Proceedings of the Design Society, Volume 5: ICED25
https://doi.org/10.1017/pds.2025.10081

Augmented design automation: leveraging parametric designs using large language models

Fabian Schöfer and **Arthur Seibel** ✉

Leuphana University Lüneburg, Germany

✉ arthur.seibel@leuphana.de

ABSTRACT: Traditional design automation enables parameterized customization but struggles with adapting to abstract or context-based user requirements. Recent advances in integrating large language models with script-driven CAD kernels provide a novel framework for context-sensitive, natural-language-driven design processes. Here, we present augmented design automation, enhancing parametric workflows with a semantic layer to interpret and execute functional, constructional, and effective user requests. Using CadQuery, experiments on a sandal model demonstrate the system's capability to generate diverse and meaningful design variations from abstract prompts. This approach overcomes traditional limitations, enabling flexible and user-centric product development. Future research should focus on addressing complex assemblies and exploring generative design capabilities to expand the potential of this approach.

KEYWORDS: computer aided design (CAD), design automation, user centred design, machine learning, large language models

1. Introduction

The automated generation of parametric designs is a rapidly growing field in product development and engineering (Rädler & Rigger, 2022). Describing the workflow that leads to a final product through its creation steps allows designers to define variables whose values represent variations of a design. Design automation has a wide range of applications, ranging from the design of electric motors (Umland et al., 2024) to medical applications, such as creating patient-specific implants based on computer tomography data (Burge et al., 2023).

Additionally, the emerging field of prompt engineering demonstrates considerable potential for product development and remains an area of active research (Sahoo et al., 2024). By integrating a prompt-based system with a script-driven CAD kernel, the ability to generate CAD models using natural language has been successfully demonstrated (Badagabettu et al., 2024). However, this approach still faces challenges in reasoning capabilities.

The implementation of design automation workflows into a prompt-based system creates a new form of design automation. Rather than relying on a rigid algorithmic workflow dependent on predefined input variables, this approach enables context-sensitive design to be described through its functions, effects, and construction (Ponn & Lindemann, 2011). By enhancing the design generation with a semantic layer, human language can be used to specify design changes. These change requests are not restricted to the geometric features of the CAD model but can include contextual information, such as "make this product suitable for outdoor use," which can be automatically interpreted and translated into meaningful feature adjustments. At the same time, users maintain the ability to directly modify specific geometric features. Moreover, the system is able to understand specific product requirements, such as weight or stiffness—capabilities that go beyond traditional design automation or CAD systems. As a result of these enhanced capabilities, this approach is referred to as augmented design automation.

2. Related work

2.1. Prompt engineering

Prompt engineering is an innovative methodology aimed at extending the functional capabilities of pre-trained large language models (LLMs) (Sahoo et al., 2024). It allows models to perform novel tasks by providing them with task-specific information, eliminating the need to retrain model parameters (Chen et al., 2023). The core approach involves crafting targeted instructions that enable the model to perform in various applications, including, but not limited to, question answering, commonsense reasoning, and even complex domains such as healthcare (Wang et al., 2023; Hartsock & Rasool, 2024).

The study and refinement of prompt patterns remain an active and dynamic area of research, containing a rich array of strategies and methodologies. These include role prompting, where roles or perspectives are assigned to improve the model's responses, few-shot prompting, where minimal examples help the model to learn new tasks, and the inclusion of reasoning methods, such as chains or trees of thought, to enable deeper and more sophisticated text understanding (Sahoo et al., 2024; Gu et al., 2023).

2.2. Text to CAD

The conversion of human instructions into computer-aided design (CAD) models has been effectively demonstrated by Badagabettu et al. (2024) in their seminal paper, Query2CAD. This study underscores the innovative application of LLMs to create functional code that connects natural language inputs with digital design outputs. By leveraging the advanced capabilities of LLMs, a text-to-CAD model system was successfully implemented using the Python library CadQuery, known for its powerful features and simplicity in parametric modelling. The work signifies a significant advancement in automated design, providing insights into how natural language can be utilized for sophisticated design tasks traditionally reserved for skilled engineers, thereby democratizing access to design processes.

2.3. Product design

The Munich Product Concretization Model represents a suitable framework for the systematic design of technical products (Ponn & Lindemann, 2011). This model is composed of three interconnected layers, each addressing a distinct aspect of the design process. These layers include the functional layer, which focuses on defining the product's intended functions, the effective layer, which converts these functions into specific effects, and the constructional layer, which materializes these effects into physical shapes and components. Together, they provide a structured and logical pathway from abstract requirements to a fully developed product design.

2.4. Design automation

Design automation is a specialized domain within CAD that utilizes product knowledge, sophisticated algorithms, and computational tools to simplify and enhance design processes. It is particularly valuable in scenarios where extensive product customization is required to meet specific customer needs. Beyond supporting customization, design automation serves multiple purposes, such as reducing human errors in complex design tasks, shortening lead times, and improving operational efficiency (Rädler & Rigger, 2022).

