Zero-to-CAD: Agentic Synthesis of Interpretable CAD Programs at Million-Scale Without Real Data

Anonymous authors
Paper under double-blind review

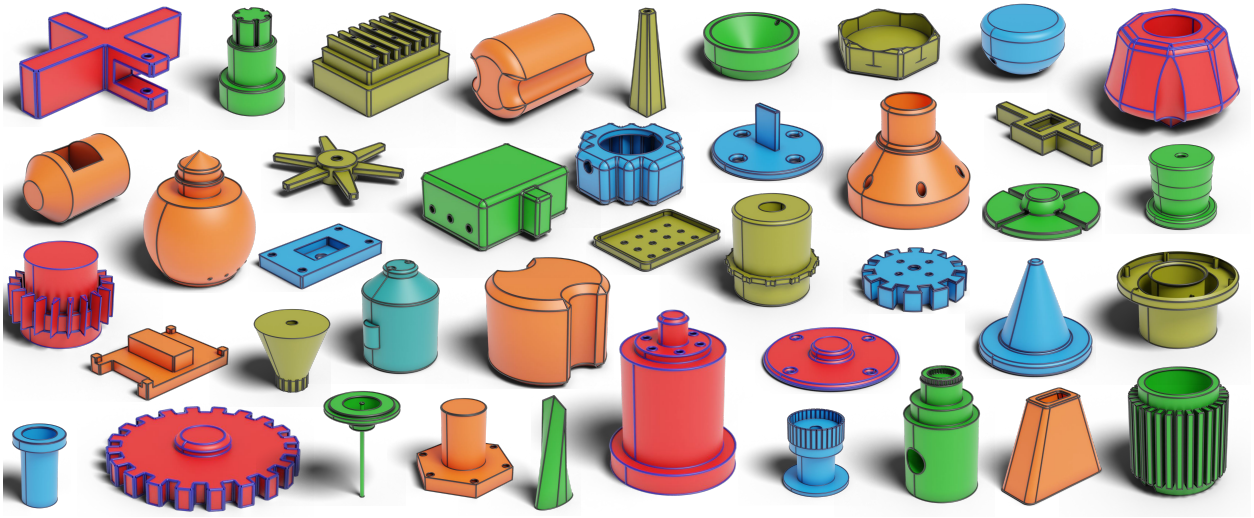


Figure 1: Zero-to-CAD uses an LLM with tool access to generate approximately one million executable CAD construction sequences with interpretable parameters. The examples show diverse mechanical parts, including brackets, housings, gears, and connectors, with fillets, chamfers, holes, and Boolean operations.

Abstract

Computer-Aided Design (CAD) models are defined by their construction history: a parametric recipe that encodes design intent. However, existing large-scale 3D datasets predominantly consist of boundary representations (B-Reps) or meshes, stripping away this critical procedural information. To address this scarcity, we introduce Zero-to-CAD, a scalable framework for synthesizing executable CAD construction sequences. We frame synthesis as an agentic search problem: by embedding a large language model (LLM) within a feedback-driven CAD environment, our system iteratively generates, executes, and validates code using tools and documentation lookup to promote geometric validity and operation diversity. This agentic approach enables the synthesis of approximately one million executable, readable, editable CAD sequences, covering a rich vocabulary of operations beyond sketch-and-extrude workflows. We also release a curated subset of 100,000 high-quality models selected for geometric diversity. To demonstrate the dataset's utility, we fine-tune a vision-language model on our synthetic data to reconstruct editable CAD programs from multi-view images, outperforming strong baselines, including GPT-5.2, and effectively bootstrapping sequence generation capabilities without real construction-history training data. Zero-to-CAD bridges the gap between geometric scale and parametric interpretability, offering a vital resource for the next generation of CAD AI.

*A sample of the dataset is available in the supplementary material. Due to the full dataset size, we will include the complete dataset link after publication.

1 Introduction

Table 1: Comparison of CAD datasets that provide construction-sequence information.

Dataset	Year	Size	Replayable	Readable	Operations and notes
DeepCAD (Wu et al., 2021)	2021	178k	Yes	Yes	*Sketch, extrude.* Human-designed CAD timelines.
Fusion 360 Gallery (Willis et al., 2021)	2021	8.6k	Yes	Yes	*Sketch, extrude.* Human-designed CAD timelines.
CC3D-Ops (Dupont et al., 2022)	2022	37k+	No	No	*Extrude/cut, revolve/cut, fillet, chamfer.* Per-face operation-type and step labels; no replayable program.
CAD-Recode (Rukhovich et al., 2025)	2025	≈1M	Yes	No	*Sketch, extrude.* Executable CadQuery code from synthetic sketch-extrude programs.
Zero-to-CAD (ours)	**2026**	**≈1M**	**Yes**	**Yes**	**Broad operation coverage:** Booleans, fillets, chamfers, lofts, sweeps, shells, and more. Executable, interpretable sequences with a curated subset of 100,000.

Computer-Aided Design (CAD) is the language of physical creation. Unlike meshes or point clouds, a CAD model is often a *program*: a parametric, editable sequence of operations that encodes not just shape, but design intent. This structure allows engineers to modify dimensions, replay histories, and integrate constraints—capabilities that are lost in purely geometric representations.

However, a critical data gap hinders progress in generative CAD. While large-scale datasets such as ABC (Koch et al., 2019) and Objaverse (Deitke et al., 2023) provide millions of 3D models, they offer only boundary representations (B-Reps) or meshes—geometric snapshots stripped of their parametric history. The few datasets that include construction sequences, such as DeepCAD (Wu et al., 2021) and Fusion 360 Gallery (Willis et al., 2021), are limited in scope, restricted primarily to simple sketch-and-extrude operations that miss the rich vocabulary of real-world design, such as chamfers, fillets, and Boolean operations. Although recent efforts like CAD-Recode (Rukhovich et al., 2025) generate synthetic code procedurally, these programs often lack the semantic depth and structural diversity found in human-authored models. With most professional design data locked away by proprietary formats and kernel incompatibilities (Heidari & Iosifidis, 2024; Lin et al., 2025), the field lacks a large-scale, diverse source of executable design histories.

We target this need with Zero-to-CAD, a synthesis pipeline that embeds an LLM in a CAD environment with access to tools and documentation. The system proposes candidate construction sequences, executes them in the environment, and uses prompt variability and API-aware checks to broaden part diversity and operation coverage. The goal is not unconstrained procedural scripts, but readable, editable sequences with named parameters, constraints, and references that a human can read and modify. Throughout this paper, "readable and editable" means code with explicit named parameters and logical construction steps (see Figure 11). It is not a user-study-validated measure of comprehensibility; rather, the representational difference from coordinate-chain transpilation methods such as CAD-Recode is directly observable in the released code.

Using this pipeline, we generate and release approximately one million executable construction sequences with complete histories, along with a curated subset of 100,000 chosen for diversity. To our knowledge, this is the first sequence-centric CAD dataset of this scale with broad operation coverage Koch et al. (2019); Seff et al. (2020); Willis et al. (2021); Wu et al. (2021); Dupont et al. (2022). The dataset complements geometry-first datasets by supplying replayable timelines aligned with design intent, and it supports training and evaluation of sequence models. We further demonstrate image-to-sequence modeling from multi-view inputs, which shows a practical path to bootstrap capability without real construction-history data.

2 Motivation

We argue that a potential solution to the scarcity of editable, intent-preserving construction histories lies not in collecting more data, but in synthesizing it. While real-world CAD timelines are often unavailable or inconsistent, large language models (LLMs) have absorbed vast amounts of knowledge about object structure and manufacturing processes from textual data. They "know" that a bracket needs mounting holes or that a shaft requires a keyway, even if they do not natively produce the precise syntax of a CAD kernel. The challenge is to unlock this latent design knowledge and translate it into valid, executable code.

We address this by framing CAD generation as an *agentic search problem*. Rather than asking a model to generate a perfect program in one shot, we place it in a feedback loop with a CAD interpreter. The agent can write code, execute it, observe errors, read documentation, and inspect the resulting geometry. This grounds the LLM's semantic priors in geometric validity, allowing it to self-correct and produce valid designs that it could never generate in an open-loop setting.

