buildings

MDPI

Article

Execution-Supervised Text-to-CAD Generation for Reliable AI-Assisted Parametric Design

Lizhi Niu [1], Jisheng Zhao [2,*] and Wei Li [3]

[1] College of Civil Engineering, Tongji University, No. 1239 Siping Road, Shanghai 200092, China; lizhi_niu@tongji.edu.cn
[2] Floating AI, No. 1500 Shenkun Road, Minhang District, Shanghai 201107, China
[3] Independent Researcher, Hangzhou 311100, China; liwei1996er@gmail.com
* Correspondence: zhaojisheng@foda.ai

Abstract

Reliable text-to-CAD generation remains challenging because syntactically valid programs may fail during geometric execution or produce inaccurate geometric models. Existing approaches often treat geometric execution primarily as a post-generation validation step, limiting its role in guiding the generation process. To address this limitation, this study proposes a unified execution-supervised framework that incorporates geometric feedback throughout data representation, model optimization, inference-time correction, and self-training. The framework first converts raw boundary representation (B-Rep) data into compact, executable CadQuery programs. A cost-aware hierarchical verification mechanism is then established by combining lightweight Mock-CadQuery (Mock-CQ) screening with authoritative Open CASCADE Technology (OCCT) execution. Furthermore, Chamfer-Distance-guided reinforcement learning, execution-guided iterative repair, and execution-verified self-training are integrated to improve geometric fidelity, execution reliability, and targeted modeling coverage. Experiments demonstrate that the proposed framework significantly reduces the invalid output rate, while also decreasing the mean Chamfer Distance among valid geometric outputs. These findings highlight the effectiveness of integrating execution feedback into CAD program generation and provide a practical approach to producing reliable, executable, and editable parametric models for AI-assisted engineering design.

Keywords: text-to-CAD; parametric design; large language models; geometric verification; execution supervision; design automation

Check for updates

Academic Editor: Ahmed Senouci

Received: 22 September 2026
Revised: 16 October 2026
Accepted: 20 October 2026
Published: 23 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Computer-Aided Design (CAD) plays a fundamental role in architecture, engineering design, product development, and additive manufacturing. Although a completed solid is commonly represented using a boundary representation (B-Rep) composed of faces, edges, and vertices, its design intent is often more naturally expressed through the sequence of geometric operations used to construct it. A parametric procedure such as Sketch–Extrude–Fillet provides an explicit description of the modeling process and offers several practical advantages, including editability, inspectability, reusability, and direct execution by a geometric kernel. Such properties are particularly important in computational and parametric design workflows, where generated models are expected not only to reproduce geometry but also to support subsequent modification, optimization, and engineering use.

CadQuery provides a Python-based interface for this purpose, while geometric construction is carried out using Open CASCADE Technology (OCCT). Consequently, a generated CAD program can serve as an editable and executable engineering representation rather than merely a static geometric output.

Such executable representations are particularly suitable for automated parametric design because they preserve construction logic and support subsequent modification and downstream engineering integration. Natural-language-driven CAD generation could further reduce the technical barrier between design intent and geometric modeling. Its practical value, however, depends on whether the generated program can consistently produce valid and reusable geometry rather than merely syntactically plausible code.

Recent advances in code-oriented large language models (LLMs) support natural-language program generation. Qwen2.5-Coder provides a code-specialized backbone [1]; Code Llama [2], StarCoder [3], and DeepSeek-Coder [4] offer related programming capabilities. CAD generation studies follow three main representation choices. DeepCAD models sketch–extrude sequences [5], SkexGen separates construction codebooks [6], and hierarchical neural coding supports controllable completion [7]. BrepGen directly generates structured boundary geometry [8]. Multimodal approaches use additional shape observations: CAD-MLLM combines textual, visual, and point-cloud conditioning [9], Cadrille incorporates online reinforcement learning [10], and Img2CAD conditions on a single image [11]. Point2CAD reconstructs boundary models from point clouds [12].

A third formulation generates inspectable CAD programs. CAD-Llama adapts language models to parametric modeling [13]. Text2CAD produces sequential CAD designs from descriptions at different levels of detail [14]. Text-to-CadQuery generates executable Python CAD programs [15], while CAD-Coder incorporates chain-of-thought and geometric rewards [16]. OpenECAD uses a visual language model to produce editable design commands [17]. These approaches motivate representations that preserve construction logic and can be inspected, modified, executed, and integrated with existing CAD tools.

Despite this progress, the practical reliability of text-to-CAD generation remains a major challenge. Syntactically plausible code does not necessarily produce a valid geometric model. A generated program may reference a nonexistent topological entity, construct a self-intersecting profile, assign an infeasible fillet or shell parameter, or return an empty result after a Boolean operation. Such failures are difficult to identify from token-level probabilities alone because their validity depends on the geometric state created during execution. This distinguishes CAD code generation from general-purpose code generation, where functional correctness can often be evaluated using predefined input–output tests. In CAD modeling, correctness depends on geometric-kernel behavior and therefore requires explicit geometric verification.

The consequences of these failures extend beyond code validity. In practical parametric design, a failed geometric operation can interrupt subsequent modeling steps, prevent geometry export, or invalidate later modifications that depend on the resulting topology. Even when a program executes, small errors in dimensions, Boolean operations, feature selection, or construction order may propagate through the modeling sequence and lead to a result that deviates substantially from the intended design. Therefore, improving the reliability of text-to-CAD generation requires feedback mechanisms that are aware of geometric execution rather than relying exclusively on linguistic or syntactic likelihood. From this perspective, geometric execution is not merely an evaluation tool but also a potentially valuable source of supervision for model optimization.

Moreover, executability alone is not sufficient to ensure a satisfactory CAD result. A program can execute successfully while still producing an object with incorrect dimensions, topology, proportions, or semantic features. Accordingly, reliable text-to-CAD generation

should distinguish among at least three complementary requirements: syntactic validity, kernel-level executability, and geometric or semantic fidelity. Existing approaches often use geometric execution primarily as a post-generation validity check. This strategy can identify unsuccessful outputs, but it does not fully exploit execution information as a learning signal for improving the generation process itself. A further difficulty is that repeated invocation of a full-scale geometric kernel during training can impose substantial computational cost, making direct execution-based supervision difficult to apply at high frequency.

To address these limitations, this study introduces an execution-supervised text-to-CAD framework in which executability is incorporated throughout the modeling pipeline rather than being used only for final validation. The core idea is to combine frequent, low-cost execution-related supervision with authoritative geometric-kernel verification. Direct invocation of OCCT for every intermediate candidate would provide accurate geometric information but would also introduce considerable computational overhead during training and iterative generation. Therefore, a lightweight execution proxy, termed Mock-CadQuery (Mock-CQ), is introduced to detect common parameter- and operation-level geometric violations efficiently. OCCT is retained as the authoritative verifier whenever real geometry must be constructed and accepted. This cost-aware separation allows execution information to be used more frequently during optimization without replacing the geometric kernel on which final geometric validity depends.

In building information modeling, editable geometry must also participate in a structured design process. Succar's BIM framework situates model creation within information delivery and stakeholder coordination [18]. Sacks et al. identify interactions between BIM functionality and lean construction that depend on reusable, consistent model information [19]. For existing buildings, uncertainty and incomplete documentation introduce additional modeling requirements [20]. These observations motivate generation of inspectable parametric programs whose geometric validity can be checked before they are passed to an engineering workflow.

The overall framework is illustrated in Figure 1. First, raw B-Rep representations are converted into compact and executable CadQuery programs. The model is then optimized using supervised fine-tuning, Chamfer-Distance-based reinforcement learning, and execution-aware verification, while EV-RFT expands the reachable training distribution. During inference, verifier feedback is used to repair residual failures, and OCCT provides the final geometric validity check. The framework therefore separates frequent low-cost execution supervision from authoritative kernel-level verification.

Accordingly, this study aims to establish a unified mechanism through which execution information contributes to representation learning, geometric optimization, inference-time correction, and distribution expansion. Executable CadQuery programs are investigated as compact generation targets, continuous geometric rewards and execution-aware verification are introduced during optimization, localized execution feedback is used to repair residual failures, and verifier-filtered self-training is employed to improve generalization beyond the original training distribution. Together, these components form a coordinated execution-supervised workflow for reliable and editable parametric CAD generation; the contribution is their verified interaction and allocation of verification cost, rather than each constituent learning algorithm is independently novel.

The main contributions of this study are summarized as follows:

- Execution supervision throughout the text-to-CAD pipeline. Executability is formulated as a supervision signal spanning model optimization, inference-time repair, and verifier-filtered self-training.
- Cost-aware geometric verification and coordinated optimization. A hierarchical verification strategy combines lightweight Mock-CQ with authoritative OCCT verification.

By integrating geometric reinforcement learning, execution-guided repair, and verifier-filtered self-training, the framework jointly improves execution reliability, geometric quality, and generation robustness.

- Extension to building-oriented parametric design. The proposed framework is applied to the automatic generation of representative building components, enabling the conversion of natural-language descriptions into executable CAD programs and geometric models. Kernel verification and multi-view visualization demonstrate its potential for rapid component modeling and AI-assisted engineering design.

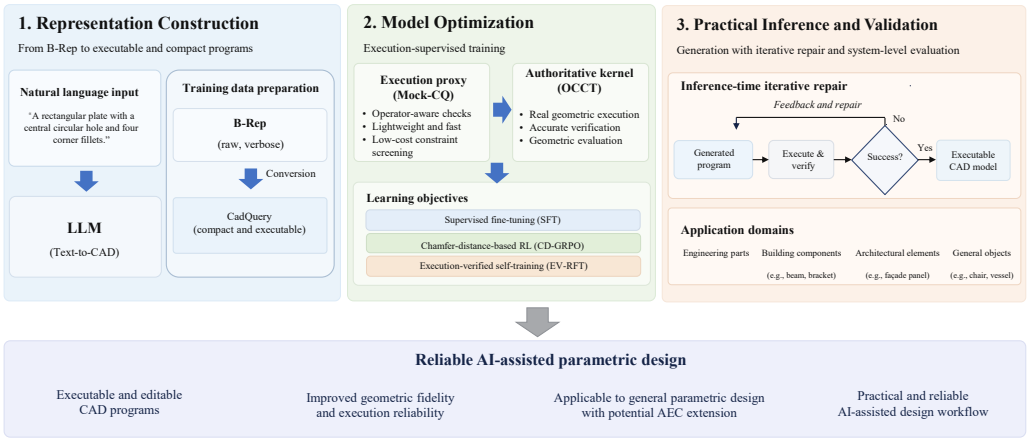


Figure 1. Overall framework of the proposed execution-supervised text-to-CAD method. The workflow comprises three stages: representation construction, execution-supervised model optimization, and practical inference and validation. B-Rep data are converted into compact executable CadQuery programs; Mock-CQ provides low-cost execution-related feedback, whereas OCCT is retained for authoritative geometric verification. SFT, CD-GRPO, EV-RFT, and execution-guided iterative repair are integrated to improve execution reliability and geometric fidelity. Mock-CQ is a bounded proxy rather than a replacement for kernel execution, and the final inference model applies this verification strategy to category-expanded parametric design tasks.

2. Related Work

2.1. CAD Generation and Representation

Learning-based CAD generation has evolved from domain-specific sequence modeling toward multimodal reconstruction and program synthesis. Early methods commonly represent CAD construction histories using discrete geometric operations. DeepCAD [5], for example, models sketch–extrude sequences as tokenized construction programs. Subsequent approaches, including SkexGen [6], Hnc-CAD [7], BRepGen [8], ComplexGen [21], and CAD-as-Language [22], extend this idea through hierarchical representations, probabilistic modeling, and language-based formulations. These compact representations reduce sequence length compared with raw B-Rep serialization and make learning of construction logic more tractable. However, many such representations remain domain-specific and are not directly executable by general geometric kernels such as OCCT [23].

A second line of research reconstructs CAD geometry from multimodal observations. CAD-MLLM [9] is a representative example that unifies multimodality-conditioned CAD generation within a multimodal large language model, allowing textual, visual, and geometric information to contribute to the generation process. Related approaches, including Cadrille [10], Img2CAD [11], Point2CAD [12], Free2CAD [24], and CAD-Recode [25], further investigate CAD reconstruction from images, point clouds, or other geometric observations. These methods benefit from direct observations of the target shape and therefore address a different conditioning setting from text-only CAD generation.

More recently, CAD modeling has increasingly been formulated as program generation. Text2CAD [14] and CAD-Llama [13] generate parametric representations from descriptions. Text-to-CadQuery [15] and CAD-Coder [16] exploit code-capable language models to generate executable CAD programs. OpenECAD [17] instead conditions a visual language model on design images to produce editable commands. This formulation is particularly attractive for parametric design because the output remains editable, inspectable, and reusable. It also allows CAD generation to benefit from the programming priors from Qwen2.5-Coder [1]. Code Llama focuses on code specialization [2], StarCoder provides an open code model [3], and DeepSeek-Coder extends this direction with large-scale programming data [4]. Nevertheless, executable correctness remains a major challenge. In many existing systems, geometric execution is primarily used to validate generated programs after inference rather than to supervise the generation process itself.

