CAD-VIGA: Agentic Vision-Guided Refinement for Parametric CAD Generation

G. Abdullin A. Blinov A. Narimov R. Gilmetdinov M. Mousatat

Innopolis University

Abstract

Generating useful computer-aided design (CAD) models from reference images requires executable, editable parametric programs rather than only plausible three-dimensional silhouettes. We present CAD-VIGA, an agentic workflow in which a general-purpose coding agent plans a target object, writes CadQuery code, inspects multi-view renderings and geometric diagnostics, and revises the model through an execution-grounded feedback loop.

We compare CAD-VIGA with a one-shot generation baseline on a difficulty-balanced 15-example subset of the BenchCAD Vision2Code task, using three independent attempts per example and method. CAD-VIGA produces executable models in 45 of 45 attempts (100%), compared with 17 of 45 attempts (37.8%) for the one-shot baseline. Conditional on successful execution, CAD-VIGA achieves a mean Voxel IoU of 0.3800 and a mean BenchCAD composite score of 0.5073, while the baseline achieves 0.4127 and 0.5236, respectively. CAD-VIGA therefore improves execution reliability but does not improve conditional geometric similarity in this experiment, and its mean runtime is approximately 7.4 times higher. These results show that pipeline completion, geometric fidelity, and computational cost must be reported separately for tool-mediated CAD agents.

1 Introduction

Computer-aided design is a central representation for engineering artifacts because it stores shapes as editable construction procedures rather than only as surface samples. A CAD model can encode sketches, extrusions, cuts, fillets, holes, repeated patterns, and part relationships, making it more suitable for manufacturing-oriented workflows than meshes or point clouds. Recent research has therefore shifted from generic 3D generation toward systems that recover or synthesize CAD construction sequences from images, text, or other geometric signals.

The difficulty is that CAD generation is both visual and symbolic. A generated model should look like the requested object, but it must also compile, have valid topology, preserve the intended holes and cutouts, and remain editable as parametric code. Prior work illustrates several complementary approaches. DeepCAD treats CAD construction as a sequential modeling language and trains a Transformer over operation histories [16]. Text2CAD extends this line by generating sequential CAD models from natural-language prompts of different expertise levels [7]. CAD-GPT combines image or text conditioning with a spatial reasoning mechanism for CAD command sequence synthesis [14]. These systems are important because they show that CAD programs can be learned or predicted from large training corpora. However, our project deliberately explores a different question: given a general-purpose coding agent, can CAD generation be improved by surrounding the agent with execution, rendering, verification, and artifact-management tools instead of training a new CAD-specific model?

Agentic CAD systems take a different route. Instead of asking a model to produce the final CAD program in one pass, they place a language or vision-language model inside a feedback loop: write code, execute it, render or validate the result, then revise. Adler et al. describe this broader family as Agent-Aided Design and introduce the AADvark prototype, arguing that CAD

agents benefit from strong verification signals such as compilation errors, solver feedback, and renderings [1]. VIGA develops a closely related inverse-graphics loop for Blender scenes: the agent edits executable symbolic programs, observes rendered outcomes, compares them with the target, and carries an evolving multimodal memory across iterations [21]. CAD-VIGA follows this same core insight but moves the target domain from general 3D scenes to parametric CAD code.

Our project focuses on CAD generation from a prompt together with a reference image. The reference image acts as the visual specification for topology, feature placement, orientation, and proportions, while the prompt supplies semantic and functional intent. CAD-VIGA exposes this task through the Model Context Protocol (MCP): an MCP-capable coding agent calls CAD-specific tools, submits CadQuery code to a backend, receives rendered views and mesh statistics, and stores artifacts for later inspection. The workflow is deliberately engineering-oriented. It requires an up-front plan, an independent plan verification step, iterative execution and visual verification, semantic labels for meaningful parts, and a final comparison gate. These components are designed to reduce common failure modes such as over-cutting, misplaced holes, floating subparts, and accepting a watertight but semantically wrong solid.

This work makes three principal contributions. First, we present an execution-grounded MCP toolchain that connects a general-purpose coding agent to parametric CAD execution, multi-view rendering, geometric diagnostics, and structured experiment artifacts. Second, we provide a controlled, repeated comparison with one-shot generation on a difficulty-balanced BenchCAD subset: 90 total attempts separate the reliability benefit of the full workflow from its conditional geometric quality and runtime cost. Third, we introduce a stage-separated evaluation methodology that reports artifact delivery, conditional reconstruction quality, stochastic variance, and computational cost independently, preventing executable-code reliability from being conflated with geometric accuracy.

2 Related Work

2.1 CAD as a Generative Sequence

DeepCAD introduced a deep generative model for CAD construction sequences, representing shapes as ordered operations rather than as voxels, point clouds, or polygon meshes [16]. This formulation is important for our work because CAD-VIGA also treats the executable construction program as the primary artifact. However, CAD-VIGA does not train a new sequence model. It uses a general coding agent to synthesize and revise CadQuery code through execution and visual feedback.

Text2CAD further emphasizes natural-language control over CAD sequence generation [7]. Text2CAD is especially relevant because it studies beginner-to-expert textual prompts, showing that language alone can carry useful design intent. CAD-VIGA uses language in this same spirit, but the language prompt is paired with a reference image and placed inside an execution loop. The agent can therefore compare the generated result against a visual target rather than rely only on a text prompt.

