CAD-as-Code and Agentic AI for Function-Aware Engineering Design

Elias Berger[1,2*], Jan Mehlstäubl[2], Bernhard Saske[1], Kristin Paetzold-Byhain[1]

[1] Dresden University of Technology, 01062 Dresden
[2] MAN Truck & Bus SE, 80995 Munich
*Corresponding author. E-Mail address: elias.berger@tu-dresden.de

Mechanical Computer-Aided Design (CAD) remains a challenge in product development and engineering design due to steep learning curves. Hence, large language models (LLM) gain popularity to generate parametric geometry programmatically (CAD-as-Code). This paper examines CAD-as-Code as a methodological paradigm with implications for engineering design practice. We present a hybrid agentic framework, that uses CAD-as-Code with physics-in-the-loop and human validation to generate structurally sound designs. We prove our concept by generating physically valid designs based on load bearing requirements. Finally, we derive implications for engineering design practice from our prototype observations.

Keywords:
engineering design automation, generative ai, agentic ai, computer-aided design, cad-as-code

1 Introduction

In engineering design, CAD represents a persistent bottleneck. Translating functional requirements into geometric models requires specialized expertise and substantial time investment, restricting design capabilities to trained specialists. As product complexity increases and development cycles compress, this expertise barrier increasingly constrains innovation velocity and scalability.

Therefore, applying generative artificial intelligence (AI) to CAD is gaining momentum. Recent research has established technical feasibility: LLMs generate geometric models optimized for dataset similarity [1–3], while Vision-Language Models provide visual feedback in iterative loops [4, 5]. These approaches converge on, what we term "*CAD-as-Code*", by defining CAD designs as executable scripts. They treat it instrumentally as an interface convenience rather than examining it as a paradigm with practical implications. This reveals two gaps. Architecturally, current work optimizes for geometric similarity to training data rather than functional correctness. Second, existing work does not examine how CAD-as-Code impacts engineering design conceptually.

This work is a concept paper that examines CAD-as-Code as a methodological paradigm with implications for engineering design. We argue that CAD-as-Code represents a fundamental development, not merely a technical convenience. By formulating engineering design as executable code, it transforms CAD from a human-operated graphical tool into a computational platform with bidirectional programmatic access: AI agents generate parametric models as scripts, and physics-based tools evaluate them deterministically. However, realizing this potential requires addressing several challenges such as limitations in spatial reasoning of LLMs, interoperability with human operators, and questions of trust and professional responsibility.

We provide of proof of concept through a prototype implementation and derive opportunities and challenges from this exploratory development. Our prototype makes use of established knowledge-

based engineering tools for assessing structural integrity, such as Finite Element Analysis (FEA). CAD-as-Code enables closed-loop validation where, rather than training LLMs to approximate physical laws from data, we integrate knowledge-based simulation directly into agentic decision-making made possible by the CAD-as-Code paradigm. This allows us to advance from visual inspection to physics-grounded validation, increasing trust in AI systems.

We discuss the proposed paradigm shift critically by answering the following research questions:

1. Why is CAD-as-Code as a methodological paradigm for AI-assisted mechanical CAD in engineering a promising direction of research?
2. How can CAD-as-Code be applied in a hybrid agentic framework for design automation?
3. What opportunities and challenges emerge from using programmatic CAD together with agentic AI?
4. What are the implications for engineering design practice and professional responsibility?

To answer these questions, we contribute: (1) conceptual positioning CAD-as-Code as aparadigm enabling agentic AI and hybrid design automation, (2) prototype implementation demonstrating technical feasibility through five diverse structural scenarios, (3) empirically grounded analysis of opportunities and challenges derived from the prototype results considering the broader engineering design discipline.

2 Background

2.1 CAD Software as an Engineering Design Tool

Modern CAD systems are built on geometric modeling kernels that represent three-dimensional objects through boundary representation (B-rep) or constructive solid geometry (CSG) approaches [6]. Users interact with these kernels through graphical user interfaces (GUI) that provide visual feedback. However, this GUI-centric interaction presents fundamental challenges for AI automation. CAD GUIs require spatial reasoning about visual layouts, coordinate transformations for mouse positioning, context-dependent menu navigation, and interpretation of graphical feedback. These capabilities remain largely beyond current AI systems [7]. This interface mismatch between AI capabilities and CAD interaction motivates programmatic alternatives like CAD-as-Code.

