AIM-CAD: CAUSALITY-INSPIRED AUDITING OF HISTORICAL EDITS FOR TRAINING-FREE CAD EDITING

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-guided parametric CAD editing requires realizing user-specified modifications while preserving unaffected content and maintaining a valid, editable model. However, obtaining large-scale, high-quality editing supervision is costly, while task-specific fine-tuning requires additional model-specific adaptation. This motivates training-free editing that reuses historical edits as external experience. Such reuse remains challenging: new requests may differ from available examples in instruction or CAD structure, and historical traces may contain redundant or context-specific operations. Moreover, plausible edit plans may not translate faithfully into executable CAD operations. We propose Audited Intent–Operation Memory for CAD Editing (AIM-CAD), an experience-guided, training-free framework for reusing historical edits. AIM-CAD builds an audited memory by organizing historical edits as intent–operation graphs and applying causality-inspired interventions to check instruction-level executable sufficiency and prune redundant operation attributions. These interventions assess functional support within a fixed CAD program. Relevant audited experience is then retrieved across instruction semantics and CAD structure. A Graph-Guided Dual-Output Generator produces both structural and executable realizations, whose consistency is checked by Graph–Script Consistency Validation before output. Experiments on adapted CAD-Editor and BenchCAD benchmarks across multiple LLM backbones demonstrate the effectiveness of AIM-CAD. Code will be available at `https://anonymous.4open.science/r/aim-cad-4F2B/`.

1 INTRODUCTION

Language-guided parametric CAD editing modifies an existing construction history according to a user instruction while preserving unaffected design content and maintaining a valid, editable model. Recent methods address this task through locate-and-infill and iterative refinement, often relying on trained CAD-specific components and aligned editing data (Yuan et al., 2025; 2026). Complementary work explores geometry-driven editing and executable CAD reconstruction (Ma et al., 2025; Rukhovich et al., 2025). However, constructing high-quality aligned editing data is difficult and costly because the instruction, parametric operation sequence, and resulting geometry must remain mutually consistent. This motivates a complementary training-free setting that reuses available editing experience at inference time without task-specific parameter updates.

LLMs provide a promising foundation for this goal because they combine language understanding, structured reasoning, and code generation, capabilities already explored for CAD generation and manipulation (Li et al., 2025; Wang et al., 2025; Li et al., 2026; Alrashedy et al., 2025). Nevertheless, direct prompting leaves a fixed LLM to infer unfamiliar and construction-dependent edits from scratch. Structured memory has enabled language agents to retrieve and recombine past experiences across tasks (Zhang et al., 2026b). In CAD editing, however, historical examples are often only partially relevant because their operations depend on the original construction context. Effective reuse therefore requires identifying transferable experience and adapting it to the current CAD model.

Achieving this goal presents three key challenges. (i) Limited coverage of historical edits. High-quality CAD editing examples are costly to obtain, and new requests rarely have exact historical counterparts. Useful precedents must therefore be identified across variations in both instruction semantics and CAD structure. (ii) Ambiguous intent–operation evidence in historical edits. A his-

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

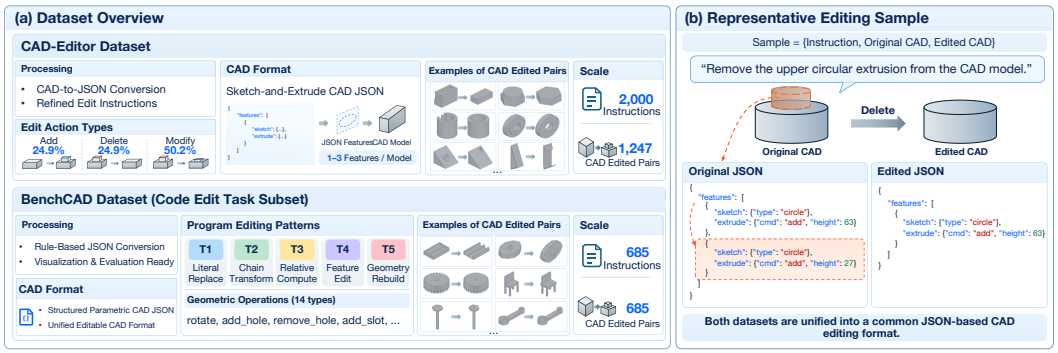

Figure 1: Overview of the adapted CAD editing benchmarks and unified representation. (a) CAD-Editor and the CodeEdit subset of BenchCAD, showing their editing annotations, representative edit pairs, and dataset statistics. A single original–edited CAD pair in CAD-Editor may be associated with multiple edit instructions. (b) A representative editing sample in the unified JSON-based CAD representation, consisting of an instruction, an original CAD model, and its corresponding edited CAD model together with their structured JSON representations.

torical edit trace may contain multiple effects, redundant operations, or construction-specific modifications, making it unclear which operations actually support the intended modification. (iii) Gap between edit plans and executable realizations. A plausible editing plan may not be faithfully realized through parametric operations, as feature dependencies and operation ordering can cause the executed result to diverge from the intended structural modification.

To address these challenges, we introduce AIM-CAD, a training-free framework for reusing historical CAD editing experience. AIM-CAD retrieves relevant experience by jointly considering instruction semantics and CAD structure rather than relying on exact historical matches. Candidate intent–operation relations are audited through executable interventions to retain functionally supported evidence. A graph-guided dual-output generator then produces structural and executable realizations whose consistency is validated before output.

Our contributions are summarized as follows: (i) We introduce AIM-CAD as an early exploration of training-free parametric CAD editing, enabling frozen LLMs to reuse historical editing experience without task-specific parameter updates. The framework supports multiple LLM backbones without backbone-specific training. (ii) We develop an intervention-audited memory of historical intent–operation graphs, checking instruction-level executable sufficiency and pruning redundant operation attributions. Retrieved experience guides a Graph-Guided Dual-Output Generator with Graph–Script Consistency Validation. (iii) We adapt CAD-Editor (Yuan et al., 2025) and the editing subset of BenchCAD (Zhang et al., 2026a) to a unified structured CAD editing setting, and establish an editing-aware evaluation protocol for assessing geometric fidelity, structural preservation, unintended modifications, and instruction alignment. (iv) Experimental results across two benchmark settings and multiple LLM backbones demonstrate the effectiveness of AIM-CAD, with ablation studies further validating its key design choices.