CAD-GPT uses a multimodal large language model to synthesize CAD construction sequences from either a single image or a textual description, and introduces special spatial tokens for sketch-plane orientation, 3D location, and 2D sketch coordinates [14]. This supports the motivation for CAD-VIGA's explicit attention to spatial reasoning. Our system does not train a new multimodal CAD model or add learned spatial tokens; instead, it makes spatial reasoning external and inspectable through planning text, multi-view semantic color render feedback, and verifier reports.

The emergence of large programmatic CAD datasets made sequence-based learning practical. The Fusion 360 Gallery introduced 8,625 human-authored sketch-and-extrude construction sequences and an environment for programmatic CAD reconstruction [15]. At the sketch

level, SketchGen models heterogeneous primitives and geometric constraints as a sequential language [11]. SkexGen subsequently separated topology, geometry, and extrusion variation into disentangled codebooks [19], while Hierarchical Neural Coding represented design variation at multiple levels to support controllable generation and completion [20]. These systems demonstrate the value of structured construction histories, although they do not provide the execution-and-render feedback loop used by CAD-VIGA.

2.2 Agentic Inverse Graphics

VIGA frames inverse graphics as an interleaved multimodal reasoning process in which an agent alternates between symbolic program edits and rendered visual observations [21]. A key similarity between VIGA and CAD-VIGA is their use of executable programs and rendered feedback. In both systems, executable code is the action space, rendered images expose spatial discrepancies, and feedback guides the next iteration. The main difference is the representation and success condition. VIGA targets Blender scene reconstruction, where objects, materials, and spatial arrangement are rendered as scene code. CAD-VIGA targets CadQuery solids and assemblies, where the output must be executable CAD code and where holes, booleans, part interfaces, and feature semantics matter.

Adler et al. describe agent-aided design for dynamic CAD models and identify the same high-level loop: an agent writes code or structured assembly definitions, compiles them, visualizes the result, and iteratively refines the design from feedback [1]. Their system focuses on dynamic assemblies with joints and moving parts, using FreeCAD and a constraint solver. CAD-VIGA currently focuses on CadQuery code refinement for static parametric CAD models and assemblies, but it adopts the broader AAD principle that CAD agents should be coupled to strong domain tools rather than judged only by a language model's first response.

Recent code-centric and verification-driven systems further motivate our design. CAD-Recode maps point clouds to executable Python CAD programs and demonstrates that its generated code can support interpretation and downstream editing [12]. CADCodeVerify uses vision-language validation questions and corrective visual feedback to improve generated CAD code [2]. CADSmith combines execution repair, exact kernel measurements, and a visual judge [3], while IterCAD formulates generation and editing as closed-loop interaction with an executable CAD sandbox [6]. These systems place CAD-VIGA within the broader shift from one-shot generation toward execution-grounded and iteratively verified CAD agents.

2.3 Geometry-to-CAD Reconstruction

A complementary line of work recovers editable CAD structure from existing geometry. Zone Graphs combines geometric partitioning with learned search to infer sketch, extrusion, and Boolean sequences from boundary representations [18]. Point2Cyl decomposes point clouds into extrusion cylinders [13], and SECAD-Net learns sketch–extrude reconstruction without direct sequence supervision [8]. More recently, Ma et al. reconstruct CAD construction sequences from point clouds using multimodal diffusion [10], while Point2CAD combines learned segmentation with analytic surface fitting to recover CAD topology [9]. These methods primarily assume point-cloud or B-rep evidence; CAD-VIGA instead starts from rendered images and uses repeated execution to test a generated program.

2.4 Language Models for Executable CAD Code

Recent work increasingly generates general-purpose CAD scripts rather than closed command vocabularies. Text-to-CadQuery translates natural-language descriptions directly into CadQuery and studies scaling across open language models [17]. CAD-Coder fine-tunes a vision-language model to produce editable CadQuery programs from images [4]. These approaches are especially

close to CAD-VIGA in output representation; our focus differs in using a general coding agent with an external planning, execution, rendering, and verification loop rather than relying only on a single learned mapping from an input to code.

3 Approach

3.1 System Overview

CAD-VIGA is implemented as a local MCP server that connects an MCP-capable coding agent to a CAD execution backend. The agent receives a prompt and, optionally, a reference image. The current code-refinement workflow exposes the following core tool sequence:

```
set_reference_image / generate_from_image
initialize_plan, verify_plan
execute_and_evaluate, verify, investigate / undo_last_step
compare_and_replan, compute_metrics, end
```

Together these tools register the reference image, create and verify a four-phase geometric plan, execute CadQuery code, inspect rendered views, record verifier feedback, optionally compute metrics when a reference STL exists, and run a final comparison gate before finalizing an experiment folder. If `compare_and_replan` blocks acceptance, the next step is not to stop; the agent starts a targeted correction plan, validates it with `verify_plan`, executes the focused revision, and compares again. Earlier planning helpers such as cumulative step planners are still useful for some experiments, but the main CAD-VIGA workflow now begins with `initialize_plan` and immediately validates that plan with `verify_plan`.

The backend executes CadQuery code, exports mesh or solid artifacts, computes mesh statistics, and renders six camera views. The local MCP server converts rendered PNG images into MCP image content so the agent can inspect its own output. The workflow stores the plan, generated code, rendered images, STL artifacts, mesh statistics, verifier reports, and experiment summaries. This artifact trail is useful because CAD failures are often not visible from a single final model: a later valid mesh can still encode the wrong feature semantics if the agent over-cuts a pocket or moves a repeated feature away from its intended centerline.

