BENCHCAD: A COMPREHENSIVE, STANDARD-ANCHORED BENCHMARK FOR PROGRAMMATIC CAD

Anonymous authors
Paper under double-blind review

ABSTRACT

Industrial Computer-Aided Design (CAD) code generation requires models to produce executable parametric programs from visual or textual inputs. Beyond recognizing the outer shape of a part, this task involves understanding its 3D structure, inferring engineering parameters, and choosing CAD operations that reflect how the part would be designed and manufactured. Despite the promise of Multimodal large language models (MLLMs) for this task, they are rarely evaluated on whether these capabilities jointly hold in realistic industrial CAD settings. We present BenchCAD, a synthetic, standard-anchored benchmark for industrial CAD reasoning. BenchCAD contains 17,900 execution-verified CadQuery programs across 106 industrial part families, including bevel gears, compression springs, twist drills, and other reusable engineering designs. It evaluates models through visual question answering, code question answering, image-to-code generation, and instruction-guided code editing, enabling fine-grained analysis across perception, parametric abstraction, and executable program synthesis. Across 10+ frontier models, BenchCAD shows that current systems often recover coarse outer geometry but fail to produce faithful parametric CAD programs. Common failures include missing fine 3D structure, misinterpreting industrial design parameters, and replacing essential operations such as sweeps, lofts, and twist-extrudes with simpler sketch-and-extrude patterns. Fine-tuning and reinforcement learning improve in-distribution performance, but generalization to unseen part families remains limited. An answerability audit narrows the mean code–image QA gap from 27.8 to 11.6 percentage points, showing that the original comparison partly reflected questions that rendered views cannot answer. BenchCAD supports diagnosis of programmatic CAD capabilities within this synthetic setting. BenchCAD has also been used in publicly reported model evaluations by OpenAI and Anthropic. Released under CC-BY-4.0.

1 INTRODUCTION

Multimodal large language models (MLLMs) (OpenAI, 2026a; Anthropic, 2026a; Google DeepMind, 2026) now combine visual perception, code generation, and multi-step reasoning in a single interface, performing tasks that previously required dedicated systems. A natural next question is whether such models can move beyond recognizing visual content to producing executable programs that define and modify physical objects. Computer-aided design (CAD) provides a canonical testbed for this question. Unlike a mesh, point cloud, or rendered image, a CAD model is typically an editable parametric program: geometric operations such as extrusion, cutting, sweeping, filleting, and patterning construct a solid object through variables and constraints (CadQuery Contributors, 2024). A capable CAD agent must therefore connect visual evidence, 3D structure, symbolic operations, and executable program synthesis.

Real industrial designs are rarely arbitrary shapes. They belong to reusable families such as gears, springs, brackets, fasteners, pipes, and bearings (Figure 1), where small local features and parameter relations decide whether a part is merely visually similar or actually useful as an editable design. A spring is not fully specified by its helical silhouette: its end coils, pitch profile, and cross-section encode design intent. Similarly, a gear-like outline is insufficient without the correct tooth construction and parameterization. **The geometry passes the eye but fails the caliper.**

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

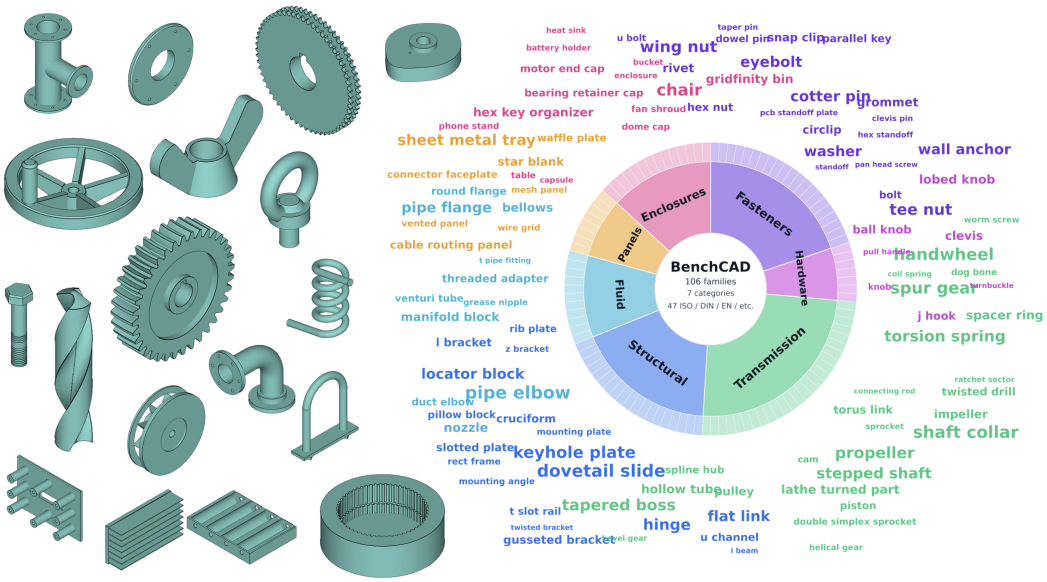


Figure 1: **BenchCAD overview.** BenchCAD is a unified, capability-decomposed evaluation framework for industrial CAD reasoning, consisting of 17,900 expert-verified parametric CadQuery parts (*left*) drawn from 106 industrial families spanning fasteners, transmission components, structural elements, fluid fittings, panels, hardware, and enclosures. The 7-category functional taxonomy (*right*) covers 52 specification-anchored families (51 tied to ISO/DIN/EN/ASME/IEC standards and one to an open-source specification), with each part realized as an executable parametric program testable for geometry, parameters, and edits.

Existing evaluations capture only parts of this problem. Prior CAD code-generation benchmarks (Wu et al., 2021; Khan et al., 2024; Guan et al., 2025; Rukhovich et al., 2025; Kolodiazhnyi et al., 2026; Elistratov et al., 2026) score whether a model's rendered output matches a target shape through a single geometric metric such as IoU or Chamfer distance, while program editing (Alrashedy et al., 2025; Yuan et al., 2025) and design question answering are studied in isolation. However, two programs may produce similar outer envelopes while differing in editability, operation choice, and engineering detail, so a single shape-matching score can overestimate capability and obscure which sub-ability — visual perception, parametric abstraction, or code synthesis — drives the remaining gap.

For practical CAD agents, the relevant question is not only "does the rendered geometry match?", but also "does the model understand the operations, parameters, constraints, and editable structure that produced it?" We therefore formalize CAD reasoning as a four-level capability hierarchy, from part-level visual recognition to CAD-operation understanding, parametric abstraction, and executable code synthesis (Figure 3), because CAD failures are often compositional: a model may identify the family but pick the wrong operation, infer the rough scale but miss a standard relation, or edit the requested feature while changing unrelated parts.

We introduce BenchCAD, a capability-decomposed benchmark for executable, editable, and constraint-aware CAD reasoning. BenchCAD contains 17,900 execution-verified CadQuery programs across 106 industrial part families covering twist drills, bevel gears, compression springs, propellers, brackets, flanges, eye bolts, fasteners, and other reusable industrial designs (Figure 1). It exercises a substantially broader CadQuery operation surface than prior released corpora, including helical sweeps, lofts, twist-extrudes, and parametric involute-gear construction. Rather than unconstrained primitives, it instantiates part families with parameter relations and standard-derived dimensions (Figure 2). It evaluates three practical CAD-agent workflows through four task families: VISION2CODE (`img2cq`, image-to-code generation from multi-view renders), VISION QA and CODE QA (`qa_img`, `qa_code`, design question answering from visual or program inputs), and CODE EDIT (`edit_code`, instruction-guided program editing).

Across 10+ frontier MLLMs and open CAD-specialized baselines (Kolodiazhnyi et al., 2026; Elistratov et al., 2026), BenchCAD reveals a consistent gap between apparent geometric similarity and

true parametric understanding: models often recognize the part family but miss local details, choose simplified or incorrect operations, or fail to apply localized edits faithfully. After excluding questions that normalized renders cannot answer, the mean code–image QA gap decreases from 27.8 to 11.6 percentage points across ten models. This narrower comparison distinguishes an information-access confound from the remaining modality difference. Supervised fine-tuning (SFT) and reinforcement learning (RL) on BenchCAD improve operation coverage and executable generation, but substantial out-of-distribution gaps remain, especially for held-out industrial families requiring advanced operations and precise design constraints.

Public model reports from OpenAI and Anthropic include evaluations on BenchCAD (OpenAI, 2026c;b; Anthropic, 2026b;f;d;c;e). Appendix J (Figures 7 and 8) summarizes these results and their source-specific settings.

Contributions. (i) We release BenchCAD, a domain-expert-verified benchmark of 17,900 executable CadQuery programs across 106 industrial part families with multi-view renders, design QA pairs, and curated edit examples. (ii) We propose a unified CAD-agent evaluation framework covering image-to-code generation, image- and code-based QA, and instruction-guided code editing. (iii) We provide a capability-decomposed evaluation of frontier MLLMs and open CAD-specialized models, showing that current systems remain limited by local detail recognition, CAD operation reasoning, parametric abstraction, and edit fidelity.

2 Related Work

CAD code-generation models. Recent work establishes a common SFT+RL post-training pipeline on procedural sketch-and-extrude corpora. DeepCAD (Wu et al., 2021) and Text2CAD (Khan et al., 2024) introduced command-sequence corpora; CAD-Coder (Guan et al., 2025) reformulated the data as CadQuery and added GRPO with a Chamfer reward; the CAD-Recode (Rukhovich et al., 2025)/cadrille (Kolodiazhnyi et al., 2026)/CADEvolve (Elistratov et al., 2026) lineage drove DeepCAD IoU to $\sim$92% via multi-modal SFT and online RL on 1M–2.7M-script corpora, establishing CadQuery as a promising generation target for VLMs. Two structural gaps remain: (i) training is dominated by sketch+extrude — helical sweeps and parametric involute-gear construction are absent from every released corpus; and (ii) evaluation lacks family-level taxonomy and standard-table grounding, conflating visual perception, parametric abstraction, and code synthesis into a single end-to-end IoU score.

Table 1: **BenchCAD positioning.** Benchmark overview for MLLM CAD capabilities.

	BenchCAD
Primary purpose	evaluation benchmark
Industry domain experts checked	Yes
Named industrial families	**106**
ISO/DIN/EN/ASME/IEC codes	**47**
Verified edit pairs	**748**
Paired numeric-QA items	**2,400**
Capability-decomposed task suite	✓
Rotation + scale-invariant scoring	✓

CAD evaluation benchmarks. The closest prior CAD code/edit benchmarks — CADPrompt (Alrashedy et al., 2025), CAD-Editor (Yuan et al., 2025), CADialogue (Zhou et al., 2025), and Hist-CAD (Dong et al., 2026) — each cover a slice but none combines (i) execution-verified parametric edit pairs, (ii) instruction-guided editing with verified target solids, and (iii) capability-decomposed sub-tasks. Detailed protocol comparison is in Appendix B.

Position of BenchCAD. BenchCAD is a unified, capability-decomposed evaluation framework and the first public CAD benchmark to combine four properties simultaneously: (i) *execution-verified at scale* (17,900 parts); (ii) *standard-anchored* (51 families tied to ISO/DIN/EN/ASME/IEC tables, plus one open-source specification); (iii) *operation-rich* (46 CadQuery operations including `makeHelix`, `twistExtrude`, `polarArray`); (iv) *capability-decomposed* (four tasks probing visual recognition, parametric abstraction, code synthesis, and verified program editing, with an answerability-filtered image/code comparison). See Table 1 and Appendix C for the full per-axis comparison.

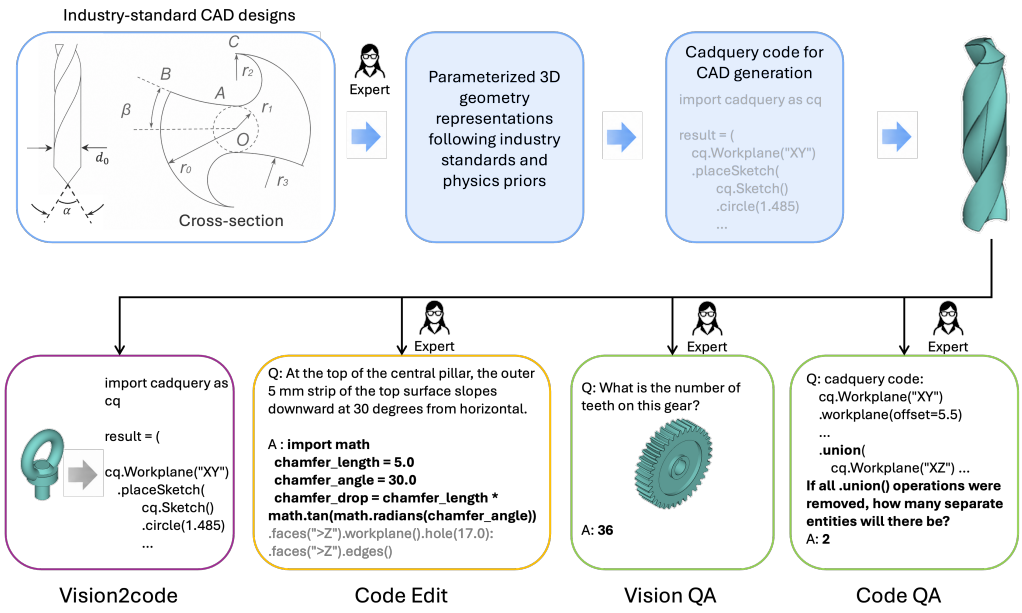


Figure 2: **BenchCAD generation pipeline and task suite.** *Top:* parts originate from industry-standard engineering designs (e.g., DIN 338 twist-drill cross-section), are realized as parameterized 3D geometry that respects standard parameter relations and physical priors, and are emitted as executable CadQuery code with verified geometry. *Bottom:* the four BenchCAD evaluation task categories operationalized on these parts — `img2cq` (image-to-code), `edit_code` (instruction-guided program editing), `qa_img` (image-based design QA), and `qa_code` (code-based design QA).

3 BenchCAD: Dataset and Tasks

3.1 Dataset

Design principles. BenchCAD rests on four principles. *(P1) Expert-authored geometry*: domain experts implement reusable parametric families using standard-derived relations where available and explicit proportional design rules otherwise; standard anchoring does not certify full manufacturing compliance. *(P2) Standard-table anchoring*: where a part has an industrial counterpart, parameters sample from real specification tables (e.g., ISO 22 V-belt cross-sections, DIN 338 twist-drill diameters, ISO 23509 bevel-gear pitch–module relations; summary in Appendix D); 52/106 families (49%) are specification-anchored: 51 reference 47 unique ISO/DIN/EN/ASME/IEC codes, and one uses an open-source specification. *(P3) Family-level taxonomy*: records are grouped into 106 named families (*coil_spring*, *helical_gear*, . . .); a family may further branch into *subfamilies* that capture mating or construction variants of the same part type (e.g., male/female fasteners). Each (sub)family is a small Python module exposing a typed parameter schema, sampler, validator, and deterministic builder, making coverage measurable (Appendix E). *(P4) Operation breadth*: BenchCAD exercises a broader CadQuery operation surface than prior corpora, spanning primitives, 2D sketching, advanced solid ops (helical sweeps, lofts, shells), boolean composition, holes, arrays, and finishing features, including operations (*makeHelix*, *twistExtrude*, *polarArray*) rare or absent in DeepCAD/Fusion360-derived corpora (per-corpus breakdown in Appendix R).