2 Related Work

2.1 Language-Guided CAD Generation and Editing

Parametric CAD research has progressed from sequence-based generation to language-conditioned and LLM-assisted modeling (Wu et al., 2021; Khan et al., 2024; Li et al., 2025). CAD editing additionally requires modifying an existing construction history while preserving unaffected structure. CAD-Editor adopts a locate-and-infill formulation and reports an instruction-retrieval in-context baseline (Yuan et al., 2025). CADMorph uses a geometry-driven plan–generate–verify process (Ma et al., 2025), while CAD-Refiner combines multimodal interpretation, sequence injection, and geo-

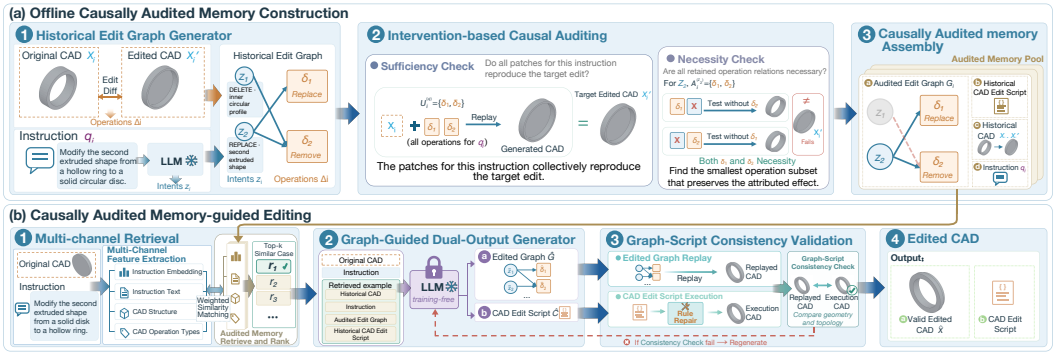

Figure 2: Overview of AIM-CAD, which constructs a causally audited memory from historical edits offline and retrieves from this memory to guide new CAD edits at inference time. (a) Historical edits are converted into intent–operation graphs and processed by Intervention-based Causal Auditing. Sufficiency and necessity checks evaluate the executable coverage and redundancy of candidate operation attributions and produce a reusable memory of audited graphs and edit scripts. (b) For a new edit, relevant cases are retrieved from this memory to guide generation of an edited graph and CAD edit script. The graph is replayed and the script is executed, and their resulting CAD models are cross-validated before producing the final CAD.

metric feedback (Yuan et al., 2026). Related work also studies executable CAD reconstruction and inference-time visual feedback (Rukhovich et al., 2025; Li et al., 2026).

CAD-Editor and BenchCAD provide instruction-guided editing data for parametric sequences and executable programs, respectively (Yuan et al., 2025; Zhang et al., 2026a), while neuralCAD-Edit provides expert-authored multimodal requests (Perrett et al., 2026). Because geometrically similar models may have different parametric realizations (Wu et al., 2021; Dupont et al., 2024), prior evaluation considers both geometry and construction structure.

2.2 Training-Free Adaptation and External Memory

Prior inference-time adaptation methods use reasoning, feedback, and episodic memory without updating model parameters (Yao et al., 2023; Madaan et al., 2023; Shinn et al., 2023). Experience-based methods further extract and reuse information from prior trajectories or interactions (Zhao et al., 2024; Lingam et al., 2025). Recent memory studies also show that memory granularity, redundancy, organization, and retrieval can substantially affect downstream performance (Pan et al., 2025; Wu et al., 2025).

2.3 Causality-Inspired Intervention-Based Evidence Auditing

Controlled interventions are widely used to assess whether selected variables or components are functionally relevant to an observed outcome. Causal representation learning and counterfactual reasoning provide perspectives on interventions and their effects (Schölkopf et al., 2021; Saengkyongam et al., 2024; Ravfogel et al., 2025; Vashishtha et al., 2026), while activation patching examines functional attribution through interventions on internal activations, with conclusions depending on the intervention design and evaluation metric (Zhang & Nanda, 2024).

3 Method

3.1 AIM-CAD Framework

Given a parametric CAD model X and a natural-language editing instruction q, AIM-CAD produces an edited model $\hat{X}$ while aiming to preserve unrelated design structure. We represent a CAD document as $X = (F, M)$, where $F = [f_1, \ldots, f_n]$ is an ordered list of editable features or construction operations and M contains fields that must remain unchanged.

Historical edits form a collection $\mathcal{H} = \{(X_i, Q_i, X_i')\}_{i=1}^{N}$, where X_i and X_i' are source and target models and Q_i contains descriptions of the edit. AIM-CAD converts these examples into an external memory of intent–operation graphs and executable edit scripts. Offline, instruction decomposition and patch alignment produce candidate graphs, which are audited through executable interventions. Online, a retrieved example guides a frozen language model to jointly generate a graph and a script. Both branches undergo executable validation.

3.2 Historical Edit Graph Generator

Executable edit representation. For each historical pair, we compute an ordered JSON patch

$$\Delta_i = [\delta_{i1}, \ldots, \delta_{iK_i}] = \mathrm{Diff}(F_i, F_i'). \tag{1}$$

Each δ_{ik} is an RFC 6902 `add`, `remove`, or `replace` operation on the editable feature subtree. These are document modifications, not necessarily individual geometric CAD commands. Full replay must exactly recover F_i' without modifying M_i. An adaptive diff selects between nested edits and parent-level replacement according to serialized patch size.

Instruction decomposition and alignment. For each $q \in Q_i$, the model first extracts atomic intents $Z_i^{(q)} = [z_{i1}^{(q)}, \ldots, z_{ir_i^{(q)}}^{(q)}]$ using only that instruction and fixed decomposition rules. An intent contains an action, a textual target, and optional typed parameters.

A separate model call aligns these intents to Δ_i. It receives the current instruction, its intents, and patch descriptions containing operation indices, paths, and applicable before/after values. Local values are extracted by sequential replay to account for earlier patch operations. Each instruction is processed independently. Alignment may be many-to-many, and an intent may remain unaligned rather than being assigned a speculative edge.

The raw graph is

$$G_i^{\mathrm{raw}} = (Z_i, \Delta_i, E_i^{\mathrm{raw}}), \qquad Z_i = \bigsqcup_{q \in Q_i} Z_i^{(q)}, \tag{2}$$

where the disjoint union preserves instruction-specific intent identities. An edge $(z_{ij}^{(q)}, \delta_{ik})$ is a candidate attribution, not a verified causal relation. Patch order is retained separately from edges.

3.3 Intervention-based Causal Auditing

We audit candidate attributions by executing subsets of the historical patch. Each intervention evaluates whether the operations attributed to an instruction provide executable support for reproducing the observed historical edit. Let $\mathcal{K}_i = \{1, \ldots, K_i\}$. For $S \subseteq \mathcal{K}_i$,

$$X_i^S = \mathrm{Apply}(X_i, \Delta_i, S) \tag{3}$$

replays the selected operations in their original order. Our auditing procedure is causality-inspired rather than a causal identification method. We use executable interventions within a fixed historical CAD program to assess the functional support of candidate intent–operation relations.

Sufficiency Check. For intent $z_{ij}^{(q)}$, define its aligned index set and the union for the complete instruction as

$$A_{ij}^{(q)} = \{k : (z_{ij}^{(q)}, \delta_{ik}) \in E_i^{\mathrm{raw}}\}, \qquad U_i^{(q)} = \bigcup_j A_{ij}^{(q)}. \tag{4}$$

Sufficiency is evaluated at the instruction level. We test whether the union of operations attributed to its atomic intents reproduces the historical target:

$$d_{\mathrm{geo}}\left(X_i^{U_i^{(q)}}, X_i'\right) \le \tau_{\mathrm{suff}}, \tag{5}$$

where d_{geo} is symmetric squared Chamfer distance. This instruction-level test measures executable sufficiency under the prescribed equivalence criterion. Insufficient instructions are excluded from usable memory.

Necessity Check. For each intent, we test whether attributed operations can be removed while preserving their executable effect. Let $A = A_{ij}^{(q)}$:

$$S_{\text{local}} \in \arg\min_{S \subseteq A} |S| \quad \text{s.t.} \quad X_i^S \simeq X_i^A, \tag{6}$$

where $\simeq$ denotes the applicable audit equivalence criterion. Local equivalence alone does not justify removal. We also test omitted operations in the full edit context and retain context-dependent or structurally required operations. Dependency safeguards may retain operations beyond the local minimum. Auditing removes attribution edges, not operations from the stored patch.