3.2 Planning Before Generation

The first design decision is to require a structured plan before code generation. The plan records the primary shape family, approximate bounding box, complete feature inventory, functional interpretation, proportional ratios, construction strategy, boolean operation order, pattern operations, and phase-based build sequence. This step is inspired by VIGA's explicit reasoning trajectory but adapted to CAD-specific constraints: the agent must identify which features are additive, subtractive, repeated, cosmetic, or functional before writing CadQuery.

CAD-VIGA also includes a mandatory plan-verification step. The agent rechecks the reference image against its own proposed inventory and records hallucinations or corrections. This is intended to reduce a common failure mode in image-guided CAD generation: the model invents plausible engineering features that are not actually visible, then spends later iterations preserving those hallucinated features.

3.3 Iterative Code, Rendering, and Verification

After planning, the agent enters the generation loop. In each iteration it submits complete CadQuery code along with a short explanation of the revision. The backend executes the code and returns rendered views plus mesh statistics. The agent then records verifier feedback before making the next revision.

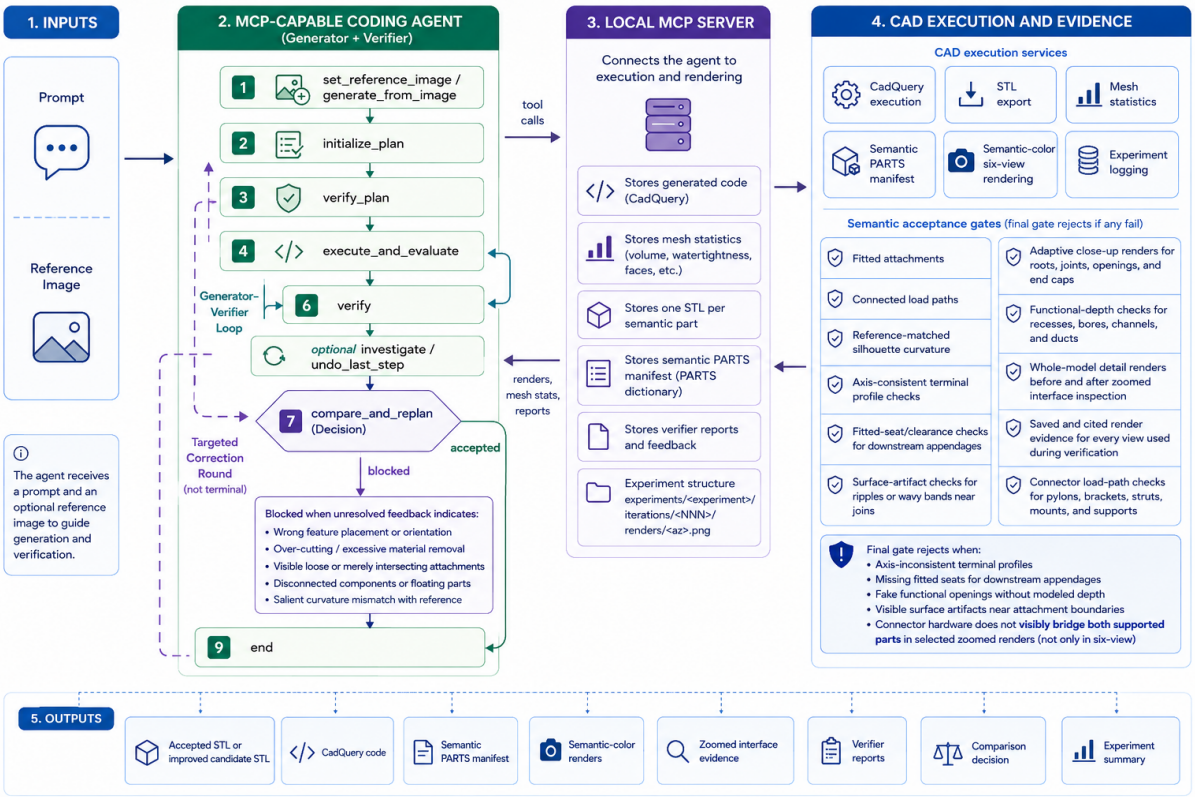


Figure 1: CAD-VIGA pipeline. A general-purpose coding agent uses MCP tools to register a prompt and reference image, create a four-phase geometry plan, verify that plan, generate CadQuery code, execute and render the model, inspect semantic colored feedback, and refine the code through verifier reports. A blocked comparison gate sends the agent into a targeted correction round; an accepted gate finalizes the STL, code, renders, reports, and summary. The figure summarizes the workflow at a conceptual level; the implementation exposes the tool names listed in the text.

This loop is the direct point of contact with VIGA. VIGA uses symbolic scene programs and rendered observations to drive visual-to-code feedback; CAD-VIGA uses CadQuery programs and rendered CAD views for the same iterative pattern. The difference is that CAD-VIGA's feedback must include CAD-specific signals: whether the code executed, whether the geometry has finite volume, whether features are attached, whether semantic parts are present, and whether holes or reliefs match the intended cut type.

3.4 Semantic Part Labels and Assembly Interfaces

For multi-part outputs, CAD-VIGA supports a top-level `PARTS` dictionary with stable semantic labels. The backend can export labeled parts and render them with distinct colors, allowing verifier feedback to refer to specific components rather than visually ambiguous regions. This mechanism is influenced by AADvark's use of enhanced renderings and instance labels to strengthen visual grounding [1]. It is part of the broader CAD-VIGA system design, but the present BenchCAD evaluation consists primarily of individual mechanical parts and does not separately measure the effect of semantic labeling or assembly-interface verification.