Generation pipeline. Each family supports three difficulty tiers — easy / medium / hard — defined by parameter complexity, where higher tiers expand parameter ranges and activate optional features (full tier definition in Appendix E). For each family×subfamily×tier bucket we sample parameters under standard-table constraints, emit the CadQuery program, and sandbox-execute it; programs that fail to compile, exceed a 30 s runtime budget, or produce degenerate (zero or inverted) volume are quarantined (full failure-mode taxonomy in Appendix E). Figure 2 illustrates the pipeline (top) and the four downstream evaluation tasks (bottom).

Expert verification and geometric audit. Before generation, four CAD experts review ten rendered samples and their source code per family against the relevant dimension tables and reference drawings; identified issues are corrected and re-reviewed, and disagreements among the experts are resolved by discussion until consensus. Each instance passes six automated checks: build success, non-degenerate geometry, timeout detection, family-specific parameter validation, realism filtering, and a clean-namespace code-to-solid round trip. Of approximately 20,000 generated instances, a stratified 10% received detailed human review, identifying issues in about 4% of the reviewed families. After module correction, full-corpus visual screening and family rebalancing, 17,900 instances remain. A subsequent reverse-measurement audit found 351 instances whose declared holes lacked matching B-rep evidence, including 321 no-material-removal operations. We corrected the affected modules and added this audit to the pipeline; Appendix E describes its scope and the distinction between geometry and operation-label validity.

What a standard fixes. Anchoring is parameter-specific. For a roller-chain sprocket, ISO 606 fixes pitch, roller diameter and derived tooth geometry; tooth count, bore, hub and keyway are generator-defined. For a bearing retainer cap, ISO 15 fixes the bearing envelope, not every cap dimension. These references do not imply certification of the complete generated part.

Three released datasets. *BenchCAD* contains 17,900 verified CadQuery code parts and serves as the primary evaluation set. *BenchCAD-QA* provides 2,400 paired image/code numeric-QA items. *BenchCAD-Edit* provides 748 curated edit pairs across dimensional, additive, subtractive, and multi-step categories. The datasets are hosted at `https://huggingface.co/datasets/BenchCAD/BenchCAD`, with data under CC-BY-4.0 and evaluation code under MIT. Appendix K summarizes the dataset documentation.

3.2 TASKS

VISION2CODE. Models generate executable CadQuery code from four canonical orthographic views. The primary metric is IoU between the rendered prediction and the ground-truth occupancy grid; we also report Chamfer distance, feature score, essential-op recall, and execution rate.

CODE EDIT. Given an original CadQuery program and a natural-language edit instruction, the model must produce a minimally modified program whose rendered solid matches the target. Items cover five edit types T1–T5: literal replacement, chained transformation, relative computation, feature editing, and geometry rebuilding (Table 17). We use `Accuracy` as the headline metric, which measures headroom-normalized improvement over the original-to-target gap (Eq. (1)) and controls for varying per-pair difficulty. Unlike from-scratch generation or QA, this task tests whether a model can apply a specified change to an existing valid program while preserving the rest. Because the instruction supplies target values, the default code-plus-text task does not isolate visual perception.

VISION QA and CODE QA. We organize the QA bank along a four-level capability hierarchy (Figure 3), from low-level perception at the base to high-level synthesis at the apex: *(i) Holistic Visual Recognition,* L_1 — recognize the part family and integrate multi-view renders into a coherent volume; *(ii) CAD Operations Understanding,* L_2 — read CadQuery (or infer from geometry) and map operations to features in execution order; *(iii) Industrial Parametric Abstraction,* L_3 — abstract geometry into the parametric structure an engineer would write, respecting standard-table relations; *(iv) Compositional Spatial-Code Reasoning,* L_4 — plan spatially consistent CAD operations, maintain dependencies across parameters and coordinate frames, and produce valid, executable code. The paired-modality comparison is restricted to image-answerable questions: four CAD experts identified items requiring source code, construction history, or absolute dimensions unavailable in normalized renders; we excluded them and rescored existing predictions for ten models without additional inference. The resulting gap measures the benefit of code access on this restricted subset; it is not a causal isolation of visual perception from reasoning. Results on the original bank remain supplementary code-comprehension and dataset diagnostics (Appendix L).

Question-bank construction, edit-pair protocol, rotation-invariant IoU, and scale-invariance ablation appear in Appendices L, P, O, and Q.

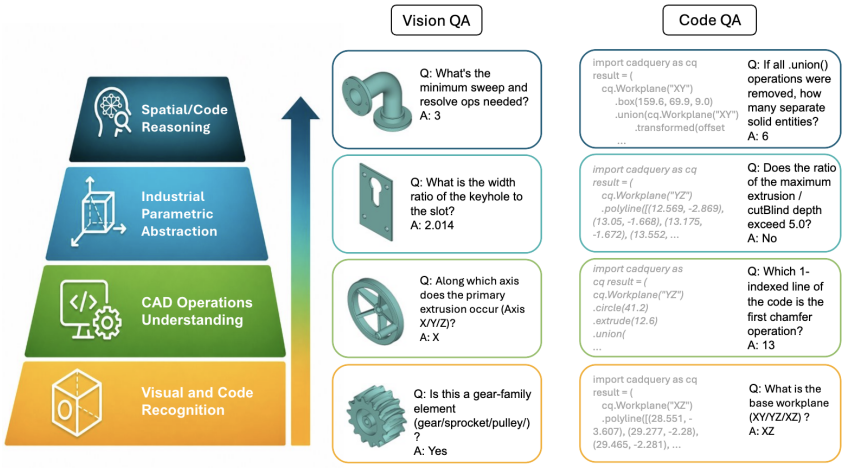


Figure 3: **BenchCAD-QA capability hierarchy.** Four-level organization of the original question bank (L1 Holistic Visual Recognition → L4 Spatial/Code Reasoning). Only questions judged answerable from normalized renders enter the main paired-modality comparison; source-code and construction-history questions are treated as code comprehension.

4 Experiments

4.1 Setup

For each task, we include a *blind baseline* that preserves the output format but removes the informative input: black views for Vision2Code/Vision QA, no code for Code QA, and unchanged original programs for Code Edit. These baselines measure gains beyond dataset priors and metric floors across all relevant metrics.

4.2 Models

Three model classes are evaluated: *(i) Frontier proprietary MLLMs:* GPT-4o (OpenAI, 2024), GPT-5.3 thinking / non-thinking (OpenAI, 2026a), Claude Opus 4.7 thinking / non-thinking (Anthropic, 2026a), Gemini 3.1 Pro thinking / non-thinking (Google DeepMind, 2026), OpenAI o3 (OpenAI, 2025b), Moonshot Kimi (Moonshot AI, 2025). *(ii) Open-source MLLMs / code LLMs:* Qwen3-VL (Bai et al., 2025), InternVL3 (Chen et al., 2025), gpt-oss-120b (OpenAI, 2025a), nemotron-3-super-120b-a12b (Chandiramani et al., 2026). *(iii) CAD-specialist lineage:* cadrille-RL (Kolodiazhnyi et al., 2026) and CADEvolve v3 (Elistratov et al., 2026). Closed models are queried through their official APIs; open-weights models via OpenRouter (OpenRouter, 2024) or the official repository.

4.3 Evaluation

For Vision2Code we report `exec_pct`, mean voxel `IoU`, mean Chamfer distance `CD`, mean Hausdorff distance `HD`, Feature-F1, essential-op recall `ess`, and a composite `total` score (Table 11). The composite uses $0.60\,\mathrm{IoU} + 0.20\,\mathrm{ess} + 0.10\,\mathrm{FeatureF1} + 0.05\,\mathrm{CD}_{score} + 0.05\,\mathrm{HD}_{score}$. Essential-operation matching allows family-specific alternatives rather than requiring an identical source program (Appendix H); rotation-invariant IoU is diagnostic only. For Code Edit, with rendered solid S_g, original solid S_o, and target solid S_t, let $\mathrm{IoU}(S_a, S_b)$ denote the voxel intersection-over-union. Over n edit records, we report normalized accuracy, which controls for pair-specific bias induced by varying original–target IoU:

$$\mathrm{Acc}_{\mathrm{norm}} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{clip}\left(\frac{\mathrm{IoU}(S_g^i, S_t^i) - \mathrm{IoU}(S_o^i, S_t^i)}{1 - \mathrm{IoU}(S_o^i, S_t^i)}, 0, 1\right). \tag{1}$$

Accuracy is a headroom-normalized improvement, where 1 means the model fully traversed the original-to-target gap, and all records satisfy $\mathrm{IoU}(S_o^i, S_t^i) < 0.99$. For Vision QA and Code QA, the

Table 2: **QA after answerability filtering.** Existing predictions are rescored without new inference. The original bank includes questions requiring code, construction history or absolute scale; the image-answerable subset is the primary paired comparison. Gap is Code minus Image. Means are across the ten models; gaps use underlying precision and may differ from subtraction of rounded entries.

	Original bank			Image-answerable		
Model	Image	Code	Gap	Image	Code	Gap
o3	.327	.708	.381	.798	.897	.099
Gemini 3.1 Pro	.587	.836	.249	.771	.919	.148
Opus 4.7 thinking	.530	.829	.299	.768	.878	.110
Opus 4.7	.526	.801	.275	.760	.873	.113
Gemini 3.1 Pro thinking	.576	.838	.262	.756	.938	.182
GPT-5.3	.513	.823	.310	.747	.874	.127
GPT-5.3 thinking	.514	.821	.307	.722	.870	.148
GPT-4o	.464	.726	.262	.669	.729	.060
Moonshot 8k	.447	.620	.173	.624	.708	.084
Moonshot 128k	.442	.700	.258	.623	.713	.090
Mean	.493	.770	.278	.724	.840	.116

released evaluator uses exact match for integer, count and Boolean questions. Dimension and ratio answers receive $\min(|p|, |g|)/\max(|p|, |g|)$ for nonzero predictions p and targets g with matching signs; opposite signs score zero, and a zero-valued pair scores one only if both values are zero. We report mean numeric scores, following the released QA scoring implementation. The primary modality comparison uses the image-answerable subset of the original 2,400-item bank (§3.2); full-bank axis results are supplementary.

Sampling sensitivity. We additionally evaluated ten API models on three disjoint sets of 200 stems (6,000 outputs in total). We report unconditional IoU, counting execution failures as zero, and the standard deviation across sets (Appendix I), which measures variation across sampled inputs, not multi-seed variability on a fixed test set.

4.4 Training

We also test whether BenchCAD's operation breadth and standard-anchored families yield trainable capability gains that transfer beyond the trained families. We train an open Qwen3-VL-2B baseline reported in the bottom block of Table 11, comparing matched-compute SFT and RL settings that differ only in training data composition. **SFT.** We train three variants with the same optimizer, schedule, and held-out evaluation set: *iid* uses BenchCAD plus extrusion-heavy auxiliary data, *ood* follows the same recipe but removes 10 mechanical families, and *baseline* uses no BenchCAD data. **RL.** Starting from each SFT checkpoint, we apply an on-policy GRPO-style objective with reward $r{=}0.2\,\mathrm{ess}{+}0.8\,\mathrm{IoU}$ and $r{=}-1$ for parse-error outputs; the OOD-RL run also excludes the held-out OOD families. We report per-operation recall ablations in Table 12, with full training details and curves in Appendix F.

5 Results

5.1 Overall Performance of 4 Tasks

Vision2Code. The unified leaderboard (Table 11, Appendix G) shows two patterns: (i) the specialist CAD lineage transferred from DeepCAD scores well on IoU but underperforms on non-extrude operations; (ii) frontier MLLMs remain below the strongest trained baseline (gemini-3.1-pro-thinking total 0.3457); the effect of image resolution depends on reasoning mode (Table 20).

Code Edit. Table 3 reports the BenchCAD-Edit leaderboard (mean-normalized improvement). Overall scores show a clear gap between model families, but the per-type breakdown in Figure 4 reveals a more diagnostic pattern: simple API-level edits are nearly solved by modern models, while compositional edits remain difficult. On the 100-pair input ablation, replacing the instruction with a target render substantially reduces performance, while adding the original render to code plus text shows no consistent aggregate benefit (Figure 11) — a null result under the current instruction distribution, where target values are already stated, not evidence that images are generally unnecessary for

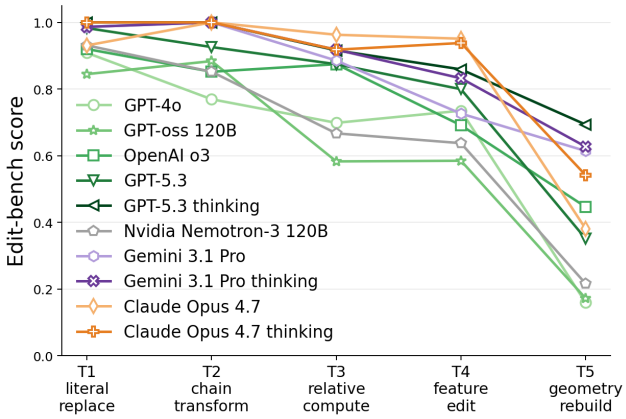


Figure 4: **BenchCAD-Edit by task type.**

Table 3: **BenchCAD-Edit Accuracy.**

Model	Thinking	Acc.
GPT-5.3	✓	**0.865**
Claude Opus 4.7	✓	0.853
Gemini 3.1 Pro	✓	0.837
Claude Opus 4.7	–	0.811
Gemini 3.1 Pro	–	0.795
GPT-5.3	–	0.740
OpenAI o3	✓	0.708
GPT-4o	–	0.615
Nemotron-3 120B	–	0.608
GPT-oss 120B	–	0.561
Baseline (no change)	–	0.000

CAD editing. Per-model F-code distributions, the full L1–L4 attribution, and the image-conditioned variant are in Appendix P.

VISION QA and CODE QA. Table 2 shows that answerability filtering materially changes the modality comparison. Across ten models, mean image and code scores are 0.724 and 0.840 on the filtered subset, and the mean gap falls from 0.278 to 0.116. The ordering also changes: o3 rises from 0.327 to 0.798 on image QA and becomes the highest-scoring image model among these ten. Thus, part of the original gap reflected unavailable information rather than weak visual reasoning; code access still helps on retained questions, but the smaller gap does not isolate a single capability. Tables 9 and 10 retain the original full-bank axis breakdown as supplementary diagnostics, not as a clean visual-perception comparison.

5.2 ERROR ANALYSIS

Across tasks, most failures trace to deficiencies in three core capabilities.