3.4 Multi-channel Retrieval

Each memory item $m_i = (X_i, Q_i^{\text{ret}}, G_i^{\text{audit}}, C_i)$ stores the source model, retained instructions, audited graph, and a historical Python edit script C_i, which is generated for the corresponding source–target pair and retained only if its execution on X_i reproduces X_i'.

Let $\mathcal{M}$ denote the candidate memory collection. Retrieval combines instruction embedding similarity, lexical overlap, CAD structure, and available operation metadata. Given input CAD X and instruction q, we select the top-ranked candidate using a weighted score:

$$m^* = \arg\max_{m_i \in \mathcal{M}} s(m_i \mid X, q). \tag{7}$$

Only eligible audited content is exposed to the generator. Retrieval uses no test target information.

3.5 Graph-Guided Dual-Output Generator

The graph indicates which historical patch operations are attributed to each intent, while the script supplies implementation patterns. The model is instructed to follow the test instruction, relocate paths and indices in the test CAD, and avoid copying unrelated historical edits.

A frozen language model jointly generates

$$(\hat{G}, \hat{C}) = f_\theta(X, q, m^*), \qquad \hat{G} = (\hat{Z}, \hat{\Delta}, \hat{E}), \tag{8}$$

with no parameter updates. The graph contains predicted intents, ordered patches, and alignment edges, and the Python script independently implements the edit on the input CAD JSON. Since the two branches are generated jointly but instantiated independently, their agreement is not guaranteed and is therefore explicitly verified through Graph–Script Consistency Validation.

3.6 Graph–Script Consistency Validation

The graph and script branches provide two executable realizations of the predicted edit. We first validate them independently. The generated graph $\hat{G}$ is checked for structural validity and valid patch paths, and is then replayed on the input CAD X, producing X_{graph}. The generated script $\hat{C}$ is executed in a restricted environment to produce X_{script}. Both results must preserve immutable fields and pass CAD reconstruction and geometric validity checks.

We then perform Graph–Script Consistency Validation by comparing X_{graph} and X_{script} in geometry and topology. The two branches are considered consistent only when they realize the same CAD modification under the prescribed equivalence criterion. If either branch fails executable validation or the two results are inconsistent, the frozen LLM regenerates the outputs.

A prediction $\hat{X}$ is accepted only when both branches are valid and their executed results are consistent. Otherwise, regeneration is triggered until the bounded repair budget is exhausted. Graph–Script Consistency Validation therefore focuses on agreement between the structural and executable realizations of the predicted edit.

4 Experiments

4.1 Experimental Setup

Datasets. For CAD-Editor (Yuan et al., 2025), we convert source–target programs into a canonical CAD JSON representation and regenerate compatible editing instructions. CAD-Editor is re-

evaluated on this adapted benchmark rather than using the results reported in the original paper. The evaluation set contains 2,000 editing instructions from the test split, while the Causally Audited Memory is constructed from 8,349 candidate edits drawn exclusively from the training split. For BenchCAD (Zhang et al., 2026a), we adapt the CodeEdit subset to the same canonical CAD JSON representation. The resulting benchmark contains 685 source–instruction–target triplets from 101 CAD families. We use five-fold family-disjoint evaluation, with retrieval memory and audit calibration restricted to the four non-test folds.

Implementation Details. We construct the historical intent–operation graphs with Gemma and reuse the same offline memory across generation backbones. Unless otherwise stated, AIM-CAD uses the top-ranked reference for each instruction, jointly generates an edit graph and a Python edit script, and allows up to three generation attempts. Experiments involving Gemma use Gemma-4-31B-Instruct (Gemma Team, 2026), with its 4-bit quantized version for local inference. Gemini-3.8 Flash (Doshi & Popa, 2026) and GPT-6 Astra (OpenAI, 2026b) use medium reasoning. All backbones share the same reference protocol and validation budget.

Evaluation Metrics. We evaluate CAD editing using seven metrics: Validity Rate (VR) for executability, Pairwise Chamfer Distance (CD) and HD95 for geometric fidelity, Jensen–Shannon Divergence (JSD) for distributional similarity, Sequence Similarity (SeqSim) for structural agreement, Unintended Change Rate (UCR) for preservation, and Directional CLIP similarity (D-CLIP) for instruction alignment. VR is the percentage of predictions that pass CAD reconstruction and geometric validity checks. HD95, a percentile-based Hausdorff measure (Taha & Hanbury, 2015), is computed here as the 95th percentile of pooled bidirectional nearest-neighbor surface distances, complementing CD by characterizing larger surface deviations. JSD measures the distribution difference between generated and target CAD models, D-CLIP evaluates whether the geometric change aligns with the instruction, and SeqSim is the normalized token-level Levenshtein similarity (Levenshtein, 1966) between predicted and target canonical CAD sequences. VR is computed over all instructions, whereas the remaining quality metrics are computed over each method's valid predictions and should therefore be interpreted jointly with VR. For Pairwise CD, unlike the original CAD-Editor protocol, we use the paired source–instruction–target correspondence:

$$\text{Pairwise CD} = \frac{1}{|V|} \sum_{i \in V} \text{CD}(\hat{X}_i, X'_i), \tag{9}$$

where V is the method-specific set of valid predictions and CD uses symmetric squared nearest-neighbor distances between surface point clouds. We measure unintended modifications using

$$\text{UCR} = \frac{\text{FP}}{\text{TP} + \text{FP}}, \tag{10}$$

where TP and FP denote correct and extra edits in direction-sensitive added/deleted voxel sets. Lower UCR indicates fewer modifications outside the ground-truth edit region. We additionally report No-Edit Rate, the percentage of valid predictions whose canonical CAD JSON is identical to the input, to identify cases where low geometric distance is achieved without performing the requested edit.

Baselines and Ablations. For end-to-end comparison, we use CAD-Editor and Direct Sequence Editing on the CAD-Editor-derived benchmark, and Direct CAD JSON Editing on BenchCAD. All direct-generation baselines use the same validation budget as AIM-CAD. For ablations, *w/o Reference* removes historical conditioning, *w/o Causal Auditing* uses the unaudited historical graph, and *w/ Random Ref.* and *w/ Top-3 Refs.* vary the retrieval strategy. The script-only ablations include *w/ Script-only Ref. + Script-only Output* and *w/ Full Ref. + Script-only Output*. The former removes the reference graph, while the latter retains the full graph–script reference. Both generate only a CAD edit script.

4.2 Main Results

4.2.1 Results on the CAD-Editor-derived Benchmark

AIM-CAD improves the reliability of language-guided CAD editing across all three backbones (Table 1). Compared with direct sequence editing, it consistently achieves higher VR and SeqSim and

Table 1: Across LLM backbones on the CAD-Editor-derived benchmark, AIM-CAD consistently improves sequence similarity and reduces unintended changes over direct editing; it also yields large validity gains with Gemini-3.8 Flash and GPT-6 Astra.