3.5 Validity as a Sanity Gate

CAD-VIGA treats successful execution, finite volume, and watertightness as necessary but insufficient conditions. An executable model may still place a hole on the wrong face, convert a blind

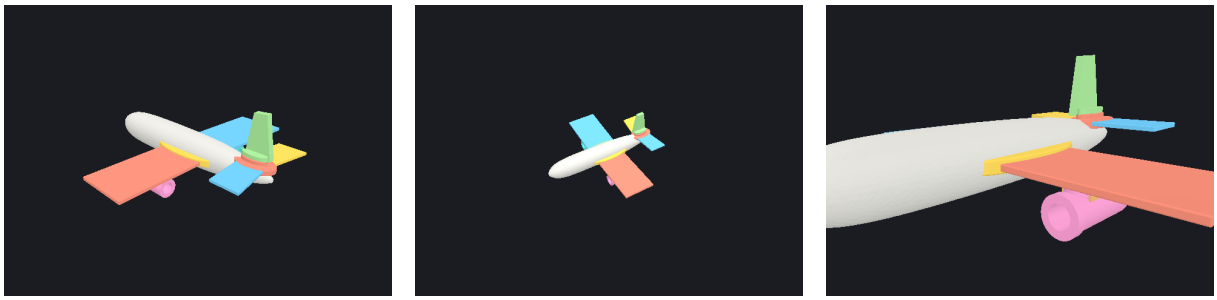


Figure 2: Semantic-color rendering example from the latest CAD-VIGA experiment. The generated twin-engine aircraft is rendered in solid mode from three selected color-mode camera angles. Distinct colors correspond to logical `PARTS` entries such as the fuselage, left and right wings, engines, pylons, vertical tail, and stabilizers.

relief into a through-cut, use incorrect proportions, or omit a reference-critical feature. Verification therefore considers feature placement and cut type in addition to low-level mesh validity. The BenchCAD experiment evaluates final geometry with reference-based metrics, but it does not isolate the contribution of individual semantic or assembly-specific verification checks.

4 Experiments

4.1 Research Questions

The evaluation addresses four questions: (RQ1) whether CAD-VIGA produces executable parametric models from multi-view images; (RQ2) whether its execution reliability differs from a controlled one-shot baseline; (RQ3) how the two methods compare in geometric similarity; and (RQ4) how reliability, quality, variance, and runtime change across BenchCAD difficulty levels.

4.2 Benchmark and Sampling

We use the BenchCAD Vision2Code task, an execution-grounded benchmark containing 17,900 execution-verified CadQuery programs across 106 industrial part families [22]. Each selected record supplies a 2×2 composite of four rendered views, family metadata, a difficulty label, and reference CadQuery code. We execute the reference code locally to obtain a STEP solid for scoring; neither the code nor the STEP solid is exposed to the generation method.

We sample five records from each of the `easy`, `medium`, and `hard` levels after a streaming shuffle with buffer 512 and seed 42. The same 15 records and reference images are used by both methods. Each method is run independently three times per record, producing 45 CAD-VIGA attempts and 45 baseline attempts. The repeats measure stochastic generation variability; the sampling seed still fixes only one small subset of BenchCAD.

4.3 Compared Methods

CAD-VIGA. The coding agent receives the four-view image and a generic mechanical-CAD instruction, constructs and checks a geometric plan, generates CadQuery code, uses MCP-backed execution and rendering, and may revise the program from visual and geometric feedback.

One-shot baseline. The baseline uses the same coding model, input image, generic reconstruction instruction, selected records, CadQuery execution environment, and BenchCAD scorer. It requests one complete CadQuery program without CAD-VIGA planning, MCP rendering, verifier feedback, or iterative revision. Three independent one-shot calls are made for each record; "one-shot" describes the absence of an internal refinement loop, not the number of repeated trials.

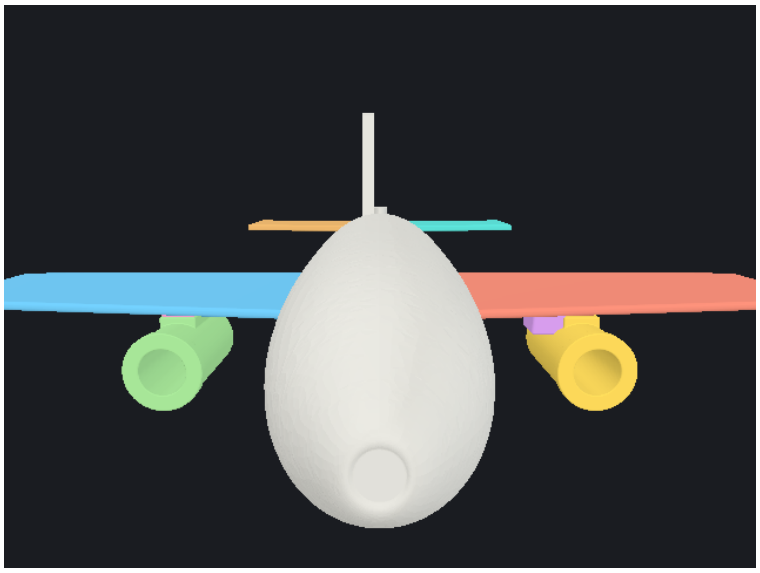


Figure 3: Example verifier failure case from an intermediate aircraft run. The overall aircraft outline appears roughly left–right symmetric, but the detailed geometry is not reliably symmetric: engine and pylon details drift between sides, and the fuselage contains an erroneous circular depression on the nose-facing body. This render motivated explicit checks for symmetric detail placement and fuselage surface validity, rather than accepting a model from coarse silhouette similarity alone.

