OF AI LANGUAGE MODELS IN GENERATING 3D PRINTABLE MODELS

Nicolae-Răzvan MITITELU, Roxana HOBJÂLĂ, Vasile ERMOLAI, Marius-Ionuț RÎPANU, Cristian BIȘOG

Abstract: *This paper presents aspects regarding the integration of artificial intelligence (IA) with OpenScad to generate STL files used in 3D printing. Conventional CAD modeling and manual writing of parametric scripts often require advanced technical knowledge and are prone to errors. It simplifies this process by interpreting the instructions in natural language and the automatic generation of optimized Openscad scripts. The experimental study focused on the design of a grooved tree-functional bore system, demonstrating the ability to ensure geometric accuracy, functional adaptability and perfect integration in the sectioning software. The results obtained confirm the efficiency, reproducibility and flexibility in the design, positioning as a transforming tool in parametric workflows for 3D printing.*
Keywords: *Artificial intelligence, 3D parametric design, geometrical optimization, AI CAD design.*

1. INTRODUCTION

3D printing has become a key technology in industrial manufacturing, rapid prototyping, and even home applications. For a part to be printed correctly, a 3D models is required, usually in SLT format (i.e., Standard Tessellation Language), which is an simplified representation of the 3D model's surfaces [1]. However, exporting an STL file, depends on the CAD software specifics. In this paper, an alternative method was addressed, one that does not involve the use of conventional CAD design sofware for 3D modeling, the method being manual code writing to define the geometry of objects through parametric modeling using OpenSCAD [2].

This process is also time-consuming, as in conventional 3D modeling, being prone to errors and requiring a deep understanding of geometric logic and programming language [3].

Artificial Intelligence (AI), through advanced language models, can simplify this process. AI can understand commands and transform them into functional OpenSCAD scripts, thus democratizing access to parametric design for a wider range of users [4].

The question underlying the conception of this paper was: can AI efficiently develop functional code for generating parametric STL files while maintaining geometric precision, parameter adaptability, and production reliability?

The use of artificial intelligence, in the process of generating STL documents, may be of interest to different fields such as the following: the manufacturing industry of additives necessary in quickly generating mechanical components and prototypes [5]. In the field of medicine for the design of personalized medical devices [6], or in the architectural field for creating model parameters for complex structures [7]. Once created, the models can easily be modified and replicated without causing repercussions on other variables [3].

An important detail is the fact that by applying you in the original design phase, companies could reduce the waste of resources and can shorten including the production terms. In this combination of OpenSCAD, together with the automated generation of AI assisted scripts, it could allow the producers to adapt the design of a real -time product to the market requirements [2]. Generating STL files using OpenSCAD is a process that requires not only technical knowledge about the syntax of programming language, but also solid knowledge about parametric geometry and how physical objects

are transposed into 3D models. Openscad uses a script code language, based on geometric definitions and mathematical relationships, which can be difficult for unpro -experienced users as the occurrence of small errors in the code structure or in geometric parameters can generate defective or impossible printed models [4]. Also, in the absence of a graphic interface, the generation of STL files using programming languages, example Python, requires time and patience for understanding geometric logic being unproductive compared to traditional CAD computer design.

This problems was the main reason to find a solution by using tools that can intervene to simplify the process by automating the generation of the OpenSCAD program. In the integration process of generating 3D models, the development of a clear work flow was aimed. Replicable, which allow users to generate functional STL models based on natural language instructions. This basic structure applied in the present work included the following stages:

a) Defining standardized instructions.

In order to achieve predictable results, standard instructions models have been established that you can interpret correctly. Examples include commands such as: "generates a grooved cylinder with a diameter of 10 mm, 16 grooves and a tolerance of 0.2 mm".

b) Generation of the OpenSCAD code.

Based on the instructions received, you have generated codes necessary for the OpenSCAD program in compliance with the requirements imposed. At this stage it is necessary to verify several scenarios to formulate the requirement to identify the optimal instructions.

c) Optimization of the generated code.