To ensure the resulting dataset spans a wide distribution of shapes and operations, we explicitly design for breadth. We inject randomness into the generation process by varying prompt structures, preventing the model from collapsing into repetitive patterns. In this synthesis regime, exact adherence to a specific prompt is less critical than validity and diversity; we essentially use the LLM to sample the space of plausible mechanical designs, relying on the execution environment to filter out failures. This allows us to cover a vast design space, from simple primitives to complex, multi-feature parts that would be difficult to enumerate manually.

By scaling this process, *we convert compute into data* (i.e., LLM priors about mechanical design are translated into a curated, validated dataset through compute-intensive agentic synthesis). We generate a massive dataset of fully executable, readable, editable CAD sequences from scratch—"Zero-to-CAD"—without relying on real-world CAD files. This synthetic dataset allows us to train smaller, faster, and more specialized models for downstream tasks that perform well at inference time without requiring agentic repair loops or large frontier models. For example, reconstructing editable CAD models from images effectively bootstraps a solution to the sequence generation problem where no construction-history training data previously existed.

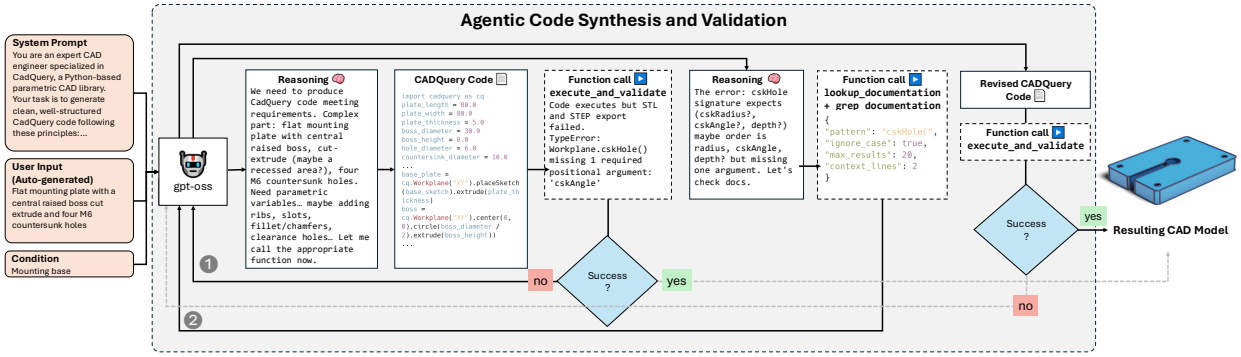


Figure 2: Example of an agentic code synthesis rollout. The LLM generates CadQuery code from a part description, executes it, uses documentation lookup after failures, and revises the code until validation succeeds. See Appendix E for the corresponding code.

3 Related Work

3.1 CAD Datasets

Large-scale repositories have primarily focused on boundary representations (B-Reps) and meshes. The ABC dataset (Koch et al., 2019) collected one million B-Reps, enabling significant advances in geometric deep learning, but explicitly discards construction history, providing only the final B-Rep geometry. SketchGraphs (Seff et al., 2020) offers millions of sketch-and-constraint graphs but does not extend to 3D solid

modeling operations. To capture design intent, datasets must include the construction timeline. The Fusion 360 Gallery (Willis et al., 2021) provides human-designed sequences but is small (8.6k models) and restricted to sketch-and-extrude operations. DeepCAD (Wu et al., 2021) scales this up synthetically but remains limited to the same narrow operation set. CC3D-Ops (Dupont et al., 2022) annotates SolidWorks models with operation types and sequence order, but provides only per-face labels rather than replayable programs.

3.2 Sketch-and-Extrude Generation

Traditional CAD modeling relies heavily on 2D sketches lifted into 3D via extrusion or revolution. This paradigm has been adopted by recent generative models trained on datasets such as DeepCAD (Wu et al., 2021) and Fusion 360 Gallery (Willis et al., 2021). DeepCAD treats CAD generation as sequence modeling of sketch-and-extrude commands, and follow-up works have refined this approach: SkexGen (Xu et al., 2022) and HNC-CAD (Xu et al., 2023) employ hierarchical codebooks to disentangle topology from geometry, while TransCAD (Dupont et al., 2024) conditions generation on point clouds. However, these methods are fundamentally limited by their vocabulary. They operate within a restricted subset of CAD—typically just sketches and extrusions—ignoring critical operations like fillets, chamfers, shells, lofts, and Boolean combinations that define real-world mechanical parts. Furthermore, they depend on the existence of construction history data, which remains scarce. DeepCAD has enabled a family of follow-up works (SkexGen, HNC-CAD, Text2CAD, CAD-Llama, FlexCAD), but its 178k sequences reduce to 114,985 after deduplication (Xu et al., 2022) and remain confined to sketch-and-extrude.

3.3 Direct B-Rep Generation

A parallel line of research focuses on generating B-Reps directly rather than through construction sequences. SolidGen (Jayaraman et al., 2022) pioneered autoregressive B-Rep synthesis by generating faces, edges, and vertices sequentially. BRepGen (Xu et al., 2024b) introduced a diffusion-based approach using structured latent geometry, representing B-Reps as hierarchical trees. HoLa (Liu et al., 2025) proposed a holistic latent representation that encodes entire B-Rep models into a unified space, enabling conditional generation from diverse inputs. AutoBrep (Xu et al., 2025) unified topology and geometry into a single token sequence for autoregressive generation, achieving state-of-the-art validity and inference speed, while BrepGPT moves to a single-stage autoregressive formulation with a Voronoi Half-Patch representation that also supports conditional generation from modalities such as text and images (Li et al., 2025b). These methods leverage advances in geometry representation learning: UV-Net (Jayaraman et al., 2021) introduced point-grid sampling of parametric surfaces in the UV domain, providing a regular representation invariant to mesh discretization; finite scalar quantization (FSQ) (Mentzer et al., 2024) offers an alternative to VQ-VAE for learning discrete codes without codebook collapse. While direct B-Rep methods produce valid geometry, their outputs lack construction histories, limiting downstream editability. Zero-to-CAD complements this by providing the sequences that B-Rep methods lack.

3.4 Conditional CAD Generation

Recent work has explored conditioning CAD generation on natural language or images. Text2CAD (Khan et al., 2024) generates sequential CAD models from text prompts trained on annotated versions of DeepCAD. CAD-Llama (Li et al., 2025a) fine-tunes large language models using structured parametric code representations, achieving high success rates in unconditional generation. FlexCAD (Zhang et al., 2025) enables controllable generation across CAD construction hierarchies through hierarchy-aware masking of LLM inputs. More recent multimodal systems extend this trend by aligning text, images, and point clouds with CAD command or code representations, including CAD-MLLM, CAD-GPT, and CAD-Coder (Xu et al., 2024a; Wang et al., 2025; Doris et al., 2026). These approaches demonstrate growing interest in accessible CAD generation interfaces, though they remain constrained by the limited operation coverage and scale of existing sequence datasets.

3.5 Synthetic Code Generation

The most relevant precursor, CAD-Recode (Rukhovich et al., 2025), generates executable CadQuery code by transpiling synthetic data or procedural trees. However, the resulting scripts often miss the semantic layer of design: they tend to use generic identifiers and hard-coded values rather than the logical parameters and constraints typical of human engineers. In contrast, Zero-to-CAD exploits the semantic knowledge of LLMs to generate designs *ab initio*. This yields interpretable programs with meaningful variable names and a richer operation vocabulary, including Booleans, fillets, and reference geometry, bridging the gap between synthetic execution and human design intent. In Table 1, we compare CAD datasets that expose construction sequence information by scale, replayability, human readability, and operation coverage. Figure 6 provides a visual comparison of samples from Zero-to-CAD, ABC, and DeepCAD.