By automating routine and repetitive design activities, design automation allows engineers to focus on more creative and innovative aspects of product development. Moreover, design automation is essential for exploring vast solution spaces, identifying optimal design configurations, and ensuring compliance with both technical and regulatory requirements. It is crucial in industries that demand high precision, adaptability, and responsiveness to evolving market demands, such as the automotive, aerospace, and consumer electronics sectors. Its applications not only improve productivity but also drive innovation by enabling rapid prototyping, simulation, and evaluation of different design alternatives.

3. Proposed approach

In classical design automation, customizing an existing parametric design to suit specific preferences is typically achieved by adjusting the relevant parameters. By augmenting the design automation process chain with a semantic layer, a new form of design automation can be achieved.

To integrate a semantic layer into the design workflow, the design must be represented in a format that is interpretable by LLMs and enriched with contextual information regarding its functions, effects, and construction. Moreover, the CAD kernel involved in the process must be accessible by the LLM. Here, the Python library CadQuery[1] is used, as it has been demonstrated to effectively create functional CAD instructions based on human language (Badagabettu et al., 2024).
Transforming the design workflow into a script-based format creates a machine-readable representation of the design (Figure 1, left). This representation is extended by contextual information about the overall product framework (Figure 1, centre). Through this process, the LLM acquires a semantic understanding of the product, creating an interconnection between itself, the user, and the design (Figure 1, right).

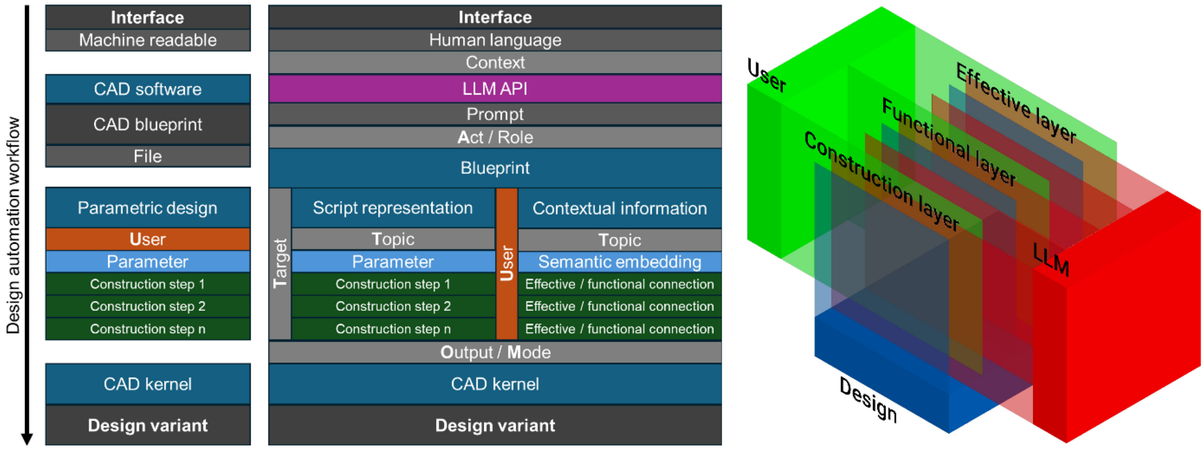


Figure 1. Enhanced user interaction through applying prompt engineering to a design workflow (left and centre), creating an interconnection between the user, the LLM, and the design (right)

3.1. Prompt design

The extension of the design workflow with a semantic layer—and thus its connection with an LLM—is achieved through a prompt. The prompt is created according to the AUTOMAT scheme (Vogel, 2024), a widely used structure for LLM prompting. It contains a role, which describes the task of creating CAD models using human language as input, a target, which provides information about the CAD kernel and its usage, and the base code of the design, which defines the creation steps of the product. Hallucination reduction is realized through chain-of-thought and chain-of-knowledge prompting (Sahoo et al., 2024). For human and machine readability, the prompt is structured using markdown language, see Table 1.