Therefore it was necessary to apply successive requirements, until the model AI was able to respect the capacity of the compatibility of the code with the particular language of the OpenSCAD program. This aspect is important because each programming language has certain features, which is why a code generated for this program, in the same form, cannot be compatible with other similar programs, for example Zoo Modeling, which is used for the same reason, Generating 3D models using code language, has certain features that must be respected.

d) Printing parts and checking compatibility

This stage can be considered less important, if the generated models do not have to comply with a high level of precision, but when the requirements are that the resulting piece respects a certain precision, this stage becomes the most important of the whole process.

2. EXPERIMENTAL PART

From the outset, the artificial intelligence algorithm was employed to determine compatibility between a shaft and a gear. This way the functional requirements were specified as input parameters only for the shaft-type part. The AI algorithm received instructions to independently determine the necessary parameters for the bore-type part and ensure proper coupling between the two components through the grooves.

The initial parameters were entered as the following text: "Generate a code for a cylinder with a diameter of 10 mm, a height of 2 mm, 16 uniformly distributed grooves, each grooves having a depth of 4 mm and a width of 1 mm. Add a tolerance to facilitate assembly. The cylinder must couple with another bore-type part for which you need to determine the parameters to ensure proper coupling of the 3D-printed parts. The bore-type part must include several holes with a diameter of 3 mm for mounting dowel pins, radially arranged on a part diameter of 50 mm". Based on the defined specifications, the next stage involved generating OpenSCAD code using AI. Two distinct scripts were generated: one for the grooves shaft (see Fig. 1) and one for the mating bore component (see Fig. 2). Each script was analyzed to ensure clarity, coherence, and compliance with the technical parameters.

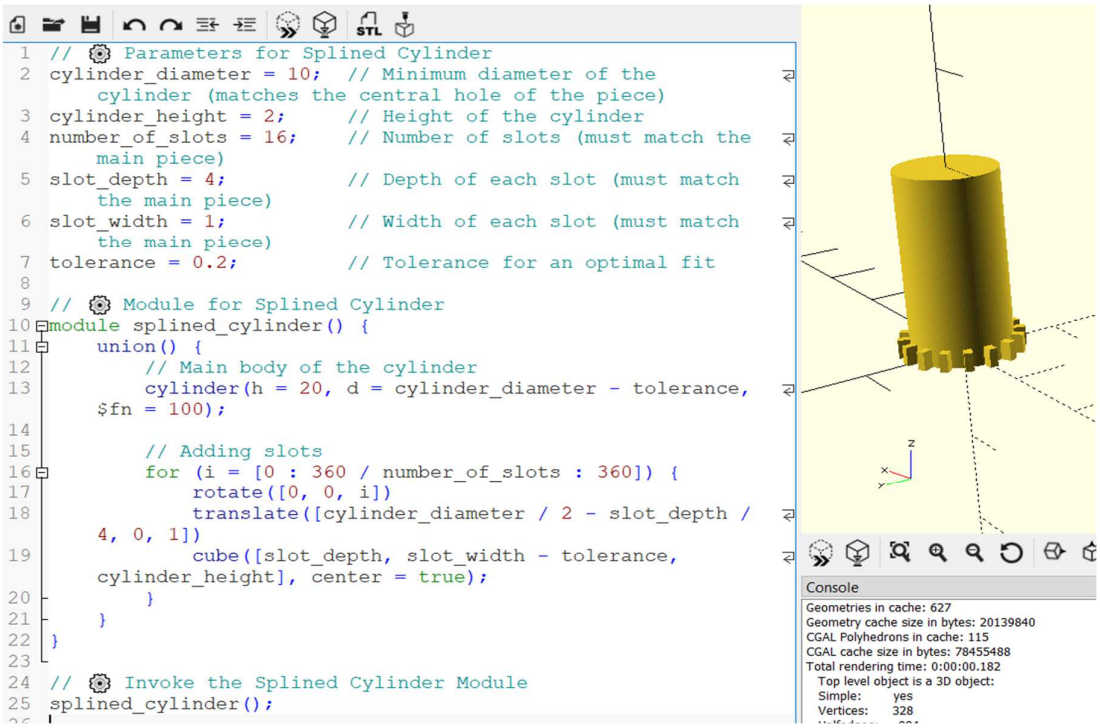
After generation, the codes obtained were imported into the OpenSCAD software for rendering and visualization. These three-dimensional models obtained were visually analyzed to verify compliance with the initially established parameters (fig. 3, 4). In the obtained images, there is a representation of both pieces, including the respective grooves and dimensions. The rendering process was carefully checked to identify any errors before exporting the STL files.

```
cylinder_diameter = 10;    // Minimum diameter of the cylinder
cylinder_height = 2;       // Height of the cylinder
number_of_slots = 16;      // Number of evenly spaced slots
slot_depth = 4;            // Depth of each slot
slot_width = 1;            // Width of each slot
tolerance = 0.2;           // Tolerance for fitting

// ⚙ Module for Splined Cylinder
module splined_cylinder() {
    union() {
        // Main body of the cylinder
        cylinder(h = cylinder_height, d = cylinder_diameter - tolerance, $fn = 100);

        // Adding slots
        for (i = [0 : 360 / number_of_slots : 360]) {
            rotate([0, 0, i])
            translate([cylinder_diameter / 2 - slot_depth / 2, 0, cylinder_height / 2])
            cube([slot_depth, slot_width, cylinder_height], center = true);
        }
```