Explicit topology is also relevant to the representation choice. UV-Net encodes surface parameterizations together with face adjacency [26]. BRepNet performs message passing through oriented topological entities [27]. Vitruvion generates sketch primitives and their parametric constraints [28]. JoinABLe predicts joints between existing CAD parts [29]. These studies show that surface geometry, feature constraints, and assembly relations carry distinct information. An executable construction program offers a complementary representation that exposes operations and parameters directly to verification.

2.2. Execution-Aware Learning and Self-Correction

Feedback from program execution has become an important mechanism for improving code generation. General code-generation methods such as CodeRL [30], AlphaCode [31], and RLEF [32] use unit tests or functional-correctness signals during optimization, while formal methods such as AlphaVerus [33] and PSV-Verus [34] use verification results to filter or improve generated programs. Reinforcement-learning approaches further extend this paradigm by optimizing model behavior with explicit feedback. GRPO [35], together with later variants such as DAPO [36] and Dr.GRPO [37], provides a practical framework for policy optimization without requiring a separate value network.

Self-training provides a complementary route for exploiting generated supervision. Methods such as Self-Instruct [38], STaR [39], and ReST [40] iteratively improve a model using its own generated outputs, while rejection-sampling fine-tuning retains samples satisfying an external criterion [41]. Its relation to policy optimization has been studied explicitly [42], and failure-informed agent tuning provides another route to generated supervision [43]. The same principle is particularly relevant to CAD generation, where a generated program can be evaluated not only syntactically but also through geometric execution. However, CAD verification differs from ordinary program testing because there is rarely a simple input–output oracle. A program may be syntactically correct yet fail when constructing geometry, or it may execute successfully while producing a shape that does not match the intended design.

Execution feedback can also be introduced at inference time. Self-Refine [44], Reflexion [45], Self-Debug [46], Self-Edit [47], and RISE [48] revise initial model outputs according to error messages or constraint violations. For CAD generation, the quality and cost of such feedback are especially important. Lightweight checks are inexpensive but cannot fully capture geometric validity, whereas real-kernel execution provides more authoritative information at a substantially higher computational cost. The proposed framework addresses this trade-off through a two-level verification strategy: Mock-CQ provides inexpensive operation-level feedback, while OCCT performs final geometric verification. Execution information is therefore used not only for post-generation checking, but also for reinforcement learning, iterative correction, and verifier-filtered self-training.

This separation also relates to neuro-symbolic verification in engineering. For example, the SYNAPSE architecture couples probabilistic AI components with deterministic structural checks to prevent unverified predictions from directly controlling engineering decisions [49]. Our setting differs in that the symbolic object is an executable parametric CAD program and the final oracle is a boundary-representation kernel, but the same principle applies: learned generation should be bounded by an independently evaluated acceptance condition.

The use of executable intermediate representations has broader precedent. DreamCoder learns reusable program abstractions together with a synthesis policy [50]. PAL delegates generated symbolic computations to a runtime [51]. CodeT selects candidate programs using generated tests and execution agreement [52]. CAD requires a richer oracle because geometric construction can depend on evolving topology and numerical tolerances even when the program is syntactically valid.

2.3. Positioning and Evaluation of Text-to-CAD Methods

Evaluation of generated CAD models generally requires more than a single validity measure. Geometric similarity is commonly assessed using Chamfer Distance (CD) [53], F-score [54], intersection over union (IoU), and iterative closest point (ICP) alignment [55]. DeepCAD [5], for example, evaluates bidirectional squared CD after normalizing the geometry to $[-0.5, 0.5]^3$, whereas CAD-MLLM [9] also reports F-score under a specified distance threshold. Datasets such as ABC [56] and Fusion 360 Gallery [57] are widely used for geometric reconstruction and CAD evaluation.

These metrics, however, describe only part of the problem. For text-to-CAD generation, three levels of correctness should be distinguished. Syntactic validity concerns whether the generated text forms a valid program. Kernel executability determines whether the program can construct a nonempty geometric model under a real CAD kernel. Geometric and semantic fidelity evaluates whether the resulting model actually matches the requested design. A method can perform well at one level and still fail at another.

This distinction also clarifies the position of the proposed method. Domain-specific token models emphasize compact representation, multimodal reconstruction methods benefit from direct visual or geometric observations, and code-LLM approaches exploit pretrained programming knowledge. The present study focuses instead on how execution information can be incorporated throughout the generation pipeline and how verification cost should be distributed between optimization and final acceptance. Mock-CQ is therefore used where feedback must be frequent and inexpensive, whereas OCCT is retained where geometric validity must be determined reliably. At the same time, execution rate is evaluated separately from geometric similarity because an executable program may still produce an inaccurate shape. Accordingly, this study treats execution reliability and geometric fidelity as complementary evaluation objectives for executable parametric CAD generation.

For architecture, engineering, and construction (AEC) workflows, this distinction is consequential because an executable component remains only an intermediate design artifact. BIM objects and structural or geotechnical components must additionally satisfy dimensional, material, interface, and code-based requirements. Low-carbon pile design demonstrates the need to evaluate parametric alternatives against embodied-carbon objectives [58]. Industry studies identify additional barriers to foundation decarbonization [59]. These studies motivate potential AEC uses of editable generated geometry.

2.4. Natural Language Modeling and Verification in AEC

Natural language interfaces in AEC encompass retrieval, model generation, and regulatory checking, each with different acceptance requirements. Lin et al. translate natural-

language queries into information retrieval over cloud BIM [60]. BIMS-GPT combines language interpretation with model information search and visualization [61]. Text2BIM uses a language model-based multiagent workflow to generate editable early-stage building models [62]. The present method focuses on executable component-level parametric programs, which can serve as geometric inputs to these larger information workflows after additional semantic and interoperability checks.

Established BIM verification provides a useful methodological reference. Eastman et al. organize automated building-design checking around rule interpretation, model preparation, evaluation, and reporting [63]. Solihin and Eastman classify rules by computational complexity and required model information [64]. Semantic NLP can extract requirements from regulatory documents for later compliance evaluation [65]. Automated safety checking additionally depends on construction sequences and hazard-specific predicates [66]. These systems demonstrate why geometric executability is a prerequisite rather than a complete engineering acceptance criterion. A review of AI in construction similarly identifies data, integration, and validation challenges across applications [67]. The contribution of the present framework is to make execution information available throughout parametric generation while preserving an authoritative kernel decision at final acceptance.

3. Methodology

The proposed framework converts natural-language descriptions into executable CadQuery programs through a sequence of execution-supervised stages. The method consists of four closely related components. First, raw CAD data are converted into compact and executable program representations. Second, a cost-aware verification strategy combines the lightweight Mock-CQ proxy with the OCCT geometric kernel. Third, geometric reinforcement learning and execution-guided correction are used to improve both geometric quality and execution reliability. Finally, execution-verified self-training and execute-and-select inference are introduced to improve distributional coverage and practical generation robustness.

3.1. Executable CadQuery Representation

Raw OmniCAD samples are represented as Fusion 360 B-Rep JSON sequences. Although B-Rep preserves detailed geometric and topological information, its serialized form is often substantially longer than the effective context window available during code-LLM fine-tuning. Long sequences are therefore prone to truncation and make it difficult for the model to preserve dependencies among successive geometric operations.

To obtain a more compact generation target, a B-Rep-to-CadQuery converter is constructed to transform the original representation into executable parametric programs. The conversion consists of four main operations: (i) recognizing Sketch + ExtrudeFeature sequences; (ii) recovering common geometric primitives, including rectangles, circles, and regular polygons; (iii) merging or removing collinear, duplicated, and degenerate curves; and (iv) batching adjacent coplanar profiles and combining symmetric bidirectional extrusions. The resulting representation preserves the construction logic of the original CAD model while substantially reducing the sequence length. Its influence on downstream generation performance is evaluated in Section 4.4.

Floating-point parameters are additionally shortened to a fixed number of significant digits to further reduce unnecessary sequence complexity. Because numerical truncation may alter a geometric constraint near a boundary, each modified value is checked with Mock-CQ. If truncation changes the corresponding constraint judgment, the original value is retained. This procedure provides a compact representation without introducing avoid-

able geometric inconsistencies. The operation-wise verification procedure implemented in Mock-CQ is summarized in Algorithm 1.

Algorithm 1 Mock-CQ verification

Require: CadQuery program c, mocked geometric kernel M, constraint family $\mathcal{F}$
Ensure: Verification result (v, e), where $v \in \{\top, \bot\}$ and e is an optional diagnostic

if c cannot be parsed as a valid program **then**
  return $\bot$
end if
Initialize geometric state $S \leftarrow \emptyset$
for each geometric operation $g_t \in c$ **do**
  Select $\mathcal{F}_{g_t} \subseteq \mathcal{F}$
  Validate g_t under M
  if any $f_i \in \mathcal{F}_{g_t}$ is violated **then**
    return $(\bot, f_i)$
  end if
  Update $S \leftarrow M(S, g_t)$
end for
if S contains a valid terminal result object **then**
  return $\top$
else
  return $\bot$
end if

3.2. Cost-Aware Geometric Verification

CadQuery relies on Open CASCADE Technology (OCCT) to construct real geometry. Although kernel execution provides authoritative geometric validation, its computational cost is too high for the frequent feedback required during model optimization. To address this mismatch, we introduce Mock-CQ, a lightweight geometric-constraint verifier that approximates OCCT executability without explicitly constructing the complete B-Rep.

Mock-CQ maintains a compact geometric state that records the operation history and local dimensional information. During program interpretation, sketch primitives, extrusions, revolutions, Boolean operations, fillets, chamfers, arrays, and related operators are checked against a predefined constraint family $\mathcal{F}$ (Table 1). A program is rejected immediately when a constraint is violated or when no valid terminal result object is produced.

The implemented constraints cover exactly nine families, including positive geometric dimensions, polygon edge count, rotation-angle range, torus self-intersection, shell thickness, fillet and chamfer size, hole-diameter bounds, array count, and result-object binding (Table 1). Mock-CQ is designed as a lightweight execution proxy for screening common parameter- and operation-level failures rather than as a complete geometric verifier. It therefore provides inexpensive screening for common failures, while OCCT remains the authoritative acceptance mechanism. Programs outside these operator patterns can be falsely rejected; when Mock-CQ is used for reward gating, such a false negative down-weights a geometrically valid candidate and can bias learning toward the covered API subset. The nonzero gate floor and final OCCT check limit, but do not eliminate, this risk.

Let C_M denote the cost of one Mock-CQ evaluation, C_K the cost of one OCCT execution, and p_M the fraction of candidates forwarded to the real kernel. For a proxy-first cascade, the expected verification cost is

$$\mathbb{E}[C_{\text{cascade}}] = C_M + p_M C_K. \tag{1}$$

Direct execution of every candidate would instead incur a cost of C_K. The proxy is therefore computationally advantageous when

$$C_M \ll (1 - p_M)C_K. \tag{2}$$

This reduction in computational cost does not imply equivalence between the two verifiers. Mock-CQ evaluates local constraints, whereas OCCT resolves the actual curves, surfaces, topology, Boolean operations, and numerical tolerances of the generated geometry. Consequently, Mock-CQ is used for frequent intermediate feedback, while OCCT is retained for final validation.

Table 1. The nine geometric-constraint families implemented by Mock-CQ. They screen common local violations and are not a complete specification of CadQuery/OCCT executability.

No.	Constraint Family
1	Strictly positive dimension parameters for sketch and 3D primitives
2	Regular polygon edge count ≥ 3
3	Rotation angle $\theta \in (0, 360]$
4	Torus self-intersection forbidden (tube radius $<$ ring radius)
5	Shell thickness $\leq 0.80\times$ main radius
6	Fillet/chamfer size $\leq$ minimum dimension/1.8
7	Hole radius $<$ main radius
8	Array count ≥ 1
9	Result-object binding and type validity

3.3. Execution-Supervised Optimization and Inference Repair

After supervised fine-tuning, geometric information is introduced through execution-supervised reinforcement learning. Let $G(c) \in \mathbb{R}^{n\times 3}$ denote the point cloud sampled from the geometry produced by program c, and let G^* denote the corresponding reference point cloud. Both point clouds are normalized according to the longest axis of their axis-aligned bounding boxes:

$$\hat{G} = \mathrm{norm}_{[-0.5,0.5]^3}(G), \qquad \hat{G}^* = \mathrm{norm}_{[-0.5,0.5]^3}(G^*), \tag{3}$$

$$d(\hat{G}, \hat{G}^*) = \frac{1}{2|\hat{G}|}\sum_{p\in\hat{G}} \min_{q\in\hat{G}^*} \|p-q\|^2 + \frac{1}{2|\hat{G}^*|}\sum_{q\in\hat{G}^*} \min_{p\in\hat{G}} \|p-q\|^2, \tag{4}$$

$$R_{\mathrm{CD}}(c) = \exp\left[-\beta d\left(\hat{G}(c), \hat{G}^*\right)\right]. \tag{5}$$

During CD-GRPO, $n = 2048$ surface points are sampled per valid geometry and $\beta = 8.0$ in the exponential reward. The independent evaluation protocol uses 8192 points per geometry. If program c cannot generate a valid point cloud, $R_{\mathrm{CD}}(c) = 0$. Compared with a binary execution reward, this continuous formulation preserves information about the geometric distance between an executable result and its reference shape.