Table 1. Schematic structure of the system prompt

```
# Role
- You act as a program translating human language into CAD models.
# Target
- Update the provided base code according to the user input.
- Use only commands from the Python CadQuery library.
- Limit responses to Python code.
## Base code <BaseCode>
<BaseCode>
```

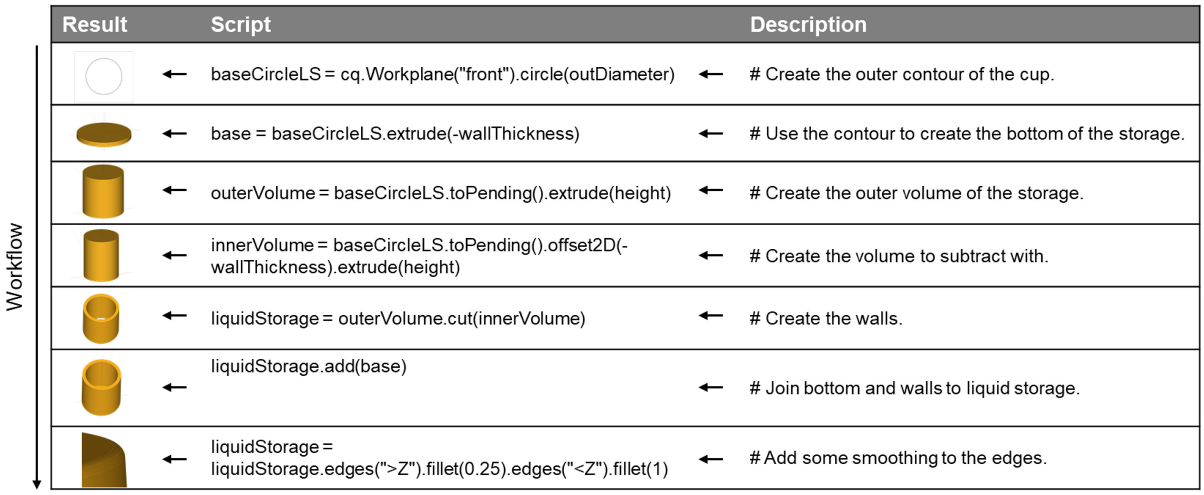
3.2. Base code

The base code serves both as part of the system prompt and as blueprint for the design. It is implemented using the Python library CadQuery. By appending additional information to design-relevant code lines that explain the construction process and its step-by-step contribution to the functionality and effects of the product, a semantic connection is established. This allows designers to integrate their intentions for flexibility and usability into the initial design. The base code is organised into three main layers: features, parts, and assembly.

[1] https://cadquery.readthedocs.io/

3.2.1. Features

Features are written and commented on inline, ensuring that each CAD step is linked to the functional, effective, and constructional layers of the final product. This approach enables the LLM to target specific code lines relevant to the requested design changes. An example of a step-by-step description of feature construction is shown in Figure 2.



Result	Script	Description
	baseCircleLS = cq.Workplane("front").circle(outDiameter)	# Create the outer contour of the cup.
	base = baseCircleLS.extrude(-wallThickness)	# Use the contour to create the bottom of the storage.
	outerVolume = baseCircleLS.toPending().extrude(height)	# Create the outer volume of the storage.
	innerVolume = baseCircleLS.toPending().offset2D(-wallThickness).extrude(height)	# Create the volume to subtract with.
	liquidStorage = outerVolume.cut(innerVolume)	# Create the walls.
	liquidStorage.add(base)	# Join bottom and walls to liquid storage.
	liquidStorage = liquidStorage.edges(">Z").fillet(0.25).edges("<Z").fillet(1)	# Add some smoothing to the edges.

Figure 2. Step-by-step description of feature construction on the example of a cup

3.2.2. Parts

Multiple features together add up to a part. The part is initially described in a header, which contains the name and a descriptive connection to the functional and the effective layers of the product, see Table 2.

Table 2. Header part

# Part: Name
# Functions: <Functional description>
# Effects:
# <Effect>: <Effective description>

A distinct connection is established between a part and its effects and functions. In contrast to the feature description, this approach enables the LLM to handle entire groups of instructions that together form a specific part.

3.2.3. Assembly

Even though CadQuery offers constraint-based assembly capabilities, this work focuses solely on using Boolean operations to assemble the final product. Such as the parts of the product, the assembly process is first introduced and described in a header comment, see Table 3.

Table 3. Step-by-step description of the assembly

<Assembly step 1> # <Step description>
<Assembly step 2> # <Step description>
...

Following that, the assembly instructions are implemented and commented on step by step, such as with the feature instructions.

4. Experiments

To illustrate the proposed approach, a basic sandal design was created, and experiments were conducted using functional, effective, and constructional prompt requests. The system prompt used to generate the designs is provided in Appendix A.

A constructional request is defined by the topological and geometrical aspects of the product. Functional requests, on the other hand, do not contain specific constructional information; instead, they are making demands related to the functionality of the product. Effective requests contain neither constructional nor functional information. They relate to the effective layer of the product and contain information such as weight or friction.

Figure 3 shows the structure of the initial design used in the experiment. It consists of three parts: a sole, a strap, and a profile, which together form the assembly of the final product. The initial design provides three basic functions: fit to the foot, protection of the foot, and ensuring sufficient grip to the ground. Its effects are protection, fit, weight, strength, and friction, which are closely related to the functionality of the product.