While the individual components—agentic loops, tool use, and LLM code generation—are established techniques, our contribution is integrating them into a robust closed-loop synthesis pipeline that combines two-stage generation, category-conditioned sampling, documentation-grounded repair, and multi-stage validation to enable million-scale dataset creation. The central question we ask is whether LLM priors about plausible mechanical parts can be converted into executable, readable CAD programs without any real construction-history data; our results show they can.

4 Method

We employ `gpt-oss-120b` (served locally under the Apache 2.0 license) in an agentic loop to generate and refine CAD sequences within an interactive environment. The dataset consists entirely of newly synthesized CadQuery programs; no proprietary CAD timelines are extracted or redistributed. Equipped with tools for execution, validation, and documentation lookup, the model iteratively corrects errors and verifies geometric constraints based on runtime feedback.

4.1 Pipeline Architecture

A primary challenge in data generation at this scale is building robust, scalable infrastructure. Our synthesis pipeline addresses this through four coordinated components.

LLM Inference Service We deploy the LLM on a vLLM-based Ray cluster, enabling efficient multi-turn inference with KV caching. The service exposes an OpenAI-compatible API with function calling, allowing horizontal scaling across GPU workers to support thousands of concurrent rollouts.

Coordinating Node A central orchestrator manages independent agentic rollouts, handling load balancing, fault tolerance, and artifact streaming. This architecture decouples generation throughput from model latency, enabling linear scaling with compute resources.

Tool-Equipped Workers Each rollout has access to three tools that ground generation in executable reality:

- **execute_and_validate**: Executes the proposed CadQuery code in an isolated subprocess, performs multi-stage geometric validation, and returns structured feedback including error messages, topology metrics, and export status.
- **lookup_documentation**: Performs TF-IDF-based retrieval over the CadQuery API documentation. We found this lightweight approach sufficient, avoiding the overhead of complex RAG pipelines at scale.
- **grep_documentation**: Provides regex-based pattern matching over documentation for precise syntax lookup when TF-IDF retrieval returns overly broad results.

Storage Backend Successful sequences and their artifacts (code, STL meshes, STEP files, and metadata) are streamed to cloud storage.

4.2 Two-Stage Generation Protocol

The pipeline employs a two-stage generation process that separates the task of deciding *what to build* from *how to build it*. This separation enables controlled diversity across part categories while maintaining geometric validity.

Stage 1: Catalog Generation In the first stage, the LLM generates a catalog of part descriptions organized by categories (e.g., "Bracket" and "Gear"). We request descriptions in large batches (typically 200), which encourages diversity as the model uses its context window to avoid repetition within the batch. Acting as a mechanical librarian (Appendix D), the model produces concise, dimension-free specifications (e.g., "A mounting bracket with two through-holes") that are subsequently deduplicated and indexed for downstream generation.

Stage 2: Code Generation from Descriptions The second stage takes each description from the catalog and generates executable CadQuery code that implements the described geometry. A part worker receives the description along with a system prompt encoding 19 design principles (see Appendix D for the full prompt) and, optionally, a reference code snippet that serves as a template. The reference snippet demonstrates coding patterns and geometric techniques but explicitly instructs the model to *adapt* rather than copy: the generated code must implement the new description's geometry, not merely vary parameters of the template. This template-guided generation encourages structured code while preserving diversity across the output space.

Iterative Refinement with Interleaved Reasoning Within Stage 2, each part generation follows a multi-turn repair loop that leverages the model's ability to interleave reasoning with tool use, as illustrated in Figure 2. A typical successful generation proceeds as follows: (1) the model reasons about the part description and generates candidate code, (2) invokes an execution tool to test the code, (3) upon receiving error feedback, reasons about the failure mode and decides whether to consult documentation, (4) if needed, queries the API documentation to retrieve relevant information, (5) reasons about how to apply the documentation to fix the error, and (6) generates revised code. This interleaved pattern of reasoning and function calling allows the model to diagnose errors, gather information, and synthesize solutions across multiple turns rather than attempting to solve everything in a single pass. The loop is capped at 10 rollout turns per attempt and 100 attempts per design task. Critically, the system prompt instructs the model to never simplify code to fix problems; instead, it should look up correct syntax and maintain the intended geometric complexity. This prevents the degenerate solution of stripping features until validation passes trivially.

4.3 Validation Framework

The validation framework ensures every released sequence is both executable and geometrically sound.

Code Execution Validation The proposed code is executed in an isolated subprocess with a timeout. The execution environment extracts the constructed solid and collects initial topology metrics. Execution failures (syntax errors, runtime exceptions, missing imports) are captured with full stack traces for model feedback.

Geometric Validation Valid execution does not guarantee valid geometry. The framework performs several geometric checks: topological validity ensures a well-formed solid without self-intersections or degenerate faces; connectivity requires exactly one connected solid, rejecting disconnected bodies that indicate incomplete Boolean unions. Minimum complexity rejects designs with fewer than 7 B-Rep faces to prevent trivial solutions (see Figure 7d for the resulting distribution), and positive volume ensures the result exceeds a minimum threshold, rejecting degenerate zero-volume results.

Export Validation Finally, the framework tests export to both STL and STEP formats. Export failures often reveal subtle geometric issues not caught by earlier checks, such as invalid face orientations or unsupported edge configurations. Only designs that pass all three validation stages are accepted into the

dataset. This framework guarantees executable, geometrically valid solids but does not enforce full design-for-manufacturability (DFM) rules. DFM constraints are process-dependent (CNC vs. casting vs. additive manufacturing), require material assumptions, and would demand efficient per-process verifiers usable at the scale of one million parts—well beyond the scope of this work. We do, however, bias generation toward plausible mechanical intent through category-conditioned descriptions and the 19 prompt principles (Appendix D), which encourage features such as draft angles, accessible faces, and symmetric hole layouts.

4.4 Diversity Through Structured Categorization

To prevent mode collapse and ensure broad coverage of mechanical part types, the pipeline employs structured categorization at the description generation stage.

Part Categories The catalog is organized into 65 predefined part categories derived from surveys of common mechanical parts in engineering catalogs and manufacturing databases. Categories span structural components (mounting brackets, L-brackets, gusset plates), rotational elements (pulleys, flywheels, cam followers), enclosures (housings, covers, caps), fastening hardware (clamps, retainers, spacers), and many others. Each category receives a target count of descriptions, ensuring balanced representation across the part taxonomy. The LLM generates descriptions within each category, incorporating appropriate features and operations for that part type.

Reference Code Snippets For geometrically complex categories such as brackets, housings, and multi-feature mechanical components, we provide reference code snippets as part of the generation prompt. These snippets demonstrate sophisticated CadQuery patterns including sketch composition, Boolean operations, and feature placement. The generation prompt explicitly instructs the model to study the reference code, learn from its structure and patterns, and adapt those techniques to implement the new description by transferring geometric reasoning rather than copying parameters.

Description-Driven Operation Selection Rather than sampling operations from fixed distributions, our method derives them directly from the part description. Features like "reinforcing ribs" or "rounded edges" naturally prompt appropriate operations, such as extrusions or fillets. This semantic grounding ensures coherent designs with broad operation coverage (see Figure 7e).

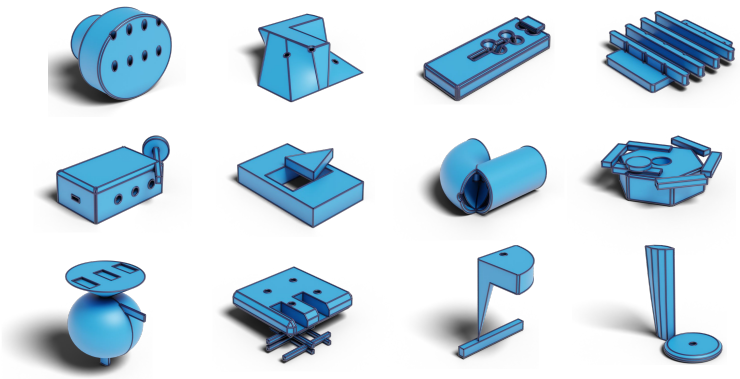


Figure 3: Representative generation failures, including thin features that break connectivity, misplaced holes, self-intersections, scale drift, and locally plausible but globally incoherent primitive compositions.