Mock-CQ is incorporated into the reward through multiplicative execution gating. For a program that fails the proxy verification, the geometric reward is down-weighted according to

$$R(c) = R_{\mathrm{CD}}(c)[w_{\mathrm{mock}} + (1 - w_{\mathrm{mock}})\mathbb{1}[\mathrm{MockCQ}(c)]], \tag{6}$$

where $w_{\mathrm{mock}} = 0.2$. A program that passes Mock-CQ but fails OCCT execution still receives zero geometric reward because no valid point cloud can be obtained. The proxy therefore supplies an additional execution-aware signal without replacing real-kernel geometric evaluation.

To improve optimization stability, two modifications from recent GRPO variants are adopted. Following DAPO [36], the upper PPO clipping bound for positive advantages

is relaxed using Clip-Higher. Following Dr.GRPO [37], standard-deviation normalization of the advantage is removed to avoid suppressing useful reward differences when the distribution becomes narrow. A KL regularization coefficient $\beta_{\mathrm{KL}} = 0.02$ is retained to limit policy drift, and the run uses group size $G = 4$. These are training choices: changing them requires re-optimization and cannot be represented by post hoc rescoring of frozen candidates.

For aggressive learning-rate settings, ZClip [68] is additionally used to suppress occasional gradient spikes. Let μ_t and σ_t^2 denote the exponential-moving-average mean and variance of the gradient norm, respectively, and let $\|g_t\|$ denote the gradient norm at step t. The standardized deviation is

$$z_t = \frac{\|g_t\| - \mu_t}{\sigma_t}. \tag{7}$$

When $z_t > z_{\mathrm{thresh}}$, the adaptive clipping threshold is defined as

$$\tau_t = \mu_t + \frac{z_{\mathrm{thresh}}^2 \sigma_t}{z_t}. \tag{8}$$

Execution feedback is also used during inference through an iterative repair procedure. For each prompt, the model is allowed at most $T_{\max} = 3$ generation attempts. The first attempt is greedy; later attempts use temperature 0.7 and top-$p = 0.95$. A candidate is first evaluated by Mock-CQ. If a local constraint is violated, the corresponding error is returned to the model and a revised candidate is generated. Candidates that pass the proxy are subsequently executed with OCCT. Kernel-level exceptions, including empty geometry or invalid shell construction, are likewise returned as correction signals. The iteration terminates when OCCT produces an accepted geometry or when the maximum number of attempts is reached.

This two-stage repair mechanism separates inexpensive local feedback from authoritative geometric verification. Mock-CQ handles common parameter- and operation-level failures, whereas OCCT is invoked only when the candidate requires real geometric validation. The procedure therefore improves execution reliability without relying on unrestricted language-model self-critique.

3.4. Execution-Verified Self-Training and Practical Inference

Although OmniCAD contains a large number of CAD samples, its distribution is dominated by common engineering components. Geometric categories outside this distribution remain comparatively sparse. To improve the coverage of such cases, we introduce execution-verified rejection-sampling fine-tuning (EV-RFT), which expands the training data using generated samples that pass both execution and semantic verification.

For a set of out-of-distribution captions, the current model first generates candidate CadQuery programs. Each candidate is then subjected to two filters:

1. Execution filter. The generated program is first checked by Mock-CQ and subsequently executed with OCCT. Only programs that construct valid geometry are retained.
2. Semantic filter. The executed geometry is rendered and compared with the input caption [69]. Samples with rendering–caption cosine similarity below the specified threshold $\tau = 0.25$ are discarded.

Only samples passing both criteria are added to the subsequent training set. The execution filter removes geometrically invalid programs, whereas the semantic filter reduces the probability of retaining executable but semantically incorrect samples. The resulting training distribution is therefore expanded without requiring manual CAD annotation.

CLIP is a selection proxy rather than a human acceptability measure, however, and self-generated samples may reinforce the current model's construction and rendering biases. The complete EV-RFT procedure is summarized in Algorithm 2.

Algorithm 2 Execution-Verified Rejection-Sampling Fine-Tuning (EV-RFT)

Require: Base model θ_0, initial training set $\mathcal{D}_0$, number of rounds K
1: **for** $k = 1, \ldots, K$ **do**
2: Train $\theta_k \leftarrow \mathrm{SFT}(\theta_{k-1}, \mathcal{D}_{k-1})$
3: Construct an OOD caption set $\mathcal{C}_k$
4: Generate $\mathcal{R}_k = \{(x, \theta_k(x)) : x \in \mathcal{C}_k\}$
5: Retain samples passing Mock-CQ and OCCT
6: Retain samples with CLIP similarity $\geq \tau$
7: $\mathcal{D}_k \leftarrow \mathcal{D}_{k-1} \cup \mathcal{R}_k$
8: **end for**
9: **return** θ_K

The curriculum progresses from relatively simple OOD geometries toward more complex combinations of primitives and Boolean operations. This gradual increase in geometric complexity avoids introducing difficult samples before the current model can generate them reliably. The same verification loop can be applied to additional geometric categories without changing the overall training procedure.

For practical inference, candidate generation is combined with isolated OCCT execution and execute-and-select sampling. The model first produces a greedy candidate. If this candidate fails real-kernel execution, additional programs are sampled according to a predefined temperature schedule. For candidates $c_1, \ldots, c_N$, the first executable result is selected as

$$j^* = \min\{j \in \{1, \ldots, N\} : K(c_j) = \top\}, \tag{9}$$

where K denotes isolated OCCT execution together with nonempty STEP/STL output verification.

This strategy prioritizes executable coverage when no reference geometry is available during practical inference. It does not imply that the first executable candidate has the highest geometric fidelity; controlled evaluation therefore continues to rely on CD, F-score, and qualitative inspection. Under the idealized assumption that individual candidate execution events are independent and each succeeds with probability p, the probability of obtaining at least one executable candidate is

$$1 - (1 - p)^N. \tag{10}$$

Although the independence assumption does not strictly hold for generated candidates, the expression illustrates the reliability–latency trade-off introduced by best-of-N inference. In practical use, this strategy combines language-model generation with real-kernel verification while preserving the editability and executability of the resulting parametric CAD program.

4. Experiments and Results

This section evaluates the proposed framework in terms of execution reliability, geometric fidelity, representation efficiency, verification cost, training stability, inference-time correction, and targeted distribution expansion. Unless otherwise stated, the controlled comparisons use the same fixed 500-program subset of the processed OmniCAD test partition and the same evaluation protocol.

4.1. Experimental Setup

Dataset and preprocessing. The experiments are conducted on the OmniCAD dataset [9]. Raw samples are converted from B-Rep JSON sequences into executable CadQuery programs using the procedure described in Section 3.1. The processed corpus contains approximately 210K caption–CadQuery pairs. Training and test partitions are separated before downstream model training and evaluation. The official test identifiers are processed into 20,407 reference programs. The controlled study uses a fixed 500-program subset, hereafter called the **OmniCAD-500 subset**.

Base model and training configuration. Qwen2.5-Coder-1.5B-Instruct [1] is used as the base model. Parameter-efficient fine-tuning is performed using LoRA [70], with adapters applied to the q/k/v/o attention projections and the gate/up/down MLP projections. The primary SFT training was conducted on an NVIDIA RTX 5080 GPU with 16 GB of memory using bfloat16 (bf16) precision.

A two-epoch SFT run with an effective batch size of 32, a maximum sequence length of $L = 1024$, an initial learning rate of 5×10^{-5}, cosine decay, and a 3% warmup ratio. The component-study adapter uses LoRA $r = 128$, $\alpha = 256$, and dropout $= 0.05$. AdamW was used as the optimizer with $\epsilon = 10^{-8}$ and a global gradient-norm clipping threshold of 1.0. The stated hyperparameters define the training protocol. The component study uses greedy decoding; the recorded results are single training runs and are not presented as multiple-seed training estimates.

For CD-GRPO, the learning rate is $\eta = 10^{-5}$, the KL regularization coefficient is $\beta_{\mathrm{KL}} = 0.02$, the group size is $G = 4$, and 300 optimization steps are performed. The execution-related coefficient is set to $w_{\mathrm{mock}} = 0.2$, with $w_{\mathrm{CD}} = 1.0$. The optimization adopts DAPO Clip-Higher with $\epsilon_{\mathrm{high}} = 0.28$ [36]. Standard-deviation normalization is removed following Dr.GRPO [37]. For execution-guided iterative repair, the maximum number of generation turns is set to $T_{\max} = 3$.

Training data composition. The primary training set contains 181,593 OmniCAD samples. EV-RFT adds 8600 verified samples over $K = 3$ rounds, resulting in a total of 190,193 training examples. The EV-RFT samples account for approximately 4.5% of the final training set. Their composition is summarized in Table 2. The SFT row in the main controlled comparison uses this complete 190,193-example set and therefore already includes EV-RFT data; the no-EV model appears only in the incremental study in Section 4.7.

Table 2. Composition of the 190,193-example training set used by the main SFT baseline and subsequent controlled stages.

Source	Complexity	Samples	Proportion
OmniCAD	–	181,593	95.5%
EV-RFT R1	Single body of revolution	1000	0.5%
EV-RFT R2	Rotational compositions	1000	0.5%
EV-RFT R3	Multi-primitive + decor.	2600	1.4%
EV-RFT assemblies	Nested Boolean operations	1200	0.6%
EV-RFT early	Miscellaneous	2800	1.5%
Total		190,193	100%

Evaluation metrics. Normalization follows the DeepCAD protocol [5]; thresholded F-score evaluation follows CAD-MLLM [9]. Generated and reference geometries are normalized to $[-0.5, 0.5]^3$ before geometric comparison. This paper reports the invalid rate (IR), bidirectional squared Chamfer Distance (CD) at the $\times 10^3$ scale, F-score at the threshold τ, and the ICP- and 13-axis-aligned F-score. Absolute changes in percentage are reported in percentage points (pp), whereas relative changes are reported in percent.

CD and F-score are computed only when a finite geometry pair is available; IR always uses all 500 prompts. Valid-only geometry averages and all-prompt execution metrics are therefore reported separately. For an all-prompt F-score sensitivity analysis, any missing or invalid geometry receives $F = 0$.

4.2. Overall Performance

Table 3 summarizes the controlled component comparisons on the fixed OmniCAD-500 subset. All rows use text conditioning and the same geometric evaluation protocol. The table isolates the sequence of supervised learning, geometric policy optimization, and inference repair.

Table 3. Controlled text-only ablation on the same OmniCAD-500 subset. Geometric metrics are reported valid-only aggregates; IR uses all 500 prompts.

Method	Invalid Count/500	IR ↓	Mean CD ↓	F@0.05 Aligned ↑	F@0.20 Aligned ↑
SFT	38	7.60%	199.9	66.65%	93.31%
SFT + Iter	31	6.20%	199.7	66.44%	93.27%
SFT + CD-GRPO	23	4.60%	186.0	65.25%	93.95%
SFT + CD-GRPO + Iter	3	0.60%	183.8	64.73%	93.76%

Note: CD values use the $\times 10^3$ scale and 8192 sampled surface points.

SFT alone produces an invalid rate of 7.60% (38/500; Wilson 95% CI, 5.59–10.26%). Adding Iter reduces the value to 6.20%, while CD-GRPO achieves a larger reduction to 4.60%. When CD-GRPO and Iter are combined, the invalid rate decreases to 0.60%. Thus, 497 of the 500 evaluated prompts produce executable outputs (Wilson 95% CI for IR, 0.20–1.75%). The intermediate Wilson intervals [71] are 4.40–8.67% for SFT + Iter and 3.08–6.81% for SFT + CD-GRPO.

Geometric fidelity follows a related, but not identical, trend. Mean CD decreases from 199.9 for the SFT baseline to 183.8 for the complete pipeline, corresponding to an improvement of approximately 8%. The aligned F@0.20 remains above 93% throughout the controlled experiments and reaches 93.76% for the complete model.