Holistic Spatial Recognition. Visual recognition failures occur along two axes: detailed feature counting and spatial grounding.

A hex-head bolt (Figure 5A) illustrates a fine-detail failure. GPT-5.3 recovers the overall bolt body but misses small, human-obvious CAD features such as the thread structure and the top-face chamfer. Counts of source-level chamfer calls are not generally identifiable from a render, so we do not use such questions as independent evidence of visual recognition failure.

A bearing retainer cap (Figure 5B) exposes a global spatial-frame failure. The model captures several fine-grained features but fails to infer the correct extrusion direction: the generated code extrudes from the XY plane rather than the target XZ plane. This error is common for geometries generated based on XZ and YZ workplanes. It is partially diagnosed by the substantially higher 24-axis rotation-invariant IoU compared with the single-axis IoU, suggesting that the predicted shape is geometrically similar but expressed in the wrong spatial frame (Appendix O). The same example also reveals parametric-abstraction failures across multiple features.

Operation understanding. A twisted bracket (Figure 5C) is generated as two mutually perpendicular brackets without twisted connection; the requisite twist-extrusion is absent from the emitted program entirely. The model recognizes the holistic spatial (L1) but fails to map the visible torsion to the corresponding CAD operation, exposing an Op Vocabulary Gap (Appendix H) at the operation-understanding layer.

Industrial Parametric abstraction. A standardized DIN 2095 coil spring (Figure 5D) exposes a parametric-abstraction failure. The standard requires closed and ground ends: the coil pitch is locally reduced near both terminals, causing the end turns to flatten into planar bearing surfaces. The model recognizes the spring and emits a helical sweep, but collapses the end-specific pitch variation into a uniform helix with an incorrect cross-section. It thus captures the family and operation (L1 &

Table 4: **IID–OOD generalization under SFT and RL.** The ten OOD families are excluded from training only for ood-sft and ood-rl (family-held-out generalization); iid checkpoints train on all 106 families, so their OOD columns measure performance with family exposure.

	IID split				OOD split			
Checkpoint	IoU↑	Exec.↑	Ess.↑	Score↑	IoU↑	Exec.↑	Ess.↑	Score↑
ood-sft (A)	0.577	94.6%	0.789	0.6064	0.397	89.8%	0.305	0.3776
ood-rl (B)	0.757	99.1%	0.865	0.7467	0.464	**100.0%**	0.339	0.4604
iid-sft (C)	0.640	87.3%	0.857	0.6225	0.635	96.6%	0.983	0.6920
iid-rl (D)	0.753	98.9%	0.883	0.7635	**0.761**	99.6%	**0.989**	**0.8214** ⋆

L2) but misses the standard-driven parameterization an engineer would encode — which is why BenchCAD anchors families to standards: it tests not only whether models name a part or invoke the right operation, but whether they recover the design rules underlying real parametric CAD.

The layered framework thus turns an opaque "wrong code" label into level-specific diagnostics: the bracket fails at L_2, while the spring passes L_1–L_2 but fails at L_3. A single IoU score would collapse these distinctions and hide which level to strengthen.

5.3 BenchCAD as a Training Resource for Improving Model Capabilities

BenchCAD training improves geometric fidelity and the use of advanced operations such as `revolve`, `sweep`, `loft`, and `fillet` (Table 12). To distinguish these gains from family-template exposure, Table 4 separates models trained with ten families excluded from models trained on all 106 families. For `ood-sft`, IoU drops from 0.577 on seen families to 0.397 on held-out families; `ood-rl` improves these scores to 0.757 and 0.464, respectively. RL therefore improves held-out performance, but a substantial family gap remains. The `iid` checkpoints have seen all families during training, so their OOD-column scores are an exposure comparison, not evidence of unseen-family transfer (details in Appendix F).

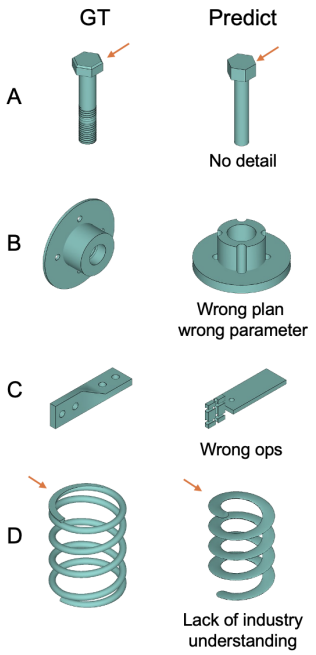


Figure 5: **Examples of failures in Vision2Code.** Zoom-in for more details.

6 Conclusion

BenchCAD evaluates programmatic CAD through image-to-code generation, paired image/code QA, and instruction-guided program editing; its synthetic, expert-authored families expose failures in fine geometry, operation selection and coordinated edits that coarse shape similarity misses. Additional validation reveals limits that motivate a narrower interpretation of modality gaps and operation-based scores: filtering image-unanswerable questions reduces the mean code–image QA gap from 27.8 to 11.6 percentage points, and a B-rep audit finds mismatches between declared hole operations and realized geometry. Training on BenchCAD improves operation coverage and geometric fidelity, but models trained without ten held-out families still show substantial seen-to-unseen gaps. BenchCAD thus supports controlled study of CAD program synthesis and family-level generalization; its scores do not establish manufacturing compliance or transfer to real industrial repositories.

AI use statement

We used generative AI tools to help implement and debug code, to polish the writing, and to prepare figures. They were not used for any other task that requires disclosure, such as generating or cleaning data, designing experiments or interpreting results. The authors reviewed all AI-assisted work and take full responsibility for the content of this paper.

Ethics statement

BenchCAD consists of procedurally generated parametric CAD parts. It contains no personal data, no proprietary designs and no safety-sensitive specifications, and no human-subjects data were collected. Engineering standards are referenced by code and name only; the benchmark samples dimensions within published ranges and does not reproduce standard documents. Scores on BenchCAD do not certify manufacturing compliance and should not be used to approve parts for production (Appendix A). Data are released under CC-BY-4.0 and evaluation code under MIT. Broader risks of CAD-capable models, such as accelerated reverse engineering of regulated components, are discussed in Appendix U.

Reproducibility statement

Section 3 describes dataset construction, verification and the four tasks; generation details, difficulty tiers and the reverse-measurement audit are in Appendix E, the standards summary in Appendix D, and dataset documentation in Appendix K. Section 4.3 defines all metrics, including the composite VISION2CODE score, the normalized edit accuracy (Eq. (1)) and the QA scoring rule; essential-operation alternatives, rotation-invariant IoU and scoring ablations are specified in Appendices H, O and Q. Question-bank construction and the answerability re-audit are described in Appendix L, and the edit-pair protocol in Appendix P. Exact prompts for all tasks are listed in Appendices M and N. Training data mixtures, hyperparameters and the family-held-out split are given in Section 4.4 and Appendix F. Appendix I reports sensitivity to the sampled inputs, and Appendix T describes how to run the three benchmarks from the released data and evaluation code. Externally reported scores in Appendix J are reproduced as published and were not re-graded unless stated. Upon publication, data and evaluation code will be released as a tagged version with fixed split membership, per-split manifests with SHA-256 checksums, the pinned evaluator version, Croissant metadata, and column and scoring definitions; hashes of all ground-truth programs will be published to support contamination checks. We additionally keep an unreleased regeneration of the benchmark (the same families with freshly sampled parameters and a different rendering scheme) as a private split for contamination audits; no reported score uses it.

References

Khaled Alrashedy et al. CAD-CodeVerify: Self-verification of vision-language models on CAD code generation. In *International Conference on Learning Representations (ICLR)*, 2025.

Anthropic. Claude Opus 4.7 System Card, 2026a. URL https://www.anthropic.com/system-cards. Accessed 2026-05-07.

Anthropic. Claude Fable 5 & Claude Mythos 5 System Card, 2026b. URL https://anthropic.com/claude-fable-5-mythos-5-system-card. Accessed 2026-09-25.

Anthropic. Claude Fable 5.1 & Claude Mythos 5.1 System Card, 2026c. URL https://www.anthropic.com/claude-fable-5-1-mythos-5-1-system-card. Accessed 2026-09-25.

Anthropic. Claude Opus 5 System Card, 2026d. URL https://www.anthropic.com/claude-opus-5-system-card. Accessed 2026-09-25.

Anthropic. Claude Opus 5.5 System Card, 2026e. URL https://www.anthropic.com/claude-opus-5-5-system-card. Accessed 2026-09-25.

Anthropic. Claude Sonnet 5 System Card, 2026f. URL https://www.anthropic.com/claude-sonnet-5-system-card. Accessed 2026-09-25.

Shuai Bai et al. Qwen3-VL Technical Report, 2025.

CadQuery Contributors. CadQuery: A python parametric CAD scripting framework based on OCCT. https://github.com/CadQuery/cadquery, 2024.

Abhishek Chandiramani et al. Nemotron 3 Super: Open, Efficient Mixture-of-Experts Hybrid Mamba-Attention Model, 2026.

Zhe Chen et al. InternVL3: Exploring Advanced Training and Test-Time Recipes for Open-Source Multimodal Models, 2025.

Xintong Dong et al. HistCAD: Geometrically constrained parametric history-based CAD dataset, 2026.

Maksim Elistratov, Marina Barannikov, Gregory Ivanov, Valentin Khrulkov, Anton Konushin, Andrey Kuznetsov, and Dmitrii Zhemchuzhnikov. CADEvolve: Creating realistic CAD via program evolution, 2026.

Google DeepMind. Gemini 3.1 Pro Model Card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/. Accessed 2026-05-07.

Yandong Guan, Xilin Ge, Shihao Yang, Wenhao Yang, Zhipeng Wei, Can Cui, Cheng Tang, Liang Zhang, and Yueting Zhuang. CAD-Coder: Text-to-CAD generation with chain-of-thought and geometric reward. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Mohammad Sadil Khan, Sankalp Sinha, Talha A. M. Uddin, 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 (NeurIPS) Datasets and Benchmarks Track*, 2024.

Kyrylo Kolodiazhnyi, Danila Rukhovich, and Djamila Aouada. cadrille: Multi-modal CAD reconstruction with online reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Moonshot AI. Kimi K2 and Kimi-Latest vision-language models, 2025. URL https://platform.moonshot.ai/docs/intro. API endpoints kimi-latest and moonshot-v1-*-vision-preview; accessed 2026-05-07.

OpenAI. GPT-4o System Card, 2024.

OpenAI. gpt-oss-120b & gpt-oss-20b Model Card, 2025a.

OpenAI. OpenAI o3 and o4-mini System Card, 2025b. URL https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf. Accessed 2026-05-07.

OpenAI. GPT-5.3 Instant System Card, 2026a. URL https://openai.com/index/gpt-5-3-instant-system-card/. Accessed 2026-05-07.

OpenAI. GPT-6 Astra: A new generation of intelligence, 2026b. URL https://openai.com/index/gpt-6-astra/. Accessed 2026-09-25.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition, 2026c. URL https://openai.com/index/gpt-5-6/. Accessed 2026-09-25.

OpenRouter. OpenRouter: A Unified Interface for LLMs, 2024. URL https://openrouter.ai. Accessed 2026-05-07.

Danila Rukhovich, Kyrylo Kolodiazhnyi, and Djamila Aouada. CAD-Recode: Reverse engineering CAD code from point clouds. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

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. In *ACM Transactions on Graphics (SIGGRAPH)*, 2021.

Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: A deep generative network for computer-aided design models. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Yu Yuan, Shizhao Sun, Qi Liu, and Jiang Bian. CAD-Editor: A locate-then-infill framework with automated training data synthesis for text-based CAD editing. In *International Conference on Machine Learning (ICML)*, 2025.

Jiwei Zhou, Jorge D. Camba, and Pedro Company. CADialogue: A multimodal LLM-powered conversational assistant for intuitive parametric CAD modeling. *Computer-Aided Design*, 2025. doi: 10.1016/j.cad.2025.103929.

A Limitations

Synthetic scope and standard anchoring. BenchCAD consists of expert-authored parametric families, not parts extracted from real engineering repositories. Of 106 families, 51 reference ISO/DIN/EN/ASME/IEC specifications and one references an open-source specification. Anchoring selected dimensions and relations does not establish full manufacturing tolerances, material compliance, assembly validity or industrial readiness. The family-held-out experiment measures transfer within this synthetic construction process; it does not demonstrate transfer to real-repository parts.

Generator self-validation. A builder and its validator can share a mistaken equation while the program, parameter labels and rendered geometry remain mutually consistent. Expert review and the B-rep audit in Appendix E reduce this risk, but do not provide independent, exhaustive seed-compliance verification for every instance and every engineering constraint.

QA answerability. Source-level operation counts, construction history and absolute dimensions are not generally recoverable from normalized renders. We therefore restrict the main image/code comparison to expert-identified image-answerable questions. The excluded questions remain useful for code comprehension. Even after filtering, a modality gap can reflect several interacting capabilities rather than visual perception alone.

Metric and edit limitations. Voxel overlap can miss small but consequential feature changes. Family-specific operation alternatives accommodate some equivalent constructions but do not prove semantic equivalence or engineering correctness. For localized edits, whole-solid agreement rewards target correctness and penalizes unintended geometric changes, but does not independently establish minimal source edits or preservation of every parameter dependency. We do not report a separate T5 preservation result or an AST-diff result in this study.

B Extended Related Work: CAD Code-Generation and Editing Benchmarks

Code generation. DeepCAD (Wu et al., 2021) introduces a 178K-model corpus of Onshape parts encoded as discrete operation tokens; Text2CAD (Khan et al., 2024) extends DeepCAD with 660K natural-language annotations across four abstraction levels. CAD-Coder (Guan et al., 2025) reformulates the same data as CadQuery Python source and applies SFT+GRPO with chain-of-thought and Chamfer reward (mean CD 6.54×10^{-3} on Text2CAD). The CAD-Recode (Rukhovich et al., 2025)/cadrille (Kolodiazhnyi et al., 2026)/CADEvolve (Elistratov et al., 2026) lineage shares a Qwen2-VL-2B backbone: CAD-Recode fine-tunes on 1M sketch-and-extrude scripts; cadrille adds multi-modal inputs and Dr.CPPO online RL with piecewise IoU reward (DeepCAD IoU 92.2, Fusion360 IoU 84.6, 0.0% invalidity); CADEvolve expands the corpus to $\sim$1.3M scripts via an LLM-driven evolutionary loop covering extrude/revolve/loft/sweep/fillet/chamfer/shell/Boolean/patterns — broadening the operation distribution but still without helical sweeps or parametric involute-gear construction. The CADEvolve Hugging Face release contains only sentence embeddings (`.npy`), not the source code, so its full operation surface is not directly auditable; the public GitHub repo ships only 46 hand-written seed programs.

Editing. The instruction-guided editing setting discussed in Section 2 differs from from-scratch generation: the input includes an existing program, and success requires applying the requested change while retaining unrelated geometry. BenchCAD provides verified before/after targets and five edit types, from literal replacement to coordinated restructuring. Its normalized geometric score measures progress toward the target but is not an independent proof of source-level minimality.