2.2 Data-based CAD Generation Methods

A comprehensive overview of this development is provided by [8] who systematically categorizes different model types, modalities, and representations. In recent months, the application of generative methods based on LLMs for CAD has gained momentum. Several studies demonstrate the ability of LLMs to generate parametric CAD models from textual [2, 9–12], image [13] or multimodal [1, 3, 14] instructions.

Agentic and iterative generation processes have been recently proposed. Preintner et al. (2025) integrates an evolutionary optimization loop to iteratively improve generated designs based on visual feedback. Similarly, Ocker et al. (2025) propose a Multi-Agent System driven by a VLM to automate CAD model generation by mirroring the structure of human engineering teams with specialized agents. Alrashedy et al. (2025) presents a process that employs commercial VLMs and prompting to generate CAD code and uses visual tests to verify if the CAD object matches the intended shape.

However, the evaluation of these generative CAD models has so far been primarily based on geometric similarity metrics such as Chamfer Distance, Intersection-over-Union (IoU), or Normal Consistency.

These metrics quantify the shape similarity between the generated and reference models but fail to capture functional properties or manufacturability. Studies such as [15] and [3] explicitly confirm this limitation: while geometric consistency improves, the evaluation lacks consideration of functional requirements and design intent like load-bearing capacity, cost, or sustainability.

2.3 Knowledge-Based and Hybrid Systems for Geometry Generation

Knowledge-based generative design systems use explicit engineering rules, physical models, and constraint formulations to create valid, context-sensitive CAD geometries. Unlike data-driven approaches, they ensure physical plausibility, manufacturability, and functional correctness from the outset. Key methods include topology optimization and parametric or rule-based CAD. These techniques are transparent and reliable, especially when data is scarce, but require substantial effort for knowledge acquisition and maintenance [16, 17].

To address the limited physical grounding of purely data-driven generative models, recent work has emphasized the renewed relevance of Knowledge-Based Engineering (KBE) [18]. In contrast to models that infer patterns from data, KBE systems derive valid geometries from explicit engineering knowledge, physical laws, and formalized constraints, enabling context-sensitive solutions aligned with functional requirements [19]. While KBE provides transparency, validity, and robustness it is resource-intensive, requiring substantial upfront knowledge acquisition and ongoing maintenance. Emerging research suggests that the future of generative CAD lies in hybrid systems that couple the creative capabilities of LLMs with the rigor of knowledge-based mechanisms such as simulation-in-the-loop, rule systems, and physics-based boundary conditions, thereby aligning generative performance with engineering objectives [20].

3 Method

This work follows a Design Science Research (DSR) methodology [21] to motivate why CAD-as-Code, combined with the increasing capabilities of agentic AI, may initiate a paradigm shift when applied to engineering design automation. We implemented a prototype to proof the concept across five structural scenarios to demonstrate feasibility and surface key opportunities and challenges. Based on empirical evidence and observations from prototype development, we derive and critically discuss emerging challenges and opportunities and contrast them against existing literature. The study is exploratory: it does not claim statistical validation, with quantitative evaluation deferred to future work.

4 Current state of generative CAD using CAD-as-Code

4.1 Coding approaches for CAD

Currently, CAD software is a visual tool, but modern software libraries enable the creation of CAD models not through GUIs, but by providing code-based interfaces to geometric modeling kernels. Code-based CAD libraries such as CadQuery [22], build123d [23], and RapidCADPy [24] enable a shift toward programmatic, parametric, and reproducible geometry generation in mechanical design. Built on mature solid modelling kernels such as OpenCascade, these systems allow users to define sketches, constraints, features, and Boolean operations through high-level Python abstractions. Further, coding approaches allow for direct integration of other engineering tools such as FEA. Previously, different tools were air-gapped for agentic AI, because of the GUI-focused data flows [7].

Advantages include explicit parameterization, version control compatibility, and seamless integration with computational pipelines—capabilities increasingly exploited in design automation and generative engineering. However, our experience developing the framework presented in this work revealed

practical limitations. Scripted CAD environments often expose only a subset of kernel functionality, making advanced operations such as robust filleting, complex blends, and surface modeling more difficult than in commercial parametric systems. Robustness issues arise from geometric degeneracies and kernel failures that can be challenging to diagnose programmatically. Furthermore, code-based workflows lack the interactive visual feedback and constraint-management of professional CAD environments, limiting their usability for detailed production design requiring frequent visual inspection and manual refinement.