Method	VR (%) ↑	Pairwise CD ↓	HD95 ↓	JSD ↓	SeqSim ↑	UCR ↓	D-CLIP ↑	No-Edit Rate (%)
CAD-Editor	96.70	0.06118	0.17734	0.006004	0.7650	0.4810	-0.00659	0.21
Gemma-4-31B								
Direct Sequence Editing	79.05	**0.05189**	0.16813	0.006853	0.6962	0.4841	-0.01648	68.18
AIM-CAD	**81.30**	0.05883	**0.16372**	**0.005728**	**0.7995**	**0.3295**	**-0.00970**	0.00
Gemini-3.8 Flash								
Direct Sequence Editing	69.85	0.06078	**0.15967**	0.007661	0.7892	0.4070	-0.01463	3.72
AIM-CAD	**98.25**	**0.05379**	0.16048	**0.004722**	**0.7963**	**0.3757**	**-0.00683**	0.00
GPT-6 Astra								
Direct Sequence Editing	74.15	**0.05414**	**0.15660**	0.007073	0.7890	0.4491	-0.00808	1.24
AIM-CAD	**97.40**	0.05428	0.16334	**0.004882**	**0.7941**	**0.3751**	**-0.00417**	0.00

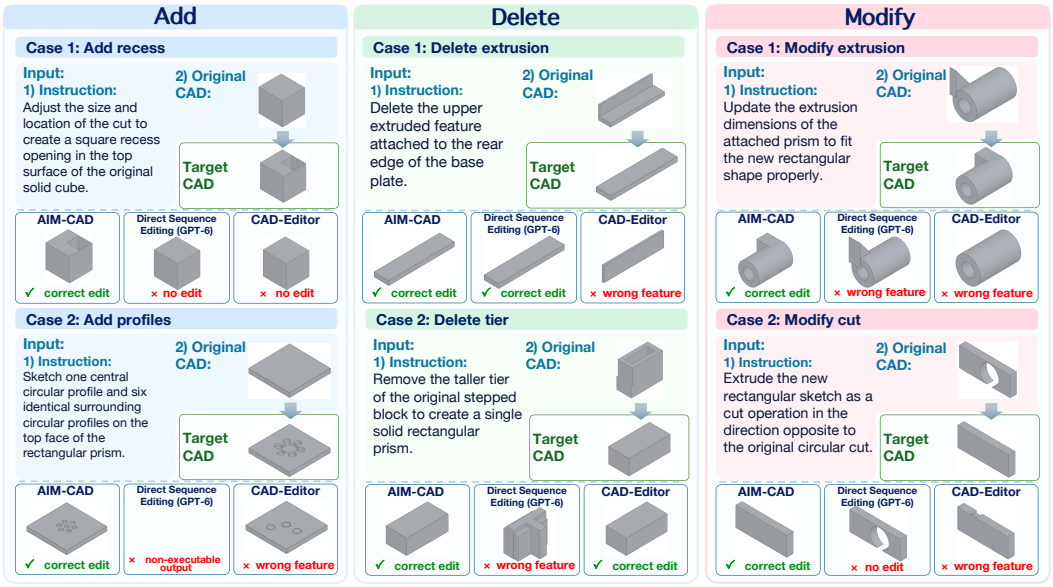

Figure 3: Representative results from the CAD-Editor-derived benchmark across the three edit categories defined in CAD-Editor: Add, Delete, and Modify. Two examples are shown for each category. AIM-CAD and Direct Sequence Editing use the same GPT-6 Astra backbone.

lower UCR. The largest VR gains occur with Gemini and GPT-6 Astra, increasing from 69.85% to 98.25% and from 74.15% to 97.40%, respectively. Among valid predictions, all three AIM-CAD backbones achieve lower CD, HD95, and UCR and higher SeqSim than the CAD-Editor baseline.

The backbones exhibit complementary strengths within AIM-CAD: Gemini achieves the best VR, CD, HD95, and JSD, Gemma achieves the best SeqSim and UCR, and GPT-6 Astra achieves the best D-CLIP. These results demonstrate that the framework supports both reliable execution and controlled editing across different language models.

Geometric metrics show a more mixed pattern. Direct Sequence Editing with Gemma achieves the lowest CD (0.05189), yet 68.18% of its valid predictions leave the input unchanged. Because source and target models can differ only locally, copying the source can yield a low global distance without completing the edit. Its weaker SeqSim, UCR, and D-CLIP further show why CD alone does not capture editing quality. Direct Sequence Editing with GPT-6 Astra retains lower CD and HD95, while the Gemini baseline has a slightly lower HD95. For both backbones, AIM-CAD achieves substantially higher VR alongside better JSD, SeqSim, UCR, and D-CLIP.

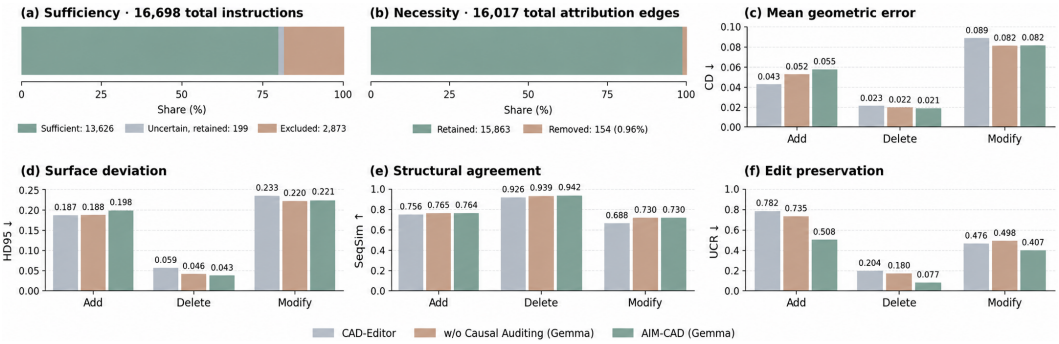

Figure 4: Historical-reference auditing and editing quality across edit types. (a) Sufficiency outcomes for 16,698 instructions from 8,349 historical edits. (b) Necessity pruning of attribution edges among the retained instructions. (c–f) CD, HD95, SeqSim, and UCR for CAD-Editor and the Gemma-based AIM-CAD variants across Add, Delete, and Modify. AIM-CAD achieves higher SeqSim and lower UCR than CAD-Editor in every category, and auditing consistently lowers UCR relative to the unaudited variant.

Figure 3 further illustrates these trends through representative examples from the three edit categories defined in CAD-Editor: Add, Delete, and Modify. We show two cases from each category. For a controlled comparison, AIM-CAD and Direct Sequence Editing both use GPT-6 Astra as the backbone. In the shown examples, AIM-CAD completes the requested edits, whereas the baselines exhibit failure modes including unchanged outputs, incorrect feature edits, and a non-executable prediction.

Overall, AIM-CAD's main advantage is reliable execution combined with stronger structural agreement and fewer unintended changes, rather than optimization of geometric proximity alone.

4.2.2 Ablation Study

The ablations show contributions from historical references, joint graph–script generation, and causal auditing (Table 2). References primarily support executable generation, while joint generation and auditing improve edit locality.

Removing references reduces VR from 81.30% to 12.55%, identifying historical conditioning as a major source of reliable execution. The no-reference variant's higher SeqSim and lower HD95 are computed from only 251 successful predictions, compared with 1,626 for AIM-CAD. This restricted coverage explains why its favorable conditional scores cannot establish stronger overall editing performance.

Removing graph output increases UCR from 0.3295 to 0.4623, while removing causal auditing increases it to 0.4507, with little change in VR or CD. These ablations support the contribution of joint generation and causal auditing to reducing unintended modifications.