4.4 Implementation Details

Component	Configuration
Coding agent	Codex CLI with `gpt-5.6-sol`
CAD stack	Python 3.11; CadQuery 2.3.0; cadquery-ocp 7.9.3.0
Runtime	Docker image based on `python:3.11-slim-bookworm`; headless OS-Mesa rendering
CAD-VIGA rendering	Six azimuth views returned for agent inspection
Timeouts	MCP startup 30 s; tool call 180 s; backend HTTP default 120 s; no overall per-example timeout
Input	Four-view BenchCAD image and generic expert mechanical-CAD prompt
Sampling	Five records per difficulty; shuffle buffer 512; seed 42
Repetition	Three independent attempts per record and method; 90 attempts total

Table 1: Experimental configuration.

4.5 Metrics and Statistical Analysis

Generation success requires an executable generated program, exportable CAD geometry, and available BenchCAD metrics. We report Voxel IoU and the BenchCAD composite score conditionally on successful execution, because a failed program has no geometry to compare. Success rate is therefore reported separately and is not hidden by conditional means.

For stochasticity, we retain all three attempt-level outcomes and compute per-record means and sample variances. Because repeated attempts of the same record are not independent benchmark items, per-record summaries are the primary unit of interpretation. We additionally report two-sided Welch tests on successful attempt-level metrics as exploratory analyses and paired tests

Method	Success	Rate	Voxel IoU	Composite
One-shot baseline	17/45	37.8%	0.4127	0.5236
CAD-VIGA	45/45	100.0%	0.3800	0.5073

Table 2: Overall comparison. Voxel IoU and composite score are conditional means over successful attempts and must be interpreted together with success rate.

on per-record means where both methods have at least one success. These tests are secondary: excluding failed baseline attempts introduces survivorship bias, and small paired subsets have low power.

5 Results

5.1 Execution Reliability and Geometric Quality

CAD-VIGA completed all 45 attempts, whereas the one-shot baseline completed 17 of 45. This corresponds to success rates of 100% and 37.8%, respectively. The baseline success rate decreased from 53.3% on easy records to 40.0% on medium records and 20.0% on hard records; CAD-VIGA remained at 100% in every tier.

Among successful attempts, the one-shot baseline has slightly higher overall conditional means than CAD-VIGA. Thus the measured advantage of CAD-VIGA is reliability rather than improved conditional similarity. An executable output can still be geometrically inaccurate: CAD-VIGA's hard turnbuckle attempts, for example, have a per-record mean IoU of only 0.0187 despite successful execution.

5.2 Qualitative Examples

Figures 4–6 show selected individual CAD-VIGA attempts for qualitative inspection. Their captions report the scores of the displayed attempts rather than the three-attempt per-record means used in the repeated quantitative comparison.

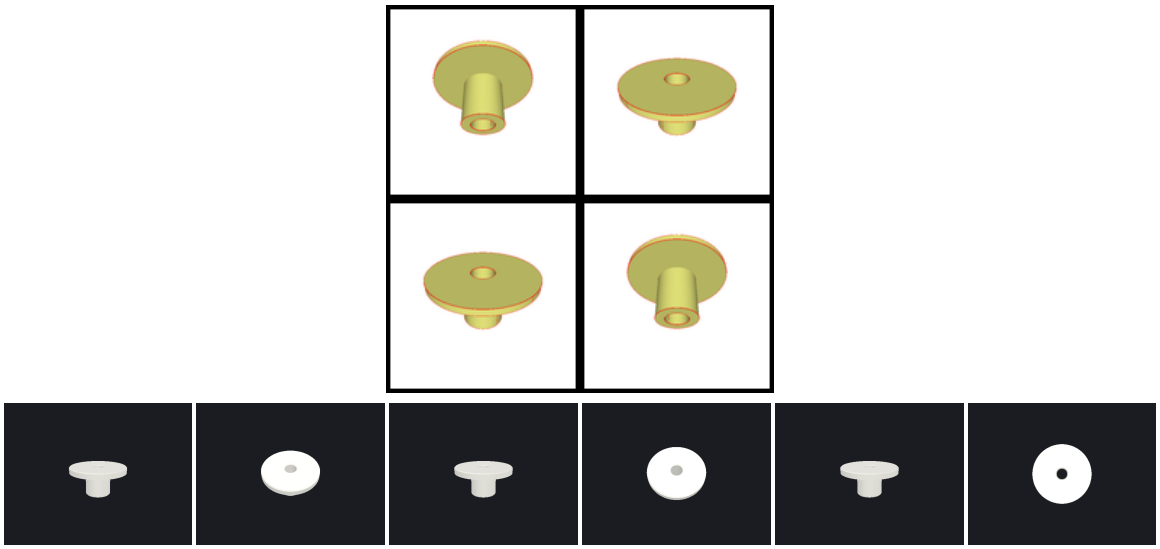


Figure 4: Easy example: BenchCAD tee-nut reference image and six CAD-VIGA renders. Voxel IoU: 0.7245; composite score: 0.7487.