C Full Comparison Against Prior CAD Code-Generation Work

Table 5 provides the full per-axis comparison referenced in §2, including all evaluation-side properties (#Industrial families, #Std codes, #Distinct CQ ops, Adv ops, Edit / QA tasks, #Eval models) across DeepCAD, Fusion360 Gallery, Text2CAD, CADPrompt, CAD-Coder, CAD-Recode, cadrille, and CADEvolve.

Table 5: **BenchCAD versus prior CAD code-generation work.** BenchCAD is a *unified, capability-decomposed evaluation framework* for large-model CAD reasoning, distinct in purpose from the CadQuery-VLM lineage (CAD-Recode, cadrille, CADEvolve), which releases training corpora alongside small fine-tuned models (Qwen 1.5B–2B) saturating ∼92% IoU on legacy sketch+extrude splits. **Type:** *C+M* = corpus+model, *D* = dataset, *B* = benchmark. **#Fam.** = named industrial part families. **#Std** = distinct ISO/DIN/EN/ASME/IEC specification codes. **Op surface**: *narrow* = sketch+extrude IR or a small CadQuery subset; *broad* = wide CadQuery API including advanced solid ops (precise per-corpus counts and protocol in App. R). **Adv** = all four advanced operation families (helix, loft/sweep, twist-extrude, parametric involute-gear) exercised. ✓ = supported, p. = partial, – = absent.

Benchmark	Yr	Type	#Sam.	#Fam.	#Std	Op surface	Adv	Edit	QA
DeepCAD (Wu et al., 2021)	'21	C+M	178K	–	0	narrow	–	–	–
Fusion360 G. (Willis et al., 2021)	'21	D	8K	–	0	narrow	–	–	–
Text2CAD (Khan et al., 2024)	'24	C+M	170K	–	0	narrow	–	–	–
CADPrompt (Alrashedy et al., 2025)	'25	B	200	–	0	broad	p.	–	–
CAD-Coder (Guan et al., 2025)	'25	C+M	163K	–	0	narrow	–	p.	–
CAD-Recode (Rukhovich et al., 2025)	'25	C+M	1M+7K	–	0	narrow	–	p.	seq.
cadrille (Kolodiazhnyi et al., 2026)	'26	C+M	1M+7K	–	0	narrow	–	–	–
CADEvolve (Elistratov et al., 2026)	'26	C+M	1.3M	–	0	broad	p.	–	–
BenchCAD (ours)	**'26**	**D+B**	**17,900**	**106**	**47**	**broad+**	✓	✓	✓

Table 6: **Standard-anchored families.** 49% of BenchCAD families (52/106) are standard-anchored: 51 bind their parameters to ISO/DIN/EN/ASME/IEC specification tables and one to an open-source specification, sampling at values drawn from real engineering ranges rather than arbitrary geometry. The bottom row counts the 54 custom families covering bespoke industrial parts (twisted brackets, lobed knobs, etc.) governed by analogous proportional rules without formal standards.

Standard	Codes covered (sample)	#Codes	#Fam.
ISO	22, 53, 113, 272, 606, 1234, 2339 …	17	18
DIN	315, 338, 471/472, 580, 660, 705, 950, 2095, …	21	21
EN	10034, 10056, 10219, 10279	4	5
ASME	B1.20.1, B16.5, B16.9	3	5
IEC	60072-1, 60086	2	2
Other (open-source)	–	–	1
Total standard-anchored		**47**	**52 (49%)**
Custom (no formal standard)		–	54

D Standard Codes Covered

Table 6 reports the standards grounding of BenchCAD. The 52 specification-anchored families comprise 51 tied to 47 ISO/DIN/EN/ASME/IEC codes and one tied to an open-source specification. Standard-defined dimensions coexist with generator-defined dimensions; Section 3 gives sprocket and retainer-cap examples. Table 6 is a summary, not a complete parameter-level compliance specification. The remaining 54 custom families cover bespoke industrial parts without a formal standard, while still using analogous proportional design rules.

E Dataset Generation Details

Subfamilies. A family may further branch into one or more *subfamilies* that capture mating, assembly, or construction variants of the same part type — for instance, a threaded-fastener family may split into male (`bolt`) and female (`nut`) subfamilies that share the same thread/pitch parameter table but differ in build chain; a retaining-ring family may split into internal (groove inside a bore) and external (groove on a shaft) variants; a thread family may split into single- and multi-start helices. Subfamilies reuse their parent family's parameter schema and standard-table anchoring, but specify

their own deterministic builder. They are sampled and rendered independently, so a single named family in the per-family analysis can contribute several subfamily×tier buckets to the verified release.

Difficulty tiers. Each (sub)family exposes three tiers — *easy* / *medium* / *hard* — defined by parameter complexity. Easy uses default parameter ranges with optional features disabled (e.g., a coil spring with constant pitch and no end treatment). Medium expands parameter ranges and enables a subset of optional features. Hard activates all optional features and uses extreme parameter ranges within the implemented family constraints (e.g., variable pitch with closed-and-ground spring ends), exercising the more advanced construction operations of the family. Sampling is balanced approximately uniformly across tiers within each (sub)family.

Sandbox failure-mode taxonomy. Each generated CadQuery program is executed in a sandbox subprocess; records hitting any of the following are quarantined and excluded from the release: (i) *parse / import error* — the program fails Python parsing or CadQuery API resolution; (ii) *runtime exception* — a CadQuery call raises (e.g., on a non-positive radius or an empty selector); (iii) *timeout* — execution exceeds a 30 s wall-clock budget, typically caused by pathological boolean or sweep construction; (iv) *degenerate volume* — the resulting solid has $\leq 10^{-6}$ mm^3 volume or an inverted (negative-determinant) transformation chain. Programs surviving all four checks are rendered into multi-view images and routed past a domain expert for visual sign-off; only records passing every stage enter the release.

Reverse measurement of declared holes. We checked each declared hole against cylindrical-face evidence in the exported B-rep, then investigated failures by deleting the operation, rebuilding the solid and comparing volumes. The audit flagged 351 of 17,900 parts (1.96%); in 321 cases, the declared operation removed no material, typically because it overlapped a larger cut or lay outside the valid face region. Of the 351 flagged parts, 166 belong to families without a standard table. The affected holes are generator-defined features rather than dimensions read from standards: examples include T-slot rail mounting holes sized as `slot_opening`$\times 0.7$ and a clevis tap-drill bore sized as $0.85 \times$ `pin_d`. The affected family modules have been corrected and the reverse check added to the generation pipeline.

Effect on existing evaluations. The audit concerns the original 17,900-part evaluation corpus. Its renders and ground-truth solids were produced from the same programs; identifying a declared no-op does not change those existing geometric comparisons. However, an unrealized hole in the reference operation list can penalize a model that faithfully reproduces the rendered solid. Consequently, operation-label validity must be distinguished from geometric agreement. Module correction is not evidence that scores on a newly regenerated release would remain unchanged. This targeted hole audit does not validate all features or all standard-derived constraints.

Construction diversity within a family. A family is not a single template. Two instances of the same family can start on different base planes, build features in a different order (for example, the hex of a grease nipple before or after its body), draw a profile in the opposite direction, or finish an edge with a chamfer where another uses a fillet. Together with the family-held-out split (Table 4), this limits how far a model can score by recalling a memorized program.

F Model Training

SFT training mixture. (1) **iid**: BenchCAD (all 106 families) + extrusion-heavy data (text2cad, cad-recode); (2) **ood**: same recipe, 10 mechanical families held out; (3) **baseline**: text2cad + cad-recode only (no BenchCAD). Mixing 33% BenchCAD / 67% HQ for (1)/(2); 100% HQ for (3). Backbone Qwen3-VL-2B.

OOD holdout selection. We hold out 10 CAD families sampled uniformly at random from BenchCAD families that are sufficiently represented and the operations are covered in the iid but not in the baseline.

SFT training setup. We fine-tune Qwen3-VL-2B-Instruct (Bai et al., 2025) under a standard supervised code-generation setup. Inputs consist of one rendered view and a textual instruction,

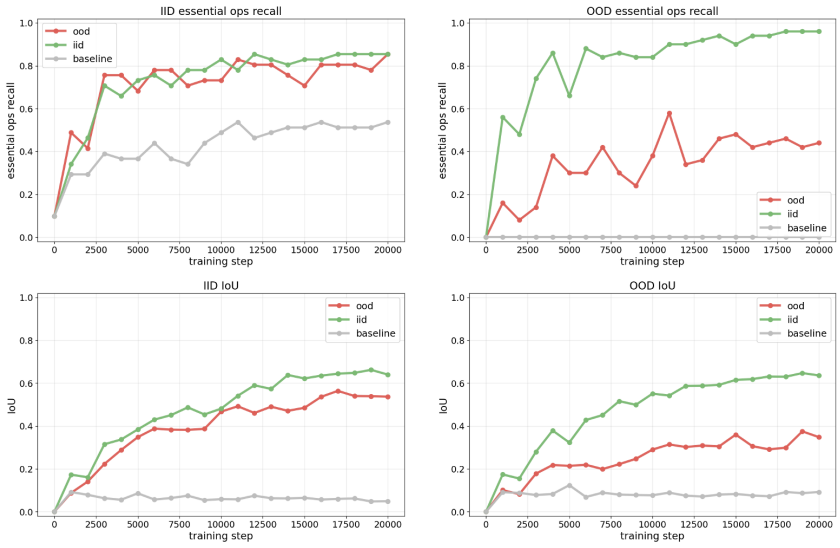


Figure 6: **The generalization gap on BenchCAD.** Qwen3-VL-2B trained on three data mixtures, evaluated on the BenchCAD validation set throughout training. *(a) OOD IoU vs. training step.* The IID-trained run (green) climbs highest; the OOD run (red, trained without the held-out family slice) plateaus mid-range; the baseline (gray, no BenchCAD) stays near the floor. *(b) OOD essential-op pass rate.* IID reaches the highest rate, OOD plateaus mid-range, baseline stays at $\sim$0% — the **CAD Operational Blindspot**. Only the OOD-trained run excludes the evaluated family slice. The gap to the IID-trained run indicates a dependence on family exposure under this training setup; it does not identify a unique cause or establish a fundamental limitation.

and targets are CadQuery programs. We use AdamW with learning rate 2×10^{-4}, cosine decay, 2k warmup steps, weight decay 0.01 with bf16 precision. The training evaluation curve (n=30) shows significant gap between model with OOD holdouts and model trained on all family (Figure 6).

RL training setup. We initialize RL from the matching SFT checkpoint: IID-SFT uses the full BenchCAD training set, whereas OOD-SFT excludes the held-out OOD families. RL is trained with a top-N GRPO-style objective without advantage normalization on a mixed prompt pool from BenchCAD, DeepCAD, and Fusion360; in the OOD setting, the held-out families remain excluded. Rewards combine geometric fidelity with an essential-operation term when available. We use learning rate 2×10^{-5}, 16 rollouts per prompt, and batch size 128.

Model Generalization. We report the full training results in Table 4 and the corresponding operation-level recall in Table 12. Compared with the pretrained baseline, both IID- and OOD-family SFT substantially improve the use of advanced CadQuery operations, indicating that BenchCAD provides effective supervision for operation-level CAD synthesis. Applying RL on top of the OOD-SFT checkpoint further improves both IID and held-out OOD performance, suggesting that geometry-based optimization improves executable fidelity beyond supervised imitation. The IID checkpoints have trained on all families, including those labelled OOD in the evaluation split. Their higher scores on that slice measure the benefit of family exposure, not generalization to families unseen during training.

G Full Per-Model Per-Task Results

Table 7 reports the dataset composition; Table 8 maps each task to its sub-capability subset; Tables 9 and 10 report supplementary original-bank QA scores; the filtered paired comparison is Table 2; Table 13 stratifies `img2cq` by family difficulty. The unified Vision2Code leaderboard (Table 11) is reported below.

Table 7: **BenchCAD dataset composition.** Three released datasets: the verified CadQuery code core (17,900 parts spanning 106 families, each with parametric source, executable STEP, four canonical-view renders, parameter JSON, and op list), the paired QA bank (2,400 questions evaluated under both visual and code conditioning), and the Edit subset (748 before/after pairs).

Dataset	#Records	#Families	GT artifacts
BenchCAD	17,900	106	code · STEP · 4 views · params · ops
BenchCAD-QA (paired img/code)	2,400 $\times$2	106	question + gold answer
BenchCAD-Edit	748	106	before/after + NL instruction

Table 8: **Capability decomposition.** BenchCAD's four tasks exercise known subsets of the four-level capability hierarchy: L_1 Holistic Visual and CadQuery Code Recognition (recognize + integrate multi-view), L_2 CAD Operations Understanding (understanding code, map ops to features), L_3 Industrial Parametric Abstraction (parameter structure, standard conventions from industry domain knowledge), and L_4 Spatial/Code Reasoning (compose all into executable program). This is a task taxonomy rather than a causal decomposition of model ability. Paired QA comparisons require answerability filtering (Table 2). "part." indicates partial coverage.

Task	Input $\rightarrow$ Output	L_1 Vis	L_1 Code	L_2 Op	L_3 Param	L_4 Synth
`img2cq`	4 views $\rightarrow$ code	✓	-	✓	✓	part.
`qa_img`	4 views + Q $\rightarrow$ text	✓	-	✓	✓	✓
`qa_code`	code + Q $\rightarrow$ text	-	✓	✓	✓	✓
`edit_img`	views + code $\rightarrow$ code	✓	✓	✓	✓	✓
`edit_code`	code + instr $\rightarrow$ code	-	✓	✓	✓	part.

Table 12 reports the recall of operations for training models.

H CAD Operational Blindspot: Per-Operation Recall

Table 14 reports operation-level recall on Vision2Code. While both models recover common sketch-and-extrude patterns such as `extrude`, `circle`, `rect`, and `workplane`, recall remains much lower for industrially important operations such as `chamfer`, `revolve`, `threePointArc`, `shell`, and counterbored holes. This gap suggests that models often approximate the final geometry with simpler primitives instead of recovering the intended parametric construction. The thinking/non-thinking differences are descriptive point estimates. Sparse operation counts and unquantified inference variability do not support a reliable ordering or a directional synthesis–execution trade-off. The more robust observation is the gap between common construction operations and several advanced operations.

Family-specific operation alternatives. Essential operations are represented as an AND of OR sets in `Vision2Code/scoring/canonical_ops.yaml`. A required set can be satisfied by an allowed alternative: for example, the `piston` and `pulley` specifications accept `revolve` or `sweep`, whereas the `coil_spring` helix requires a sweep and cannot be replaced by a revolution. Of 83 families with specifications, 66 contain at least one alternative set. These rules make essential-operation matching less dependent on one reference implementation, but do not constitute a general semantic-equivalence test. Table 14 is a per-operation diagnostic, distinct from this family-level essential-operation score.