Fig. 1. The generated script for the grooved shaft.

```
outer_diameter = 50;      // Outer diameter of the piece
piece_thickness = 5;      // Thickness of the piece
central_hole_diameter = 10; // Minimum diameter of the central hole
number_of_slots = 16;     // Number of slots
slot_depth = 4;           // Depth of each slot
slot_width = 1.1;         // Width of each slot
cylinder_hole_diameter = 4; // Diameter of circular holes
number_of_holes = 20;     // Number of holes distributed radially

// ⚙ Module for Slotted Hole
module slotted_hole() {
    // Base cylindrical hole
    cylinder(h = piece_thickness + 2, d = central_hole_diameter, $fn = 100);

    // Adding slots
    for (i = [0 : 360 / number_of_slots : 360]) {
        rotate([0, 0, i])
        translate([central_hole_diameter / 2 - slot_depth / 4, 0, -1])
        cube([slot_depth, slot_width, piece_thickness + 2], center = true);
    }
}

// ⚙ Module for Generating the Main Piece with Slotted Hole
module main_piece() {
    difference() {
        // Main body
        cylinder(h = piece_thickness, d = outer_diameter, $fn = 200);
```

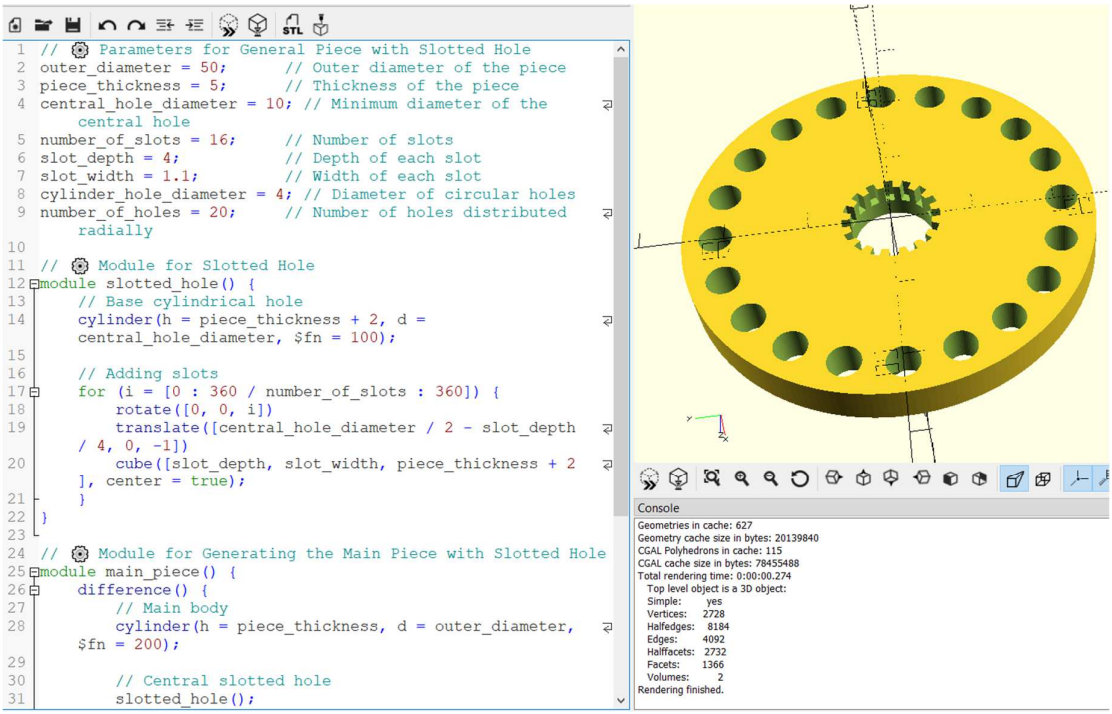
Fig. 2. The generated script for the bore-type part.



```
// Parameters for Splined Cylinder
cylinder_diameter = 10;  // Minimum diameter of the cylinder (matches the central hole of the piece)
cylinder_height = 2;     // Height of the cylinder
number_of_slots = 16;    // Number of slots (must match the main piece)
slot_depth = 4;          // Depth of each slot (must match the main piece)
slot_width = 1;          // Width of each slot (must match the main piece)
tolerance = 0.2;         // Tolerance for an optimal fit

// Module for Splined Cylinder
module splined_cylinder() {
    union() {
        // Main body of the cylinder
        cylinder(h = 20, d = cylinder_diameter - tolerance, $fn = 100);

        // Adding slots
        for (i = [0 : 360 / number_of_slots : 360]) {
            rotate([0, 0, i])
            translate([cylinder_diameter / 2 - slot_depth / 4, 0, 1])