4.2 Code-based CAD Generation using LLMs

Recent studies have increasingly applied LLMs and VLMs to CAD. These approaches follow a common workflow: the DeepCAD dataset [25] or derivatives thereof are used for training. These CAD models are first converted into executable Python code, which enable parametric modeling through code [1]. For Image-To-CAD 2D renderings are produced using a CAD kernel [3, 13] and for Text-To-CAD corresponding textual shape descriptions based on the renderings are generated using a VLM [2, 3]. The results are pairs or triplets of CAD code, image, text that form synthetic training datasets.

Open-source LLMs or VLMs are then fine-tuned on these datasets, effectively learning to reconstruct CAD operations from visual or textual input. The use of these three modalities, code, text, and image, aligns with the capabilities of existing multimodal models like Llama-3, simplifying the training process.

Other approaches directly leverage the multi-task in-context learning ability of LLMs [26]. They leverage commercial LLMs through few-shot prompting, providing example CAD objects as context to generate new parametric models as code [4, 5]. This approach omits the expensive fine-tuning process and enables rapid experimentation with frontier models.

4.3 Optimization Targets of CAD Generation

In AI experiments the algorithm is optimized towards a desired goal. Both fine-tuned and few-shot methods predominantly evaluate outputs using geometric similarity metrics such as Chamfer Distance, Intersection-over-Union (IoU), or visual comparisons using VLMs [4, 5, 15]. These metrics assess shape similarity to training data but do not evaluate functional correctness such as structural integrity, manufacturing feasibility, or performance under load. Evaluation remains geometry-centric rather than function-centric, leaving unexplored the integration of deterministic simulation into generative design loops.

5 A framework for agentic engineering design

We propose an agentic framework that makes use of CAD-as-Code by integrating generative AI, simulation-driven reasoning, and knowledge-based engineering (KBE) into a closed-loop design system (Figure 1). The architecture consists of four core components—Requirements Agent, Engineer Agent, Quality Assurance Agent, and a Knowledge-Based System—all orchestrated by a central workflow controller. The framework combines exploratory, data-driven generation with deterministic engineering validation, enabling scalable and context-aware CAD automation.

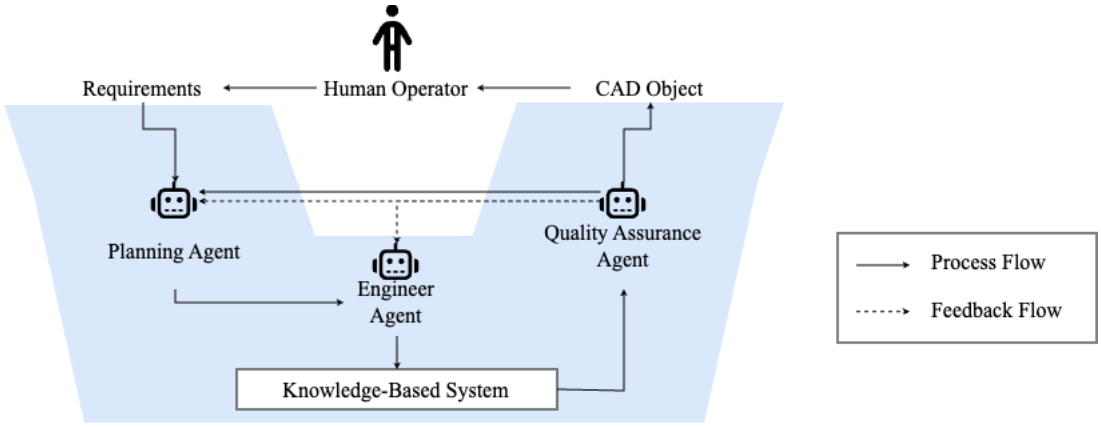

Figure 1: A framework for human-agentic-AI collaboration for engineering design. Under the CAD-as-Code paradigm, agents based on LLMs can create geometries using Python libraries and interact with knowledge-based systems (e.g. Finite-Element Analysis). The quality assurance agent evaluates the CAD design visually and structurally, providing feedback to the planner and engineer agents for iterative refinements. Graphic created by Authors, adapted from [5].

5.1 Planning Agent: Specification Interpretation and Task Decomposition

The Planning Agent interfaces with the human operator and translates high-level requirements into structured design plan, constraints, and parameter ranges. By leveraging natural language understanding and domain-specific rule sets, it decomposes ambiguous or under-specified requirements into actionable engineering tasks. This ensures that early-stage concept generation remains aligned with functional intent and boundary conditions.