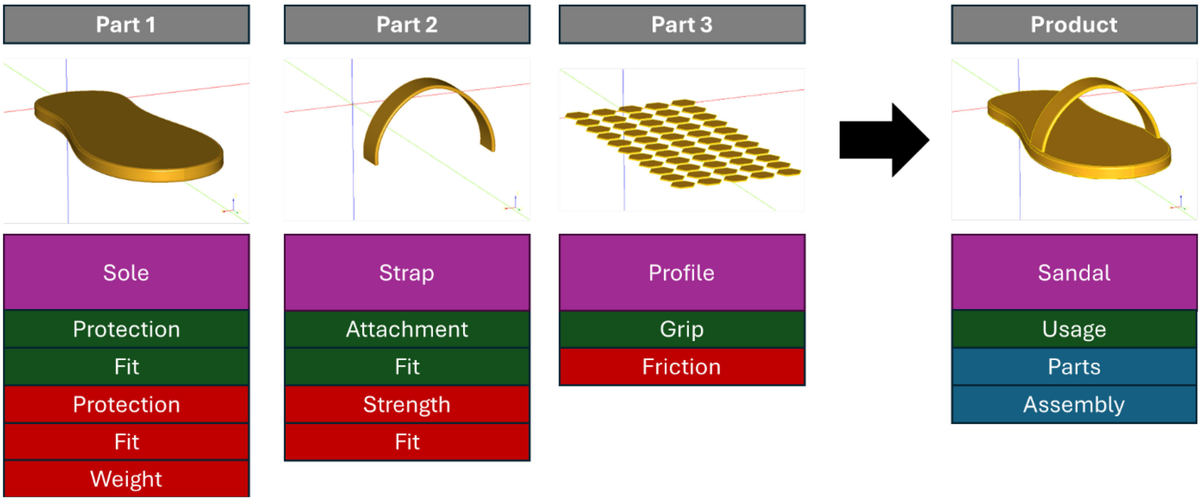


Figure 3. Workflow to create a basic design of a sandal, together with its constructional (blue), functional (green), and effective (red) properties

All experiments were performed using the API of GPT-4o (OpenAI), with a temperature setting of 1.0.[2] Although other LLMs could also be used, they were not examined in this paper.

The experiments were structured according to the layers of the Munich Product Concretization Model, namely the constructional, functional, and effective interpretations of the design.

4.1. Constructional interpretation

Figure 4 shows a sample of design variations based on constructional user requests. The variations are independent of each other, and all based on the initial design (1). Design variation (2) modifies the shape of the profile, changing from a polygon pattern defined by side count and radius to a rectangular pattern defined by width and height. Design variation (3) introduces an angular sole, accomplished by altering the contour curve from a spline to a polyline. In design variation (4), the edges of the sole are smoothed. This change results in two inline parameter adjustments of variables that were not initially included as design parameters. In design variation (5), the strap thickness is reduced, which involves adjusting the thickness of the strap, one of the original design variables.

4.2. Functional interpretation

Figure 5 shows a sample of design variations based on functional user requests. The variations are again independent of each other, and all based on the initial design (1). Design variation (2) asks for “outdoor use,” which results in several design changes aimed at improving protection and grip. Design variation (3) specifies a fit based on user measurements, directly influencing the fit of the sole. Design variation (4) requests a specific shoe size, with length, width, and height determined by the LLM according to its understanding of the request. In design variation (5), a personal statement is used to express a functional demand. By addressing the strap tightness, the height of the strap is increased, which was identified by the LLM itself.