Retrieval introduces a further trade-off. Random references degrade VR and all reported quality metrics, supporting relevance-based selection. Top-3 references improve VR, CD, HD95, SeqSim, and D-CLIP, while Top-1 achieves lower UCR and JSD. Additional examples may supply useful implementation patterns while increasing exposure to unrelated edits.

As shown in Figure 4, auditing refines the historical references and affects downstream editing performance. Sufficiency filtering removes 2,873 instructions, and necessity pruning removes 154 attribution edges without changing the stored patches. Across Add, Delete, and Modify edits, AIM-CAD achieves higher SeqSim and lower UCR than CAD-Editor. Compared with the unaudited variant, auditing consistently lowers UCR with little change in SeqSim. The largest reduction is observed for Add edits, from 0.735 to 0.508.

Table 2: Ablations on the CAD-Editor-derived benchmark. Removing historical references sharply lowers VR, while disabling causal auditing or generating only the script increases UCR. Top-3 references improve several metrics relative to Top-1 but yield higher UCR and JSD. Full Ref. combines the audited edit graph and historical CAD edit script.

Method	VR (%) ↑	Pairwise CD ↓	HD95 ↓	JSD ↓	SeqSim ↑	UCR ↓	D-CLIP ↑
AIM-CAD	81.30	0.05883	0.16372	**0.005728**	0.7995	**0.3295**	-0.00970
w/o Reference	12.55	0.05869	**0.13142**	0.030754	**0.8351**	0.3300	-0.02238
w/ Script-only Ref. + Script-only Output	82.05	0.06264	0.16584	0.006404	0.7928	0.4594	-0.01024
w/ Full Ref. + Script-only Output	82.05	0.06159	0.17239	0.006118	0.7961	0.4623	-0.01021
w/o Causal Auditing	81.55	0.05839	0.16252	0.006243	0.7987	0.4507	-0.00994
w/ Random Ref.	72.85	0.06979	0.17586	0.007442	0.7767	0.4442	-0.01283
w/ Top-3 Refs.	**82.45**	**0.05764**	0.16195	0.006217	0.8036	0.4426	**-0.00867**

Table 3: Generalization across datasets. Results on the BenchCAD editing subset show that AIM-CAD remains effective beyond the CAD-Editor-derived benchmark, outperforming direct CAD JSON editing across both backbones on validity and the principal editing-quality metrics.

Method	VR (%) ↑	Pairwise CD ↓	HD95 ↓	JSD ↓	SeqSim ↑	UCR ↓	D-CLIP ↑
AIM-CAD (Gemma-4-31B)	**88.91**	**0.01187**	**0.03755**	**0.007759**	**0.9175**	**0.1420**	**-0.00133**
Direct CAD JSON Editing (Gemma-4-31B)	69.20	0.01874	0.04834	0.010177	0.8899	0.1611	-0.00922
AIM-CAD (GPT-6 Astra)	**96.50**	0.00700	**0.02535**	**0.006560**	**0.9398**	**0.0765**	**-0.00037**
Direct CAD JSON Editing (GPT-6 Astra)	75.33	**0.00692**	0.03056	0.008432	0.9039	0.0877	-0.00198

Together, these findings support AIM-CAD's design: retrieve relevant experience, audit its executable attribution, and jointly generate the edit graph and script to improve execution reliability and preservation.

4.2.3 Generalization to BenchCAD

AIM-CAD also improves editing reliability and preservation on BenchCAD, demonstrating applicability beyond the CAD-Editor-derived representation (Table 3). With Gemma, VR increases from 69.20% to 88.91%, accompanied by improvements in all reported quality metrics. With GPT-6 Astra, VR increases from 75.33% to 96.50%, together with better HD95, JSD, SeqSim, UCR, and D-CLIP.

Although Direct CAD JSON Editing with GPT-6 Astra achieves a slightly lower CD of 0.00692 compared with 0.00700 for AIM-CAD, the difference is marginal, while AIM-CAD attains a substantially higher VR and better performance on the other quality metrics.

Overall, the results extend AIM-CAD's benefits to a different construction representation: more instructions yield valid CAD models, with improved structural agreement and fewer unintended changes among valid predictions.

5 Conclusion

We presented AIM-CAD, a training-free framework for language-guided parametric CAD editing based on Causally Audited Memory. The framework audits historical intent–operation graphs through executable interventions and reuses the resulting memory to guide joint graph–script generation with cross-representation consistency validation. Experiments on the adapted CAD-Editor and BenchCAD benchmarks show improved validity, structural agreement, and unintended-change control across multiple backbones. Ablations highlight the complementary roles of historical references, causal auditing, and joint graph–script generation in supporting executable and controlled editing. These results demonstrate the value of combining audited historical experience with executable validation for training-free CAD editing.

AI USE STATEMENT

We used generative AI in data preparation and CAD editing. For the adapted CAD-Editor data, a language model with visual input capabilities generated editing instructions from structured representations of source and target CAD models and rendered views, as well as executable Python functions implementing the corresponding edits. Instruction refinement included manual spot checks. Generated functions were executed on the source representations and retained only when their outputs matched the target representations, excluding the raw sequence field used only for provenance. This execution check verifies transformation consistency, not the semantic accuracy of every generated instruction. In AIM-CAD, language models generate CAD edits using retrieved historical memory, without task-specific parameter updates, as described in Section 3.

REFERENCES

Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. Generating CAD code with vision-language models for 3D designs. In *International Conference on Learning Representations*, 2025.

ByteDance Seed. Seed2.0 Model Card: Towards Intelligence Frontier for Real-World Complexity. *arXiv preprint arXiv:2607.00248*, 2026. doi: 10.48550/arXiv.2607.00248.

Tulsee Doshi and Raluca Ada Popa. Introducing Gemini 3.8 Flash and 3.8 Flash Cyber. Google Blog, 2026. URL https://blog.google/innovation-and-ai/models-and-research/gemini-models/3-8-flash-and-3-8-flash-cyber/. Accessed September 24, 2026.

Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. TransCAD: A hierarchical transformer for CAD sequence inference from point clouds. In *Computer Vision – ECCV 2024*, pp. 19–36. Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-73030-6_2.

Gemma Team. Gemma 4 Technical Report. *arXiv preprint arXiv:2607.02770*, 2026. doi: 10.48550/arXiv.2607.02770.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. In *Advances in Neural Information Processing Systems*, volume 37, pp. 7552–7579. Curran Associates, Inc., 2024. doi: 10.52202/079017-0242.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

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

Xueyang Li, Jiahao Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. Seek-CAD: A self-refined generative modeling for 3D parametric CAD using local inference via DeepSeek. In *The Fourteenth International Conference on Learning Representations*, 2026.

Vijay Lingam, Behrooz Omidvar Tehrani, Sujay Sanghavi, Gaurav Gupta, Sayan Ghosh, Linbo Liu, Jun Huan, and Anoop Deoras. Enhancing language model agents using diversity of thoughts. In *International Conference on Learning Representations*, 2025.

Weijian Ma, Shizhao Sun, Ruiyu Wang, and Jiang Bian. CADMorph: Geometry-driven parametric CAD editing via a plan-generate-verify loop. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine:

Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2019.

OpenAI. GPT-5.6 model guidance. OpenAI API documentation, 2026a. URL https://developers.openai.com/api/docs/guides/latest-model?model=gpt-5.6. Accessed September 24, 2026.