            cube([slot_depth, slot_width - tolerance, cylinder_height], center = true);
        }
    }
}

// Invoke the Splined Cylinder Module
splined_cylinder();
```

```
Console
Geometries in cache: 627
Geometry cache size in bytes: 20139840
CGAL Polyhedrons in cache: 115
CGAL cache size in bytes: 78455488
Total rendering time: 0:00:00.182
  Top level object is a 3D object:
  Simple:     yes
  Vertices:   328
```

Fig. 3. Generating the tree-type landmark.



```
// Parameters for General Piece with Slotted Hole
outer_diameter = 50;       // Outer diameter of the piece
piece_thickness = 5;       // Thickness of the piece
central_hole_diameter = 10; // Minimum diameter of the central hole
number_of_slots = 16;      // Number of slots
slot_depth = 4;            // Depth of each slot
slot_width = 1.1;          // Width of each slot
cylinder_hole_diameter = 4; // Diameter of circular holes
number_of_holes = 20;      // Number of holes distributed radially

// Module for Slotted Hole
module slotted_hole() {
    // Base cylindrical hole
    cylinder(h = piece_thickness + 2, d = central_hole_diameter, $fn = 100);

    // Adding slots
    for (i = [0 : 360 / number_of_slots : 360]) {
        rotate([0, 0, i])
        translate([central_hole_diameter / 2 - slot_depth / 4, 0, -1])
        cube([slot_depth, slot_width, piece_thickness + 2], center = true);
    }
}

// Module for Generating the Main Piece with Slotted Hole
module main_piece() {
    difference() {
        // Main body
        cylinder(h = piece_thickness, d = outer_diameter, $fn = 200);

        // Central slotted hole
        slotted_hole();
```

```
Console
Geometries in cache: 627
Geometry cache size in bytes: 20139840
CGAL Polyhedrons in cache: 115
CGAL cache size in bytes: 78455488
Total rendering time: 0:00:00.274
  Top level object is a 3D object:
  Simple:     yes
  Vertices:   2728
  Halfedges:  8184
  Edges:      4092
  Halffacets: 2732
  Facets:     1366
  Volumes:    2
Rendering finished.
```

Fig. 4. Generation of the bore type reference.