At the stricter threshold of 0.05, however, the reported valid-only aligned F-score decreases from 66.65% to 64.73%. The substantial improvement in execution validity is therefore not accompanied by a uniform improvement across all geometric metrics. This behavior indicates that the proposed optimization is more effective at recovering globally plausible and executable geometry than at uniformly improving fine-scale surface correspondence. It may also reflect a change in the set of outputs entering the valid-only average: repair adds previously invalid, potentially difficult cases.

CD-GRPO and Iter address different aspects of the problem. CD-GRPO directly optimizes an aggregate discrepancy between generated and reference geometry, whereas Iter is activated by execution failures and responds to localized verification feedback. A repaired program may therefore become executable while remaining slightly different from the reference at local geometric features.

The controlled comparison within OmniCAD provides the most direct evidence for the contribution of the proposed components. Table 3 reports same-protocol component comparisons. Related methods are discussed as methodological context in Section 4.8.

4.3. SFT Output Distribution and Geometric Fidelity

A more detailed analysis of the SFT baseline is conducted using the 500-prompt OmniCAD evaluation. Among these samples, 462 programs are executable, yielding

$$\mathrm{IR} = \frac{38}{500} = 7.60\%. \tag{11}$$

Of the generated outputs, 444 have finite DeepCAD-normalized CD and F-score records. The corresponding mean and median CD values are 199.889 and 157.574, respectively, at the $\times 10^3$ scale. A 20,000-resample paired-row bootstrap [72] with seed 20261008 gives a 95% interval of 183.442–219.017 for the mean CD. The mean aligned F-scores are 66.650% (95% bootstrap interval, 64.136–69.146%) at threshold 0.05 and 93.312% (92.296–94.306%) at threshold 0.20. The progression of invalid rates across the principal stages of the proposed framework is presented in Figure 2, with the in-domain and OOD evaluations shown separately.

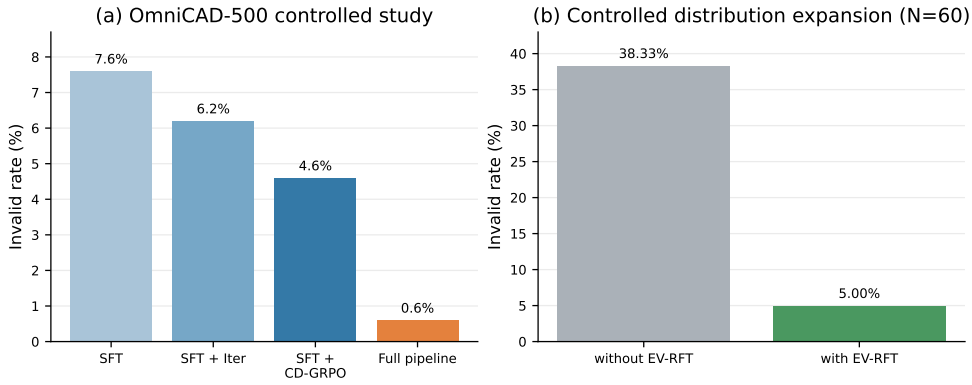


Figure 2. Progression of invalid rate across the principal stages of the proposed framework. The in-domain OmniCAD evaluation and the OOD test are shown separately because they represent different evaluation settings. The plotted in-domain values are single-run aggregate counts on the fixed OmniCAD-500 subset.

To expose valid-only survivorship bias, we also assign $F = 0$ to every output without a finite geometric record and average over all 500 prompts. The resulting all-prompt F@0.05 and F@0.20 are 59.18% and 82.86%, respectively. The distribution of finite CD values and the corresponding execution and aligned F-score statistics are presented in Figure 3.

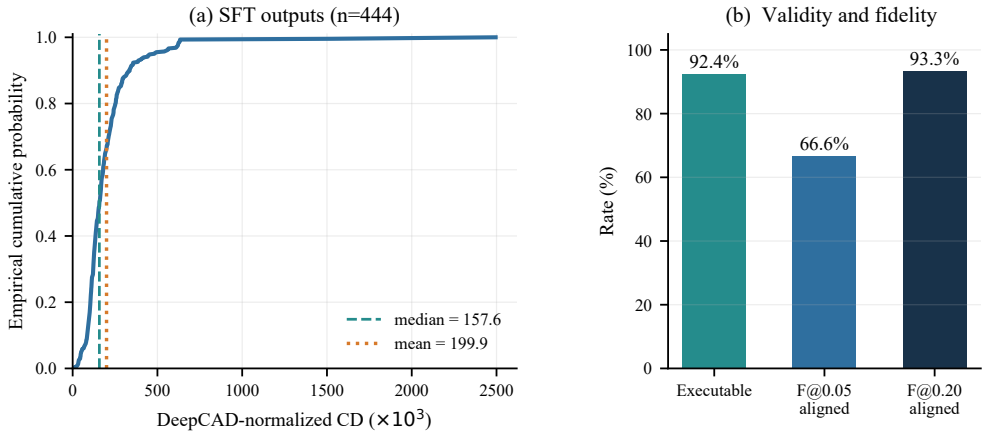


Figure 3. Distribution of the recorded SFT evaluation results. (**a**) Empirical cumulative distribution of finite DeepCAD-normalized CD values. (**b**) Execution rate and aligned F-scores obtained on the 500-prompt evaluation. Geometry distributions use the 444 finite records; execution uses all 500 prompts.

The distribution provides information that is not captured by the mean alone. Half of the 444 finite-CD outputs have CD values below approximately 157.6, whereas a small number of cases exceed 2000 and increase the mean to 199.9. A similar distinction can be observed between F@0.20 and F@0.05: the much higher value at the relaxed threshold indicates that global shape agreement is stronger than strict local correspondence.

This behavior is consistent with the compact program representation adopted in this study. The dominant body, symmetry, or Boolean structure can be recovered correctly even when local edge placement, profile proportions, or fillet radii remain imperfect. Representative execution failures, geometric-detail errors, and their corresponding improvements are illustrated in Figure 4.

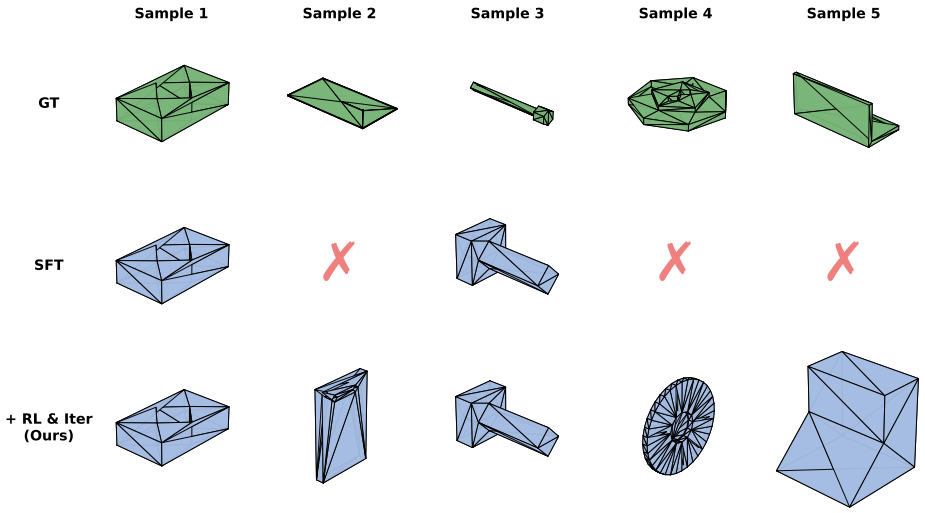


Figure 4. Diagnostic comparison of five deliberately selected OmniCAD cases illustrating both execution failures and geometric-detail errors. SFT generates invalid programs for samples 2, 4, and 5 and misses important geometric details in samples 1 and 3. Adding CD-GRPO and Iter repairs the execution failures and improves the topology or proportions in these examples. The selection criterion was visibility of a documented failure or repair mode.

4.4. Mock-CQ Verification and Representation Ablation

The first set of ablation experiments evaluates the lightweight execution proxy introduced in Section 3.2. The verification performance of Mock-CQ is evaluated against direct OCCT execution using the constraint families defined in Table 1.

Mock-CQ is compared with direct OCCT execution on 200 reference CadQuery programs. Every reference program is OCCT-executable, so this is a positive-only recall audit. Defining an OCCT-executable program as positive and Mock-CQ acceptance as a positive prediction gives $TP = 192$, $FN = 8$, $FP = 0$, and $TN = 0$. Recall and accuracy on this single-class set are both 96.0%; precision is 100% conditional on the absence of negative cases, whereas specificity is undefined. The eight disagreements are false rejections, not "no false negatives". Mixed generated-candidate results are reported separately in Section 5.3. The corresponding verification performance and computational costs are summarized in Table 4.

Table 4. Timing and positive-class recall audit on 200 OCCT-executable reference programs. Timing scopes are reported separately.

Method and Scope	Time per Sample	Accepted	Recall
OCCT kernel, in process	17.0 ms	200/200	100% (reference)
OCCT isolated worker + export	1577 ms	200/200	100% (reference)
Mock-CQ, in process	0.37 ms	192/200	96.0%

The 17.0 ms value measures kernel construction in the same process, whereas 1577 ms includes a fresh interpreter, CadQuery/OCCT import, isolated execution, and STEP export. The values therefore are not contradictory and must not be used interchangeably. The results support Mock-CQ for frequent low-cost feedback, but its false rejection rate affects reward gating and its single-class agreement cannot establish specificity or population accuracy. OCCT remains the final acceptance mechanism.

The verifier is further evaluated on a fixed expanded test population in Section 5.3. All 500 held-out reference programs are retained, and 200 predefined single-fault derivatives introduce negative examples. This larger study supplements the original positive-only check without changing the verifier or selecting cases according to its outcome.

The second ablation examines the B-Rep-to-CadQuery representation described in Section 3.1. Token statistics for 100 randomly selected OmniCAD samples are reported in Table 5. Under the Qwen2.5-Coder tokenizer, raw B-Rep JSON requires an average of 9922 tokens per sample, with a median of 3640 tokens and a 75th percentile of 9221. The converted CadQuery representation requires only 955 tokens on average.

Table 5. Sequence-length comparison between raw B-Rep JSON and CadQuery using the Qwen2.5-Coder tokenizer ($n = 100$).

Representation	Characters	Token Mean	Token Median	Token p75
B-Rep JSON (raw)	24,824	9922	3640	9221
CadQuery	1570	955	281	1028
Reduction	15.8×	10.4×	13.0×	9.0×

The mean sequence length is therefore reduced to approximately 1/10 of its original length. Floating-point normalization was logged for 190,189 processed records. For all samples subjected to this check, the Mock-CQ geometric-constraint judgment remains unchanged before and after numerical normalization.

To determine whether sequence compression also affects downstream learning, two generation targets are evaluated under the same SFT setting: caption-to-B-Rep JSON and caption-to-CadQuery. Most raw JSON sequences exceed 4096 tokens, resulting in extensive truncation and incomplete generations. The corresponding invalid rate reaches 73.4%.

By contrast, more than 95% of the converted CadQuery programs fit within the $L = 1024$ sequence budget. The resulting invalid rate is 7.60%, with an aligned F@0.05 of 66.65% and a mean CD of 199.9. The downstream generation performance of the two representations is compared in Table 6.

Table 6. Effect of the generation representation on downstream performance on the OmniCAD test set.

Generation Target	*L*	IR ↓	F@0.05 Aligned ↑	CD ↓
B-Rep JSON (raw)	4096 (truncated)	73.4%	32.1%	295.7
CadQuery	1024	7.60%	66.65%	199.9

These results show that the representation change is not merely a reduction in storage or sequence length. It also produces a generation target that is better matched to the available context length and to the programming priors of the pretrained code model.

4.5. Training Stability with ZClip

The principal SFT configuration described in Section 4.1 does not require ZClip. A separate experiment with a more aggressive learning rate is used to evaluate training stability.

Under this configuration, pronounced loss spikes occur approximately 100–150 optimization steps after the learning rate reaches its maximum. Reducing the fixed global gradient-norm threshold from 1.0 to 0.2 does not remove the instability. In the most extreme event, the pre-clipping gradient norm reaches approximately 685× its exponential-moving-average mean, corresponding to a z-score of 234.

When ZClip is enabled, 292 gradient-norm anomalies are detected and adaptively clipped. The resulting training process completes 11,900 steps and reaches a final evaluation loss of 0.140.

This experiment is intentionally separated from the main SFT configuration. Its purpose is to evaluate optimization stability under a more aggressive learning-rate setting. The training stability of different gradient clipping strategies is compared in Table 7.

Table 7. Training-stability ablation under the aggressive-learning-rate configuration.