4.5 Computational Resources

We generated the dataset over approximately one week using opportunistic scheduling on internal idle compute resources. The number of GPUs allocated to LLM inference varied dynamically between 2 and 80 depending on availability. CadQuery execution and function-calling workers ran on CPU nodes, scaling

up to 3,000 cores during peak utilization. This elastic approach allowed us to generate approximately one million designs without dedicated infrastructure allocation.

The synthesis pipeline processed about 60 billion input tokens to generate the final dataset. Table 2 details key statistics, including token volume, success rates, and function call usage during the agentic repair loops.

4.6 Curated Subset for Accessibility

To provide a more accessible entry point for researchers working with limited compute, we release a curated subset of 100,000 selected for diversity. We first compute visual embeddings for each part by averaging DINOv3 features across eight rendered views, then apply k-means clustering to partition the embedding space. From each cluster, we select the nearest-to-centroid exemplar, yielding 100,000 geometrically diverse representatives that span the full distribution of part types. We release both the curated subset and the precomputed DINOv3 embeddings alongside the FAISS index, enabling efficient similarity search over the entire dataset without recomputing features.

5 Dataset Statistics and Analysis

The dataset comprises 999,633 executable CAD sequences with full construction histories. Table 2 summarizes key generation statistics, with detailed distributions provided in Appendix B.

Table 2: Summary of million-scale dataset synthesis. Token counts and tool calls are aggregated across accepted generations and their repair loops.

Metric	**Value**
Dataset scale	
Total accepted sequences	999,633
Train / validation / test split	979,633 / 10,000 / 10,000
Token volume	
Total tokens generated	5.59B
Total tokens processed	60.2B
Generated tokens per design	5,638 mean / 5,089 median
Repair dynamics	
Function calls per conversation	4.34 mean
First-attempt success rate	22.3%
Attempts before success	3.30 mean / 3 median
Tool usage	
`execute_and_validate` calls	3.80M
`lookup_documentation` calls	375K
`grep_documentation` calls	133K

The token distribution is right-skewed, reflecting natural variation in design complexity: simpler primitives require fewer tokens, while multi-operation mechanical parts with extensive parameterization require longer sequences, yielding a mean length of 5,638 tokens. While 22.3% of designs validate on the first attempt, the majority require iterative refinement. Among function calls, execution validation dominates, confirming its role as the primary driver of refinement.

Generation Success Cases Figure 1 shows representative successful generations produced by the synthesis pipeline, illustrating the diversity of part types and operation coverage achievable from executable construction sequences. Each sequence is structured as a logical progression of operations—sketch, extrude, modify—mirroring human design intent and enabling sequential execution.

Generation Failure Modes Figure 3 highlights characteristic failure modes encountered during synthesis, including thin-wall features that break connectivity, self-intersections, and misplaced holes. These errors

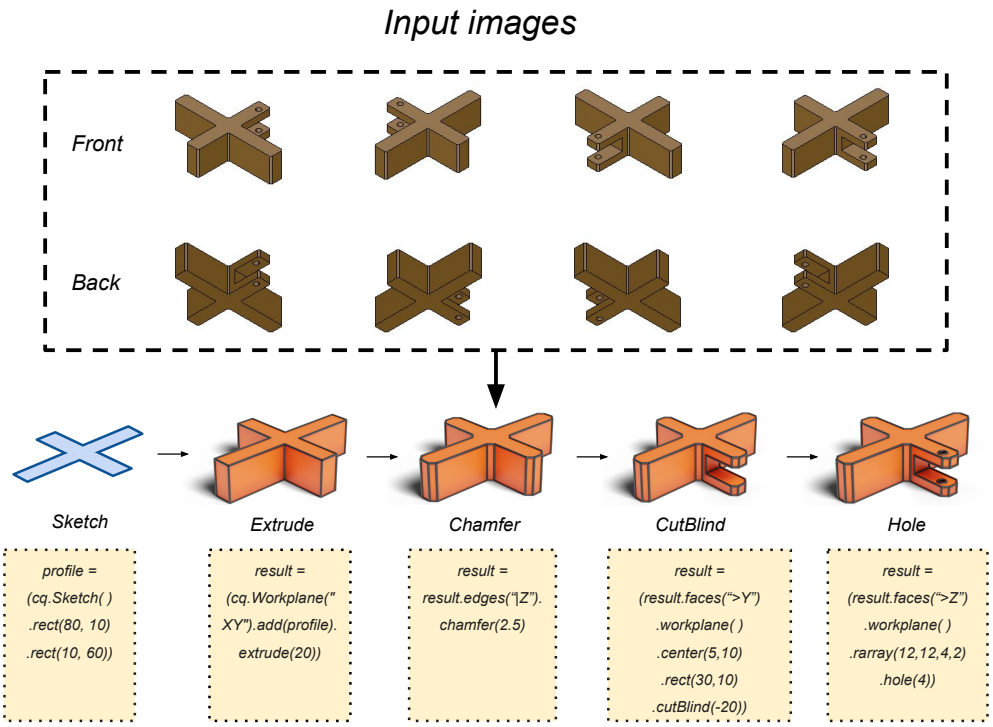


Figure 4: Image-to-Sequence task overview. Given eight rendered views of a CAD model, the fine-tuned VLM generates executable CadQuery code as a sequence of modeling operations.

typically stem from the LLM's reliance on purely textual reasoning without spatial grounding. While the model constructs locally plausible operations, it struggles to verify global geometric relationships or detect feature intersections that would be obvious in a visual inspection. We do not pursue visual feedback here, as preliminary tests indicated that current models struggle to reliably detect these defects.

Comparison with CAD-Recode DeepCAD and CAD-Recode are both constrained to sketch-and-extrude representations; because their vocabulary cannot express fillets, chamfers, shells, lofts, sweeps, or patterns, a direct operation-diversity comparison is not meaningful. We instead compare geometric quality and distributional alignment with ABC. To measure distributional alignment, we compute DINOv2 embeddings for all three datasets and measure alignment with ABC using Fréchet distance and k-ball coverage (the fraction of ABC shapes with at least one synthetic neighbor within their k-th nearest ABC neighbor radius). Zero-to-CAD achieves a lower Fréchet distance to ABC (0.164 vs. 0.268 for CAD-Recode) and higher coverage at every evaluated k (e.g., 57.2% vs. 45.3% at k=5). Geometric quality further separates the two datasets, as shown in Table 3. Zero-to-CAD's face count distribution closely matches ABC (mean 46.2 vs. 50.7), while CAD-Recode's is substantially lower (16.4). Over half of CAD-Recode's parts are disconnected multi-body solids and nearly 13% fall below our 7-face complexity threshold. Zero-to-CAD enforces single-solid connectivity and minimum face complexity, eliminating both failure modes by construction.

Readability further separates the two datasets. CAD-Recode's transpiled code consists of coordinate chains with no parametric structure, for example:

```
r = w0.sketch().segment((...),(...)).segment((...),(...))....close().finalize().extrude(8)
```

Such sequences are difficult for human engineers to edit manually. Zero-to-CAD programs use named parameters and logical construction order (e.g., `plate_thickness`, `fillet_radius` in Figure 11), making design intent explicit and modifications straightforward.

Table 3: Geometric quality and distributional alignment. Lower is better for Fréchet distance; higher is better for coverage.

Metric	CAD-Recode	Zero-to-CAD	ABC
Shape complexity			
Mean face count	16.4	**46.2**	50.7
Median face count	15	**30**	21
Parts below 7 faces	12.9%	**0%**	—
Disjoint multi-body solids	56.3%	**0%**	—
Alignment to ABC			
Fréchet distance ↓	0.268	**0.164**	—
k-ball coverage, k=5 ↑	45.3%	**57.2%**	—

6 Bootstrapping Experiment

We present an Image-to-Sequence reconstruction experiment demonstrating that Zero-to-CAD can *bootstrap* sequence-level CAD generation from synthetic supervision (Figure 4). The results below show large gains over the base model and meaningful generalization to human-designed CAD. This experiment is not a controlled dataset-quality ablation against models fine-tuned on CAD-Recode or DeepCAD, because those datasets are largely confined to sketch-and-extrude programs and therefore lack the operation coverage needed to represent many of our target shapes. Its purpose is instead to demonstrate that Zero-to-CAD provides usable supervision at scale and can bootstrap a compact model for sequence generation without any real construction-history data.