[2] https://openai.com/index/openai-api/

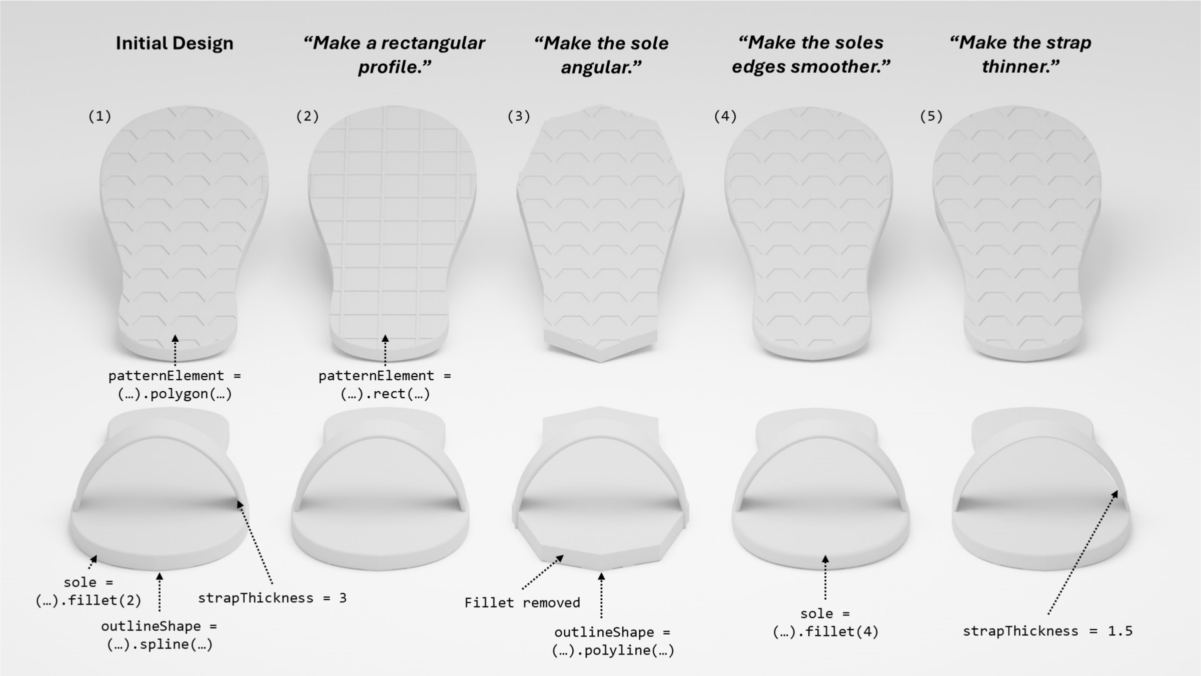

Initial Design
"Make a rectangular profile."
"Make the sole angular."
"Make the soles edges smoother."
"Make the strap thinner."
(1)
(2)
(3)
(4)
(5)
patternElement = (…).polygon(…)
patternElement = (…).rect(…)
sole = (…).fillet(2)
strapThickness = 3
outlineShape = (…).spline(…)
Fillet removed
outlineShape = (…).polyline(…)
sole = (…).fillet(4)
strapThickness = 1.5

Figure 4. Sample results of constructional interpretation in the experiment

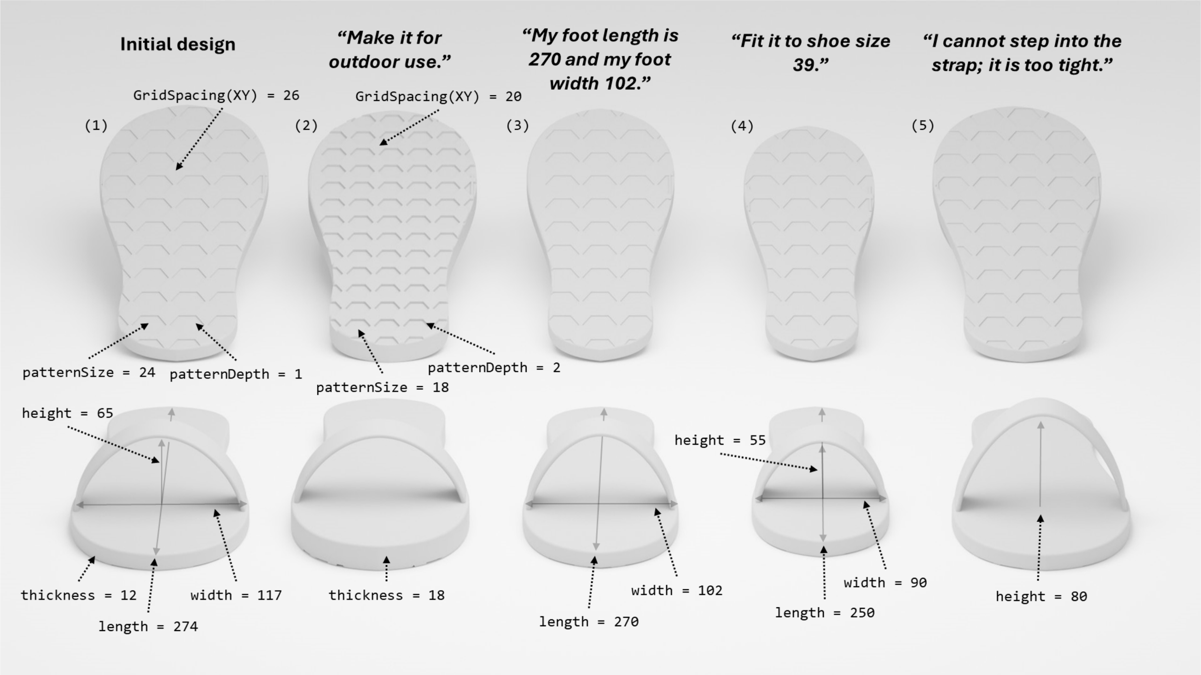


Figure 5. Sample results of functional interpretation in the experiment

4.3. Effective interpretation

Figure 6 shows a sample of design variations based on effective user requests. As before, the variations are independent of each other, and all based on the initial design (1). In design variation (2), the request for reduced friction leads to a decrease in profile depth, which expands the technical description of the sole. Design variation (3) transforms the personal observation of the shoe being too heavy into a weight reduction by decreasing the thickness of the sole. In design variation (4), two simultaneous requests are made, targeting friction and weight, which lead to changes in both profile and sole thickness. In design variation (5), the weight of the strap is specifically addressed, which leads to a reduction in strap width and thickness. The relationship between width, thickness, and consequently weight was not included in the initial technical description of the strap—it is independently identified by the LLM.