OpenAI. GPT-6 Astra model. OpenAI API documentation, 2026b. URL https://developers.openai.com/api/docs/models/gpt-6-astra. Accessed September 24, 2026.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Jianfeng Gao. SeCom: On memory construction and retrieval for personalized conversational agents. In *International Conference on Learning Representations*, 2025.

Toby Perrett, Matthew Bouchard, and William McCarthy. neuralCAD-Edit: An expert benchmark for multimodal-instructed 3D CAD model editing. *arXiv preprint arXiv:2604.16170*, 2026. doi: 10.48550/arXiv.2604.16170.

Shauli Ravfogel, Anej Svete, Vésteinn Snæbjarnarson, and Ryan Cotterell. Gumbel counterfactual generation from language models. In *International Conference on Learning Representations*, 2025.

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

Sorawit Saengkyongam, Elan Rosenfeld, Pradeep Ravikumar, Niklas Pfister, and Jonas Peters. Identifying representations for intervention extrapolation. In *International Conference on Learning Representations*, 2024.

Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021. doi: 10.1109/JPROC.2021.3058954.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377.

Abdel Aziz Taha and Allan Hanbury. Metrics for evaluating 3D medical image segmentation: Analysis, selection, and tool. *BMC Medical Imaging*, 15:29, 2015. doi: 10.1186/s12880-015-0068-x.

Aniket Vashishtha, Qirun Dai, Hongyuan Mei, Amit Sharma, Chenhao Tan, and Hao Peng. Executable counterfactuals: Improving LLMs' causal reasoning through code. In *International Conference on Learning Representations*, 2026.

Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. Text-to-CAD generation through infusing visual feedback in large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 65326–65345. PMLR, 2025.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations*, 2025.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Meng Yuan, Dawei Lin, Hongxia Xie, Tieru Wu, and Rui Ma. CAD-Refiner: A unified framework for CAD generation and iterative editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3244–3253, 2026.

Yu Yuan, Shizhao Sun, Qi Liu, and Jiang Bian. CAD-Editor: A locate-then-infill framework with automated training data synthesis for text-based CAD editing. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 73588–73603. PMLR, 2025.

Fred Zhang and Neel Nanda. Towards best practices of activation patching in language models: Metrics and methods. In *International Conference on Learning Representations*, 2024.

Haozhe Zhang, Kaichen Liu, Miaomiao Chen, Lei Li, Shaojie Yang, Cheng Peng, and Hanjie Chen. BenchCAD: A comprehensive, industry-standard benchmark for programmatic CAD. *arXiv preprint arXiv:2605.10865*, 2026a. doi: 10.48550/arXiv.2605.10865.

Zhenkui Zhang, Wendong Bu, Kaihang Pan, Bingchen Miao, Wenqiao Zhang, Guoming Wang, Wei Ji, Rui Tang, Juncheng Li, and Siliang Tang. Evolving generalist virtual agents with generative and associative memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(15): 13006–13014, 2026b. doi: 10.1609/aaai.v40i15.38300.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

A Implementation Details

Intervention-based Causal Auditing For each attributed operation set, we exhaustively enumerate all subsets, including the empty subset, and select the minimum-cardinality feasible subset under the equivalence criterion defined in the main text. Invalid or unreliable replays are excluded from candidate subsets. Ties are resolved by geometric residual and then by patch-index order. When replay dependencies, structural identity, or geometric comparison are unreliable, the corresponding attribution is conservatively retained. The intervention thresholds are calibrated on held-out data and fixed before evaluation.

Multi-channel Retrieval On the CAD-Editor-derived benchmark, retrieval combines semantic, lexical, CAD-structure, and operation-metadata similarities with weights $(0.50, 0.20, 0.25, 0.05)$, respectively. BenchCAD omits operation metadata and renormalizes the remaining weights. The highest-scoring eligible reference is used for generation.

Graph-Guided Dual-Output Generator Gemma experiments use Gemma-4-31B-Instruct with 4-bit quantization and no task-specific fine-tuning. Unless explicitly varied in an ablation, all AIM-CAD variants share the same retrieval configuration, auditing thresholds, and generation budget. Each sample is allowed at most three total generation attempts, including regeneration.

B Dataset Adaptation Details

CAD-Editor adaptation. We construct the CAD-Editor-derived benchmark from the paired source and target CAD models provided by CAD-Editor. Their CAD programs are converted into our canonical JSON representation, and duplicate or geometrically invalid source–target pairs are removed.

The original CAD-Editor instructions are not directly aligned with our structured editing representation. We therefore regenerate editing instructions using Doubao-Seed-2.0 (ByteDance Seed, 2026). For each source–target pair, the model is provided with rendered views of the source and target CAD models together with their corresponding pre-edit and post-edit CAD JSON representations. We generate both parameterized and non-parameterized descriptions of the underlying modification. A subset of the generated instructions is manually inspected to verify consistency with the corresponding CAD edit.

The resulting evaluation set contains 2,000 editing instructions. The historical candidate pool used by the retrieval pipeline contains 8,349 training edits. Because the instructions differ from those used in the original CAD-Editor evaluation, all CAD-Editor baselines reported in our experiments are re-evaluated using the same regenerated instructions rather than directly adopting numbers from the original paper.

BenchCAD adaptation. We additionally evaluate on the editing subset of BenchCAD, which provides paired source CAD programs, editing instructions, and target CAD programs. To obtain a representation compatible with the CAD-Editor-derived benchmark, we manually map and normalize CadQuery operations into their canonical JSON counterparts, using GPT-5.6 (OpenAI, 2026a) only as an auxiliary tool for interpreting program structure when necessary. The converted JSON programs are subsequently reconstructed into executable CAD models and manually inspected for geometric correctness, consistency with the annotated edit, and unintended modifications.

The final benchmark contains 685 source–instruction–target triplets from 101 CAD families. We perform five-fold family-disjoint evaluation. For each fold, the test families are excluded from both the retrieval memory and the calibration data used by the intervention-based auditing procedure; only examples from the remaining four folds are available during inference. This split prevents examples from the same CAD family from appearing simultaneously in the test set and its retrieval memory.