Next, the models were exported as STL files directly from the software. The export was made without errors, which means that the model is a successful one, and the resulting STL files have also been checked to respect the geometric conditions and defined parameters.

3. 3D MODELS SLICING AND PRINTING

After exporting the STL files, they were imported into PrusaSlicer 2.9.0 slicing software, to prepare the models for printing (see Fig. 5). Slicing parameters such as layer thickness, infill density, and printing speed were customized. The resulting image 3D model preview shows that the models were processed without errors and that the slicing can be performed correctly.

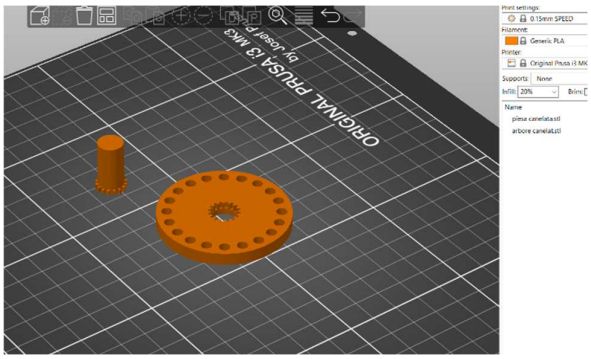


Fig. 5. 3D models import into PrusaSlicer.

The models were then physically validated using a Prusa i3 MK3S+ printer, using a red (for the saft) and a green (for the wheel) PolyTerra PLA as shown in Fig. 6 and Fig. 7.

Fig. 6. 3D Printed shaft-type component.

The resulting parts were physically analyzed to verify compliance with the dimensional parameters, the structural integrity and the quality of the surfaces.

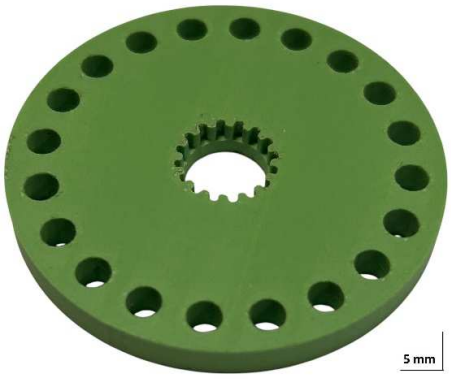


Fig. 7. 3D Printed bore-type component.

An important detail is the configuration of the bore, ensuring that the shaft-type part can interlock. This detail was also automatically generated by artificial intelligence, aiming to introduce additional improvements that were accepted from the initial stage of code preparation (see Fig. 8).

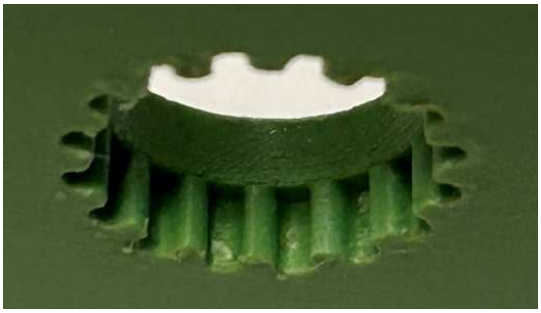


Fig. 8. Self-Locking shaft configuration.

The assembly of the two components was tested to confirm their functionality. The results showed a precise fit between the two components, with proper clearance and relatively low- alignment errors (see Figs. 9 and 10).

Fig. 9. Front view assembled parts.

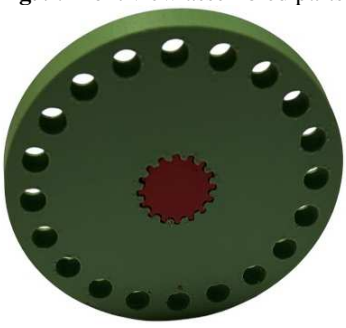


Fig. 10. Back view assembled parts.