I Sampling Sensitivity of Vision2Code

Ten API models were evaluated on three disjoint sets of 200 stems each, yielding 6,000 graded outputs. Table 15 reports the mean and standard deviation across these three sets. IoU is unconditional: every sampled input is included, and non-executing generations receive zero. This is a separate sampling-sensitivity analysis, not a replacement for the main leaderboard under an assumed identical sample or averaging convention.

Table 9: **Supplementary image QA on the original, unfiltered bank.** Axis scores are retained for comparison with the initial evaluation. Some items require source code, construction history or absolute scale and cannot be answered from normalized renders. These scores must not be interpreted as a pure visual-reasoning measure; Table 2 gives the primary filtered comparison.

Model	Holistic Visual Recognition	CAD Operation Understanding	Industrial Parametric Abstraction	Spatial Reasoning	Total score
opus-4.7	0.699	0.464	0.426	0.668	0.526
opus-4.7-thinking	0.715	**0.485**	0.421	0.614	0.530
gemini-3.1-pro	**0.750**	0.462	0.536	**0.688**	**0.587**
gemini-3.1-pro-thinking	0.722	0.426	**0.551**	0.669	0.576
gpt-4o	0.599	0.408	0.431	0.396	0.464
gpt-5.3-chat-latest	0.636	0.423	0.488	0.548	0.513
gpt-5.3-thinking	0.650	0.429	0.482	0.534	0.514
moonshot-v1-128k	0.556	0.387	0.427	0.334	0.442
moonshot-v1-8k-vision	0.600	0.246	0.465	0.181	0.447
openai-o3	0.328	0.188	0.398	0.560	0.327
blank image (baseline)	0.376	0.325	0.418	0.296	0.375

Table 10: **Supplementary code QA on the original, unfiltered bank.** Per-model numeric scores use the same axis definitions as Table 9. Because some questions are not image-answerable, differences between these full-bank tables confound modality with information availability. Use Table 2 for the filtered paired comparison; small point-estimate differences here are not evidence of a reliable ranking.

Model	CadQuery Code Recognition ↑	CAD Operation Understanding ↑	Industrial Parametric Abstraction ↑	Spatial Reasoning ↑	Total score ↑
opus-4.7	0.868	0.800	0.793	0.595	0.801
opus-4.7-thinking	0.891	0.781	0.851	0.632	0.829
gemini-3.1-pro	**0.914**	0.782	0.867	0.537	0.836
gemini-3.1-pro-thinking	0.907	0.783	**0.876**	0.537	**0.838**
gpt-4o	0.865	0.593	0.732	0.688	0.726
gpt-5.3-chat-latest	0.879	**0.805**	0.815	0.731	0.823
gpt-5.3-thinking	0.885	0.802	0.811	0.730	0.821
moonshot-v1-128k-v	0.842	0.551	0.692	**0.792**	0.700
moonshot-v1-8k-v	0.772	0.603	0.555	0.536	0.620
openai-o3	0.804	0.701	0.689	0.492	0.708
google-gemma-4-31b-it	0.791	0.674	0.606	0.528	0.664
openai-gpt-oss-120b	0.790	0.732	0.656	0.379	0.689
nvidia-nemotron-3-120b	0.771	0.660	0.661	0.293	0.671
blank code (baseline)	0.040	0.257	0.290	0.420	0.223

The largest rank change across sets is two places. The Opus thinking/non-thinking means are effectively tied at this resolution, and the GPT-5.3 difference is small relative to the observed between-set variation. These results do not support a directional thinking-mode advantage or a synthesis–execution trade-off. Repeated inference on a fixed sample and uncertainty estimates for individual QA and operation-recall comparisons are not provided by this analysis.

J External Evaluations on BenchCAD

Public model reports from OpenAI and Anthropic include Vision2Code results on BenchCAD (OpenAI, 2026c;b; Anthropic, 2026b;f;d;c;e). Figures 7 and 8 collect them together with our own runs. Two settings are distinguished: a plain run (*no tools*) and an agentic run (*with tools*), in which the

Table 11: **Vision2Code unified leaderboard on `BenchCAD`.** Specialist CAD lineage, frontier MLLMs, and our open Qwen3-VL-2B baseline. Best per block in **bold**.

Model	IoU ↑	CD ↓	ess_score ↑	exec_pct ↑	total ↑
Specialist CAD lineage (∼2B)					
Cadrille-RL (Kolodiazhnyi et al., 2026)	0.0683	0.3507	0.3118	91.2	0.3382
CADEvolve v3 (Elistratov et al., 2026)	**0.7497**	**0.0080**	**0.3715**	**92.7**	**0.6014**
Frontier MLLMs					
gpt-4o	0.2003	0.0603	0.4487	91.0	0.2393
gpt-5.3 (no thinking)	0.2298	0.0558	0.4872	81.5	0.2465
gpt-5.3 (thinking)	0.2187	0.0524	0.5128	82	0.2497
claude-sonnet-4.6 (no thinking)	0.2415	0.0486	0.5385	79.5	0.2599
claude-sonnet-4.6 (thinking-high)	0.2567	0.0465	0.5897	86.5	0.2979
claude-opus-4.7 (no thinking)	0.2740	0.0459	0.5064	95.5	0.3075
claude-opus-4.7 (thinking)	0.2790	0.0453	0.5705	**96.5**	0.3238
gemini-3.1-pro (no thinking)	0.3140	0.0432	0.5641	88.5	0.3315
gemini-3.1-pro (thinking)	**0.3546**	**0.0387**	**0.7050**	81.5	**0.3457**
openai/o3	0.2256	0.0468	0.4930	54.0	0.1978
moonshot-v1-128k	0.1281	0.0744	0.1871	12.5	0.0609
moonshot-v1-8k	0.1270	0.0847	0.1935	10.0	0.0595
Ours — Qwen3-VL-2B					
qwen3-2b-sft-ood	0.5630	0.0174	0.7380	94.2	0.5877
qwen3-2b-sft-iid	0.6400	0.0108	0.8710	88.1	0.6282
qwen3-2b-rl-ood	0.7140	0.0047	0.8100	**99.1**	0.7231
qwen3-2b-rl-iid	**0.7520**	**0.0041**	**0.8920**	98.9	**0.7682**
qwen3-2b (baseline)	0.0032	0.3652	0.0042	14.6	0.0084
gpt-4o (black img)	0.0802	0.1011	0.2244	87.0	**0.1102**

model has a Python sandbox with the task images and CAD libraries and can render, measure and revise its program before submitting.

Provenance. Claude Sonnet 4.6, GPT-5.3, Gemini 3.1 Pro and Claude Opus 4.7 are our runs from Table 11, reported as IoU-score (voxel IoU × execution rate). Kimi K3 and Grok 4.5/4.6 were also graded with our scorer, and the agentic Grok runs use our harness. All other points are self-reported by the model developers and were not re-graded: GPT-5.5 and GPT-5.6 Terra/Sol/Luna (OpenAI, 2026c); GPT-6 Astra (OpenAI, 2026b); Claude Opus 4.8, Mythos Preview and Mythos 5 (Anthropic, 2026b); and Claude Sonnet 5, Opus 5, Fable 5.1 and Opus 5.5 (Anthropic, 2026e).

Comparability. Developer-reported scores are voxel IoU rather than IoU-score and do not share one split. Anthropic reports the no-tools results for Opus 4.8, Mythos Preview and Mythos 5 on the full set and its other results on a random 1,000-file subset; GPT-6 Astra is reported on the 1,000-file subset with tools only; OpenAI's GPT-5.6 launch table does not state the split, metric details or attempt budget. In Figure 8, the Anthropic curves are digitized from the cost figure in Anthropic (2026e) and the OpenAI curves from OpenAI's published BenchCAD launch figure (OpenAI, 2026c;b); the Grok curves are our runs, priced at the published Grok 4.5 token rates (Grok 4.5 sweeps a 10/25/100-round turn budget; Grok 4.6 is a single 100-round point). These points are therefore not strictly comparable with each other or with Table 11; we include them only to document external use of BenchCAD and the gap between plain and agentic settings.

K Datasheet for BenchCAD

We follow the datasheet template of Gebru et al. (2021); the responses below summarize the construction and scope of the evaluated datasets.

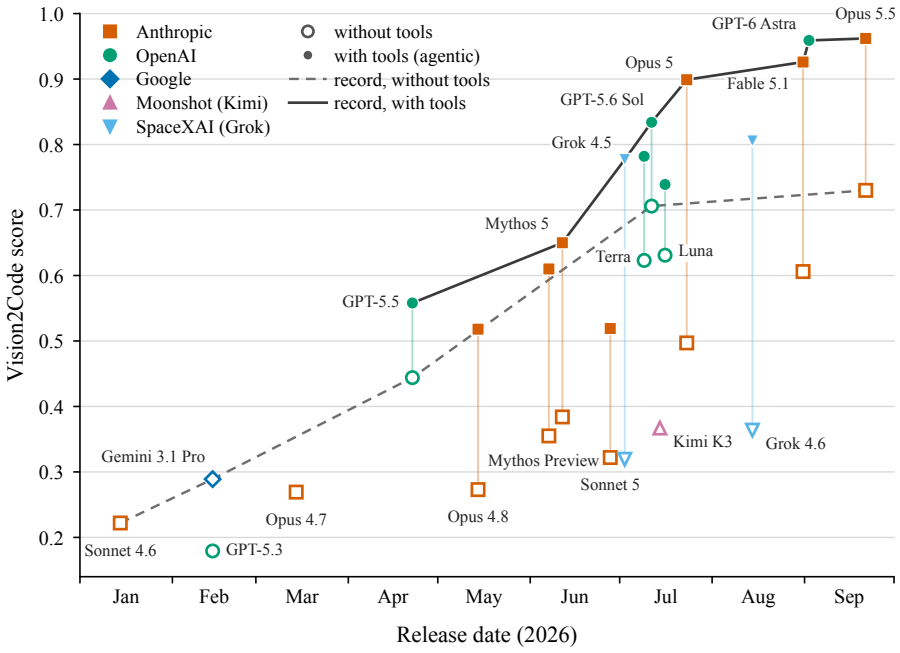


Figure 7: **Vision2Code scores of frontier models by release date (January–September 2026).** Hollow markers are no-tools runs and filled markers are agentic runs with a Python sandbox; vertical segments connect the two settings of the same model. The dashed and solid lines connect successive record scores without and with tools. Colors and marker shapes denote the developer. Scores combine our re-graded runs (IoU-score) with developer-reported voxel IoU on different splits (see text).

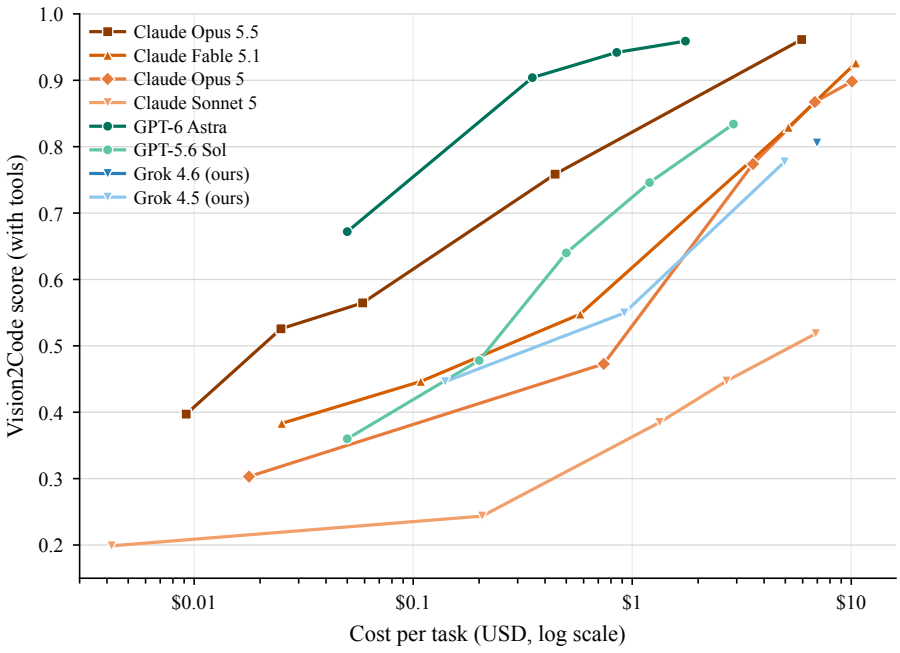


Figure 8: **Vision2Code score versus cost per task** (USD, log scale). All curves are agentic runs with tools. Anthropic and OpenAI curves are digitized from the developers' published figures; Grok curves are our runs.

Table 12: **Per-operation recall after BenchCAD training.** We compare the pretrained baseline, OOD-family SFT, and IID SFT on Vision2Code operation recall. BenchCAD training substantially improves recall on BenchCAD-distinguishing operations, especially `revolve`, `fillet`, `loft`, `sweep`, and array-based operations.

Operation	baseline	ood-sft	iid-sft	|GT|
Basic ops (shared with prior corpora: cad-recode / text2cad)				
`extrude`	**100.0**	91.7	95.8	48
`cut`	87.2	70.2	**89.4**	94
`union`	**91.7**	41.7	79.2	48
`circle`	82.8	79.3	**96.6**	58
`cylinder`	5.9	88.2	**94.1**	34
`box`	2.6	84.6	**92.3**	78
`rect`	5.0	65.0	**90.0**	40
`workplane`	0.0	77.4	**88.7**	106
Advanced ops (BenchCAD-distinguishing)				
`hole`	0.0	82.8	**93.1**	58
`chamfer`	0.0	**85.3**	**85.3**	68
`revolve`	0.0	54.8	**93.5**	62
`fillet`	0.0	25.0	**87.5**	16
`loft`	0.0	22.2	**77.8**	18
`rarray`	0.0	77.8	**100.0**	18
`sweep`	0.0	25.0	**75.0**	16
`polarArray`	0.0	33.3	**83.3**	12
`shell`	0.0	80.0	**100.0**	10
`sweep+helix`	0.0	**100.0**	50.0	4
`twistExtrude`	0.0	**100.0**	**100.0**	2
Macro recall (16 ops)	17.5%	59.9%	**84.0%**	—
Exec rate	90.0%	94.0%	**99.0%**	—

Motivation. **For what purpose was the dataset created?** BenchCAD was created to evaluate the parametric-CAD capabilities of large vision-language and code-language models along three sub-capabilities (visual perception, parametric abstraction, code synthesis) on industrially representative part families with execution-verified ground truth. **Who created the dataset and on behalf of which entity?** The authors. **Who funded the dataset?** The authors' individual funds.

Composition. **What do the instances represent?** Each instance is a parametric CAD part: an executable CadQuery Python program, four canonical-view PNG renders with their 2×2 composite, and metadata (family, variant, difficulty tier, base plane and referenced standard). The released renders are 128 px previews; the evaluation harness renders 256 px views from the executed program. STEP solids for VISION2CODE and QA are not stored: they are obtained deterministically by executing the program, and the evaluation code includes a script that builds every ground-truth STEP. Edit pairs ship both programs and both STEP files. **How many instances are there in total?** 17,900 verified parts in BenchCAD, 2,400 paired QA questions in BenchCAD-QA (each evaluated under both visual and code conditioning, yielding 4,800 records), and 748 curated edit pairs in BenchCAD-Edit. **Does the dataset contain all possible instances?** No — BenchCAD samples a subset of all the industry CAD generation part families, and sampled a subset of parameter space per family; the schema permits unbounded sampling under the documented constraints. **Is there a label/target?** The CadQuery program, and the STEP solid obtained by executing it, serve as ground truth; for QA tasks, gold answers are derived deterministically from the generator parameters and released with each question; for edit tasks, target code and target STEP are paired with the original.