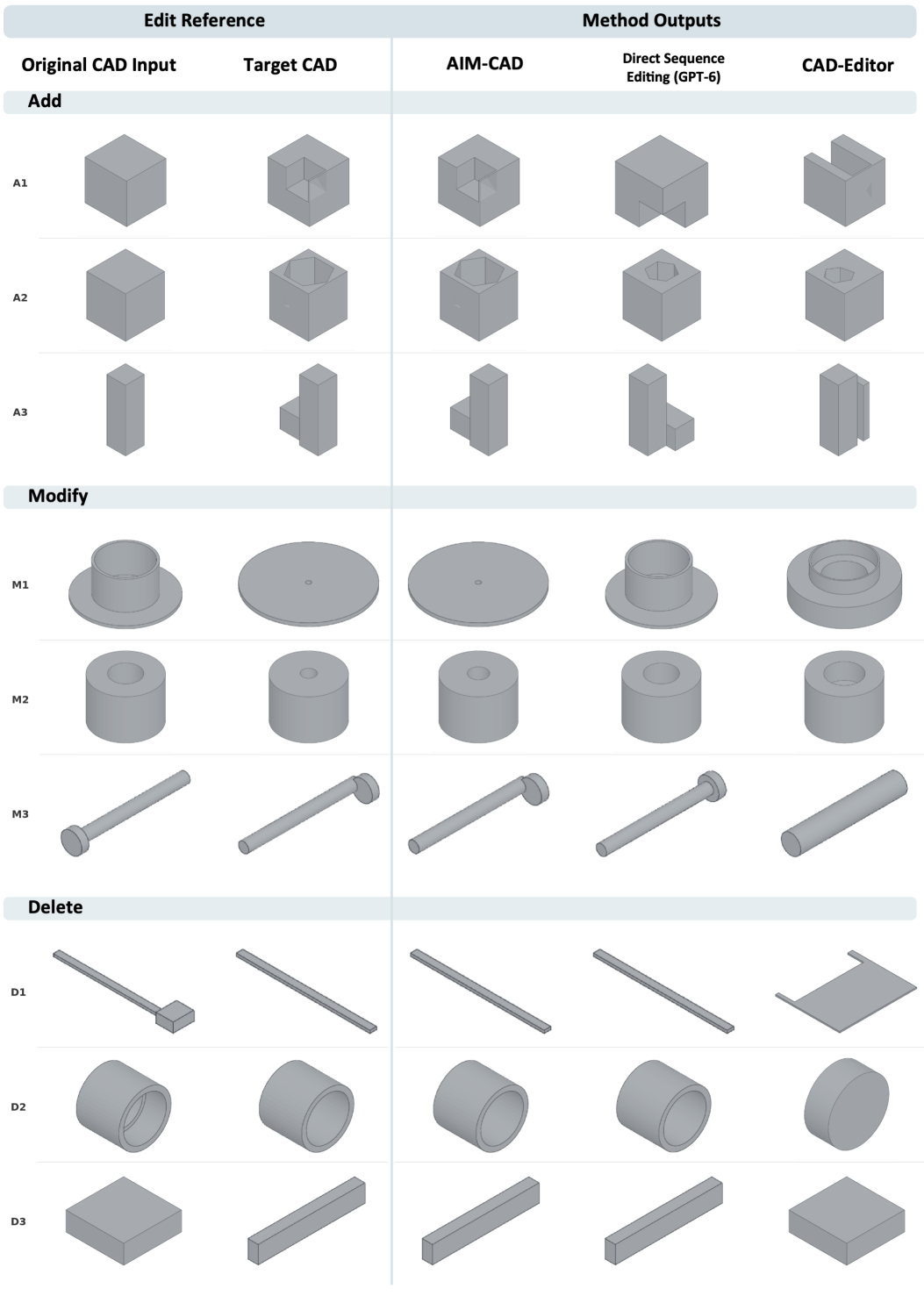

Figure 5: Additional qualitative results for the CAD editing benchmark. We provide more examples from the CAD-Editor-derived benchmark across the three edit action types. AIM-CAD and Direct Sequence Editing use the same GPT-6 Astra backbone.

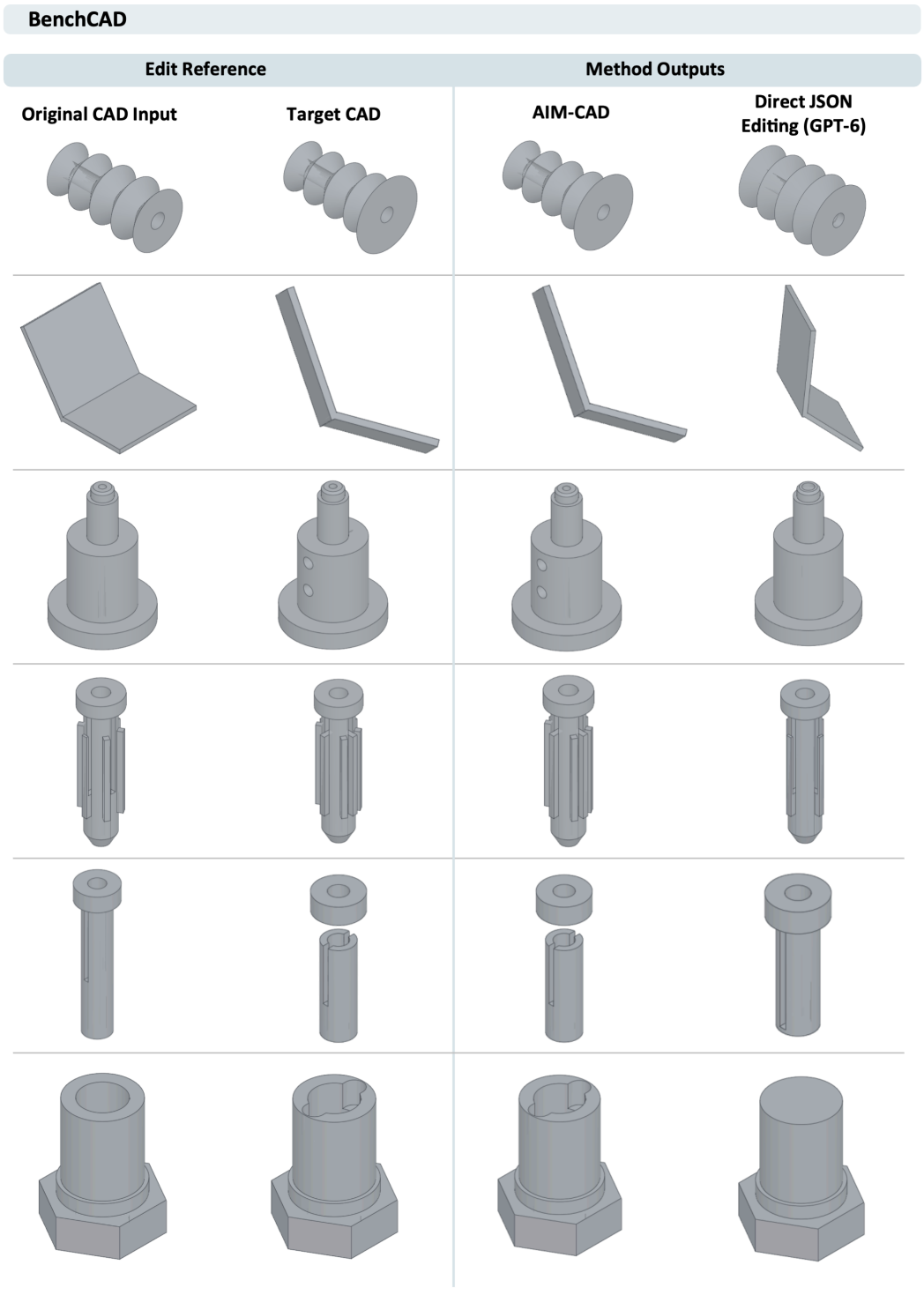

Figure 6: Representative qualitative results on the CodeEdit subset of BenchCAD. AIM-CAD and Direct CAD JSON Editing use the same GPT-6 Astra backbone.

C Evaluation Details

This section provides additional details on the evaluation, including metric interpretation, the difference between our Pairwise Chamfer Distance and the original CAD-Editor protocol, and the treatment of validity-conditioned quality metrics.

C.1 METRIC DETAILS

Chamfer distance. For each valid prediction, the predicted and target CAD models are represented by nonempty, independently normalized surface point sets. Following the CAD-Editor evaluation convention, we compute symmetric squared Chamfer distance as

$$\begin{aligned} \mathrm{CD}(P_{\mathrm{pred}}, P_{\mathrm{gt}}) = \frac{1}{|P_{\mathrm{pred}}|} \sum_{x \in P_{\mathrm{pred}}} \min_{y \in P_{\mathrm{gt}}} \|x - y\|_2^2 \\ + \frac{1}{|P_{\mathrm{gt}}|} \sum_{y \in P_{\mathrm{gt}}} \min_{x \in P_{\mathrm{pred}}} \|y - x\|_2^2. \end{aligned} \tag{11}$$

Lower values indicate closer global surface agreement.