Gradient Clipping Strategy	Training Completed	Final Eval. Loss
Fixed norm cap 1.0	No (peak +155 steps)	–
Fixed norm cap 0.2	No (peak +115 steps)	–
ZClip + AdamW $\epsilon = 10^{-6}$	Yes	0.140

4.6. Interaction Between CD-GRPO and Iter

The contribution of Iter depends strongly on the model to which it is applied. As shown in Table 8, applying Iter directly to SFT reduces IR by 1.40 pp, from 7.60% to 6.20%. When Iter is applied after CD-GRPO, IR decreases from 4.60% to 0.60%, corresponding to a reduction of 4.00 pp.

Table 8. Effect of execution-guided iterative repair on different upstream models.

Baseline	Baseline IR	IR After Iter	ΔIR
SFT	7.60%	6.20%	−1.40 pp
SFT + CD-GRPO	4.60%	0.60%	−4.00 pp

One plausible explanation is that CD-GRPO changes the distribution of residual failures. After SFT, some remaining errors involve deeper geometric or topology-dependent failures, such as unstable intersections in nested Boolean operations. Such errors are difficult to localize using operator-level feedback alone.

After CD-GRPO, a larger fraction of the remaining failures appears amenable to execution-guided correction. These cases include parameter, API, selector, and related errors for which localized feedback is more informative. The present results are consistent with this interpretation.

The turn-level statistics provide further evidence. With $T_{\max} = 3$, the complete pipeline produces 477 successful outputs at the first turn. Fifteen additional samples are repaired at the second turn and five at the third, leaving three failures. The turn-level success and repair statistics for the two model configurations are summarized in Table 9.

Table 9. Turn distribution of execution-guided iterative repair on the OmniCAD test set ($N = 500$).

Model	Turn 1 Success	Turn 2 Repair	Turn 3 Repair	Failure
SFT + CD-GRPO + Iter	477	15	5	3
SFT + Iter	462	3	4	31

For SFT + Iter, seven of the 38 initially invalid samples are repaired during the second and third turns. Following CD-GRPO, Iter repairs 20 of the 23 programs entering the repair stage. The corresponding conditional repair rates are

$$\frac{7}{38} = 18.4\%, \qquad \frac{20}{23} = 87.0\%. \tag{12}$$

These results indicate that the effectiveness of iterative repair depends on both the quality and the error characteristics of the upstream generation model. Most successful corrections also occur during the second turn: 15 of the 20 repaired samples after CD-GRPO require only one additional generation attempt, whereas five require a third turn.

Because Iter is applied only to an upstream model's failed cases and an already successful first-turn result is retained, the recorded turn counts define nested paired outcomes. An exact two-sided McNemar [73] test gives $p = 0.0156$ for SFT versus SFT + Iter (7 improved, 0 worsened) and $p = 1.91 \times 10^{-6}$ for SFT + CD-GRPO versus the complete pipeline (20 improved, 0 worsened).

4.7. Effect of EV-RFT on Controlled Distribution Expansion

The EV-RFT experiment evaluates whether verifier-filtered self-training can extend the generation distribution beyond the dominant engineering-part categories in OmniCAD. The controlled OOD test contains six categories: vessels, chair, chess rook, L-bracket, candlestick, and circular plate. Each category contains 10 manually designed caption variants, resulting in 60 prompts. The predefined prompt set contains descriptions and category labels but no reference geometry. The categories were targeted by the EV-RFT curriculum, so semantic category overlap is expected even though the exact test prompts are held out. We therefore call this a controlled distribution-expansion test, not an open-world or unseen-category benchmark.

A model trained using OmniCAD alone is compared with a model trained using OmniCAD together with the EV-RFT increment. Both models are evaluated on the same in-domain OmniCAD test set and on the same OOD test set. The corresponding execution reliability results are summarized in Table 10.

Table 10. Effect of EV-RFT on execution reliability.

Configuration	Invalid/Total	IR (Wilson 95% CI) ↓
In-domain: OmniCAD-500 subset		
Without EV-RFT	39/500	7.80% (5.76–10.49%)
With EV-RFT	38/500	7.60% (5.59–10.26%)
Controlled distribution-expansion test: 6 categories × 10 prompts		
Without EV-RFT	23/60	38.33% (27.09–50.98%)
With EV-RFT	3/60	5.00% (1.71–13.70%)

The in-domain IR changes by only −0.20 percentage points. A larger execution difference is observed on the controlled distribution-expansion test, where IR decreases from 38.33% to 5.00%. CD and F-score are not evaluated for this prompt set because no reference geometry is available.

In terms of sample counts, the number of invalid OOD outputs decreases from 23 of 60 to 3 of 60. This corresponds to 20 fewer execution failures and an 87.0% relative reduction. The contrast between the in-domain and OOD results is consistent with the intended role of EV-RFT: expanding the range of construction patterns accessible to the model while largely preserving its behavior on the original OmniCAD distribution.

About the scope of this experiment, The OOD set contains only 60 manually constructed and category-balanced prompts. Therefore, it provides a controlled test of distribution expansion. EV-RFT is self-generated and category targeted; consequently this experiment cannot rule out semantic leakage or reinforcement of the source model's preferred construction patterns.

4.8. Comparison with Related Methods and Residual Failures

Cross-method comparisons require caution because existing approaches differ in both conditioning information and benchmark configuration. CAD-MLLM [9] reports F@0.05 values of 90.88% using point-cloud input and 76.70% using image input on OmniCAD. These settings provide direct geometric or visual observations of the target object and therefore address a different conditioning problem from the text-only setting considered in this work. The corresponding model is also substantially larger. Representative outputs from the controlled distribution-expansion test, including both executable and failed cases, are shown in Figure 5.

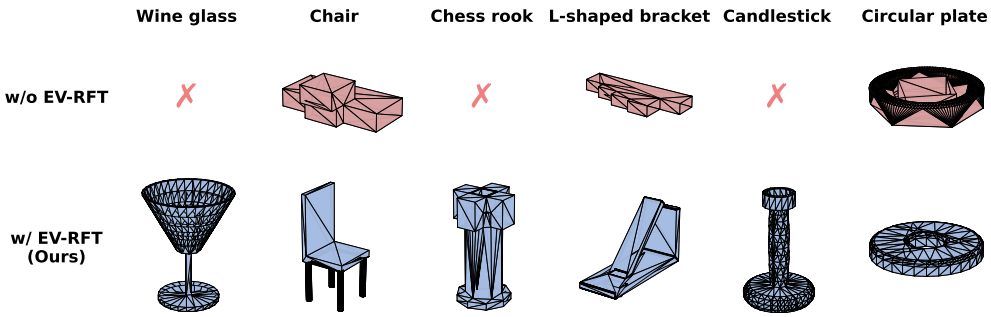


Figure 5. Selected diagnostic examples from the six controlled distribution-expansion categories, including executable and failed outputs. They illustrate observed construction differences.

Text2CAD [14] generates domain-specific construction sequences, whereas CAD-Recode [25] reconstructs editable programs from point clouds. These representation and conditioning differences affect the information available to the model and the acceptance protocol. Invalid rate is less sensitive to geometric normalization than CD, but it still depends on the dataset, program representation, and execution criterion.

Accordingly, the principal quantitative conclusions of this study are drawn from controlled comparisons using the same OmniCAD-500 subset, model family, and evaluation pipeline. A same-protocol evaluation of external text-to-CAD methods has not been performed. Comparative conclusions would require harmonized caption-to-program representations, processed reference geometry, surface sampling, invalid-output handling, and OCCT execution conditions. Published cross-dataset values provide context, the required controlled external comparison remains future work.

Residual invalid programs provide additional information about the remaining limitations. Among the 31 programs that remain invalid for SFT + Iter, three failure patterns occur repeatedly: (i) nested Boolean operations that produce unstable or empty intersections in OCCT; (ii) floating-point tolerance boundaries that introduce defects between nominally coplanar faces; and (iii) ambiguous edge references in complex fillet or chamfer operations.

These cases illustrate the distinction between local execution constraints and topology-dependent geometric failures. Some errors arise from parameters or selectors that can be corrected from localized feedback. Others become apparent only after the B-Rep topology has evolved inside the geometric kernel and therefore cannot be fully resolved by Mock-CQ.

These observations motivate future extensions toward topology- and tolerance-aware correction, including more structured kernel diagnostics, selector provenance, and

robustness-aware geometric parameter generation. Their broader engineering implications are discussed further in Section 6.

5. Practical Inference and System-Level Validation

The controlled experiments in Section 4 evaluate the proposed execution-supervised framework under standardized benchmark conditions. This section further examines whether the same execution-aware principle remains effective in a practical inference setting, where reference geometry is generally unavailable and generated programs must be validated directly through the CAD kernel.

The final inference model was trained independently of the component-study model evaluated in Section 4. Although both configurations use Qwen2.5-Coder-1.5B-Instruct as the backbone and generate executable CadQuery programs, they differ in their training datasets and LoRA adapter configurations. The controlled experiments in Section 4 evaluate the contributions of individual optimization and repair mechanisms, whereas the present section examines the practical generation of executable parametric models without reference geometry. Accordingly, the results reported here should be interpreted as a complementary system-level evaluation of the execution-aware design principle rather than as additional ablations or a continuation of the same trained model.

5.1. Final Model and Training Configuration

The final model uses Qwen2.5-Coder-1.5B-Instruct as the backbone and LoRA with $r = 64$, $\alpha = 128$, and dropout $= 0.05$. Its training data combine a category-structured synthetic CAD corpus with sampled OmniCAD examples. The synthetic corpus contains 163,306 caption–CadQuery pairs spanning geometric primitives, rotational bodies, furniture-like objects, mechanical components, industrial parts, organic profiles, and more complex composite constructions. The supervised learning objective and execution-verified program representation follow the framework described in Section 3.

An additional 12,000 OmniCAD samples are incorporated into the training data, and 1000 records are reserved for validation. The resulting training set therefore contains 174,306 examples. Category expansion increases the diversity of executable construction patterns used for final inference, while the 190,193-example component study quantifies the effects of the learning stages on the fixed benchmark. Unlike the controlled experimental configuration in Section 4, the final inference model does not employ the CD-GRPO-trained adapter. Accordingly, the practical inference results provide complementary system-level evidence for the execution-aware framework rather than representing a further stage of the same controlled ablation sequence.The training configuration of the final inference model is summarized in Table 11.

Table 11. Training configuration of the final execution-supervised inference model.

Configuration Item	Setting
Base model	Qwen2.5-Coder-1.5B-Instruct
LoRA rank	$r = 64$
LoRA scaling	$\alpha = 128$
LoRA dropout	0.05
Synthetic CAD samples	163,306
OmniCAD samples	12,000
Validation samples	1000
Training samples	174,306

The final training recipe uses two supervised epochs, a learning rate of 2×10^{-4} with cosine decay and 3% warmup, a per-device batch size of one with 16 gradient accumulation

steps, and a 1024-token packed context. The final model is selected by validation loss. During reference-free inference, the model is held fixed and OCCT feedback drives candidate acceptance. Geometric policy optimization is evaluated where reference geometry is available, whereas execute-and-select uses the first successfully constructed candidate as defined by the same kernel acceptance condition.

5.2. Kernel-Verified Execute-And-Select Inference

In practical text-to-CAD generation, a reference point cloud is generally unavailable. Candidate selection therefore cannot rely on CD or F-score. Instead, generated programs are evaluated through direct OCCT execution, with successful construction of a nonempty geometric object used as the acceptance criterion.

The evaluation uses a predefined set of 60 English prompts ordered as 20 short object names, 20 compound scenes, and 20 detailed object descriptions. No target CAD or reference geometry is supplied. Accordingly, the audit measures executability, verifier confusion, and measured computational cost.

The inference procedure follows the execute-and-select strategy introduced in Section 3. A greedy candidate is generated first. If this candidate fails geometric execution, up to four additional candidates are sampled at temperatures 0.6, 0.8, 1.0, and 0.6 (top-$p = 0.9$; candidate seeds 1–4). The fixed audit seed is 20261008 and the maximum output length is 1024 tokens. For a candidate set $c_1, \ldots, c_N$, the selected candidate is

$$j^* = \min\{j \in \{1, \ldots, N\} : K(c_j) = \top\}, \tag{13}$$

where $K(c_j)$ denotes successful OCCT execution and construction of a nonempty CAD object.

This procedure does not attempt to identify the geometrically closest candidate to an unavailable reference model. Instead, it addresses a different objective: increasing the probability that at least one candidate is executable under the real CAD kernel. Consequently, kernel execution remains the final acceptance criterion.