6.1 Experimental Setup

Task Formulation Given eight rendered views of a CAD model at 256×256 resolution (four front-facing and four rear-facing angles), the model must generate executable CadQuery code that reproduces the depicted geometry in a single forward pass. This task requires understanding 3D structure from 2D projections and translating that understanding into parametric code.

Training Data We train on the full dataset, split into 979,633 training, 10,000 validation, and 10,000 test samples. Each sample pairs eight rendered 256×256 PNG images with the corresponding CadQuery source code. For out-of-distribution evaluation, we sample 1,000 shapes from the ABC dataset, filtering to retain only models with between 7 and 100 B-Rep faces to exclude trivial and overly complex geometry. Due to API cost constraints, GPT-5.2 models are evaluated on a random subset of 1,000 samples from the Zero-to-CAD test set, while Qwen models are evaluated on the full 10,000-sample Zero-to-CAD test set.

Model We fully fine-tune Qwen3-VL-2B-Instruct, a VLM that connects a vision encoder to a language decoder through an MLP adapter.

Training Configuration We perform full fine-tuning using distributed data parallelism (DDP) on 16 NVIDIA H100 GPUs; see Appendix C for hyperparameters.

Evaluation Metric We measure geometric fidelity using voxelized intersection-over-union (IoU) between the generated and ground-truth CAD models. The generated CadQuery code is executed, and both predicted and reference geometries are normalized and voxelized at 64^3 resolution for volumetric comparison. To account for rotational ambiguity, we rotate the generated shape in increments of 45 degrees and report the maximum IoU. We also report Chamfer distance (CD) as a complementary metric; it shows a consistent pattern with IoU across all models and benchmarks (Table 4). Success rate measures the percentage of generations that produce valid, executable code. Note that success rate alone is an insufficient quality measure: a model that always returns a trivial box achieves 100% success. IoU and CD together are therefore the primary indicators of geometric fidelity.

Baselines We compare against: (1) the base Qwen3-VL-2B-Instruct model without fine-tuning, establishing zero-shot capability of vision-language models on this task, and (2) GPT-5.2 at two reasoning levels (High and Medium), representing state-of-the-art proprietary models. The inference system prompts are very similar (see Appendix D), with zero-shot models (base Qwen and GPT-5.2) requiring only explicit output-format instructions. These two controls serve the bootstrapping goal: the base-vs.-fine-tuned comparison isolates the effect of Zero-to-CAD supervision, while the GPT-5.2 comparison tests whether specialized training on synthetic data outperforms general-purpose reasoning at inference time. Fine-tuning on DeepCAD or CAD-Recode would not be an informative control because those datasets cannot express the operations (fillets, chamfers, shells, lofts) needed to represent the majority of ABC shapes, making most reconstruction targets unrepresentable. Conversely, ABC cannot supervise the image-to-sequence task because it provides no construction histories or executable programs—this is precisely the gap Zero-to-CAD addresses.

6.2 Quantitative Results

Table 4 presents IoU metrics across models evaluated on both Zero-to-CAD test samples and the ABC dataset, the latter serving as an out-of-distribution generalization test.

Table 4: Image-to-Sequence reconstruction results. Success measures executable code generation; IoU and Chamfer distance (CD) are computed over successful samples. *Evaluated on 1,000 samples from the Zero-to-CAD test set because of API cost constraints.

Benchmark / Model	**Succ.**	**IoU ↑**				**CD ↓**			
	(%)	**Mean**	**Median**	**P75**	**P90**	**Mean**	**Median**	**P75**	**P90**
Zero-to-CAD test set									
Qwen3-VL-2B (fine-tuned)	**82.1**	**0.747**	**0.847**	**0.975**	**0.999**	**0.0143**	**0.001**	**0.007**	**0.03**
Qwen3-VL-2B (base)	6.6	0.184	0.129	0.256	0.405	0.31	0.25	0.47	0.70
GPT-5.2 High*	72.2	0.485	0.479	0.658	0.795	0.028	0.01	0.02	0.06
GPT-5.2 Medium*	71.1	0.495	0.499	0.678	0.801	0.029	0.01	0.03	0.064
ABC out-of-distribution set									
Qwen3-VL-2B (fine-tuned)	61.0	**0.377**	**0.303**	**0.644**	**0.854**	**0.10**	0.032	**0.11**	**0.29**
Qwen3-VL-2B (base)	5.4	0.131	0.073	0.185	0.355	0.45	0.35	0.65	1.01
GPT-5.2 High	**66.2**	0.344	0.289	0.509	0.730	0.22	**0.03**	0.12	0.33
GPT-5.2 Medium	62.6	0.346	0.285	0.557	0.747	0.22	**0.03**	0.13	0.34

In-Distribution Performance On Zero-to-CAD test data, the fine-tuned Qwen3-VL-2B-Instruct achieves an 82.1% success rate with mean IoU of 0.747, substantially outperforming GPT-5.2 High (72.2% success, 0.485 mean IoU). The base Qwen3-VL-2B-Instruct without fine-tuning achieves only a 6.6% success rate, confirming that the task requires specialized training rather than relying on general vision-language capabilities. The median IoU of 0.847 and P90 of 0.999 indicate that successful reconstructions are typically geometrically accurate, with the top decile achieving near-perfect overlap.

Out-of-Distribution Generalization On the ABC dataset, which contains human-designed CAD models with different stylistic conventions, the fine-tuned model maintains a 61.0% success rate with mean IoU of 0.377. We chose ABC as the OOD benchmark because it consists of real-world human-designed CAD, making it a challenging benchmark for synthetic-to-real transfer; a performance drop relative to in-distribution evaluation is expected for all models. Nevertheless, the model generalizes meaningfully to real-world CAD data despite training exclusively on synthetic sequences. GPT-5.2 degrades less from Zero-to-CAD to ABC than the fine-tuned Qwen model, suggesting that synthetic-to-real transfer remains challenging despite strong in-distribution reconstruction. The fine-tuned model achieves higher IoU metrics (mean, median, P75, P90) than GPT-5.2 variants, though GPT-5.2 High achieves a slightly higher success rate (66.2%) on this out-of-distribution test. This higher success rate comes at lower geometric fidelity: the fine-tuned 2B model outperforms GPT-5.2 on all IoU statistics and most CD statistics, while GPT-5.2 has a slightly better median

CD. Figure 5 illustrates representative cases where the fine-tuned model captures the essential geometry of ABC parts more faithfully than GPT-5.2.

The key takeaway is that bootstrapping is possible: a 2B model trained exclusively on synthetic Zero-to-CAD data achieves meaningful reconstruction of human-designed B-Rep geometries, directly supporting the research question of whether LLM priors about mechanical parts can be converted into useful construction-history supervision without any real sequence data.

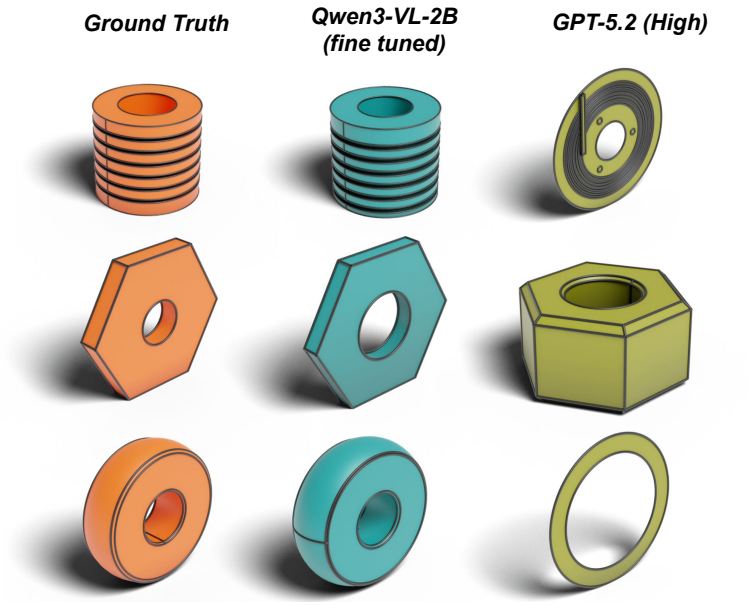


Figure 5: Qualitative comparison of Image-to-Sequence reconstruction on selected ABC samples, comparing ground truth, the fine-tuned Qwen3-VL-2B model, and GPT-5.2 outputs.