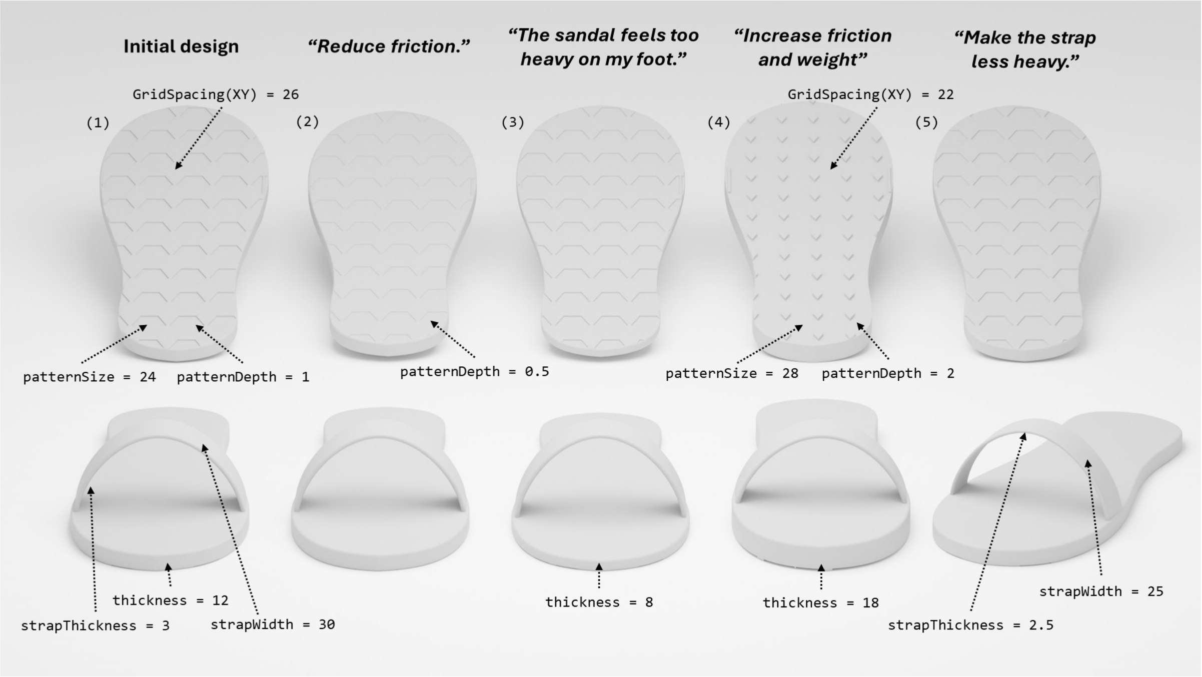


Figure 6. Sample results of effective interpretation in the experiment

5. Discussion

Constructional design changes were highly successful, particularly when made within the boundaries of the design. In some cases, the system was able to extend beyond the initial code by altering fundamental shapes, such as the contour of the sole or the pattern of the profile, without altering the functionality of the product. Incorrect code changes may occur, but in many cases, they can be automatically corrected by feeding the raised error message back into the system.

Functional changes greatly benefit from the LLM's general knowledge. For example, when asked for a different shoe size—a parameter not initially specified—the system demonstrated its ability to integrate standard norms. In addition, the request for "outdoor use" illustrated the system's ability for contextual understanding, as it interpreted and integrated new requirements into the model raised by the request. Effective changes, such as the reduction of friction, were also successfully demonstrated. For example, by connecting the relationship between the sandal's weight and the sole's thickness with the weight of the strap—a relationship not included in the initial design description—the system showed its ability to transfer knowledge on the effective layer from one element to another.

However, due to the nature of the chosen example, addressing the effective layer was underrepresented in the experiment—an example from engineering could better showcase the possibilities. Furthermore, more complex assemblies are also underrepresented in this work, primarily because of the limited token length of the LLM used.

In this work, design validation is carried out manually. Non-functional design changes are not detected automatically and must be manually corrected by the user through additional input or by rephrasing the initial request. Integrating machine learning-based design validation—similar to the methodology used in Query2CAD—could enhance system performance by preventing non-functional design changes.