5.2 CAD Engineer Agent: Generative Geometry and Iterative Refinement

The Engineer Agent serves as the primary generative component. Using a code-driven CAD backend it produces parametric models that can be validated, edited, and integrated into downstream engineering workflows. Through interaction with deterministic oracles (FEA, CAM, LCA), the system can discover design strategies that are not captured in static datasets, bridging data-driven creativity with physics-based rigor.

5.3 Quality Assurance Agent: Engineering Validation and Multi-Criteria Assessment

The Quality Assurance (QA) Agent evaluates candidate designs using simulation tools and knowledge-based rules, ensuring compliance with constraints such as structural, manufacturability, sustainability, and cost. Based on this assessment, the QA agent also generates feedback for the Engineer Agent, flagging issues such as geometric invalidity, constraint violations, meshing failures, or manufacturing infeasibility. Feedback to the planning agent is raised if the proposed plan fails to meet functional requirements.

5.4 Knowledge-Based System: Physics, Rules, and Domain Logic

At the foundation of the framework lies a deterministic knowledge-based system encompassing engineering rules, ontologies, constraint solvers, manufacturability checks, and physics-based models. It acts as a reliable validation oracle, ensuring safety, consistency, and domain compliance. The KBE layer provides grounding for AI-generated concepts, which remain physically plausible and manufacturable even when training data is scarce.

5.5 Orchestration Layer and Workflow Management

A central orchestrator and workflow engine manages agent communication, task routing, and error handling. It ensures that requirements flow toward generation, evaluation flows back as feedback, and validated designs propagate to the human operator.

6 Proof of Concept

6.1 Setup and Implementation Details

To validate the proposed concept, we implemented the multi-agent system illustrated in Fig. 1. The setup was built using the LangGraph framework, which provides structured orchestration of agent interactions and tool calls. The agent architecture was powered by an LLM backbone based on Anthropic Claude Sonnet 4.5 which handled code generation, reasoning, and dynamic tool utilization. CAD creation and modification were executed programmatically via the RapidCADPy[1] library, allowing agents to construct and iteratively update parametric geometry. For physics-based evaluation, structural simulations were performed using torch-fem[2], which supplied quantitative feedback to the agents and enabled closed-loop refinement of the generated designs.

6.2 Functional Requirements

For our prototype the inputted functions are defined as load bearing requirements. We evaluated the framework across five structural design scenarios with varying geometries, loading conditions, and boundary constraints. The system's objective is to minimize weight, while fulfilling the load bearing requirements. The design specifications were processed by the Requirements Agent, which translated them into a formalized set of engineering constraints for the downstream agents.

6.3 Observations from Prototype Development

Each scenario began with functional requirements specified as load-bearing constraints with applied forces and boundary conditions. The agentic system generated initial designs, evaluated them via FEA given the load bearing requirements, and iteratively refined geometry based on quantitative structural feedback.

The results demonstrate progressive design improvement across iterations (see Figure 2). For example, the bridge scenario (Row 1) evolved from a solid rectangular profile to an optimized beam structure with strategic material removal in low-stress regions, reducing weight while maintaining structural integrity. The tower scenario (Row 2) progressively added internal bracing as the system identified stress concentrations requiring reinforcement. Across all scenarios, the system produced fully parametric, editable CAD models—unlike topology optimization's voxelized density. These models maintain geometric intent, preserve design history, and integrate directly into standard engineering workflows.

7 Discussion

In line with DSR methodology, we treat the prototype as a research artifact and derive opportunities and challenges through reflective abstraction of empirical observations. The resulting implications are then triangulated within existing scientific literature.

7.1 Opportunities

7.1.1 Programmatic CAD for Design Automation

CAD-as-Code enables LLMs to apply their encoded engineering knowledge directly to design generation, bypassing human CAD operators. Beyond leveraging existing VLMs, CAD-as-Code offers structural advantages. Treating models as source code one can apply software development techniques to design: version control, reuse across projects, and modularization into standardized feature libraries. It enhances design reproducibility and adaptability, as parameterized scripts can regenerate variants

[1] https://github.com/rapidcad/rapidcadpy
[2] https://github.com/meyer-nils/torch-fem

automatically. Moreover, code-based models integrate seamlessly with simulation and manufacturing pipelines through application programming interfaces (APIs), supporting automated and reproducible engineering workflows.