The final-model evaluation was performed once with the stated decoding protocol. Greedy generation produced 55 executable programs out of 60 (91.67%; Wilson 95% CI, 81.93–96.39%). Execute-and-select recovered all five failures, producing 60/60 executable programs (100%; Wilson lower bound 93.98%). Two prompts were recovered by the second candidate and three by the third; no fourth or fifth candidate was required. Thus, the procedure required 60 first-pass calls, five second-candidate calls, and three third-candidate calls, totaling 68 LLM calls and 68 OCCT calls. The corresponding execution success rates and computational call counts are summarized in Table 12.

Table 12. Final-model execution and call-count evaluation on the fixed 60-prompt inference test.

Protocol	Executable/60	Wilson 95% CI	LLM Calls	OCCT Calls
Greedy first pass	55/60	81.93–96.39%	60	60
Execute-and-select ($N \leq 5$)	60/60	93.98–100%	68	68

These results demonstrate that the five first-pass execution failures in this prompt set can be resolved by eight additional candidates. The scope of this conclusion is limited to execution validity: the first executable candidate is not necessarily the most geometrically or semantically faithful candidate.

The audit made 68 LLM calls and 68 isolated OCCT calls, or 1.133 calls of each type per successful prompt. On an NVIDIA GeForce RTX 5070 Ti, model generation consumed 629.07 s and isolated OCCT execution/STEP export consumed 163.85 s; total measured generation-plus-execution time was 792.92 s, or 13.22 s per successful prompt (model

loading: 4.61 s; audit wall time after loading: 793.40 s). Mean per-call times were 9.25 s for generation and 2.41 s for the isolated OCCT worker.

5.3. Expanded Validation of the Lightweight Verifier

The positive class is a program that successfully executes under OCCT and exports a nonempty STEP model within a 30 s isolated-worker limit. Mock-CQ acceptance is a predicted positive. The expanded evaluation retains all 500 reference programs in the fixed held-out subset. A seed-20261009 permutation selects 200 source programs for single-fault injection, with 50 cases each for a missing result binding, an undefined translation parameter, a zero-radius profile, and an oversized fillet. These derivatives test common execution and geometric-parameter failures. The verifier is unchanged, every case is included, and labels are assigned by actual OCCT execution.

Re-execution accepts 482 of the 500 reference programs. The remaining 18 fail under the stated evaluation conditions, including 12 three-point arc construction failures, four other topology or Boolean-construction failures, one null-curve failure, and one timeout. All 200 fault derivatives fail execution. Consequently, the mixed population contains 482 positive and 218 negative cases. Mock-CQ yields $TP = 482$, $FP = 18$, $TN = 200$, and $FN = 0$, giving precision 96.40%, recall 100%, specificity 91.74%, and accuracy 97.43%. Each of the four predefined fault types is detected in 50/50 cases. The 18 false acceptances demonstrate that topology and tolerance-dependent construction remain the responsibility of OCCT even when local parameter checks pass. The corresponding prediction results across the evaluated populations are summarized in Table 13.

Table 13. Mock-CQ predictions against OCCT execution.

Population	TP	FP	TN	FN	Precision	Recall	Specificity	Accuracy
500 held-out references	482	18	0	0	96.40%	100%	0%	96.40%
200 single-fault derivatives	0	0	200	0	–	–	100%	100%
700-case diagnostic mixture	482	18	200	0	96.40%	100%	91.74%	97.43%
68 naturally generated candidates	48	3	5	12	94.12%	80.00%	62.50%	77.94%

The diagnostic mixture measures coverage of the specified fault mechanisms, whereas the 68 naturally generated candidates evaluate the verifier under final-model output variation. This second population contains 60 OCCT-executable and eight non-executable candidates, yielding $TP = 48$, $FP = 3$, $TN = 5$, and $FN = 12$. The earlier 200-program positive-only check provides a further reference-distribution recall result of 96%. These confusion statistics and the per-fault detection results are visualized in Figure 6.

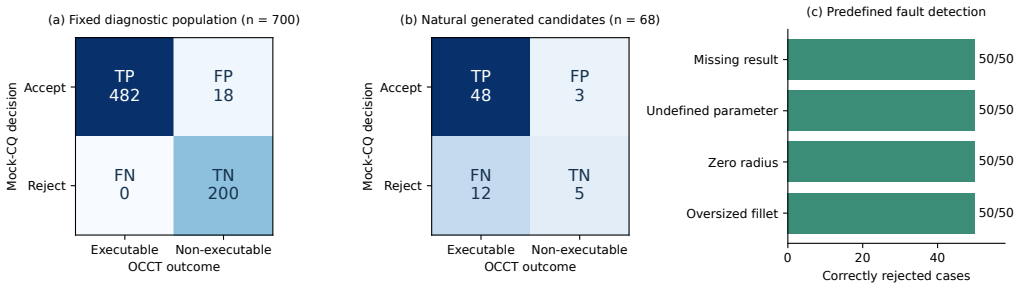


Figure 6. Expanded verifier evaluation. (**a**) Confusion matrix for all 700 predefined diagnostic cases; (**b**) confusion matrix for 68 naturally generated candidates; (**c**) detection of four predefined fault mechanisms, each with 50 tested cases. Counts are actual OCCT execution outcomes. The two matrices represent different program distributions.

Among the 17 candidates rejected by Mock-CQ in the generated sample, eight failures are attributed to unsupported solid construction, five to conservative fillet constraints, and four to API calls or identifiers outside the proxy's supported namespace. To further characterize the verification behavior across different geometric operations, the 68 generated candidates are classified into overlapping operation categories: 64 involve Boolean operations, 56 primitive solids, 26 revolutions, sweeps, or lofts, 19 sketch or profile construction, 18 edge or shell finishing operations, and 17 extrusions. Among the false negatives, 12 involve Boolean operations, 11 primitive solids, four extrusions, three sketch or profile operations, and three edge or shell operations, whereas no false negatives are observed among the 26 candidates involving revolutions, sweeps, or lofts. These results characterize the verification behavior of Mock-CQ on the fixed generated sample and highlight differences in its coverage of geometric operation patterns. The fixed expanded evaluation population is therefore used as the primary basis for quantitative validation, while the boundary-case results provide complementary insights into the coverage of the lightweight verifier.

False positives and false negatives in Mock-CQ have different implications for the execution-supervised framework. A false positive primarily introduces additional computational overhead, as the candidate must undergo OCCT verification and will be rejected if geometric execution fails. In contrast, a false negative may incorrectly penalize a geometrically valid candidate through the reward-gating mechanism in Equation (6), potentially biasing model optimization toward the operation patterns supported by Mock-CQ. Therefore, although Mock-CQ effectively identifies common parameter- and operation-level errors, its predictions cannot replace authoritative geometric verification. By combining lightweight constraint checking with OCCT execution, the proposed framework enables frequent, low-cost supervision during optimization while ensuring that the final acceptance of generated CAD programs is determined by the actual geometric kernel.

5.4. Representative Parametric CAD Results

Figures 7 and 8 present representative parametric CAD models generated by the final inference model, illustrating its capability to construct diverse geometric objects from natural-language descriptions. Eight models successfully verified by OCCT are presented, including five building-related examples—an I-beam, a connection flange, an architectural column, a gusseted bracket, and a lighthouse tower—and three general-purpose CAD objects comprising a spur gear, a wine glass, and a mug. These examples were selected from two additional prompt batches following the quantitative 60-prompt evaluation to demonstrate representative modeling capabilities rather than to provide an independent statistical benchmark. All eight models were generated without manual code modification. Seven were successfully constructed through greedy decoding, while the gusseted bracket was accepted on the second candidate ($T = 0.6$, seed 1) using the same kernel-verified execute-and-select strategy.

The generated models exhibit a variety of parametric construction operations and geometric features. The I-beam is constructed by extruding an I-shaped profile, while the connection flange combines a circular disk, a central bore, and six circumferentially distributed holes. The architectural column consists of a revolved shaft together with plinth and capital elements. The gusseted bracket combines perpendicular plates and a reinforcing gusset, whereas the lighthouse incorporates a tapered tower, gallery, lantern room, and roof. The general-purpose examples further demonstrate periodic tooth replication, the construction of thin-walled revolved profiles, and the integration of a swept handle with a hollow vessel. Based on the exported STEP models, the eight generated B-Reps contain 14, 10, 13, 30, 13, 85, 13, and 9 faces, respectively. These results illustrate the diversity

of geometric construction operations and topological configurations represented by the generated CAD programs.



Figure 7. Representative parametric CAD models generated by the final inference model and successfully verified by OCCT. The first five examples (blue tags) illustrate building-related components and structures, while the remaining three (gray tags) demonstrate general-purpose CAD modeling capabilities. All models were generated without manual code modification.

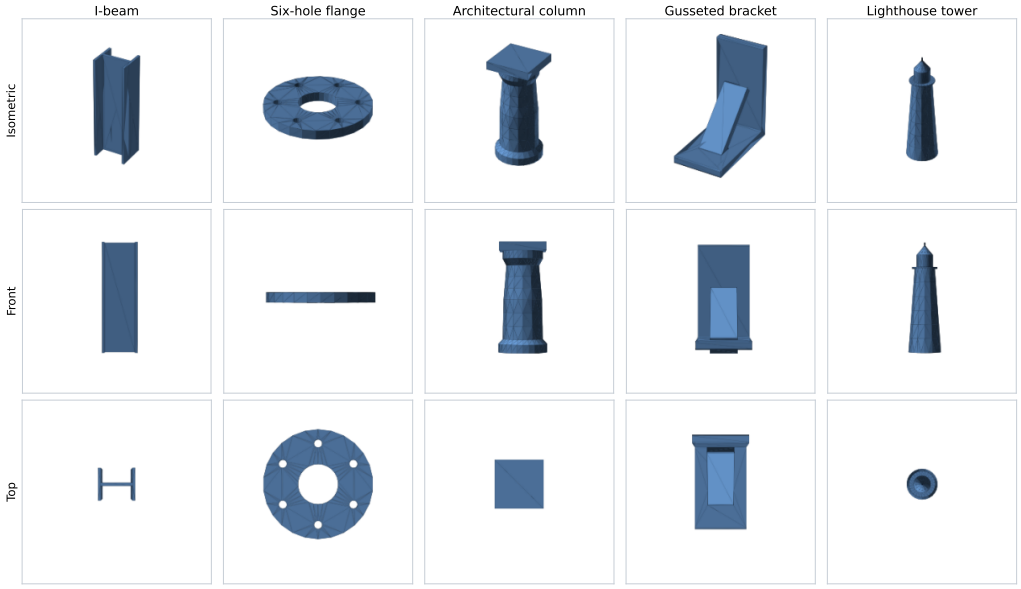


Figure 8. Isometric, front, and top views of the five building-related parametric CAD models shown in Figure 7. The multi-view visualizations illustrate their geometric configurations, including sectional profiles, openings, orthogonal plate connections, and axial features. All five models were successfully constructed and exported through OCCT without manual code modification.

The multi-view results in Figure 8 further demonstrate the geometric characteristics of the building-related examples. The isometric views provide an overall representation of the generated structures, while the front and top views reveal sectional profiles, geometric proportions, openings, and component arrangements. These complementary views facilitate the inspection of geometric features and construction details that may not be apparent from a single perspective. Together with the successful OCCT verification, these results demonstrate the ability of the proposed framework to generate executable and inspectable parametric models with diverse geometric configurations.

It should be emphasized that successful geometric execution establishes the validity of the generated CAD construction under OCCT, rather than complete compliance with specific engineering design requirements. For example, the generated spur gear exhibits repeated tooth features without verification against an exact involute specification, while the building-related examples demonstrate geometric construction rather than validated structural performance or building-code compliance. Similarly, the generated mug is not evaluated against ergonomic or manufacturing specifications. These distinctions are further discussed in Section 6.

Overall, the representative results demonstrate that the proposed framework can translate natural-language descriptions into executable, editable, and geometrically diverse CAD programs. The combination of parametric construction, kernel-level verification, and multi-view visualization provides a practical foundation for automated component modeling, geometric inspection, and subsequent design refinement in AI-assisted engineering workflows.

6. Discussion

6.1. Effects of Execution Supervision

The results indicate that the proposed framework improves text-to-CAD generation through complementary mechanisms operating at different stages of the modeling pipeline. Supervised fine-tuning (SFT) establishes the mapping from natural-language descriptions to parametric CAD programs, including syntactic correctness, operation ordering, parameter definition, and valid result-object binding. The compact B-Rep-to-CadQuery representation supports this process by reducing the target sequence length and exposing executable construction patterns within the effective context window of the model. Execution-related supervision is subsequently introduced to address geometric behavior beyond program syntax.

During model optimization, CD-GRPO uses continuous geometric rewards derived from executed CAD models to favor generated shapes with smaller discrepancies from their references. At inference time, execution-guided iterative repair (Iter) addresses residual failures by revising candidate programs in response to localized parameter, API, and geometric-kernel diagnostics. EV-RFT operates at the training-data level by retaining self-generated examples that satisfy both execution verification and a semantic-similarity filter, thereby extending the range of construction patterns represented in the training corpus. These mechanisms serve distinct purposes: geometric policy optimization, failure correction, and targeted distribution expansion, respectively.