6. Conclusion

In this paper, augmented design automation was introduced to enhance parametric design workflows by incorporating a semantic layer that interprets and executes functional, constructional, and effective user requests. The integration of LLMs with script-driven CAD kernels created a novel approach to context-sensitive design, guided by natural language. The concept was demonstrated through experiments using CadQuery with a parametric model of a sandal, highlighting the system's ability to generate diverse and meaningful design variations created from abstract prompts. By addressing the limitations of traditional design automation approaches, this concept enables greater flexibility and a more user-centric product development process.

In general, supporting synthesis processes through the use of base code examples can be seen as a means of incorporating expert knowledge into a non-expert system. A specialized, fine-tuned LLM for product creation appears both feasible and desirable. In this way, the development and integration of augmented design systems could be realized. Beyond the technical possibilities, the interfacing capabilities offer an exciting area for research. Spatial understanding, automated data acquisition, as well as user interaction provide a new approach to product creation (Hossain et al., 2024).

This non-specific product development approach could open up new business models and opportunities for companies by extending their product portfolios with individualized solutions, possibly contributing to a more sustainable economy (Briem et al., 2022).

While the system is currently limited to interpreting existing designs, the creation of new and innovative design solutions should be explored in the future.

References

Badagabettu, A., Yarlagadda, S. S., & Farimani, A. B. (2024). Query2CAD: Generating CAD models using natural language queries. *arXiv preprint arXiv:2406.00144*. https://doi.org/10.48550/arXiv.2406.00144

Briem, A. K., Ziegler, D., Mathis, L. A., & Wehner, D. (2022). Sustainable product development by means of personalization–paradox or solution? In *E3S Web of Conferences* (Vol. 349, p. 07001). EDP Sciences. https://doi.org/10.1051/e3sconf/202234907001

Burge, T. A., Jeffers, J. R., & Myant, C. W. (2023). Applying machine learning methods to enable automatic customisation of knee replacement implants from CT data. *Scientific Reports*, 13(1), 3317. https://doi.org/10.1038/s41598-023-30483-5

Chen, B., Zhang, Z., Langrené, N., & Zhu, S. (2023). Unleashing the potential of prompt engineering in large language models: a comprehensive review. *arXiv preprint arXiv:2310.14735*. https://doi.org/10.48550/arXiv.2310.14735

Gu, J., Han, Z., Chen, S., Beirami, A., He, B., Zhang, G.,... & Torr, P. (2023). A systematic survey of prompt engineering on vision-language foundation models. *arXiv preprint arXiv:2307.12980*. https://doi.org/10.48550/arXiv.2307.12980

Hartsock, I., & Rasool, G. (2024). Vision-language models for medical report generation and visual question answering: a review. *Frontiers in Artificial Intelligence*, 7, 1430984. https://doi.org/10.3389/frai.2024.1430984

Hossain, S., Gohil, A., & Wang, Y. (2024). Using LLM such as ChatGPT for designing and implementing a RISC processor: execution, challenges and limitations. *arXiv preprint arXiv:2401.10364*. https://doi.org/10.48550/arXiv.2401.10364

Ponn, J., & Lindemann, U. (2011). Konzeptentwicklung und Gestaltung technischer Produkte. *Systematisch von Anforderungen zu Konzepten und Gestaltlösungen*. Springer. https://doi.org/10.1007/978-3-642-20580-4

Rädler, S., & Rigger, E. (2022). A survey on the challenges hindering the application of data science, digital twins and design automation in engineering practice. *Proceedings of the Design Society*, 2, 1699–1708. https://doi.org/10.1017/pds.2022.172

Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). A systematic survey of prompt engineering in large language models: techniques and applications. *arXiv preprint arXiv:2402.07927*. https://doi.org/10.48550/arXiv.2402.07927

Umland, N., Wiberg, A., Winkler, K., Jung, J., & Inkermann, D. (2024). Enhancing design automation for components of electric machines: a systematic approach. *Proceedings of the Design Society*, 4, 815–824. https://doi.org/10.1017/pds.2024.84

Vogel, M. (2024). The perfect prompt: a prompt engineering cheat sheet. *The Generator*. https://medium.com/the-generator/the-perfect-prompt-prompt-engineering-cheat-sheet-d0b9c62a2bba

Wang, J., Shi, E., Yu, S., Wu, Z., Ma, C., Dai, H.,... & Zhang, S. (2023). Prompt engineering for healthcare: methodologies and applications. *arXiv preprint arXiv:2304.14670*. https://doi.org/10.48550/arXiv.2304.14670