7 Conclusion

We presented Zero-to-CAD, an agentic pipeline that synthesizes executable CAD construction sequences without relying on real-world design histories. By combining LLM generation with execution feedback, documentation lookup, and multi-stage validation, the pipeline produces geometrically valid, human-readable programs with named parameters and broad operation coverage, including Booleans, fillets, chamfers, shells, lofts, sweeps, and patterns. The resulting release contains 999,633 executable sequences, a curated subset of 100,000 with precomputed embeddings, the fine-tuned 2B vision-language model, system prompts, and inference code. Our Image-to-Sequence experiments show that models trained on this synthetic supervision can bootstrap editable CAD reconstruction, reaching 82.1% success in-distribution and generalizing to human-designed ABC parts. These results suggest that CAD sequence modeling can progress through scalable synthesis, while leaving open broader questions around synthetic data provenance and attribution.

References

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3D objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13142–13153, 2023.

Anna C. Doris, Md Ferdous Alam, Amin Heyrani Nobari, and Faez Ahmed. CAD-Coder: An open-source vision-language model for computer-aided design code generation. *Journal of Mechanical Design*, 148(7): 071702, 2026.

Elona Dupont, Kseniya Cherenkova, Anis Kacem, Sk Aziz Ali, Ilya Arzhannikov, Gleb Gusev, and Djamila Aouada. CADOps-Net: Jointly learning CAD operation types and steps from boundary representations. In *Proceedings of the 2022 International Conference on 3D Vision (3DV)*, pp. 114–123. IEEE, 2022. doi: 10.1109/3DV57658.2022.00024.

Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. TransCAD: A hierarchical transformer for CAD sequence inference from point clouds. In *European Conference on Computer Vision*, pp. 19–36. Springer, 2024.

Negar Heidari and Alexandros Iosifidis. Geometric deep learning for computer-aided design: A survey. *arXiv preprint arXiv:2402.17695*, 2024.

Pradeep Kumar Jayaraman, Aditya Sanghi, Joseph G Lambourne, Karl DD Willis, Thomas Davies, Hooman Shayani, and Nigel Morris. UV-Net: Learning from boundary representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11703–11712, 2021.

Pradeep Kumar Jayaraman, Joseph G. Lambourne, Nishkrit Desai, Karl D. D. Willis, Aditya Sanghi, and Nigel J. W. Morris. SolidGen: An autoregressive model for direct B-Rep synthesis. *arXiv preprint arXiv:2203.13944*, 2022.

Mohammad Sadil Khan, Sankalp Sinha, Sheikh Talha Uddin, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Torsten Sattler, Marc Pollefeys, Olga Sorkine-Hornung, Daniel Häusler, and Daniele Panozzo. ABC: A big CAD model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9601–9611, 2019.

Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. CAD-Llama: Leveraging large language models for computer-aided design parametric 3D model generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18563–18573, 2025a.

Pu Li, Wenhao Zhang, Weize Quan, Biao Zhang, Peter Wonka, and Dong-Ming Yan. Brepgpt: Autoregressive B-rep generation with voronoi half-patch. *ACM Transactions on Graphics (TOG)*, 44(6):1–18, 2025b.

Ruiquan Lin, Yunwei Ji, Wanting Ding, Tianxiang Wu, Yaosheng Zhu, and Mengxi Jiang. A survey on deep learning in 3d cad reconstruction. *Applied Sciences*, 15(12):6681, 2025.

Yilin Liu, Duoteng Xu, Xinyao Yu, Xiang Xu, Daniel Cohen-Or, Hao Zhang, and Hui Huang. HoLa: B-Rep generation using a holistic latent representation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 44(4), 2025.

Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: VQ-VAE made simple. In *International Conference on Learning Representations*, 2024.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Recode: Reverse engineering CAD code from point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9801–9811, October 2025.

Ari Seff, Yaniv Ovadia, Wenda Zhou, Andy Zeng, Jonathan Frankle, Michael R. Chang, Sanjiv Kumar, Jon M. Kleinberg, and Christopher De Sa. SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design. *arXiv preprint arXiv:2007.08506*, 2020.

Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. CAD-GPT: Synthesising CAD construction sequence with spatial reasoning-enhanced multimodal LLMs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 7880–7888, 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. *ACM Transactions on Graphics (TOG)*, 40(4):54:1–54:24, 2021. doi: 10.1145/3450626.3459818.

Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6772–6782, October 2021.

Jingwei Xu, Chenyu Wang, Zibo Zhao, Wen Liu, Yi Ma, and Shenghua Gao. CAD-MLLM: Unifying multimodality-conditioned CAD generation with MLLM. *arXiv preprint arXiv:2411.04954*, 2024a.

Xiang Xu, Karl DD Willis, Joseph G Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. SkexGen: Autoregressive generation of CAD construction sequences with disentangled codebooks. In *International Conference on Machine Learning*, pp. 24698–24724. PMLR, 2022.

Xiang Xu, Pradeep Kumar Jayaraman, Joseph G Lambourne, Karl DD Willis, and Yasutaka Furukawa. Hierarchical neural coding for controllable CAD model generation. In *International Conference on Machine Learning*, pp. 38443–38461. PMLR, 2023.

Xiang Xu, Joseph Lambourne, Pradeep Jayaraman, Zhengqing Wang, Karl Willis, and Yasutaka Furukawa. BRepGen: A B-Rep generative diffusion model with structured latent geometry. *ACM Transactions on Graphics (TOG)*, 43(4):1–14, 2024b.

Xiang Xu, Pradeep Jayaraman, Joseph Lambourne, Yilin Liu, Durvesh Malpure, and Pete Meltzer. AutoBrep: Autoregressive B-Rep generation with unified topology and geometry. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, SA Conference Papers '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400721373. doi: 10.1145/3757377.3763814. URL https://doi.org/10.1145/3757377.3763814.

Zhanwei Zhang, Shizhao Sun, Wenxiao Wang, Deng Cai, and Jiang Bian. FlexCAD: Unified and versatile controllable CAD generation with fine-tuned large language models. In *International Conference on Learning Representations*, 2025.