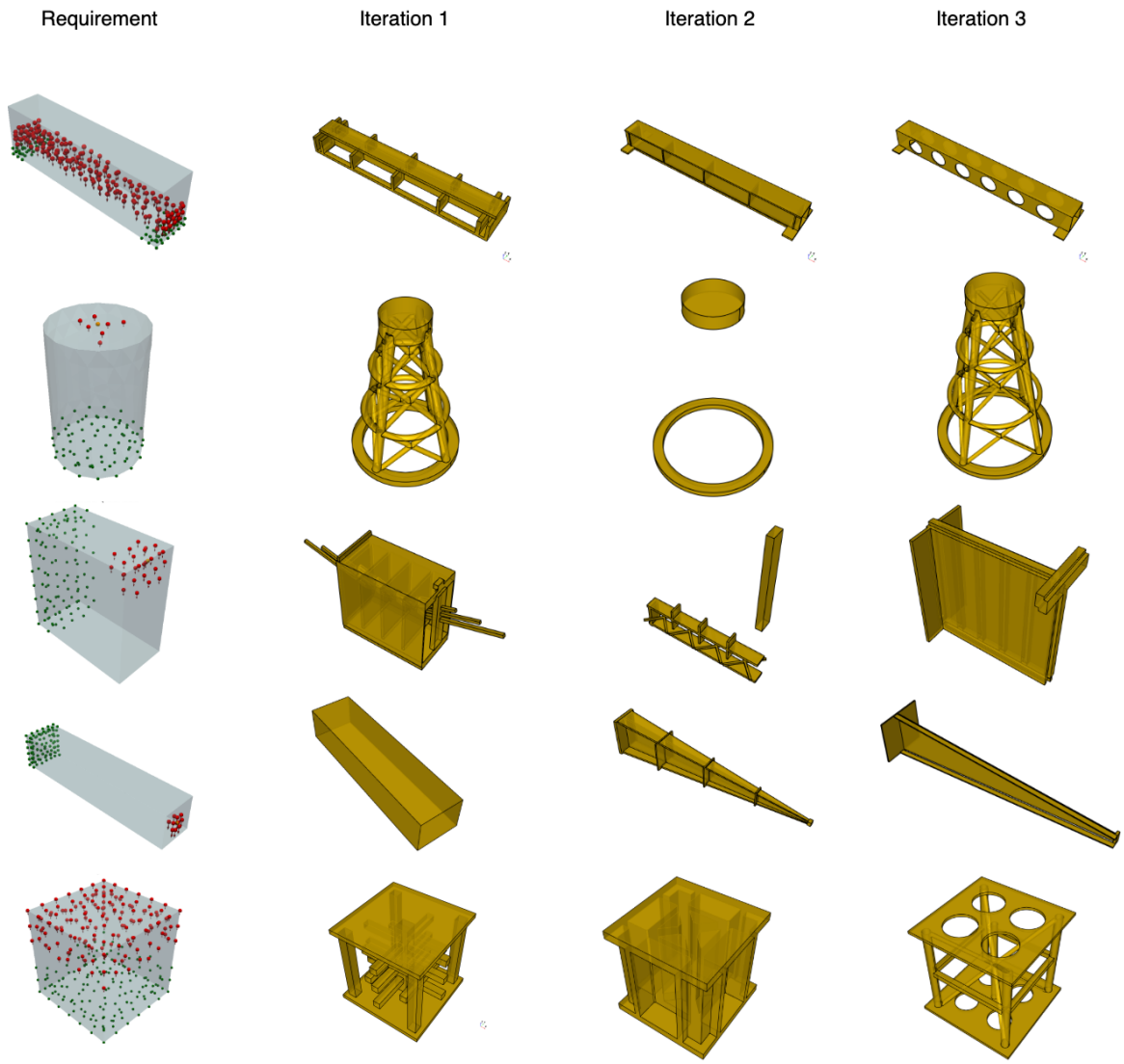

Figure 2: Left column: visualization of the load bearing requirements with applied forces in red and fixed constraints in green. This is the input for our system. Right columns: rendered outputs of each iteration from the hybrid agentic prototype, optimizing for least volume while fulfilling the load bearing requirements.