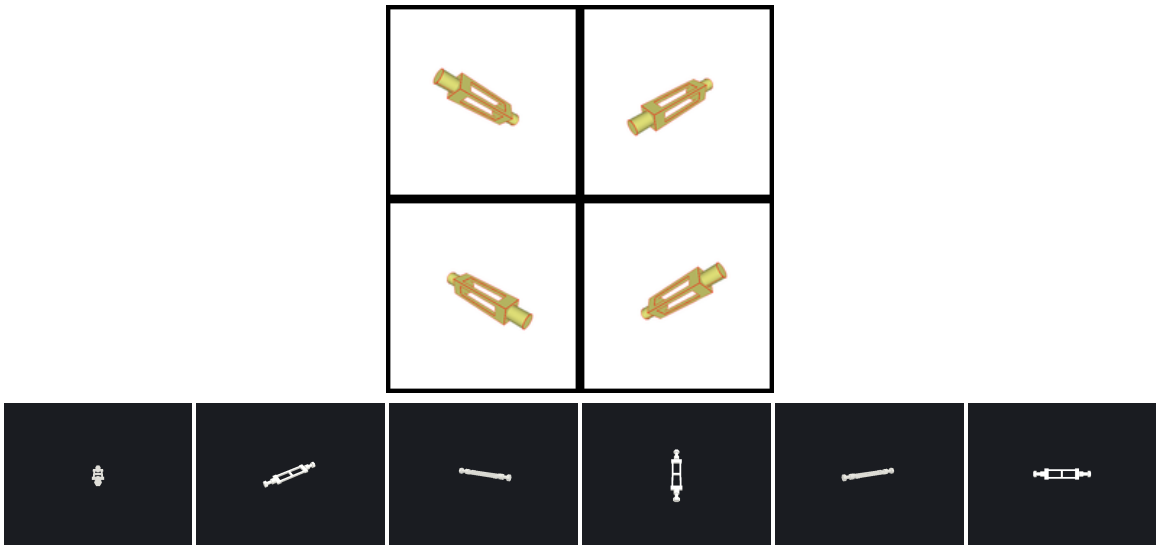


Figure 5: Medium example: BenchCAD turnbuckle reference image and six CAD-VIGA renders. Voxel IoU: 0.2139; composite score: 0.4249.

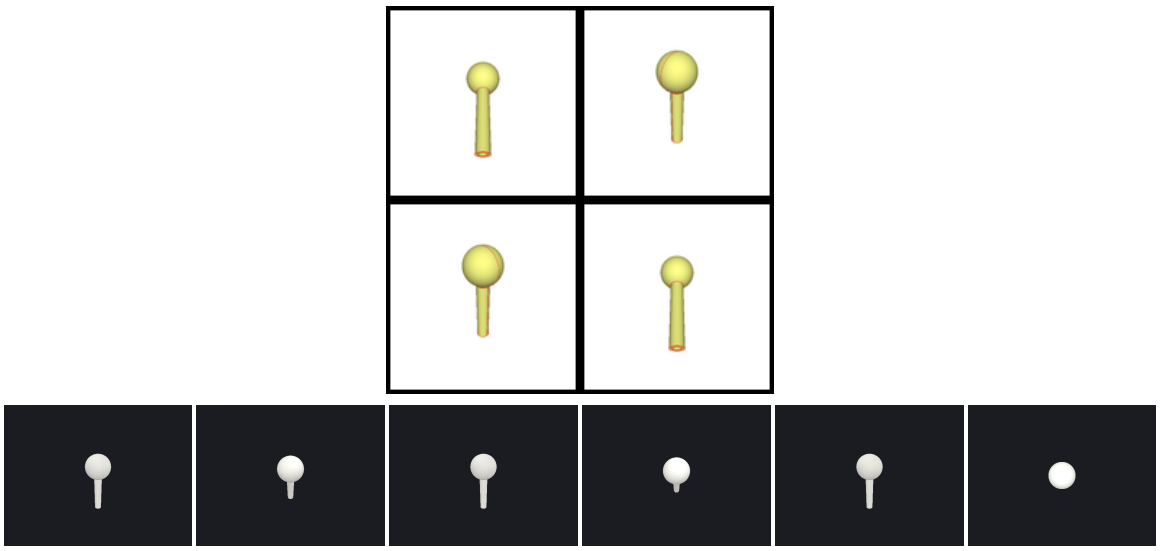


Figure 6: Hard example: BenchCAD ball-knob reference image and six CAD-VIGA renders. Voxel IoU: 0.5654; composite score: 0.6987.

Tier	Method	Success	IoU mean	IoU var.	Composite mean
Easy	Baseline	8/15	0.6099	0.0489	0.6409
Easy	CAD-VIGA	15/15	0.4540	0.0323	0.5435
Medium	Baseline	6/15	0.3113	0.0394	0.4502
Medium	CAD-VIGA	15/15	0.2712	0.0607	0.4147
Hard	Baseline	3/15	0.0894	0.0161	0.3578
Hard	CAD-VIGA	15/15	0.4148	0.1619	0.5637

Table 3: Attempt-level conditional metrics by difficulty. "Var." is sample variance over successful attempts. Baseline hard-tier means use only three successful attempts and are consequently highly selected.

5.3 Results by Difficulty

Easy and medium conditional means favor the baseline, but only 8 and 6 of its 15 attempts, respectively, reached scoring. On hard records, CAD-VIGA has higher conditional means, but the baseline comparison contains only three successful attempts. The high CAD-VIGA hard-tier IoU variance (0.1619) also shows that its mean is sensitive to strong ball-knob and hex-key-organizer results. Difficulty-level means therefore should not be interpreted without success counts and dispersion.

5.4 Runtime and Statistical Tests

The 45 CAD-VIGA attempts accumulated 20,132.3 seconds of active runtime, with a mean of 447.4 seconds (7.46 minutes), a minimum of 167.4 seconds, and a maximum of 1,184.3 seconds. Baseline attempts accumulated 2,732.1 seconds, with a mean of 60.7 seconds. CAD-VIGA was therefore approximately 7.4 times slower per attempt. Token usage, tool-call counts, and monetary cost were not recorded at sufficient granularity and are not reported.