4. CONCLUSION

The integration of AI with OpenSCAD for generating 3D printable models represents an alternative approach to traditional methods for creating 3D models, showing good efficiency

and providing access for users without advanced programming or CAD design experience.

This method enables parametric adaptability, reproducibility, and geometric precision. Even so, there are some limitations: AI struggles with generating organic shapes, relies on clear and precise instructions, and the final validation of models still requires human intervention for geometric and functional verification.

5. REFERENCES

[1] Gangl, H., Ren, Y., Urbancic, Z., *Hands-on Tropical Geometry*, Arxiv, 2023.

[2] Gonzalez, J.F., Pietrzak, T., Girouard, A., Casiez, G., *Facilitating the Parametric Definition of Geometric Properties in Programming-Based CAD*, Arxiv, 2024.

[3] Jignasu, A., Marshall, K.C., Krishnamurthy, Ganapathysubramanian, B., Balu, A., Hegde, A. *Towards Foundational AI Models for Additive Manufacturing: Language Models for G-Code Debugging, Manipulation, and Comprehension,* Arxiv, 2023.

[4] Feng, G., Yan, W., *Generative AI for Visual Programming and Parametric Modeling,* Arxiv, 2024.

[5] Li, J., Pepe, A., Gsaxner, C., Egger, J., *An Online Platform for Automatic Skull Defect Restoration and Cranial Implant Design,* 2020.

[6] Ko, J., Ajibefun, J., Yan, W., *Experiments on Generative AI-Powered Parametric Modeling and BIM for Architectural Design,* 2023.

[7] Alrashedy, K., Tambwekar, P., Zaidi, Z., Langwasser, M., Xu, W., Gombolay, M., *Generating CAD Code with Vision-Language Models for 3D Designs,* 2024.

[8] Mititelu, N.R., *Theoretical Aspects Regarding AI Implementation in Optimized STL File Generation,* International Journal of Engineering Research & Technology, 2024.

Utilizarea modelelor de inteligență artificială de limbaj în generarea de modele imprimabile 3D

Lucrarea prezintă aspecte privitoare la integrarea Inteligenței Artificiale (IA) cu OpenSCAD pentru generarea fișierelor STL utilizate în imprimarea 3D. Modelarea tradițională CAD și scrierea manuală a scripturilor parametrice necesită adesea cunoștințe tehnice avansate și sunt predispuse la erori. IA simplifică acest proces prin interpretarea instrucțiunilor în limbaj natural și generarea automată de scripturi OpenSCAD optimizate. Studiul experimental s-a concentrat pe proiectarea unui sistem arbore canelat – alezaj funcțional, demonstrând capacitatea IA de a asigura precizie geometrică, adaptabilitate funcțională și integrare perfectă în software-ul de secționare. Rezultatele obținute confirm eficiența, reproductibilitatea și flexibilitatea în proiectare, poziționând IA ca un instrument transformator în fluxurile de lucru parametrice pentru imprimarea 3D.

Nicolae-Razvan MITITELU, PhD student, "Gheorghe Asachi" Technical University of Iași, Romania, Department of Machine Manufacturing Technology, e-mail: nicolae-razvan.mititelu@student.tuiasi.ro, +40746875540

Roxana HOBJALA, PhD student, "Gheorghe Asachi" Technical University of Iași, Romania, Department of Machine Manufacturing Technology, e-mail: roxana.hobjala@yahoo.com, +40769683755

Vasile ERMOLAI, Associate Professor, "Gheorghe Asachi" Technical University of Iași, Romania, Department of Machine Manufacturing Technology, e-mail: vasile.ermolai@academic.tuiasi.ro, +40754362102

Marius-Ionut RIPANU, Professor, "Gheorghe Asachi" Technical University of Iași, Romania, Department of Machine Manufacturing Technology, e-mail: marius-ionut.ripanu@academic.tuiasi.ro, +40764556149

Cristian BIȘOG, PhD student, Department of Machine Manufacturing Technology, "Gheorghe Asachi" Technical University of Iași, Romania, e-mail: cristian.bisog@student.tuiasi.ro, +40741404419