Appendix A

System prompt for design creation

```
# Role
- You act as a program translating human language into CAD models.

# Target
- Update the provided base code according to the user input.

- Use only commands from the Python CadQuery library.
- Limit responses to Python code.

## Base code
’

import CadQuery as cq
from CadQuery import exporters

# Objective: Sandal
# Description: A simple sandal
# Parts: Sole, strap, profile

# General
## Definition of the general shape of the shoe
length = 274 # Length of foot
width = 117 # Width of foot
widthPos = length*0.666 # Position of maximum shoe width
heelWidth = width*0.666 # Calculation of heel width
heelPos = length*0.25 # Position of maximum heel width

## Helper
lengthLinePts = [(0,0),(0,length)] # Points to define the length of the shoe
widthLinePts = [(-width/2,widthPos),(width/2,widthPos)] # Points to define the width of the shoe
heelWidthLinePts = [(-heelWidth/2,heelPos),(heelWidth/2,heelPos)] # Points to define the heel width

# Part: Sole
# Functions: Protection of foot, fit to the foot
# Effects:
   # Protection: A thicker sole leads to more protection.
   # Fit: Defined by length and width of the shoe. Bigger values lead to a looser fit.
   # Weight: A thinner sole leads to a lighter shoe.

thickness = 12 # Thickness of shoe sole

outlinePts =
[lengthLinePts[1],(width/3,(length/3*2.8)),widthLinePts[1],heelWidthLinePts[1],(heelWidth/2,heelPos/
3),(0,0)] # Use the points to create the outline of the shoe.
outlineShape = cq.Workplane(“front”).spline(outlinePts) # Create an outline spline.
soleR = outlineShape.close().extrude(thickness) # Extrude the outline to a half sole.
soleL = soleR.mirror(“YZ”) # Mirror the half-sole.
sole = soleR.union(soleL) # Join both halves into a complete sole.
sole = sole.edges(“<Z”).fillet(2).edges(“>Z”).fillet(2) # Smooth the edges of the sole.

# Part: Strap
# Functions: Attaches sole to the foot, fit to the foot
# Effects:
   # Holding strength: Defined by the strap’s width and thickness
   # Fit: A bigger height leads to a looser instep.
# Strap properties
height = 65 # Height of instep. Increase if foot is too big.
strapWidth = 30 # Width of strap
strapThickness = 3 # Material strength of strap
offset = strapThickness*1.2 # Offset of strap to edge of sole
# Strap construction
strapPathPts = [(-width/2+offset,0),(-width/2+offset,thickness),(0,height),(width/2-offset,thickness),(-width/2-offset,0)] # Create the points for the path of the strap.
strapPath = cq.Workplane(“XZ”).spline(strapPathPts) # Create the path of the strap.
# Create the strap
strap = (cq.Workplane(“XY”).pushPoints([strapPath.val().locationAt(0)]).rect(strapThickness, strapWidth)
   .pushPoints([strapPath.val().locationAt(1)]).rect(strapThickness, strapWidth) # Add the contour shape.
   .consolidateWires()
   .sweep(strapPath, multisection=True) # Sweep the contour along the path.
   .translate((0,widthPos,0)) # Move the strap to its position.
   .edges(“>Y”).fillet(strapThickness*0.9/2) # Smooth the edges of the strap.
   .edges(“<Y”).fillet(strapThickness*0.9/2) # Smooth the edges of the strap.
   )

# Part: Profile
# Functions: Ground grip
# Effects:
   # Friction: Less pattern depth leads to less grip.

xGridSpacing = 26 # Distance pattern in width direction
yGridSpacing = 26 # Distance pattern in length direction
patternSize = 24 # Size of profile grid elements
patternSideCount = 6 # Defines count of grid element polygon
patternDepth = 1 # Depth of pattern inside sole

bb = sole.val().BoundingBox() # Get the size of the sole.
xmin,xmax,ymin,ymax = bb.xmin,bb.xmax,bb.ymin,bb.ymax # Store its limits.
xDistance,yDistance = xmax-xmin, ymax-ymin # Calculate the absolute size.
xSteps = int(xDistance // xGridSpacing) + 1 # Calculate the grid element X count.
ySteps = int(yDistance // yGridSpacing) + 1 # Calculate the grid element Y count.

pattern = [] # List to hold the elements of the grid
patternElement = cq.Workplane(“front”).polygon(patternSideCount,patternSize).extrude(patternDepth)
# The base grid element
for x in range(xSteps): # Create rows.
   for y in range(ySteps): # Create columns.
      position = (x*xGridSpacing-xDistance/2,y*yGridSpacing) # Calculate the element’s position.
      pattern.append(patternElement.translate(position)) # Add to the grid.

# Assembly: Sandal
# Description: The strap is added to the sole. The profile pattern gets cut out.
shoe = sole.union(strap) # Add the strap to the sole.
for ele in pattern:
   try: # Try.
      shoe = shoe.cut(ele) # Create the profile by cutting out the grid elements.
   except: # Continue if fail.
      continue

# Export
result = shoe # Set sandal as result to provide process consistency.
‘
```