Exploratory Welch tests on successful attempt-level outcomes find no overall difference in conditional Voxel IoU ($p = 0.690$) or composite score ($p = 0.784$). Paired tests on the 11 records with at least one success from both methods likewise find no significant difference in IoU ($p = 0.236$) or composite score ($p = 0.178$). Attempt-level hard-tier Welch tests favor CAD-VIGA, but this result is not robust: only three baseline hard attempts succeeded, and the paired hard comparison contains two records and is not significant. We therefore do not claim an overall geometric-quality improvement.

5.5 One-shot Failure Modes

All 28 unsuccessful baseline attempts failed during generated-program execution. Fourteen raised `BRep_API: command not done`; other failures involved invalid fillet or chamfer selections, null shapes, invalid wires, and one use of a nonexistent Workplane method. These outcomes show what the execution-grounded loop principally prevents in this experiment: it turns many non-executable first programs into deliverable CAD artifacts, although it does not guarantee that those artifacts match the reference geometry.

6 Discussion

6.1 What the Baseline Establishes

The controlled comparison changes the interpretation of CAD-VIGA. The agentic workflow does not outperform one-shot generation on conditional geometric metrics. Instead, it trades runtime for execution reliability: every CAD-VIGA attempt reaches an executable scored artifact, while nearly two thirds of one-shot programs fail. This supports the use of execution and repair when a usable artifact is required, but it does not show that visual refinement itself improves shape

fidelity. A one-shot program that already executes can be as accurate as, or more accurate than, the iteratively produced model on these mostly simple parts.

6.2 Reliability, Fidelity, and Cost

Execution success, geometric fidelity, and cost answer different questions. Success rate measures whether the user receives a model; conditional IoU measures the shape only after the pipeline completes; runtime measures the cost of obtaining that reliability. Reporting only conditional quality would favor the baseline by discarding 28 failures, while reporting only completion would hide low-overlap but valid CAD-VIGA outputs. The three quantities should remain separate in evaluations of tool-mediated CAD agents.

6.3 What Remains Unisolated

The baseline isolates the complete CAD-VIGA workflow from direct one-shot generation, but not the contribution of each component. Planning, plan verification, rendering, code repair, and final comparison are bundled. Component-level causal claims require ablations such as execution repair without visual feedback, planning without revision, and fixed-budget refinement. Likewise, the semantic `PARTS` protocol and assembly-interface checks are system capabilities rather than effects measured by this single-part BenchCAD experiment.

7 Limitations

Scale and sampling. The evaluation contains only 15 records selected with one sampling seed. Three attempts per record characterize stochastic generation on this fixed subset, but do not measure variation across alternative BenchCAD samples or support benchmark-wide generalization.

Statistical and causal scope. Conditional comparisons exclude failed baseline attempts and are affected by survivorship bias; repeated attempts also are not independent benchmark items. The baseline estimates the effect of the complete workflow, but no ablation separates planning, execution repair, rendering, and verification.

Engineering validity and cost. Voxel IoU and the composite score do not fully evaluate dimensions, manufacturability, assemblability, or mechanical function; MUSE similarly highlights limitations of geometry-only evaluation [5]. Inputs are controlled renders rather than real photographs. Results also depend on the Codex, MCP, Docker, CadQuery, OpenCASCADE, and rendering versions. Runtime was recorded, but token usage, tool-call counts, and monetary cost were not.

8 Conclusion

CAD-VIGA provides an execution-grounded workflow for image-guided parametric CAD generation. Across 45 attempts on 15 BenchCAD records, it produced executable models in every attempt, compared with 17 of 45 for a controlled one-shot baseline. This reliability improvement required approximately 7.4 times more runtime and did not improve overall conditional Voxel IoU or composite score. The main result is therefore not that iterative visual refinement universally improves geometry, but that execution-grounded agentic tooling can substantially increase the probability of delivering an executable CAD artifact. Larger samples, component ablations, and engineering-aware metrics are needed to determine when the additional loop cost is justified.

References

[1] Mitch Adler, Matthew Russo, and Michael Cafarella. Agent-Aided Design for Dynamic CAD Models. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pages 1142–1146, 2026. https://doi.org/10.1145/3786335.3813198.

[2] Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. Generating CAD Code with Vision-Language Models for 3D Designs. *arXiv preprint arXiv:2410.05340*, 2024. https://arxiv.org/abs/2410.05340.

[3] Jesse Barkley, Rumi Loghmani, and Amir Barati Farimani. CADSmith: Multi-Agent CAD Generation with Programmatic Geometric Validation. *arXiv:2603.26512*, 2026. https://arxiv.org/abs/2603.26512.

[4] Anna C. Doris, Md Ferdous Alam, Amin Heyrani Nobari, and Faez Ahmed. CAD-Coder: An Open-Source Vision-Language Model for Computer-Aided Design Code Generation. *arXiv:2505.14646*, 2025. https://arxiv.org/abs/2505.14646.

[5] Xiaoyu Dong, Zhi Li, and Xiao-Ming Wu. MUSE: Benchmarking Manufacturable, Functional, and Assemblable Text-to-CAD Generation. *arXiv:2605.28579*, 2026. https://arxiv.org/abs/2605.28579.