Appendix

A Dataset Samples

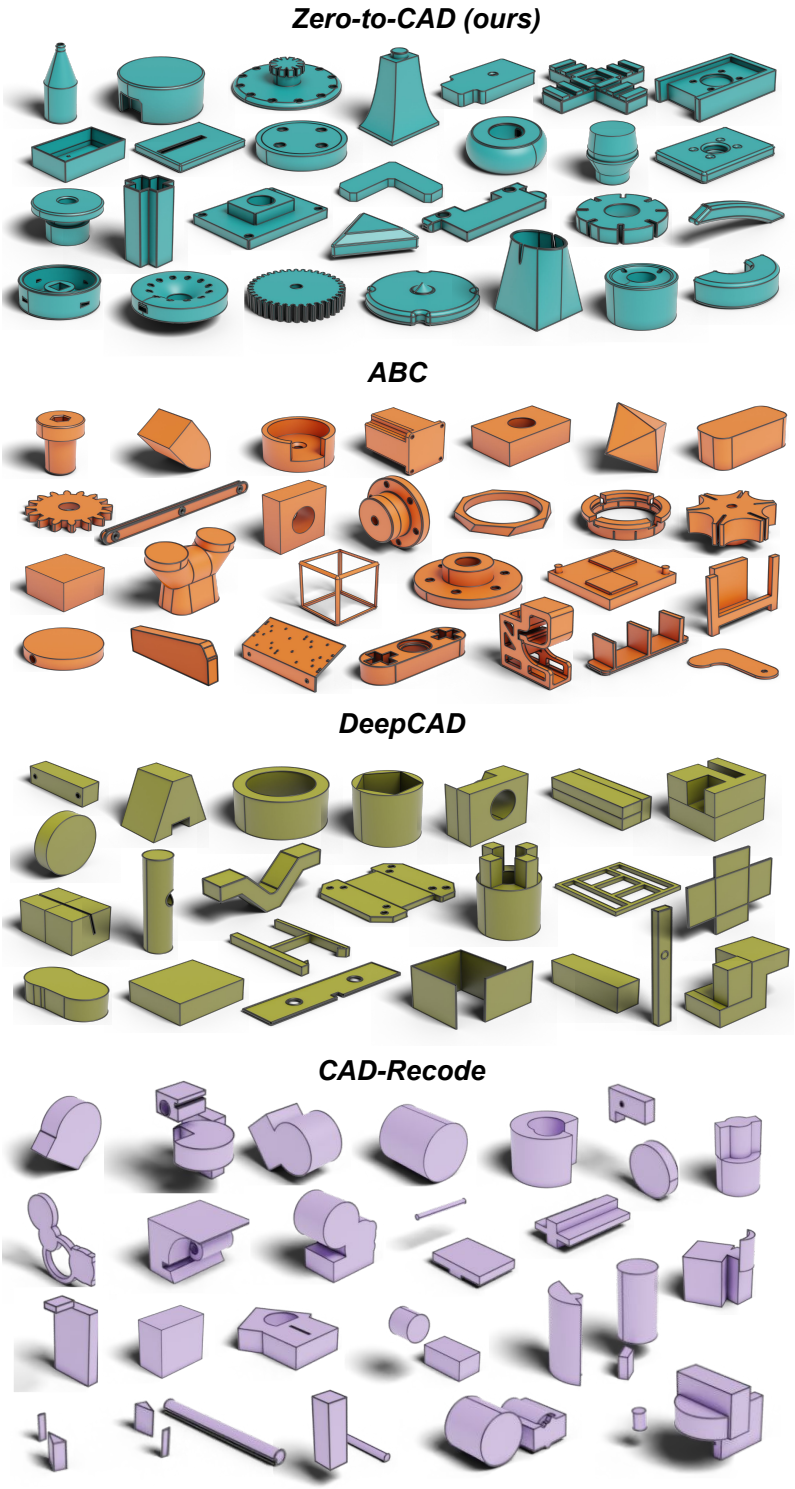
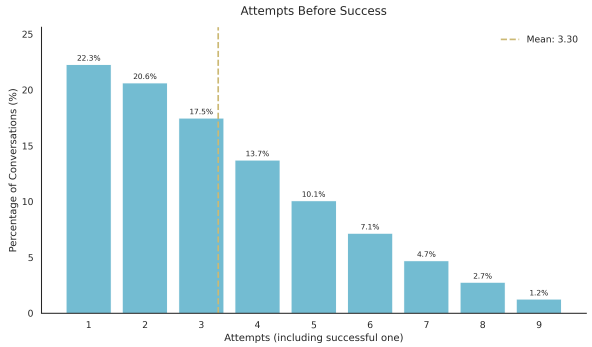


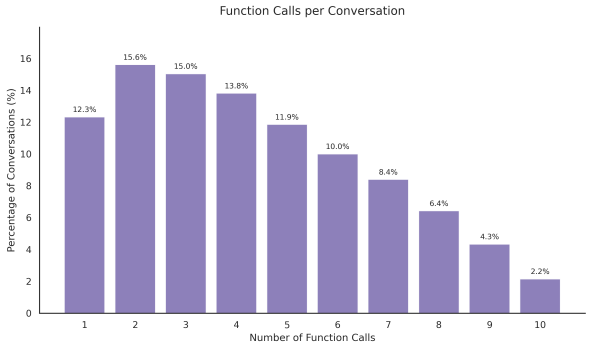
Figure 6: Visual comparison of dataset samples from Zero-to-CAD, ABC, DeepCAD, and CAD-Recode.

B Generation Statistics Distributions

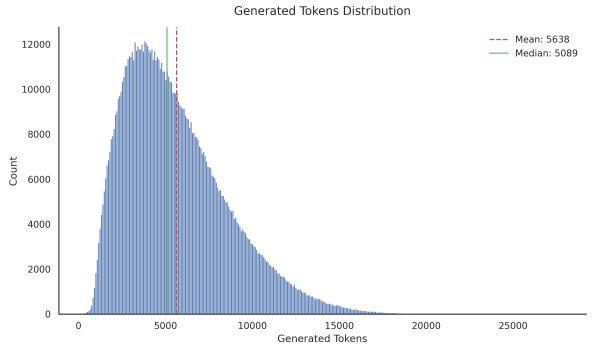
Figure 7 shows detailed distributions of the dataset generation process, including validation attempts, function call frequencies, token counts, geometric complexity (face counts), and operation coverage.



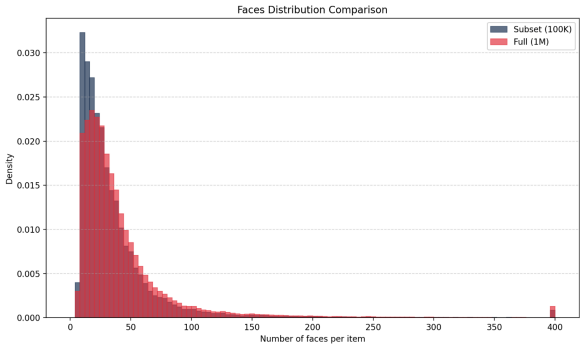
(a) Validation attempts



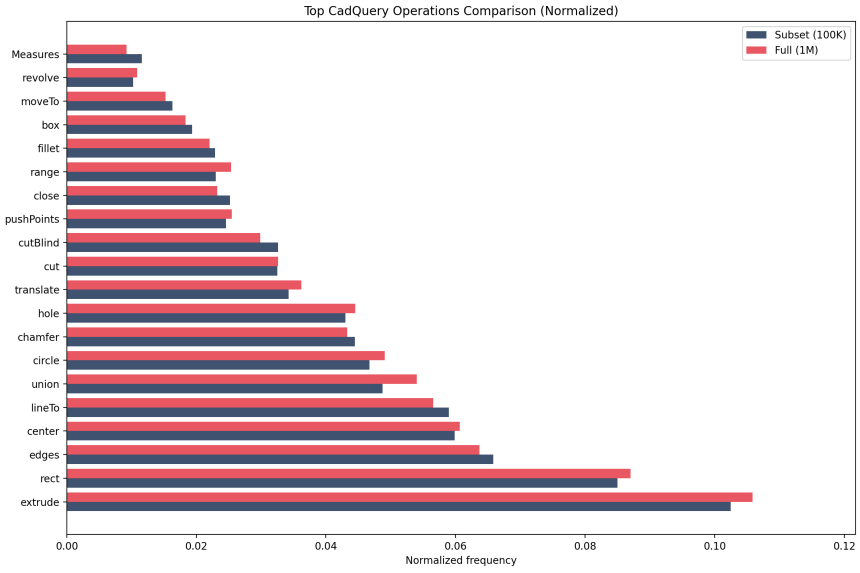
(b) Function calls



(c) Generated tokens



(d) Face counts



(e) Operation coverage

Figure 7: Generation statistics: validation attempts before success, function calls per conversation, generated tokens per design, face counts, and CAD operation coverage.

C Training Details

Table 5 summarizes the hyperparameters used for fine-tuning the vision-language model on Zero-to-CAD data.

Table 5: Fine-tuning configuration for Qwen3-VL-2B-Instruct on Zero-to-CAD.