7.1.2 Agent-Based Frameworks for Autonomous CAD Generation

A particularly promising direction in AI-assisted design is the use of multiple specialized agents working collaboratively within a coordinated framework. In such a system, distinct agents assume different roles. For example, one agent may generate a concept shape based on input requirements, another may evaluate the design against engineering constraints using simulation tools or rule-based systems, and a third may perform optimization. These agents operate within a data-driven and knowledge-driven process, with communication and task coordination managed by a central orchestrator. Recent advances such as Model Context Protocols (MCP, Anthropic, 2024) and tool calling improve the ability of AI systems to interact with external software and tools, enabling agents not only to plan design workflows but also to execute them in real-world engineering environments.

Agent-based frameworks offer key advantages beyond automation. Agents can run in parallel, enabling simultaneous design exploration, accelerated optimization, and continuous design-simulation cycles

without human intervention. By abstracting domain knowledge into modular behaviors, these systems lower barriers for non-experts—system engineers, product managers, or simulation specialists can generate prototypes or request modifications without deep CAD expertise. This broadens participation in early-phase design and accelerates iterative development. Robust AI-CAD interfaces are therefore critical for enabling autonomous, scalable, and accessible design automation.

7.1.3 Comparative Advantages of Agent-Driven Design over Topology Optimization

The interaction between engineering agent, quality assurance agent, and FEA resembles topology optimization—both iteratively refine designs based on performance feedback. However, the agent-based approach offers key advantages. First, it generates parametric, fully editable CAD models rather than voxelized density fields requiring extensive post-processing, maintaining compatibility with standard engineering workflows. Second, agents can accumulate knowledge across iterations, improving efficiency and enabling generalization—capabilities topology optimization lacks. Third, multi-agent architecture enables explicit multi-criteria reasoning, incorporating manufacturability, cost, sustainability, and safety alongside structural performance. These properties position agent-based systems as a more flexible alternative to topology optimization for industrial contexts requiring interpretable, modifiable, workflow-integrated outputs.

7.2 Challenges

7.2.1 Implementation challenges

A key limitation in current AI-driven CAD systems lies in data availability since research datasets lack the complexity found in industrial-grade CAD models [28]. This limits the training and generalization capabilities of AI models. The data-related findings align with prior reviews that identify data scarcity and heterogeneity as key bottlenecks in data-driven design and manufacturing [29, 30]. Further, LLM-driven reasoning is computationally intensive and introduces substantial execution overhead. However, the future of agentic AI lies in small, specialized LLMs that offer faster inference [31].

7.2.2 Ethical and Trust Considerations

The explainability of AI agent decision is key for the system to explain why a certain design was generated. Without such transparency, human designers may be rightly hesitant to trust AI suggestions. Initial user studies and workshops will be valuable to gauge how designers interact with these tools and what level of autonomy is acceptable for *trust* and *usability*, which mirror broader findings in human-AI interaction research emphasizing transparency, accountability, and calibrated trust as prerequisites for successful adoption [32–34]. The consensus in the field is that near-term systems will be hybrid, where AI agents assist but do not replace humans, combining the creativity and intuition of humans with the speed and computational power of AI [35, 36]. Achieving the right balance will be both a technical and a professional challenge.

7.2.3 Engineering Knowledge Limitations

Despite recent advances, current AI-driven design systems exhibit limitations in their ability to replicate the domain-specific reasoning and contextual understanding that human engineers routinely apply. Although large language models can articulate engineering best practices when prompted, they often struggle to operationalize these principles within the design process itself [37]. This discrepancy suggests that LLMs lack robust mechanisms for integrating abstract engineering knowledge with spatial reasoning [38, 39]. We hypothesize that these shortcomings stem from the inherently text-based training of LLMs, which provides limited exposure to geometric data. Consequently, LLMs may fail to internalize the causal and spatial dependencies required for consistent, functionally grounded design decisions.

8 Conclusion

This work positions CAD-as-Code as a methodological paradigm for engineering design, moving beyond its instrumental use as an CAD interface. It demonstrates that the CAD-as-Code paradigm enables agentic AI architectures to operate professional CAD software and solve engineering tasks. By integrating a multi-agent workflow with FEA, the proposed system moves beyond the limitations of geometric similarity metrics to produce structurally valid designs. Our experiments validate the feasibility of our concept. However, challenges regarding computational overhead and trust remain. Ultimately, Agentic AI is best positioned not as a replacement for human expertise, but as a sophisticated co-pilot that democratizes access to complex engineering tools and accelerates the design process.