Collection process. **How was the data acquired?** Procedurally generated by the BenchCAD pipeline (§3), with expert family-module review, stratified instance inspection, full-corpus visual screening and author-curated edit pairs.

Table 13: **The Family Cliff.** `img2cq` pass rate (%) split by family difficulty tier. The difficulty cliff is sharpest for the more capable models — gemini-3.1-pro and claude-opus-4.7 lose 11–21 points from easy to hard — while weaker models sit near the floor across all tiers. No model clears 22% on the hard tier. The 26 hard-tier families exercise advanced operations (helical sweeps, twist-extrusion, lofted Booleans) and parametric constraints (gear module–tooth–pitch consistency, spring free-length–turn-count coupling) that no evaluated model reliably preserves.

Model	Easy	Medium	Hard	Δ E–H
gemini-3.1-pro (thinking)	**34.7**	**31.6**	**21.2**	+13.6
gemini-3.1-pro (no thinking)	**34.7**	22.4	13.5	+21.3
claude-opus-4.7 (no thinking)	27.8	17.1	9.6	+18.2
claude-opus-4.7 (thinking)	22.2	21.1	11.5	+10.7
claude-sonnet-4.6 (thinking)	15.3	13.2	11.5	+3.7
gpt-5.3 (no thinking)	12.5	10.5	13.5	-1.0
gpt-5.3 (thinking)	11.1	9.2	11.5	-0.4
claude-sonnet-4.6 (no thinking)	11.1	17.1	7.7	+3.4
gpt-4o	9.7	3.9	7.7	+2.0
openai/o3	4.2	7.9	7.7	-3.5
Moonshot-v1-128k-vp	1.4	1.3	0.0	+1.4
Moonshot-v1-8k-vp	0.0	1.3	1.9	-1.9

Who was involved in data collection? The authors and four CAD experts involved in family-module verification. **Were any ethical review processes conducted?** Not applicable; no human-subjects data.

Preprocessing/cleaning/labeling. Was preprocessing/cleaning done? The six automated gates and human review described in Section 3 filter generated instances. A subsequent B-rep audit identifies unrealized declared holes (Appendix E), and a separate answerability audit defines the filtered QA comparison. The Fusion360 and DeepCAD subsets undergo additional curation: parsing failures and parts whose reconstruction yields invalid geometry are excluded. **Was the raw data saved?** Yes; the unverified pre-filter output is retained for analysis and is available on request.

Uses. Has the dataset been used for any tasks already? The four tasks defined in §3.2. **Is there anything that prevents responsible reuse?** The dataset contains no PII, no copyrighted source designs, and no safety-sensitive specifications. The standard-anchored families reference public ISO/DIN/EN/ASME/IEC standard codes by number and name; we do not redistribute the standard documents themselves.

Distribution. Will the dataset be distributed? Yes, on Hugging Face under `BenchCAD/BenchCAD`. **When?** Released with this preprint. **Under what license?** CC-BY-4.0 for data; MIT for code (evaluation harness). **Are there restrictions?** Standard CC-BY-4.0 attribution.

Maintenance. Who will support the dataset? The authors via the GitHub issue tracker and Hugging Face dataset card. **How will errata be communicated?** Versioned releases on Hugging Face with changelogs in the dataset card.

L Question Bank Construction

The numeric-QA tasks (`qa_img`, `qa_code`) draw from a per-family question bank constructed in three steps. (1) For each family, a domain-knowledge author writes 6–12 question *templates* parameterized by the family's exposed parameters — e.g. for `involute_gear`: *"what is the ratio of root-to-tip diameter?"*, *"how many teeth does the visible gear have?"*. Templates are typed as integer count, dimensional or ratio questions. (2) Templates are instantiated per-record by deterministic substitution of the verified parameter values, yielding a gold answer alongside each

Table 14: **Per-operation recall on Vision2Code.** We compare gpt-5.3 with gpt-5.3-thinking on operation-level recall and execution rate. Values are descriptive point estimates; small differences and sparsely represented operations do not establish a reliable thinking-mode ranking. Bold denotes the larger point estimate, not statistical significance.

Operation	gpt-5.3	gpt-5.3-thk	|GT|
Basic ops (shared with prior corpora: cad-recode / text2cad)			
`extrude`	86.7	**87.8**	90
`cut`	32.2	32.2	59
`union`	63.9	63.9	72
`circle`	82.7	**84.0**	75
`cylinder`	1.5	**3.0**	66
`box`	37.2	37.2	78
`rect`	**56.5**	50.0	46
`workplane`	78.2	**84.7**	124
Advanced ops (BenchCAD-distinguishing)			
`hole`	55.4	**59.0**	83
`chamfer`	**11.4**	10.0	70
`cutThruAll`	5.4	**21.6**	37
`revolve`	21.4	**32.1**	28
`fillet`	8.0	8.0	25
`cutBlind`	**47.4**	36.8	19
`loft`	11.8	**17.6**	17
`threePointArc`	0.0	**31.2**	16
`rarray`	14.3	14.3	14
`sweep`	46.2	46.2	13
`polygon`	**63.6**	54.5	11
`makeHelix`	0.0	**12.5**	8
`slot2D`	12.5	**25.0**	8
`polarArray`	0.0	**20.0**	5
`shell`	25.0	25.0	4
`mirrorY`	0.0	0.0	4
`sphere`	**100.0**	75.0	4
`cboreHole`	0.0	0.0	3
`radiusArc`	0.0	0.0	3
Macro recall (advanced, 19 ops)	22.2%	**25.7**%	—
Macro recall (all, 27 ops)	31.9%	**34.5**%	—
Exec rate	81.5%	**82.0**%	—

Table 15: **Unconditional IoU across three disjoint 200-stem sets.** Standard deviations describe between-set variation, not confidence intervals or same-input multi-seed variability.

Model	Mean IoU	SD
Claude Opus 4.7	.2408	.0204
Claude Opus 4.7 thinking	.2403	.0253
Claude Sonnet 4.6 thinking	.2296	.0108
Claude Sonnet 4.6	.2045	.0114
GPT-4o	.1721	.0093
GPT-5.3 thinking	.1549	.0212
GPT-5.3	.1507	.0322
o3	.1254	.0045
Moonshot 8k	.0111	.0043
Moonshot 128k	.0098	.0030

question. (3) A second author reviews instantiated questions for ambiguity and visibility. This initial review did not eliminate all questions that require information absent from the rendered views. The final bank contains 2,400 unique (question, gold answer) pairs sampled across 17,900 records — each evaluated under both visual (`qa_img`) and code (`qa_code`) conditioning, yielding 4,800 total QA records. The composition is approximately 73% integer count, 14% dimensional, and 13% ratio.

We release the question-template source code so that the bank can be regenerated under alternative parameter samples.

Answerability re-audit. Four CAD experts identified questions requiring source code, construction history, or absolute dimensions that normalized renders cannot establish. We removed those questions from both modalities and rescored the stored predictions of ten models, without new inference. Table 2 reports the original and retained-subset results. The mean code–image gap decreases from 0.278 to 0.116; excluded items are interpreted as code comprehension rather than visual-perception tests. The original bank remains 2,400 items; that count must not be mistaken for the size of the filtered subset.

M QA System Prompts

We provide the system prompts used for the original QA inference. Answerability filtering rescored these existing outputs rather than changing prompts or generating new responses. All models are evaluated with the same prompt template. The model is required to output only a JSON array of numbers, which enables automatic numeric grading. For binary questions, we encode "yes" as 1 and "no" as 0.

M.1 Image-based QA System Prompt

QA_IMG_SYSTEM_PROMPT

You are an expert CAD engineer. You will be shown a `DIAGONAL_VIEW_LAYOUT`. You will be given a list of numeric questions about the part. Answer each with a single number.

Rules:

- Output ONLY a JSON array of numbers, one per question, in the same order.
- No text, no keys, no explanation. Just the array.
- For yes/no questions, use 1 for yes and 0 for no.
- For count questions, use an integer, e.g., 12, not "twelve".
- For ratio questions, use a decimal, e.g., 2.5.
- For dimensional questions, answer in whatever consistent unit the code uses; the scale is arbitrary and the grader compares numeric magnitude.

Example input:

```
["How many teeth?", "What is the module?"]
```

Example output:

```
[20, 2.5]
```

M.2 Code-based QA System Prompt

QA_CODE_SYSTEM_PROMPT

You are an expert CAD engineer. You will be shown CadQuery Python code for a mechanical part. You will be given a list of numeric questions about the part this code produces.

Rules:

- Output ONLY a JSON array of numbers, one per question, in the same order.
- No text, no keys, no explanation. Just the array.
- For yes/no questions, use 1 for yes and 0 for no.
- For count questions, use an integer, e.g., 12, not "twelve".
- For ratio questions, use a decimal, e.g., 2.5.
- For dimensional questions, answer using the same scale as the numeric literals in the code.

Example input code: The code creates a gear with 20 teeth and module 2.5.

Example input questions:

`["How many teeth in the gear?", "What is the module?"]`

Example output:

`[20, 2.5]`

N VISION2CODE SYSTEM PROMPTS

To ensure reproducibility, we provide the exact prompts used for Vision2Code. The model is given a normalized four-view composite render of an industrial part and is asked to generate executable CadQuery Python code. All generated code is evaluated under the same execution protocol: `cadquery` is pre-imported as `cq`, and the final reconstructed solid must be stored in `result`.

N.1 PRIMARY VISION-TO-CAD SYSTEM PROMPT

SYSTEM_PROMPT

Generate CadQuery Python code from a `DIAGONAL_VIEW_LAYOUT`.

Renders are normalized: bbox centered at `[0.5,0.5,0.5]`, longest side maps to `[0,1]`. Match the orientation exactly—do not rotate or remap axes. World XYZ in your code must match world XYZ in the renders.

- `cadquery` is pre-imported as `cq`; no imports, no `show_object`.
- Store final solid in `result`.
- Output ONLY code, no explanation or markdown.

N.2 VISION-TO-CAD USER PROMPT

USER_PROMPT

Generate CadQuery code to recreate this industrial part shown in the 4-view composite render.

N.3 CADRILLE BASELINE SYSTEM PROMPT

CADRILLE_SYSTEM_PROMPT

You are a CadQuery expert. Given a normalized `DIAGONAL_VIEW_LAYOUT`.

Write CadQuery Python code that reproduces the geometry. Output ONLY Python code.

N.4 EDIT-TASK SYSTEM PROMPT

EDIT_CODE_SYSTEM_PROMPT

You are an expert CAD engineer. You will be given:

1. A CadQuery Python script that builds a parametric mechanical part.
2. A natural-language edit instruction describing a change to one specific feature.

Your task: return the script with the MINIMAL edit that achieves the requested feature change. Only the feature(s) the instruction explicitly mentions should change; every other feature, dimension, and detail must stay exactly as in the original. **Rules:**

- Output ONLY executable Python code, no explanation, no markdown fences.
- Make the smallest set of changes that realize the instruction. Do NOT touch unrelated literals, operations, or structure.
- Do NOT refactor, rename variables, reorder operations, or add/remove imports.

- If the instruction adds a new feature: append the operation in the most natural place; keep the rest of the chain intact.
- If the instruction removes a feature: delete just that operation block; do not rewrite surrounding code.
- If the instruction modifies a dimension/parameter: update only the literal(s) that depend on it (and, if present, the matching parameter comment).

EDIT_IMG_GT_SYSTEM_PROMPT

You are an expert CAD engineer. You will be given:

1. A CadQuery Python script that builds a parametric mechanical part(CURRENT state).
2. A 2×2 composite image of the TARGET geometry from 4 diagonal viewpoints.

Your task: return the script with the MINIMAL edit that transforms the CURRENT geometry (described by the code) into the TARGET geometry (shown in the image). The image is the only specification of the desired change, there is NO text instruction. Compare the code's geometry against the image and infer which single feature differs. Only the feature(s) that differ between code and image should change; every other feature, dimension, and detail must stay exactly as in the original code.

Rules:

- Output ONLY executable Python code, no explanation, no markdown fences.
- Make the smallest set of changes that realize the visual difference. Do NOT touch unrelated literals, operations, or structure.
- Do NOT refactor, rename variables, reorder operations, or add/remove imports.
- If the target image shows a new feature that is absent in the code: append the operation in the most natural place; keep the rest of the chain intact.
- If the target image shows a feature removed: delete just that operation block; do not rewrite surrounding code.
- If a dimension/parameter has changed: update only the literal(s) that control that dimension (and the matching parameter comment if present).

EDIT_CODE_IMG_SYSTEM_PROMPT (Ablation)

You are an expert CAD engineer. You will be given:

1. A CadQuery Python script that builds a parametric mechanical part.
2. A natural-language edit instruction describing a change to one specific feature.
3. A 2×2 composite image of the part from 4 diagonal viewpoints (CURRENT state) as an additional visual reference for the script.

Your task: return the script with the MINIMAL edit that achieves the requested feature change. Only the feature(s) the instruction explicitly mentions should change; every other feature, dimension, and detail must stay exactly as in the original.

Rules:

- Output ONLY executable Python code, no explanation, no markdown fences.
- Make the smallest set of changes that realize the instruction. Do NOT touch unrelated literals, operations, or structure.
- Do NOT refactor, rename variables, reorder operations, or add/remove imports.
- If the instruction adds a new feature: append the operation in the most natural place; keep the rest of the chain intact.
- If the instruction removes a feature: delete just that operation block; do not rewrite surrounding code.
- If the instruction modifies a dimension/parameter: update only the literal(s) that depend on it (and, if present, the matching parameter comment).

Table 16: **Effect of rotation-invariant IoU by GT base plane.** For GPT-4o, we report $\Delta = \mathrm{IoU}_{\mathrm{rot24}} - \mathrm{IoU}_{\text{single-axis}}$ over 164 valid 24-IoU samples with extrusion. Larger Δ indicates cases where the generated shape is geometrically similar but expressed in a mismatched coordinate frame.