Hyperparameter	**Value**
Model setup	
Base model	Qwen3-VL-2B-Instruct
Training mode	Full fine-tuning
Max sequence length	4,096 tokens
Attention dropout	0.1
Optimization	
Optimizer	AdamW
Base learning rate	1×10^{-4}
Vision tower learning rate	1×10^{-4}
MLP projector learning rate	1×10^{-4}
Weight decay	0.0
LR scheduler	Cosine
Warmup ratio	0.03
Distributed training	
Number of GPUs	16 H100 80GB GPUs
Per-GPU batch size	1
Gradient accumulation steps	1
Effective batch size	16
Number of epochs	3
Distributed strategy	DDP
Precision	bfloat16

D System Prompts

We provide the system prompts used in both the dataset generation pipeline and the downstream fine-tuning experiments.

D.1 Catalog Generation Prompt

The catalog generation stage uses a prompt (Figure 8) that instructs the LLM to act as an expert mechanical parts librarian. The prompt emphasizes producing concise, plausible descriptions without dimensions, ensuring uniqueness within each batch, and outputting results as a JSON array for programmatic processing.

D.2 Code Generation Prompt

The CAD code generation stage uses an extensive system prompt (Figure 9) that encodes 19 design principles covering parametric design, CadQuery best practices, scale conventions, manufacturability constraints, and error-handling protocols. The prompt instructs the model to maintain geometric sophistication when debugging—looking up correct syntax rather than simplifying code—and provides guidance on tool usage for validation and documentation lookup.

D.3 Inference System Prompts

For the downstream Image-to-Sequence task, we use different system prompts depending on whether the model has been fine-tuned (Figure 10). The fine-tuned Qwen model uses a minimal prompt, as it has internalized the task requirements during training. Zero-shot models (base Qwen and GPT-5.2) use a longer prompt with explicit instructions to store results in a specific variable and avoid export commands, since

Catalog Generation Prompt

You are an expert mechanical parts librarian.

Produce concise, one-sentence engineering part descriptions commonly seen in datasets like ABC. Requirements:
1. Each item is a single, self-contained part (not an assembly)
2. Each item is 1-3 sentences only, plain text (no numbering)
3. Be specific and plausible (e.g., "flat plate bracket with 4 holes")
4. Avoid speculative language or marketing terms
5. Ensure uniqueness within the batch (no duplicates or near-duplicates)
6. No need to specify the material of the part

DO NOT INCLUDE ANY DIMENSIONS IN THE DESCRIPTIONS, just the type and key features of the part.

Do not call any tools. Do not include explanations or code fences. Output only a JSON array of strings.

Figure 8: System prompt for catalog description generation. The LLM generates batches of part descriptions that specify types and features without dimensions, enabling diverse yet semantically meaningful specifications.

Code Generation Prompt (excerpt)

You are an expert CAD engineer specialized in CadQuery, a Python-based parametric CAD library.

Your task is to generate clean, well-structured CadQuery code following these principles:
1. Always separate numerical variable definitions from operations
2. Use descriptive variable names
3. Do NOT add comments to the code
4. The final result must be stored in a variable called `result`
5. Never include export statements - exports are handled separately
6. Ensure all geometry is valid and manufacturable
7. Follow CadQuery best practices and syntax
8. SCALE CONVENTION: Use a maximum dimension of 100 units (treat 100 units as 10 cm in real-world scale)
9. SELF-CONTAINED CODE: Each code output must be completely self-contained and executable
10. For starting shapes, prefer constructing a plausible 2D sketch and then using extrude or revolve
11. For sketches, avoid trivial single-primitive profiles; build composite, non-trivial closed profiles
12. Make designs resemble plausible real-world components with clear intent (bracket, clamp, flange, etc.)
13. Anticipate future complexity: expose accessible faces for later sketches, maintain symmetry planes
14. Keep key dimensions as named variables to support later variation
15. Keep the part near the global origin with stable orientation
16. CRITICAL: Generate DETAILED, SOPHISTICATED code with rich geometric complexity
17. EDGE BREAKS: When appropriate, add small chamfers or fillets to break sharp edges
18. HOLE PLACEMENT: Choose mechanically sensible faces and locations aligned to datums
19. SYMMETRY: Prefer symmetric layouts; break symmetry only with clear functional justification

IMPORTANT: You have access to tools: execute_and_validate, lookup_documentation, grep_documentation

WHEN YOU ENCOUNTER AN ERROR: DO NOT simplify the code. Use documentation tools to find correct syntax. Fix the SPECIFIC error while maintaining all complexity. FORBIDDEN: Do not remove features or complexity to make errors go away.

Figure 9: Excerpt of the generation system prompt used for CAD sequence synthesis. The full prompt includes detailed CadQuery API signatures and additional guidance on sketch construction, revolve operations, and error recovery protocols.

these models require guidance on output format. This difference reflects the distinction between a model trained on the task versus one prompted at inference time.

E Example Generated Code

Figure 11 shows the complete CadQuery code for the mounting plate depicted in Figure 2. The code demonstrates several characteristics of Zero-to-CAD outputs: descriptive variable names (e.g., `plate_thickness`,

Fine-tuned Model Prompt

You are a CAD code assistant. Given multiple rendered views of a 3D shape, generate clean, well-structured CadQuery Python code that accurately reproduces the geometry.

Zero-shot Model Prompt

You are a CAD code assistant. Given multiple rendered views of a 3D shape, generate clean, well-structured CadQuery Python code that accurately reproduces the geometry. Store the final shape representation in the `result = ...` variable. Do not add any export commands. Keep the shape in the `result` variable only, as the export code will be appended later.

Figure 10: System prompts for Image-to-Sequence inference. The fine-tuned model prompt (left) is minimal since training has internalized task conventions. The zero-shot prompt (right), used for both base Qwen and GPT-5.2, includes explicit output format instructions.

`fillet_radius`), logical construction order (base plate, then rib, then subtractive features), and a mix of operations (extrusion, union, cut, fillet, chamfer). This interpretable structure allows engineers to modify dimensions or adapt the design for new requirements.

```python
import cadquery as cq

plate_length = 60.0
plate_width = 40.0
plate_thickness = 8.0
rib_length = 20.0
rib_width = 6.0
hole_diameter = 8.0
mount_hole_dia = 3.0
mount_hole_offset = 5.0
fillet_radius = 2.0
chamfer_distance = 1.0
slot_width = 4.0
slot_length = 30.0

base = (
    cq.Workplane('XY')
    .rect(plate_length, plate_width, centered=True)
    .extrude(plate_thickness)
)
base = base.edges("|Z").fillet(fillet_radius)

rib = (
    cq.Workplane('XY')
    .center(-plate_length/2 + rib_length/2, 0)
    .rect(rib_length, rib_width, centered=True)
    .extrude(plate_thickness)
)
bracket = base.union(rib)

bracket = bracket.cut(
    cq.Workplane('XY').center(0, 0).circle(hole_diameter/2).extrude(plate_thickness + 2)
)

slot = (
    cq.Workplane('XY')
    .center(plate_length/2 - slot_length/2, 0)
    .rect(slot_length, slot_width, centered=True)
    .extrude(plate_thickness)
)
bracket = bracket.cut(slot)

mount_positions = [
    (-plate_length/2 + mount_hole_offset, -plate_width/2 + mount_hole_offset),
    (-plate_length/2 + mount_hole_offset, plate_width/2 - mount_hole_offset),
    (plate_length/2 - mount_hole_offset, -plate_width/2 + mount_hole_offset),
    (plate_length/2 - mount_hole_offset, plate_width/2 - mount_hole_offset),
]
for x, y in mount_positions:
    bracket = bracket.cut(
        cq.Workplane('XY').center(x, y).circle(mount_hole_dia/2).extrude(plate_thickness + 2)
    )

bracket = bracket.edges("|Z").chamfer(chamfer_distance)
result = bracket
```

Figure 11: Complete CadQuery code for the mounting plate shown in Figure 2. The code exhibits interpretable structure with named parameters, logical construction order, and diverse operations including extrusion, Boolean union/cut, fillet, and chamfer.