References

[1] A. C. Doris, M. F. Alam, A. H. Nobari, and F. Ahmed, "CAD-Coder: An Open-Source Vision-Language Model for Computer-Aided Design Code Generation," May 20, 2025, *arXiv*: arXiv:2505.14646. doi: 10.48550/arXiv.2505.14646.

[2] M. S. Khan, S. Sinha, T. U. Sheikh, D. Stricker, S. A. Ali, and M. Z. Afzal, "Text2CAD: Generating Sequential CAD Models from Beginner-to-Expert Level Text Prompts," Sep. 25, 2024, *arXiv*: arXiv:2409.17106. doi: 10.48550/arXiv.2409.17106.

[3] J. Xu, Z. Zhao, C. Wang, W. Liu, Y. Ma, and S. Gao, "CAD-MLLM: Unifying Multimodality-Conditioned CAD Generation With MLLM," Mar. 13, 2025, *arXiv*: arXiv:2411.04954. doi: 10.48550/arXiv.2411.04954.

[4] K. Alrashedy, P. Tambwekar, Z. Zaidi, M. Langwasser, W. Xu, and M. Gombolay, "Generating CAD Code with Vision-Language Models for 3D Designs," Feb. 28, 2025, *arXiv*: arXiv:2410.05340. doi: 10.48550/arXiv.2410.05340.

[5] F. Ocker, S. Menzel, A. Sadik, and T. Rios, "From Idea to CAD: A Language Model-Driven Multi-Agent System for Collaborative Design," Mar. 06, 2025, *arXiv*: arXiv:2503.04417. doi: 10.48550/arXiv.2503.04417.

[6] Ian. Stroud, *Boundary representation modelling techniques*, 1st ed. 2006. London: Springer, 2006. doi: 10.1007/978-1-84628-616-2.

[7] K. Cheng *et al.*, "SeeClick: Harnessing GUI grounding for advanced visual GUI agents," in *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 9313–9332. [Online]. Available: https://aclanthology.org/2024.acl-long.505

[8] W. Zhang, W. Chen, and L. Zhao, "A survey on generative AI for CAD and mechanical design," *Comput.-Aided Des.*, 2025.

[9] P. Govindarajan, D. Baldelli, J. Pathak, Q. Fournier, and S. Chandar, "CADmium: Fine-Tuning Code Language Models for Text-Driven Sequential CAD Design," Jan. 08, 2026, *arXiv*: arXiv:2507.09792. doi: 10.48550/arXiv.2507.09792.

[10] C. Lv and J. Bao, "CADInstruct: A multimodal dataset for natural language-guided CAD program synthesis," *Comput.-Aided Des.*, vol. 188, p. 103926, 2025, doi: 10.1016/j.cad.2025.103926.

[11] M. Usama, M. S. Khan, D. Stricker, and M. Z. Afzal, "NURBGen: High-fidelity text-to-CAD generation through LLM-driven NURBS modeling." 2025. [Online]. Available: https://arxiv.org/abs/2511.06194

[12] S. Wang *et al.*, "CAD-GPT: Synthesising CAD Construction Sequence with Spatial Reasoning-Enhanced Multimodal LLMs," *Proc. AAAI Conf. Artif. Intell.*, vol. 39, no. 8, pp. 7880–7888, Apr. 2025, doi: 10.1609/aaai.v39i8.32849.

[13] M. F. Alam and F. Ahmed, "GenCAD: Image-Conditioned Computer-Aided Design Generation with Transformer-Based Contrastive Representation and Diffusion Priors," Apr. 08, 2025, *arXiv*: arXiv:2409.16294. doi: 10.48550/arXiv.2409.16294.

[14] M. Kolodiazhnyi *et al.*, "cadrille: Multi-modal CAD Reconstruction with Online Reinforcement Learning," Sep. 19, 2025, *arXiv*: arXiv:2505.22914. doi: 10.48550/arXiv.2505.22914.

[15] T. Preintner, W. Yuan, A. König, T. Bäck, E. Raponi, and N. van Stein, "EvoCAD: Evolutionary CAD code generation with vision language models." 2025. [Online]. Available: https://arxiv.org/abs/2510.11631

[16] K. Albrecht and R. Anderl, "Information model for the integration of manufacturing restrictions into the algorithm based product development process," *Procedia CIRP*, vol. 50, pp. 819–824, 2016, doi: 10.1016/j.procir.2016.04.136.