[6] Tao Hu, Jiaxin Ai, Licheng Wen, Xueheng Li, Shu Zou, Siqi Li, Nianchen Deng, Xinyu Cai, Hongbin Zhou, Pinlong Cai, Daocheng Fu, Yu Yang, Hairong Zhang, Botian Shi, and Xuemeng Yang. IterCAD: An Iterative Multimodal Agent for Visually-Grounded CAD Generation and Editing. *arXiv:2606.13368*, 2026. https://arxiv.org/abs/2606.13368.

[7] Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating Sequential CAD Models from Beginner-to-Expert Level Text Prompts. In *Advances in Neural Information Processing Systems*, 2024. https://arxiv.org/abs/2409.17106.

[8] Pu Li, Jianwei Guo, Xiaopeng Zhang, and Dong-Ming Yan. SECAD-Net: Self-Supervised CAD Reconstruction by Learning Sketch-Extrude Operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16816–16826, 2023. https://openaccess.thecvf.com/content/CVPR2023/html/Li_SECAD-Net_Self-Supervised_CAD_Reconstruction_by_Learning_Sketch-Extrude_Operations_CVPR_2023_paper.html.

[9] Yujia Liu, Anton Obukhov, Jan Dirk Wegner, and Konrad Schindler. Point2CAD: Reverse Engineering CAD Models from 3D Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3763–3772, 2024. https://openaccess.thecvf.com/content/CVPR2024/html/Liu_Point2CAD_Reverse_Engineering_CAD_Models_from_3D_Point_Clouds_CVPR_2024_paper.html.

[10] Weijian Ma, Shuaiqi Chen, Yunzhong Lou, Xueyang Li, and Xiangdong Zhou. Draw Step by Step: Reconstructing CAD Construction Sequences from Point Clouds via Multimodal Diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27154–27163, 2024. https://openaccess.thecvf.com/content/CVPR2024/html/Ma_Draw_Step_by_Step_Reconstructing_CAD_Construction_Sequences_from_Point_CVPR_2024_paper.html.

[11] Wamiq Para, Shariq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas Guibas, and Peter Wonka. SketchGen: Generating Constrained CAD Sketches. In *Advances in Neural Information Processing Systems*, volume 34, 2021. https://proceedings.neurips.cc/paper/2021/hash/28891cb4ab421830acc36b1f5fd6c91e-Abstract.html.

[12] Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Recode: Reverse Engineering CAD Code from Point Clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9801–9811, 2025. https://openaccess.thecvf.com/content/ICCV2025/html/Rukhovich_CAD-Recode_Reverse_Engineering_CAD_Code_from_Point_Clouds_ICCV_2025_paper.html.

[13] Mikaela Angelina Uy, Yen-Yu Chang, Minhyuk Sung, Purvi Goel, Joseph G. Lambourne, Tolga Birdal, and Leonidas J. Guibas. Point2Cyl: Reverse Engineering 3D Objects from Point Clouds to Extrusion Cylinders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11850–11860, 2022. https://openaccess.thecvf.com/content/CVPR2022/html/Uy_Point2Cyl_Reverse_Engineering_3D_Objects_From_Point_Clouds_to_Extrusion_CVPR_2022_paper.html.

[14] Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. CAD-GPT: Synthesising CAD Construction Sequence with Spatial Reasoning-Enhanced Multimodal LLMs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(8):7880–7888, 2025. https://doi.org/10.1609/aaai.v39i8.32849.

[15] Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 Gallery: A Dataset and Environment for Programmatic CAD Construction from Human Design Sequences. *ACM Transactions on Graphics*, 40(4), Article 54, 2021. https://doi.org/10.1145/3450626.3459818.

[16] Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: A Deep Generative Network for Computer-Aided Design Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6772–6782, 2021. https://arxiv.org/abs/2105.09492.

[17] Haoyang Xie and Feng Ju. Text-to-CadQuery: A New Paradigm for CAD Generation with Scalable Large Model Capabilities. *arXiv:2505.06507*, 2025. https://arxiv.org/abs/2505.06507.

[18] Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl D. D. Willis, and Daniel Ritchie. Inferring CAD Modeling Sequences Using Zone Graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6062–6070, 2021. https://openaccess.thecvf.com/content/CVPR2021/html/Xu_Inferring_CAD_Modeling_Sequences_Using_Zone_Graphs_CVPR_2021_paper.html.

[19] Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. SkexGen: Autoregressive Generation of CAD Construction Sequences with Disentangled Codebooks. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 24698–24724, 2022. https://proceedings.mlr.press/v162/xu22k.html.

[20] Xiang Xu, Pradeep Kumar Jayaraman, Joseph George Lambourne, Karl D. D. Willis, and Yasutaka Furukawa. Hierarchical Neural Coding for Controllable CAD Model Generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 38443–38461, 2023. https://proceedings.mlr.press/v202/xu23f.html.

[21] Shaofeng Yin, Jiaxin Ge, Zora Zhiruo Wang, Chenyang Wang, Xiuyu Li, Michael J. Black, Trevor Darrell, Angjoo Kanazawa, and Haiwen Feng. Vision-as-Inverse-Graphics Agent via Interleaved Multimodal Reasoning. *arXiv:2601.11109*, 2026. https://arxiv.org/abs/2601.11109.

[22] Haozhe Zhang, Kaichen Liu, Miaomiao Chen, Lei Li, Shaojie Yang, Cheng Peng, and Hanjie Chen. BenchCAD: A Comprehensive, Industry-Standard Benchmark for Programmatic CAD. *arXiv:2605.10865*, 2026. https://arxiv.org/abs/2605.10865.