GT base plane	n	Mean Δ	Max Δ
XY	70	0.0463	0.5907
XZ	40	0.2305	0.8031
YZ	54	0.2140	0.7053

O ROTATION-INVARIANT IOU

We compute IoU under a configurable rotation-invariant protocol, taking the maximum over either the 6-element face-up cube symmetry group or the full 24-element cube rotation group. Before voxelization, each part is normalized by centering it at the origin and scaling it by the largest semi-axis of its bounding box. For each candidate rotation $g \in G$, we rotate the predicted voxel grid $\hat{V}$ and compute its overlap with the reference voxel grid V; the reported score is

$$\max_{g \in G} \mathrm{IoU}(g \cdot \hat{V}, V).$$

This metric is used diagnostically rather than as the primary score: a large gain from standard IoU to 24-rotation IoU indicates that the model may have generated the right geometry on the wrong construction plane. This failure is partly induced by a workplane prior: in online database, and manufacturing-oriented modeling workflows, parts are commonly initialized on the `XY` plane as the default sketching or setup plane. As a result, models may over-prefer `XY`-based constructions even when the target geometry is defined on `XZ` or `YZ`. Consistent with this interpretation, Table 16 shows substantially larger rotation-IoU gains for `XZ`/`YZ` ground-truth parts than for `XY` parts.

P EDIT PROTOCOL

Pair construction. The 748 BenchCAD-Edit pairs are drawn balanced across 106 families and four edit categories (dimensional, additive, subtractive, and multi-step). For each pair, we (i) start from a verified BenchCAD record, (ii) apply a parameter or op-list edit yielding a target part, (iii) verify the target satisfies the same verification pipeline, (iv) write a natural-language instruction by template (*"increase the bore diameter to 12 mm"*) reviewed for unambiguity by a second author, and (v) cross-validate by running two strong models and inspecting any unexpected failure modes. Pairs where models systematically disagree on instruction interpretation are revised or removed.

Scoring. See Eq. (1) for metric definitions (normalized IoU used in edit-task score). A separate preservation diagnostic can compare the prediction and original solid outside the target-feature region. It is not reported as an independent headline result here: for local edits, whole-solid target agreement already penalizes unintended changes, while returning the original unchanged scores zero under Eq. (1). This overlap does not establish source-level minimality or resolve preservation for compound T5 edits.

Task taxonomy. The five edit task types (T1–T5) are defined in Table 17. Difficulty rises monotonically with the structural complexity of the minimal correct diff, from a single literal swap (T1) to coordinated multi-block restructures (T5). The T1–T5 axis is orthogonal to the dimensional/additive/subtractive/multi-step categorization used in the main paper: T1 and T3 are predominantly dimensional, T2 and T4 cover additive/subtractive, and T5 spans the multi-step regime.

Failure taxonomy. Every failing prediction is bucketed into one of eight semantic failure modes F01–F08 (Table 18, Figure 9), each mapped to one of the four capability layers L1–L4: F01–F03 isolate L1 *Holistic Visual Recognition* (wrong value, wrong instance, wrong placement), F04–F06 isolate L2 *CAD Operations Understanding* (wrong axis, wrong selector, near no-op), F07 isolates L3 *Industrial Parametric Abstraction* (incomplete multi-instance update), and F08 isolates L4 *Compositional Spatial-Code Reasoning* (compositional geometry mismatch). Each label combines an

Table 17: **BenchCAD-Edit task taxonomy.** The 748 edit pairs are stratified into five task types T1–T5 by the structural form of the minimal correct diff. Difficulty rises monotonically from T1 (one-literal swap) to T5 (multi-block restructure with trig or coordinated sub-feature changes).

Code	Name	Definition	Diff signature	Example instruction
T1 (n=180)	literal_replace	Replace a single numeric literal that the instruction names explicitly (*from X to Y*).	one literal swap; equal lines added/removed	*"Increase the cylinder radius from 4.5 to 5.85."*
T2 (n=104)	chain_transform	Append a single chainable whole-part transform (rotate / mirror / array).	one `result = result.<op>(...)` line appended	*"Rotate the entire part by 90 degrees about the X axis."*
T3 (n=111)	relative_compute	Read an existing literal and derive another value (ratio, percentage, $2\times$, $1/4$).	one literal swap, value derived from another literal	*"Set the central bore diameter equal to twice the hub-cylinder height."*
T4 (n=199)	feature_edit	Add or delete a single CSG sub-block (hole, slot, recess, chamfer, fillet, boss).	one 3–7 line CSG block inserted or removed	*"Drill a 6 diameter through-hole along the Z axis."*
T5 (n=154)	geometry_rebuild	Non-trivial restructure: trig-derived slopes/chamfers/lofts, or paragraph-style rebuilds where two or more sub-features change in concert.	multi-block rewrite, trig in code, paragraph instruction	*"Replace the ramped arms and raised center with a single flat cross-shaped solid of uniform thickness."*

automatic detection cue (execution status, generated-vs-original/target IoU thresholds, and topology checks) with a hand audit by the authors against the predicted code and rendered geometry; the rule set and L-mapping are fixed across all ten models reported, and `ok` (strict pass) and `exec_fail` are tracked separately as the non-failure and gross-execution categories.

Image-conditioned variants. Figure 11 compares code plus instruction, code plus target views, and code plus instruction plus original views. The instructions explicitly supply target values. Replacing them with normalized target renders makes some edits underdetermined, so the image-only condition tests a different information regime. Adding original views to the text condition yields no consistent aggregate improvement in the tested subset. We report this as a null result under the current instruction distribution, without claiming that images are generally unnecessary, harmful to thinking models, or bounded by text-only performance.

Supplementary results. Figure 9 reports failure attributes of all 748 edit-pairs from 10 LLMs. Figure 10 disaggregates per-model failures into the L1–L4 buckets defined in Figure 3. Figure 11 reports the parallel image-based setting (target render in lieu of an NL instruction) broken down by T1–T5.

Q Scoring-Protocol Ablations

Table 19 reports two scoring-protocol ablations on the Vision2Code task. (i) The *24-axial vs. single-axis IoU* contrast quantifies how much credit is recovered by accounting for wrong-plane outputs (§5.2, L_1 spatial-anchoring failure mode): when the gap is large, the model's geometry is approximately correct but emitted on a non-canonical workplane. (ii) The *single-view vs. multi-view* contrast quantifies how much of a model's score is driven by the four-view evidence specifically, isolating the cost of removing back-side and depth cues from the perception input.

Table 20 reports image-resolution point estimates. For the non-thinking model, total scores at 256, 512 and 768 px are 0.291, 0.281 and 0.287; for the thinking model, they are 0.274, 0.288 and 0.291. Thinking-model IoU peaks at 512 px (0.145), while its total score is highest at 768 px. These small differences do not establish a general resolution trend or an over-elaboration mechanism.

Table 18: **Failure-mode taxonomy (F-codes).** Eight semantic failure modes used in the per-model failure analysis (Figure 9). Each F-code is mapped to one capability layer: L1 *Holistic Visual Recognition*, L2 *CAD Operations Understanding*, L3 *Industrial Parametric Abstraction*, L4 *Compositional Spatial-Code Reasoning*. Hand-labeled on the failing predictions of every BenchCAD-Edit run.

Code	Name	Definition	Detection cue	Representative failure
F01 (L1)	wrong_param_value	Right operation, wrong numerical value: dim, diameter, radius, length, or angle written off-target; geometry shape-correct but a single number misses.	$\mathrm{IoU}_{\mathrm{gen}} \geq 0.9$, output differs from target by a single literal.	Model writes radius 5.6 when prompt asks for 5.85; cylinder topology unchanged.
F02 (L1)	wrong_feature_instance	Among multiple similar features (front vs. back ring, two legs, several circles), the model edits the wrong one or perturbs a neighbor.	$\mathrm{IoU}_{\mathrm{gen}}$ swings markedly above OR below baseline; topology preserved.	Edit asks to enlarge the rear bore; model enlarges the front bore.
F03 (L1)	wrong_placement	Right operation and value, but feature placed at slightly wrong coordinate, face, or orientation.	$\mathrm{IoU}_{\mathrm{gen}} \geq 0.95$, small rigid offset against target.	Hole drilled 2 mm off-center on a face whose origin the model misread.
F04 (L2)	wrong_axis_or_direction	Rotation, mirror, or extrude direction wrong: rotate axis tuple permuted, mirror plane misidentified, sweep tangent flipped.	`status==ok`; rotated/mirrored bbox mismatches target axis.	gpt-4o rotate: `rotate((axis), (origin), deg)` with first two args swapped.
F05 (L2)	wrong_selector_or_face	`.faces()/.edges()` selector returns wrong subset: `>Y` chosen on a Z-up part, OR-selector returns multi-face, lofted-face selector silently misses.	`status==ok`; operation applied to wrong topology vs. target.	Chamfer applied to all top edges instead of the single outer rim.
F06 (L2)	near_no_op_under_edit	Output $\approx$ original: comment-only edit, deleted wrong line, retained the feature meant to be removed, or applied the edit with no observable geometric effect.	$\lvert\mathrm{IoU}_{\mathrm{gen}} - \mathrm{IoU}_{\mathrm{orig}}\rvert < 0.005$.	claude-opus-4-7 (no-think): on 49% of the failed records that rewrite the original byte-for-byte.
F07 (L3)	incomplete_multi_update	Edit requires updating multiple symmetric or repeated instances (helical-sweep points, replicated bolts/legs); model updates only a subset.	$\mathrm{IoU}_{\mathrm{gen}}$ plateaus below 1.0 across attempts; partial symmetry break.	Cross-parameter dim asks to scale all four legs; model scales only two.
F08 (L4)	compositional_geometry_mismatch	Composite or structural shape (slope strip, loft, sweep cross-section, multi-feature reorganization) is partially right but does not match GT topology.	`status==ok`; $\mathrm{IoU}_{\mathrm{gen}} \in [0.4, 0.95]$ on a T5-style edit.	Slope edit produces a chamfer of correct angle but along the wrong face strip.

R CadQuery Operation Coverage

Table 21 reports the per-category breakdown.

Static analysis protocol. For each released CadQuery corpus we extract distinct method invocations using the regex `\.([A-Za-z_]\w*)(` on the `.py` text. We then EXCLUDE three groups: (a) class/type names (`Workplane`, `Vector`, `Wire`, `Solid`, `Sketch`, `Compound`, `Edge`, `Face`, `Plane`, `Location`, `Shape`, `Edges`, `Vertices`); (b) selectors / accessors that do not modify geometry (`face`, `faces`, `edges`, `vertices`, `wires`, `val`, `vals`, `first`, `last`, `tag`,

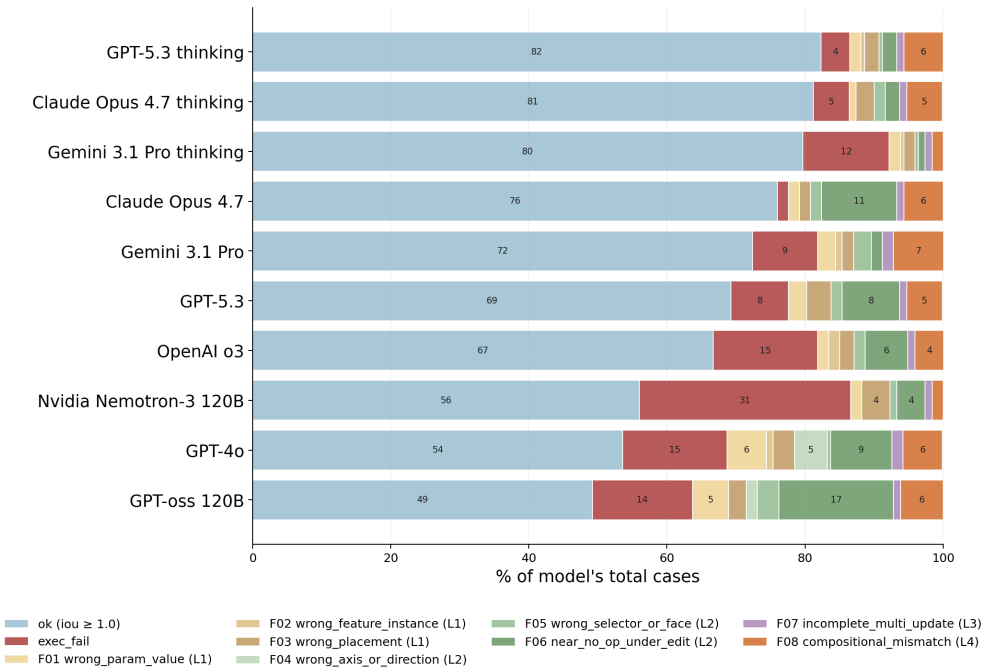


Figure 9: **Per-model failure-mode distribution on BenchCAD-Edit.** Each horizontal bar disaggregates one model's predictions into `ok`, `exec_fail`, and the eight semantic failure modes F01–F08 defined in Table 18; segments are colored by the underlying capability layer (sand = L1 *Holistic Visual Recognition*, sage = L2 *CAD Operations Understanding*, purple = L3 *Industrial Parametric Abstraction*, orange = L4 *Compositional Spatial-Code Reasoning*), so the L-mass within each bar reads off directly. Models are ordered by `ok` rate. The mass shifts systematically across model generations: older or smaller-capacity systems concentrate failures in the L2 band (F04–F06) and incur a non-trivial `exec_fail` tail; recent large models without explicit reasoning largely close the L2 gap and leave residual mass on L1 (F01–F03); reasoning-tier closed models flatten L1 and L2 alike but expose an L3–L4 ceiling (F07–F08) on multi-instance and T5-style coordinated edits, indicating that thinking lifts capability up the L-hierarchy rather than uniformly reducing all error types.

Table 19: **Scoring-protocol ablations.** (i) *24-axial vs. single-axis IoU*: comparing the cube-symmetry-group IoU (max over 24 rotations) to the standard axis-aligned IoU isolates wrong-plane errors — when 24-axial > single-axis on the same prediction, the geometry is correct but emitted on the wrong base workplane (§5.2). (ii) *Single image vs. multi-view input*: collapsing the four canonical orthographic views into a single front view removes the multi-view evidence the model relies on for depth and back-side features.

Variant	total	Δ
IoU scoring protocol (Vision2Code, fixed model output)		
24-axial rotation-invariant IoU	0.308	0.068
single-axis IoU (axis-aligned only, default)	0.240	ref
Visual input - view count (Vision2Code, single model)		
4-view canonical orthographic (default)	0.240	ref
single front view only	0.215	-0.025

`newObject`, `copyWorkplane`, `plane`); and (c) Python stdlib / numpy / utility helpers (`append`, `join`, `format`, `sin`, `cos`, `multiply`, ...). The same EXCLUDE list is applied uniformly across BenchCAD, CADEvolve seeds, CAD-Recode/cadrille, GenCAD-Code, and CADPrompt. Saturation curves: CAD-Recode at 1,274 `.py` (op count constant from 200 files onward); GenCAD-Code at 2,000 streamed samples (constant from 100 samples onward).