The quantitative findings are consistent with these complementary roles. On the fixed OmniCAD-500 subset, the invalid rate (IR) decreases from 7.60% to 0.60%, while the reported valid-only mean Chamfer Distance (CD) decreases from 199.9 to 183.8, an improvement of approximately 8%. The aligned F@0.20 remains above 93% across the controlled comparisons, whereas the stricter aligned F@0.05 decreases from 66.65% to 64.73%. Thus, the results demonstrate improved execution reliability and lower aggregate geometric discrepancy, but not a uniform improvement in fine-scale surface correspondence. Moreover, iterative repair changes the set of samples included in valid-only geometric averages.

6.2. Proxy Verification and Engineering Implications

A central feature of the proposed framework is the separation of frequent, low-cost execution-related supervision from authoritative geometric-kernel verification. Mock-CQ checks a defined set of parameter- and operation-level constraints without constructing a complete B-Rep, whereas OCCT explicitly evaluates curves, surfaces, topology, Boolean

operations, and numerical tolerances. The two verifiers therefore provide complementary information rather than interchangeable judgments of geometric validity.

This division is important for computational efficiency because execution-related feedback must be obtained repeatedly during model optimization and iterative generation. Mock-CQ can identify common local errors at low cost, reserving full geometric execution for candidates requiring kernel-level acceptance. The validation results also reveal asymmetric consequences of proxy errors. A false acceptance may result in an unnecessary OCCT execution, whereas a false rejection can reduce the reward assigned to a geometrically valid program through the gating mechanism in Equation (6), potentially favoring the proxy's supported operation patterns. Retaining OCCT as the final acceptance mechanism preserves kernel-level verification while allowing lightweight feedback to be used more frequently.

The practical inference results further illustrate the operational value of this separation. For the predefined set of 60 text prompts without reference geometry, the final category-expanded model produces 55 executable programs under greedy decoding and 60 under OCCT-verified execute-and-select inference. The generated CadQuery programs remain editable and inspectable, allowing their parameters, feature definitions, and construction sequences to be reviewed and modified. Representative building-related examples, including an I-beam, a connection flange, an architectural column, and a gusseted bracket, further demonstrate the construction of diverse component geometries. These results establish execution-verified generation capability within the evaluated settings.

In architecture, engineering, and construction (AEC) and building information modeling (BIM) workflows, potential applications include preliminary modeling of connection plates, openings, brackets, facade subcomponents, foundations, retaining structures, excavation-support components, and tunnel elements. Before such models can be accepted for engineering use, domain-specific requirements concerning units, dimensions, interfaces, materials, load paths, code compliance, constructability, and life-cycle performance must be evaluated separately. This distinction is consistent with established approaches to automated BIM rule checking [63] and classifications of model-information requirements [64]. The proposed framework therefore provides a basis for generating and inspecting editable parametric geometry that can subsequently enter more specialized engineering verification workflows.

6.3. Limitations and Future Work

The scope of the present evaluation is primarily limited to constructive parametric CAD objects. Free-form geometry, large assemblies, complex feature histories, and domain-specific engineering constraints are less represented in the tested settings. The controlled out-of-distribution (OOD) evaluation contains 60 manually designed prompts across six categories targeted by the EV-RFT curriculum. This experiment supports conclusions about execution reliability and targeted distribution expansion.

The available experimental design also defines the scope of statistical inference. The main controlled comparisons are based on a fixed OmniCAD-500 subset and single training runs. Paired outcomes support McNemar tests for the nested Iter repair stage; they do not establish a paired training effect between SFT and CD-GRPO. Consequently, aggregate geometric comparisons and uncertainty estimates must be interpreted within their respective evaluation protocols.

A further consideration is the coverage of the verification and data-selection mechanisms. Mock-CQ implements nine local constraint families and can incorrectly reject programs that are valid under OCCT, while the CLIP ViT-B/32 threshold of 0.25 used by EV-RFT provides an automated semantic-similarity filter rather than direct human assess-

ment of design intent. The influence of alternative reward-gating parameters, geometric-reward settings, group sizes, and semantic thresholds would require additional training experiments rather than post hoc rescoring of fixed model outputs.

Future research will extend execution-aware generation toward richer topological constraints, tolerance-sensitive geometric operations, and more informative verification feedback. Larger held-out benchmarks with reference geometry and independent semantic assessment will help characterize geometric quality and generalization across more diverse CAD tasks. Repeated training runs and controlled implementations of external baselines will further strengthen statistical and cross-method comparisons. For AEC-oriented applications, integrating domain-specific dimensional, material, structural, and interoperability checks will provide a pathway from executable parametric models toward engineering-validated design workflows.

7. Conclusions

This study developed a unified execution-supervised framework for reliable text-to-CAD generation, extending geometric execution from conventional post-generation validation to a supervisory mechanism throughout model optimization, inference-time repair, and self-training. The framework employs compact, executable CadQuery representations and a hierarchical verification strategy that combines lightweight Mock-CQ screening with authoritative OCCT execution to balance computational efficiency and geometric verification reliability. Furthermore, Chamfer-Distance-guided reinforcement learning, execution-guided iterative repair, and execution-verified self-training were integrated to coordinate geometric optimization, failure correction, and targeted category expansion. Experimental results demonstrate that geometric reinforcement learning and execution-guided iterative repair provide complementary benefits by reducing geometric discrepancies and correcting residual execution failures, respectively. On the OmniCAD-500 subset, the invalid output rate decreased from 7.60% to 0.60%, while the valid-only mean Chamfer Distance decreased from 199.9 to 183.8. Additional category-expansion experiments show that execution-verified self-training reduced the invalid output rate from 38.33% to 5.00%, improving generation reliability across targeted geometric categories. In a separate reference-free evaluation, OCCT-verified that execute-and-select inference successfully yielded executable outputs for all 60 test prompts. These findings demonstrate that incorporating execution feedback throughout the generation process can substantially reduce geometric execution failures and improve the robustness of automated parametric modeling. Nevertheless, the decrease in the stricter F@0.05 metric indicates that improved executability and aggregate geometric similarity do not necessarily guarantee better fine-scale geometric correspondence.

From an engineering perspective, the proposed framework enables natural-language descriptions to be converted into executable, editable, and inspectable parametric CAD models, providing a technical foundation for automated building-component modeling and AI-assisted engineering design. However, geometric executability does not necessarily imply engineering validity, and the present evaluation remains limited to selected geometric categories and execution-related requirements. Future research will extend the framework toward more complex geometries and topological operations, improve geometric and semantic fidelity, and incorporate structural performance, engineering constraints, and domain-specific verification mechanisms to support reliable automated engineering design.

Author Contributions: L.N.: Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing—Original Draft, Visualization. J.Z.: Conceptualization, Methodology, Resources, Writing—Review and Editing, Supervision, Project Administration, Funding Acquisition. W.L.: Methodology, Validation, Formal Analysis, Software, Writing—Review and Editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Natural Science Foundation of China under Grant No. 12502030 and the Shanghai Pujiang Program under Grant No. 25PJA146.

Data Availability Statement: The processed evaluation records, model evaluation outputs, and analysis scripts supporting the reported results are available from the corresponding author upon reasonable request.

Conflicts of Interest: The Author Jisheng Zhao was employed by the company Floating AI. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Hui, B.; Yang, J.; Cui, Z.; Yang, J.; Liu, D.; Zhang, L.; Liu, T.; Zhang, J.; Yu, B.; Lu, K.; et al. Qwen2.5-Coder Technical Report. *arXiv* **2024**, arXiv:2409.12186.
2. Roziere, B.; Gehring, J.; Gloeckle, F.; Sootla, S.; Gat, I.; Tan, X.E.; Adi, Y.; Liu, J.; Sauvestre, R.; Remez, T.; et al. Code Llama: Open Foundation Models for Code. *arXiv* **2023**, arXiv:2308.12950.
3. Li, R.; Allal, L.B.; Zi, Y.; Muennighoff, N.; Kocetkov, D.; Mou, C.; Marone, M.; Akiki, C.; Li, J.; Chim, J.; et al. StarCoder: May the source be with you! *arXiv* **2023**, arXiv:2305.06161.
4. Guo, D.; Zhu, Q.; Yang, D.; Xie, Z.; Dong, K.; Zhang, W.; Chen, G.; Bi, X.; Wu, Y.; Li, Y.K.; et al. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence. *arXiv* **2024**, arXiv:2401.14196.
5. Wu, R.; Xiao, C.; Zheng, C. DeepCAD: A deep generative network for computer-aided design models. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 10–17 October 2021.
6. Xu, X.; Willis, K.D.D.; Lambourne, J.G.; Cheng, C.Y.; Jayaraman, P.K.; Furukawa , Y. SkexGen: Autoregressive generation of CAD construction sequences with disentangled codebooks. In Proceedings of the 39th International Conference on Machine Learning, Baltimore, MA, USA, 17–23 July 2022.
7. Xu, X.; Jayaraman, P.K.; Lambourne, J.G.; Willis, K.D.D.; Furukawa, Y. Hierarchical Neural Coding for Controllable CAD Model Generation. *arXiv* **2023**, arXiv:2307.00149. [CrossRef]
8. Xu, X.; Lambourne, J.G.; Jayaraman, P.K.; Wang, Z.; Willis, K.D.D.; Furukawa, Y. BrepGen: A B-rep Generative Diffusion Model with Structured Latent Geometry. In Proceedings of the ACM SIGGRAPH 2024 Conference Papers, Denver, CO, USA, 27 July–1 August 2024.
9. Xu, J.; Wang, C.; Zhao, Z.; Liu, W.; Ma, Y.; Gao, S. CAD-MLLM: Unifying Multimodality-Conditioned CAD Generation With MLLM. *arXiv* **2024**, arXiv:2411.04954.
10. Kolodiazhnyi, M.; Tarasov, D.; Zhemchuzhnikov, D.; Nikulin, A.; Zisman, I.; Vorontsova, A.; Konushin, A.; Kurenkov, V.; Rukhovich, D. Cadrille: Multi-modal CAD Reconstruction with Reinforcement Learning. *arXiv* **2025**, arXiv:2505.22914.
11. Chen, T.; Yu, C.; Hu, Y.; Li, J.; Xu, T.; Cao, R.; Zhu, L.; Zang, Y.; Zhang, Y.; Li, Z.; et al. Img2CAD: Conditioned 3D CAD Model Generation from Single Image with Structured Visual Geometry. *arXiv* **2024**, arXiv:2410.03417.
12. Liu, Y.; Obukhov, A.; Wegner, J.D.; Schindler, K. Point2CAD: Reverse Engineering CAD Models from 3D Point Clouds. *arXiv* **2024**, arXiv:2312.04962.
13. Li, J.; Ma, W.; Li, X.; Lou, Y.; Zhou, G.; Zhou, X. CAD-Llama: Leveraging Large Language Models for Computer-Aided Design Parametric 3D Model Generation. *arXiv* **2025**, arXiv:2505.04481.
14. Khan, M.; Sinha, S.; Sheikh, T.; Stricker, D.; Ali, S.; Afzal, M. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. In Proceedings of the 38th International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 10–15 December 2024.
15. Xie, H.; Ju, F. Text-to-CadQuery: A New Paradigm for CAD Generation with Scalable Large Model Capabilities. *arXiv* **2025**, arXiv:2505.06507.
16. Guan, Y.; Wang, X.; Xing, X.; Zhang, J.; Xu, D.; Yu, Q. CAD-Coder: Text-to-CAD Generation with Chain-of-Thought and Geometric Reward. *arXiv* **2025**, arXiv:2505.19713.
17. Yuan, Z.; Shi, J.; Huang, Y. OpenECAD: An Efficient Visual Language Model for Editable 3D-CAD Design. *Comput. Graph.* **2024**, *124*, 104048. [CrossRef]
18. Succar, B. Building information modelling framework: A research and delivery foundation for industry stakeholders. *Autom. Constr.* **2009**, *18*, 357–375. [CrossRef]
19. Sacks, R.; Koskela, L.; Dave, B.A.; Owen, R. Interaction of Lean and Building Information Modeling in Construction. *J. Constr. Eng. Manag.* **2010**, *136*, 968–980. [CrossRef]
20. Volk, R.; Stengel, J.; Schultmann, F. Building Information Modeling (BIM) for existing buildings—Literature review and future needs. *Autom. Constr.* **2014**, *38*, 109–127. Correction in *Autom. Constr.* **2014**, *43*, 204. [CrossRef]