HD95. We sample 10,000 points uniformly by surface area from each predicted and target mesh using deterministic seeds. Both point clouds undergo the same translation and scaling, defined by the union bounding box of the input and target models, with its longest side normalized to one. HD95 is the 95th percentile of the concatenated nearest-neighbor Euclidean distances in both directions. Distances are not squared, and the two directional percentiles are not computed separately. Lower values indicate smaller upper-tail surface deviations. We report the mean per-instance HD95 over each method's valid predictions.

Spatial distribution divergence. Following CAD-Editor, Jensen–Shannon Divergence (JSD) compares the spatial occupancy distributions of the generated and target shape collections. Unlike Pairwise CD, JSD is a collection-level metric and does not require instance-wise correspondence. Lower JSD indicates closer agreement between the generated and target shape distributions.

Instruction alignment. We adopt Directional CLIP similarity (D-CLIP) following CAD-Editor. D-CLIP compares the visual change from the input CAD to the predicted CAD with the textual direction specified by the editing instruction in the shared CLIP embedding space. It is used as a semantic alignment signal and should not be interpreted as an exact measure of CAD parameter correctness or executable validity.

Sequence similarity. SeqSim is computed using normalized token-level Levenshtein similarity on the dataset-specific canonical CAD representation, as defined in the Evaluation Metrics paragraph of the main text. Because CAD-Editor and BenchCAD use different canonical tokenizations, their absolute SeqSim values are not directly comparable across datasets.

Unintended Change Rate. UCR compares the predicted edit delta with the reference edit delta using direction-sensitive added and removed voxel sets in a shared coordinate frame. It measures the fraction of predicted changes that fall outside the ground-truth modification. When the predicted edit delta is empty, UCR is defined as zero. Therefore, a low UCR alone does not imply that the requested edit has been completed. We report No-Edit Rate separately to expose predictions that simply reproduce the input CAD.

Validity and No-Edit Rate. A prediction is considered valid only when it passes executable CAD reconstruction and geometric validity checks. Validity Rate (VR) is computed over the complete evaluation set. No-Edit Rate is computed over valid predictions and measures the fraction whose canonical CAD JSON representation exactly matches that of the input.

C.2 PAIRWISE CD VERSUS THE ORIGINAL CAD-EDITOR PROTOCOL

The original CAD-Editor evaluation computes Chamfer Distance by matching each generated CAD model to its nearest shape in a reference collection. Given generated models $\{\hat{Y}_i\}_{i=1}^{N}$ and a reference set $\{Y_j\}_{j=1}^{M}$, this protocol can be written as

Table 4: Geometric diagnostics for the Gemma-4-31B experiments on the CAD-Editor-derived benchmark, computed separately over each method's valid predictions. Mean Improvement is Input–GT CD minus Pred–GT CD; positive values indicate movement toward the paired target.

Variant	Input–GT CD	Pred–GT CD	Mean Improvement	No-Edit Rate
AIM-CAD	0.05211	0.05883	-0.00672	0.00%
Direct Sequence Editing (Gemma-4-31B)	0.05273	0.05189	+0.00085	68.18%
w/ Script-only Ref. + Script-only Output	0.05060	0.06264	-0.01204	0.00%
w/ Full Ref. + Script-only Output	0.05226	0.06159	-0.00934	0.00%
w/o Causal Auditing	0.05164	0.05839	-0.00675	0.00%
w/ Random Ref.	0.05357	0.06979	-0.01622	0.00%
w/ Top-3 Refs.	0.05060	0.05764	-0.00704	0.00%

$$\mathrm{CD}_{\mathrm{NN}} = \frac{1}{N}\sum_{i=1}^{N} \min_{j} \mathrm{CD}(\hat{Y}_i, Y_j). \tag{12}$$

This set-level formulation measures proximity to the reference distribution, but the selected reference need not be the target associated with the same editing instruction.

Our adapted benchmarks provide explicit source–instruction–target correspondence. We therefore evaluate each valid prediction directly against its paired target:

$$\text{Pairwise CD} = \frac{1}{|V|}\sum_{i \in V} \mathrm{CD}(\hat{Y}_i, Y_i), \tag{13}$$

where V denotes the method-specific set of valid predictions and Y_i is the target associated with the same editing instruction. Pairwise CD therefore measures instance-level target proximity, whereas JSD provides a complementary distribution-level comparison.

C.3 Interpreting Pairwise CD

Global geometric distance can be misleading for localized CAD editing. When the source and target differ only in a small region, an unchanged source model may remain geometrically close to the target despite failing to execute the requested edit. To examine this effect, Table 4 compares the distance from the input to the target with the distance from the prediction to the target.

Direct Sequence Editing obtains the lowest Pairwise CD among the Gemma variants, but 68.18% of its valid predictions exactly reproduce the input CAD. Thus, its favorable global distance frequently reflects geometric proximity of the unchanged source to the target rather than completion of the requested edit.

At the same time, AIM-CAD does not consistently reduce global Pairwise CD relative to the input: its mean improvement in Table 4 is negative. This indicates that global surface distance alone does not fully capture localized editing quality. A prediction may alter the requested local structure while increasing global surface discrepancy, whereas an unchanged prediction may retain a small global distance. We therefore interpret Pairwise CD jointly with HD95, SeqSim, UCR, D-CLIP, Validity Rate, and No-Edit Rate rather than as a standalone measure of editing success.

C.4 Validity-Conditioned Evaluation

VR is computed over the complete evaluation set, whereas Pairwise CD, HD95, JSD, SeqSim, UCR, and D-CLIP are computed only over each method's valid predictions. The latter metrics are therefore conditioned on different method-specific subsets and should always be interpreted together with VR.

For example, on the CAD-Editor-derived benchmark, the Gemma w/o Reference variant has only 251 valid predictions, compared with 1,626 for AIM-CAD. Its conditional quality scores therefore characterize a much smaller successful subset and do not by themselves imply better overall editing

performance. Likewise, Direct Sequence Editing with Gemma illustrates that a favorable success-conditioned Pairwise CD can coexist with a high No-Edit Rate.

C.5 Baseline and Ablation Details

End-to-end baselines. Direct Sequence Editing predicts the edited CAD sequence from the input sequence and instruction without historical references, edit scripts, or graphs. On BenchCAD, Direct CAD JSON Editing directly predicts the edited canonical CAD JSON. Both baselines use CAD validation and the same generation-attempt budget as AIM-CAD.

Ablation variants. The *w/o Reference* variant removes historical conditioning while retaining joint graph–script generation. The *w/ Script-only Ref.* + *Script-only Output* variant removes the reference graph and generates only a script, while *w/ Full Ref.* + *Script-only Output* retains the full graph–script reference but omits the generated graph. The *w/o Causal Auditing* variant uses the historical edit graph without Intervention-based Causal Auditing. The *w/ Random Ref.* variant replaces ranked retrieval with a random reference, while *w/ Top-3 Refs.* provides up to three ranked references. Except for *w/o Reference*, all variants retain the source CAD and instruction associated with their references.