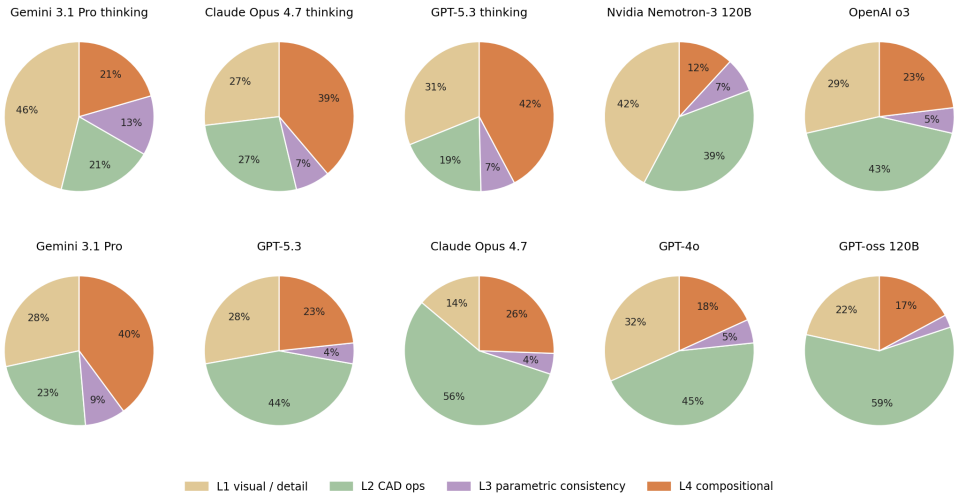


Figure 10: **Per-model failure-layer distribution on BenchCAD-Edit.** Each pie shows one model's failures aggregated by capability layer L1–L4 (collapsing the eight F-codes via the mapping in Table 18: F01–F03 → L1, F04–F06 → L2, F07 → L3, F08 → L4) and re-normalized so L1+L2+L3+L4 = 100% — i.e. the pies show the *shape* of each model's failure mode mix, independent of how often it fails overall; the title above each pie reports the absolute L-fail rate (% of all predictions) so that scale is preserved. Models are arranged in order of increasing total L-fail rate (least-broken first). The mass shifts systematically along the L-axis across model generations: older or smaller-capacity systems concentrate failures in L2 (CAD-API misuse) and incur a non-trivial `exec_fail` tail (counted separately, not shown); recent large models without explicit reasoning largely close the L2 gap and leave residual mass on L1 (visual/detail mistakes); reasoning-tier closed models flatten L1 and L2 alike but expose an L3–L4 ceiling on multi-instance and T5-style coordinated edits, indicating that thinking lifts capability up the L-hierarchy rather than uniformly reducing all error types.

Table 20: **Visual resolution ablations.** Vision2Code task score on BenchCAD across different input image resolutions. Best per block in **bold**.

Model	size (px)	IoU↑	CD↓	op_F1↑	ess_score↑	total↑
gpt-5.3-chat	256	**0.151**	**0.052**	0.455	**0.49**	**0.291**
	512	0.133	0.056	**0.497**	0.48	0.281
	768	0.142	0.055	0.494	**0.49**	0.287
gpt-5.3-thinking	256	0.121	0.060	0.474	**0.50**	0.274
	512	**0.145**	**0.053**	0.489	0.48	0.288
	768	0.143	0.054	**0.496**	0.50	**0.291**

Tier B (sketch+extrude IR works). DeepCAD, Fusion360 Gallery, and Text2CAD release a tokenized intermediate representation rather than executable CadQuery, so the CadQuery static-analysis rule does not apply directly. We instead report the IR's primitive-token alphabet: DeepCAD encodes parts as {`L`, `A`, `C`, `EXT`} plus two control tokens {`SOL`, `EOS`} (Wu et al. (2021), Sec. 3); Fusion360 Gallery r1.0.1 specifies 8 sketch curve types (`SketchLine`, `SketchArc`, `SketchCircle`, `ConicCurve`, `Ellipse`, `EllipticalArc`, `FittedSpline`, `FixedSpline`) plus extrude with 4 boolean variants (Willis et al., 2021); Text2CAD reuses the CAD-SIGNet tokenizer (∼17-token alphabet over end-of-X markers, quantized coords, and 4 extrude booleans). These counts are not directly comparable to method counts on CadQuery `.py` releases; in Table 5 we accordingly summarize each work as *narrow* (sketch+extrude IR or a small CadQuery subset) or *broad* (wide CadQuery API).

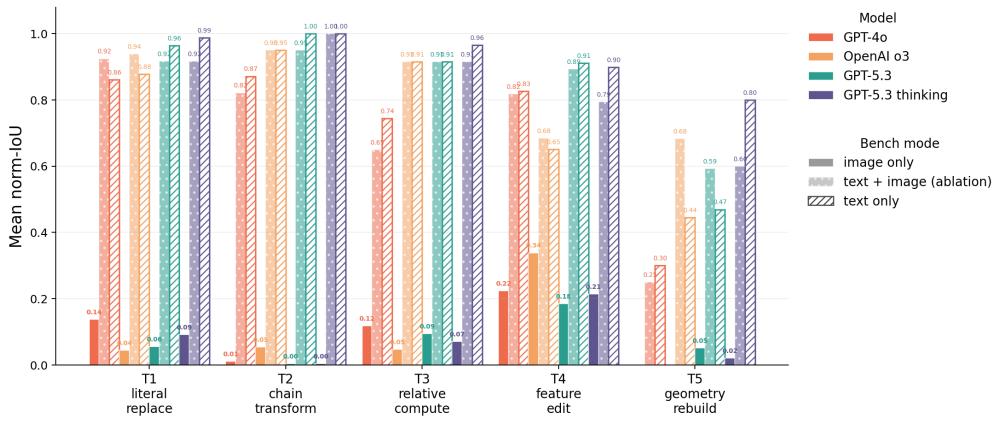


Figure 11: **BenchCAD-Edit under three input protocols.** Four OpenAI models are evaluated on the same 100-pair subset. *text*: original code and instruction; *ablation*: code, instruction and original-part views; *image-only*: code and target-part views without an instruction. Bars report normalized improvement by task type (Eq. (1)). Under this instruction distribution, adding original-part views provides no consistent aggregate benefit. Replacing the instruction removes explicit target values and reduces performance. These results do not establish an upper or lower bound, or a general causal effect of images on reasoning.

Table 21: **Per-category CadQuery operation coverage.** Distinct CadQuery method invocations per category in BenchCAD vs. the closest prior CadQuery corpus (CADEvolve), measured by the unified static-analysis rule (App. R). CADEvolve releases only 46 hand-written seed programs as source code (the expanded ∼1.3M corpus is shipped only as embeddings); its column is therefore a lower bound. Saturation curves and full op lists for all corpora (CAD-Recode, cadrille, GenCAD-Code, CADPrompt) are in App. R.

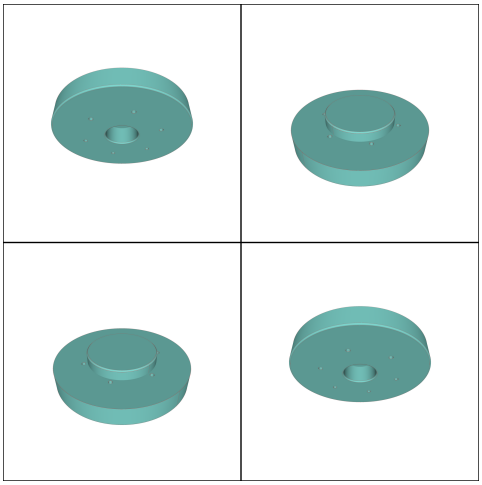
Category	BenchCAD	CADEvolve	Examples (BenchCAD)
Primitives	4	1	box, cylinder, sphere, makeTorus
2D sketch profile	15	13	circle, ellipse, spline, slot2D, threePointArc
Solid-forming	6	3	extrude, revolve, sweep, loft, twistExtrude
Advanced path	3	0	makeHelix, makeLine, assembleEdges
Boolean	5	3	cut, cutBlind, cutThruAll, union, intersect
Hole features	3	3	hole, cboreHole, cskHole
Finishing	2	1	fillet, chamfer
Arrays / Mirrors	4	3	polarArray, rarray, mirrorX, mirrorY
Transforms	3	3	transformed, rotate, workplane
Sketch builder API	1	0	placeSketch
Total	**46**	**30**	

CADEvolve caveat. CADEvolve's Hugging Face release (`kulibinai/cadevolve`, ∼2M rows, 4.7 GB) contains only `.npy` sentence embeddings keyed by component name, not Python source. The full ∼1.3M expanded scripts are referenced in the paper but are not part of the public release. We therefore report CADEvolve's operation count from the 46 hand-written seed programs in `evolution/cadquery_examples.txt` of the GitHub repo `zhemdi/CADEvolve`, yielding 30 distinct ops — a strict lower bound on the full evolved corpus.

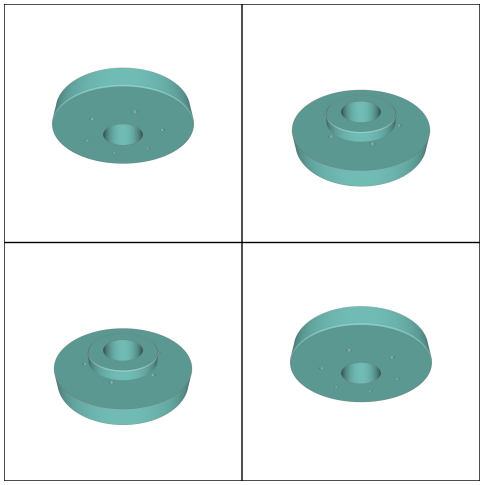
S Case Study: `motor_end_cap_bore_widen` (Case 167)

Setup. Instruction: *"Add a central cylindrical through-cut with a* 20.00 *mm radius through the flange."* Baseline IoU(orig, GT) = 0.941. Three of the four 2026 frontier models we tested return *identical* STEP outputs at IoU(gen, GT) = 0.961 — the cut takes effect on the base flange but *not* on

the upper boss, because the cut operation is placed in the middle of the build chain instead of as a final `result.cut(...)`.



Original. Motor end cap with 32.9 mm shaft hole through the flange and a 35.6 mm-radius solid boss on top.



Ground truth. Same part with the central 20 mm-radius cylindrical bore extended through both the flange and the boss.

Figure 12: **Case 167: original vs. ground-truth target.** The intended edit pierces every layer of the build chain.

Ground truth (IoU = 1.0)

```python
result = (
    cq.Workplane("XY")
    .cylinder(19.9, 69.6)              # base flange
    .faces(">Z").workplane()
    .hole(32.9)                        # original shaft hole
    .faces(">Z").workplane()
    .circle(35.6).extrude(10.4)        # upper boss (sits on flange)
    .edges(">Z").chamfer(1.1)
    .edges("<Z").fillet(1.3)
    .faces(">Z").workplane()
    .polarArray(39.79, 0, 360, 6)
    .hole(4.2)                         # 6 vent holes
)
# --- the edit: a final cylinder cut on the assembled solid
result = result.cut(cq.Workplane('XY').cylinder(100.0, 20.0))
show_object(result)
```

Idea. Build the whole part first, then `.cut` a r=20 cylinder of height 100 *after* the build chain closes. The oversized height ($100 \gg 30$) guarantees the cut pierces both the 19.9 mm flange and the 10.4 mm boss in one call.

GPT-5.3 (IoU = 0.961)

```python
result = (
    cq.Workplane("XY")
    .cylinder(19.9, 69.6)
    .faces(">Z").workplane()
    .hole(32.9)
    .faces(">Z").workplane()
    .hole(40.0)                        ← cuts only the flange
    .faces(">Z").workplane()
```

```
    .circle(35.6).extrude(10.4)        ← boss now seals the cut
    .edges(">Z").chamfer(1.1)
    .edges("<Z").fillet(1.3)
    .faces(">Z").workplane()
    .polarArray(39.79, 0, 360, 6)
    .hole(4.2)
)
show_object(result)
```

Bug. `.hole(40.0)` is inserted between the shaft hole and the boss `union`. It cuts a 40 mm-diameter through-hole on the flange's top face only; CadQuery's `.hole()` does *not* retroactively pierce a primitive `.union`-ed in afterwards. The boss is solid in its central 40 mm region.

GPT-5.3 thinking (IoU = 0.961)

```
result = (
    cq.Workplane("XY")
    .cylinder(19.9, 69.6)
    .faces(">Z").workplane()
    .hole(40.0)                        ← rewrote orig 32.9 to 40
    .faces(">Z").workplane()
    .hole(32.9)                        ← redundant:  32.9 sub of 40
    .faces(">Z").workplane()
    .circle(35.6).extrude(10.4)        ← boss seals it again
    ...
)
```

Bug. Two violations of the minimal-edit rule *plus* the same boss-seal problem: (i) modifies the existing `.hole(32.9)` (should stay) and (ii) re-adds a redundant `.hole(32.9)` that is fully contained inside the new 40 hole, hence a no-op. The geometric outcome is identical to GPT-5.3 above.

Gemini 3.1 (IoU = 0.961)

```
result = (
    cq.Workplane("XY")
    .cylinder(19.9, 69.6)
    .faces(">Z").workplane()
    .hole(40.0)                        ← replaced orig hole, in-place
    .faces(">Z").workplane()
    .circle(35.6).extrude(10.4)        ← same boss-seal failure
    .edges(">Z").chamfer(1.1)
    .edges("<Z").fillet(1.3)
    .faces(">Z").workplane()
    .polarArray(39.79, 0, 360, 6)
    .hole(4.2)
)
show_object(result)
```

Bug. The most concise of the three wrong rewrites: directly bumps the existing shaft hole from 32.9 to 40. Same structural error: cut applied before the boss is added, so the boss reseals the central column.

Failure pattern (F08, L4: spatial reasoning + code synthesis). All three IoU = 0.961 models produce the *identical* STEP file: the bore is widened to 40 mm in the 19.9 mm flange but the 10.4 mm-tall boss above it remains a solid plug. They mistake `.hole(d)` (a workplane-local cut on the current solid) for a global through-cut. The correct CadQuery idiom is to apply a final `result.cut(cq.Workplane(...).cylinder(big_h, r))`, which pierces every layer regardless of when each layer joined the build chain.

T REPRODUCTION

BenchCAD provides three independently runnable benchmarks: `CodeEdit/`, `Vision2Code/`, and `QA/`, all under a unified Python 3.11 environment managed by `uv`. After running `uv sync` and adding the required model API keys to `.env`, the full benchmark can be reproduced with `uv run python run_all.py -config prod`. Each task can also be executed individually with its own `main.py` and configuration file. Benchmark data is downloaded from the Hugging Face dataset `BenchCAD/BenchCAD`, and each task includes a small `test_data/` split for smoke testing. Model names, decoding options, and output directories are specified in YAML configuration files. We use deterministic decoding where supported and report the same task metrics as in the main paper: normalized voxel IoU for Code Edit, voxel IoU for Vision2Code, and mean symmetric ratio accuracy for QA.

U Ethical Considerations and Broader Impact

BenchCAD contains procedurally generated parametric CAD parts and references public engineering standards by code and name. The dataset contains no personal data, no proprietary designs, and no safety-sensitive specifications. Potential risks of CAD-capable models more broadly — e.g. accelerated reverse-engineering of regulated components — are not specific to BenchCAD and are best addressed at the model-deployment layer rather than the benchmark layer. We see no negative societal impact specific to this work that is not already present in the underlying open-source CAD ecosystem.