21. Guo, H.; Liu, S.; Pan, H.; Liu, Y.; Tong, X.; Guo, B. ComplexGen: CAD reconstruction by B-Rep chain complex generation. In Proceedings of the SIGGRAPH, Vancouver, BC, Canada, 7–11 August 2022.
22. Ganin, Y.; Bartunov, S.; Li, Y.; Keller, E.; Saliceti, S. Computer-Aided Design as Language. *arXiv* **2021**, arXiv:2105.02769. [CrossRef]
23. Open Cascade S.A.S. Open CASCADE Technology: Modeling Algorithms. Available online: https://occt3d.com/dev/doc/overview/html/occt_user_guides__modeling_algos.html (accessed on 19 October 2026).
24. Li, C.; Pan, H.; Bousseau, A.; Mitra, N.J. Free2CAD: Parsing Freehand Drawings into CAD Commands. *ACM Trans. Graph.* **2022**, *41*, 1–16. [CrossRef]
25. Rukhovich, D.; Dupont, E.; Mallis, D.; Cherenkova, K.; Kacem, A.; Aouada, D. CAD-Recode: Reverse Engineering CAD Code from Point Clouds. In Proceedings of the ICCV, Honolulu, HI, USA, 19–25 October 2025; pp. 9801–9811.
26. Jayaraman, P.K.; Sanghi, A.; Lambourne, J.G.; Willis, K.D.; Davies, T.; Shayani, H.; Morris, N. UV-Net: Learning From Boundary Representations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2021; pp. 11703–11712.
27. Lambourne, J.G.; Willis, K.D.; Jayaraman, P.K.; Sanghi, A.; Meltzer, P.; Shayani, H. BRepNet: A Topological Message Passing System for Solid Models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA , 20–25 June 2021; pp. 12773–12782.
28. Seff, A.; Zhou, W.; Richardson, N.; Adams, R.P. Vitruvion: A Generative Model of Parametric CAD Sketches. In Proceedings of the International Conference on Learning Representations, Virtual, 25–29 April 2022.
29. Willis, K.D.; Jayaraman, P.K.; Chu, H.; Tian, Y.; Li, Y.; Grandi, D.; Sanghi, A.; Tran, L.; Lambourne, J.G.; Solar-Lezama, A.; et al. JoinABLe: Learning Bottom-Up Assembly of Parametric CAD Joints. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA , 18–24 June 2022; pp. 15849–15860.
30. Le, H.; Wang, Y.; Gotmare, A.D.; Savarese, S.; Hoi, S.C.H. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In Proceedings of the 36th International Conference on Neural Information Processing Systems, New Orleans, LA, USA, 28 November–9 December 2022.
31. Li, Y.; Choi, D.; Chung, J.; Kushman, N.; Schrittwieser, J.; Leblond, R.; Eccles, T.; Keeling, J.; Gimeno, F.; Lago, A.D.; et al. Competition-level code generation with AlphaCode. *Science* **2022**, *378*, 1092–1097. [CrossRef] [PubMed]
32. Gehring, J.; Zheng, K.; Copet, J.; Mella, V.; Carbonneaux, Q.; Cohen, T.; Synnaeve, G. RLEF: Grounding Code LLMs in Execution Feedback with Reinforcement Learning. In Proceedings of the 42nd International Conference on Machine Learning, Vancouver, BC, Canada, 13–19 July 2025.
33. Aggarwal, P.; Parno, B.; Welleck, S. AlphaVerus: Bootstrapping Formally Verified Code Generation through Self-Improving Translation and Treefinement. *arXiv* **2024**, arXiv:2412.06176.
34. Wilf, A.; Aggarwal, P.; Parno, B.; Fried, D.; Morency, L.P.; Liang, P.P.; Welleck, S. Propose, Solve, Verify: Self-Play Through Formal Verification. *arXiv* **2025**, arXiv:2512.18160.
35. Shao, Z.; Wang, P.; Zhu, Q.; Xu, R.; Song, J.; Bi, X.; Zhang, H.; Zhang, M.; Li, Y.K.; Wu, Y.; et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv* **2024**, arXiv:2402.03300.
36. Yu, Q.; Zhang, Z.; Zhu, R.; Yuan, Y.; Zuo, X.; Yue, Y.; Dai, W.; Fan, T.; Liu, G.; Liu, L.; et al. DAPO: An Open-Source LLM Reinforcement Learning System at Scale. *arXiv* **2025**, arXiv:2503.14476.
37. Liu, Z.; Chen, C.; Li, W.; Qi, P.; Pang, T.; Du, C.; Lee, W.S.; Lin, M. Understanding R1-Zero-Like Training: A Critical Perspective. *arXiv* **2025**, arXiv:2503.20783.
38. Wang, Y.; Kordi, Y.; Mishra, S.; Liu, A.; Smith, N.A.; Khashabi, D.; Hajishirzi, H. Self-Instruct: Aligning language models with self-generated instructions. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, Toronto, ON, Canada, 9–14 July 2023.
39. Zelikman, E.; Wu, Y.; Mu, J.; Goodman, N.D. STaR: Bootstrapping reasoning with reasoning. In Proceedings of the NeurIPS, New Orleans, LO, USA, 28 November–9 December 2022.
40. Gulcehre, C.; Paine, T.L.; Srinivasan, S.; Konyushkova, K.; Weerts, L.; Sharma, A.; Siddhant, A.; Ahern, A.; Wang, M.; Gu, C.; et al. Reinforced Self-Training (ReST) for Language Modeling. *arXiv* **2023**, arXiv:2308.08998.
41. Yuan, Z.; Yuan, H.; Li, C.; Dong, G.; Lu, K.; Tan, C.; Zhou, C.; Zhou, J. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models. *arXiv* **2023**, arXiv:2308.01825.
42. Anil, G.G.; Nagaraj, D.M.; Shanmugam, K.; Shakkottai, S. Rejection Sampling Based Fine Tuning Secretly Performs PPO. In Proceedings of the Second Workshop on Test-Time Adaptation: Putting Updates to the Test at ICML 2025, Vancouver, BC, Canada, 18 July 2025.
43. Lan, L.C.; Bai, A.; Cheng, M.; Hsieh, C.J.; Zhou, T. Exploring Expert Failures Improves LLM Agent Tuning. *arXiv* **2025**, arXiv:2504.13145.
44. Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; et al. Self-Refine: Iterative refinement with self-feedback. In Proceedings of the NeurIPS, New Orleans, LA, USA, 10–16 December 2023.

45. Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; Yao, S. Reflexion: Language agents with verbal reinforcement learning. In Proceedings of the NeurIPS, New Orleans, LA, USA, 10–16 December 2023.
46. Chen, X.; Lin, M.; Schärli, N.; Zhou, D. Teaching large language models to self-debug. In Proceedings of the ICLR, Vienna, Austria, 7–11 May 2024.
47. Zhang, K.; Li, Z.; Li, J.; Li, G.; Jin, Z. Self-Edit: Fault-aware code editor for code generation. In Proceedings of the ACL, Toronto, ON, Canada, 9–14 July 2023.
48. Qu, Y.; Zhang, T.; Garg, N.; Kumar, A. Recursive introspection: Teaching language model agents how to self-improve. In Proceedings of the NeurIPS, Vancouver, BC, Canada, 10–15 December 2024.
49. Castagnone, A.; Nitti, G. A Neuro-Symbolic Framework for Ensuring Deterministic Reliability in AI-Assisted Structural Engineering: The SYNAPSE Architecture. *Buildings* **2026**, *16*, 534. [CrossRef]
50. Ellis, K.; Wong, C.; Nye, M.; Sable-Meyer, M.; Morales, L.; Hewitt, L.; Cary, L.; Solar-Lezama, A.; Tenenbaum, J.B. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual, 20–25 June 2021; pp. 835–850. [CrossRef]
51. Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; Neubig, G. PAL: Program-aided Language Models. In Proceedings of the 40th International Conference on Machine Learning, Honolulu, HA, USA, 23–29 July 2023; Volume 202, pp. 10764–10799.
52. Chen, B.; Zhang, F.; Nguyen, A.; Zan, D.; Lin, Z.; Lou, J.G.; Chen, W. CodeT: Code Generation with Generated Tests. In Proceedings of the International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023.
53. Fan, H.; Su, H.; Guibas, L.J. A point set generation network for 3D object reconstruction from a single image. In Proceedings of the CVPR, Honolulu, HA, USA, 21–26 July 2017.
54. Tatarchenko, M.; Richter, S.R.; Ranftl, R.; Li, Z.; Koltun, V.; Brox, T. What do single-view 3D reconstruction networks learn? In Proceedings of the CVPR, Long Beach, CA, USA, 15–20 June 2019.
55. Besl, P.J.; McKay, N.D. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* **1992**, *14*, 239–256. [CrossRef]
56. Koch, S.; Matveev, A.; Jiang, Z.; Williams, F.; Artemov, A.; Burnaev, E.; Alexa, M.; Zorin, D.; Panozzo, D. ABC: A big CAD model dataset for geometric deep learning. In Proceedings of the CVPR, Long Beach, CA, USA, 16–20 June 2019.
57. Willis, K.D.D.; Pu, Y.; Luo, J.; Chu, H.; Du, T.; Lambourne, J.G.; Solar-Lezama, A.; Matusik, W. Fusion 360 gallery: A dataset and environment for programmatic CAD construction from human design sequences. *ACM Trans. Graph. TOG* **2021**, *40*, 54.
58. Abushama, K.; Hawkins, W.; Pelecanos, L.; Ibell, T. Optimizing the Embodied Carbon of Concrete, Timber, and Steel Piles with a Case Study. In *1st International Conference on Net-Zero Built Environment*; Lecture Notes in Civil Engineering; Springer: Berlin/Heidelberg, Germany, 2025; Volume 237, pp. 899–911. [CrossRef]
59. Abushama, K.; Hawkins, W.; Pelecanos, L.; Ibell, T. Barriers and Enablers to the Decarbonisation of Deep Foundations in the UK: Insights from Geotechnical Industry Experts. *Environ. Sustain. Indic.* **2026**, *32*, 101521. [CrossRef]
60. Lin, J.; Hu, Z.; Zhang, J.; Yu, F. A NaturalLanguageBased Approach to Intelligent Data Retrieval and Representation for Cloud BIM. *Comput.-Aided Civ. Infrastruct. Eng.* **2016**, *31*, 18–33. [CrossRef]
61. Zheng, J.; Fischer, M. Dynamic prompt-based virtual assistant framework for BIM information search. *Autom. Constr.* **2023**, *155*, 105067. [CrossRef]
62. Du, C.; Esser, S.; Nousias, S.; Borrmann, A. Text2BIM: Generating Building Models Using a Large Language Model-Based Multiagent Framework. *J. Comput. Civ. Eng.* **2026**, *40*, 04025142. [CrossRef]
63. Eastman, C.; Lee, J.M.; Jeong, Y.S.; Lee, J.K. Automatic rule-based checking of building designs. *Autom. Constr.* **2009**, *18*, 1011–1033. [CrossRef]
64. Solihin, W.; Eastman, C. Classification of rules for automated BIM rule checking development. *Autom. Constr.* **2015**, *53*, 69–82. [CrossRef]
65. Zhang, J.; El-Gohary, N.M. Semantic NLP-Based Information Extraction from Construction Regulatory Documents for Automated Compliance Checking. *J. Comput. Civ. Eng.* **2016**, *30*, 04015014. [CrossRef]
66. Zhang, S.; Teizer, J.; Lee, J.K.; Eastman, C.M.; Venugopal, M. Building Information Modeling (BIM) and Safety: Automatic Safety Checking of Construction Models and Schedules. *Autom. Constr.* **2013**, *29*, 183–195. [CrossRef]
67. Abioye, S.O.; Oyedele, L.O.; Akanbi, L.; Ajayi, A.; Davila Delgado, J.M.; Bilal, M.; Akinade, O.O.; Ahmed, A. Artificial intelligence in the construction industry: A review of present status, opportunities and future challenges. *J. Build. Eng.* **2021**, *44*, 103299. [CrossRef]
68. Kumar, A.; Owen, L.; Chowdhury, N.R.; Gura, F. ZClip: Adaptive Spike Mitigation for LLM Pre-Training. *arXiv* **2025**, arXiv:2504.02507.
69. Radford, A.; Kim, J.W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; et al. Learning transferable visual models from natural language supervision. In Proceedings of the ICML, Virtual, 18–24 July 2021.

70. Hu, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W. LoRA: Low-rank adaptation of large language models. In Proceedings of the ICLR, Virtual, 25–29 April 2022.
71. Wilson, E.B. Probable Inference, the Law of Succession, and Statistical Inference. *J. Am. Stat. Assoc.* **1927**, *22*, 209–212. [CrossRef]
72. Efron, B. Bootstrap Methods: Another Look at the Jackknife. *Ann. Stat.* **1979**, *7*, 1–26. [CrossRef]
73. McNemar, Q. Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages. *Psychometrika* **1947**, *12*, 153–157. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.