[17] K. Herrmann, O. Altun, P. Wolniak, I. Mozgova, and R. Lachmayer, "Methodischer aufbau von entwicklungsumgebungen nach dem generative parametric design approach," in *Proceedings of the 32nd symposium design for X, DFX 2021*, 2021. doi: 10.35199/dfx2021.14.

[18] K. Herrmann, B. Bode, J. Wurst, P. C. Gembarski, I. Mozgova, and R. Lachmayer, "Quantification of the material-related environmental impact of topology-optimized multi-material components," in *Proceedings of the 33rd symposium design for X, DFX 2022*, 2022. doi: 10.35199/dfx2022.01.

[19] G. Pahl, W. Beitz, J. Feldhusen, and K.-H. Grote, *Engineering design: a systematic approach*. Springer, 2007.

[20] E. Berger, K. Herrmann, F. Pusch, J. Mehlstäubl, R. Lachmayer, and K. Paetzold-Byhain, "From geometry to function: Towards context-aware generative AI for engineering design," in *Proceedings of the DESIGN conference*, Dubrovnik, Croatia, 2026.

[21] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," 2004.

[22] AU *et al.*, *CadQuery/cadquery: CadQuery 2.4.0*. (Jan. 2024). Zenodo. doi: 10.5281/zenodo.10513848.

[23] Gumyr and build123d contributors, *build123d: A python CAD programming library*. (2024). GitHub. doi: 10.5281/zenodo.XXXXXXX.
[24] RapidCAD Contributors, *RapidCADPy: Python interface for Autodesk Inventor*. (2025). [Online]. Available: https://github.com/rapidcad/rapidcadpy
[25] R. Wu, C. Xiao, and C. Zheng, "DeepCAD: A Deep Generative Network for Computer-Aided Design Models," Aug. 16, 2021, *arXiv*: arXiv:2105.09492. doi: 10.48550/arXiv.2105.09492.
[26] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.
[27] Anthropic, "Introducing the model context protocol." Nov. 25, 2024.
[28] E. Berger, M. P. Dammann, J. Mehlstäubl, B. Saske, F. Braun, and K. Paetzold-Byhain, "Challenges and opportunities in the integration of generative AI with computer-aided design," *Proc. Des. Soc.*, vol. 5, pp. 881–890, 2025, doi: 10.1017/pds.2025.10102.
[29] J. Luo, L. Zhao, and W. Chen, "A review of deep learning applications in engineering design: Data, models, and opportunities," *J. Mech. Des.*, vol. 144, no. 12, 2022.
[30] X. Zha and X. Du, "Challenges and opportunities of AI-driven design in manufacturing systems," *Comput. Ind.*, vol. 148, p. 103844, 2023.
[31] P. Belcak *et al.*, "Small Language Models are the Future of Agentic AI," Sep. 15, 2025, *arXiv*: arXiv:2506.02153. doi: 10.48550/arXiv.2506.02153.
[32] S. Amershi *et al.*, "Guidelines for human-AI interaction," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–13.
[33] M. Eiband, H. Schneider, M. Bilandzic, D. Fazekas-Con, M. Haug, and H. Hussmann, "Bringing transparency design into practice," in *Proceedings of the 23rd international conference on intelligent user interfaces*, 2018, pp. 211–223.
[34] D. Wang, Q. Yang, A. Abdul, and B. Y. Lim, "Designing for calibrated trust in human–AI teams," *Proc. ACM Hum.-Comput. Interact.*, vol. 4, no. CSCW1, pp. 1–26, 2020.
[35] M. H. Jarrahi, C. Lutz, and G. Newlands, "Artificial intelligence, human intelligence and hybrid intelligence based on mutual augmentation," *Big Data Soc.*, vol. 9, no. 2, p. 20539517221142824, Jul. 2022, doi: 10.1177/20539517221142824.
[36] B. Shneiderman, "Human-Centered Artificial Intelligence: Reliable, Safe & Trustworthy," *Int. J. Human–Computer Interact.*, vol. 36, no. 6, pp. 495–504, Apr. 2020, doi: 10.1080/10447318.2020.1741118.
[37] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *ArXiv Prepr. ArXiv170208608*, 2017.
[38] S. Ahmed and Y. Reich, "Human-AI collaboration in engineering design: Opportunities and challenges," *J. Eng. Des.*, vol. 34, no. 3, pp. 179–199, 2023.
[39] C. McComb, J. Linsey, and J. Cagan, "Limitations of data-driven generative design in engineering practice," *Des. Stud.*, vol. 79, p. 101091, 2022.