TOPOLOGY DIVERGES BEFORE FAILURE: FINDING THE RIGHT REPAIR POINT IN CAD PROGRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Generated CAD programs can remain executable even after their committed construction structure diverges from the intended design, making execution feedback alone insufficient to determine where repair should begin. We introduce TopoRecover, a topology-aware framework that treats repair localization as a first-class problem before patch generation. TopoRecover replays CAD command sequences with a deterministic state-machine interpreter that records completed profiles and their source commands. By comparing these profiles with a fixed feature plan specifying the required construction structure, it maps structural mismatches to the command blocks that produced them. A Large Language Model (LLM) proposes geometry-constrained local edits, followed by downstream execution repair, while preserving compatible construction history. Repaired programs are accepted only when verifier replay satisfies the plan and independent OpenCASCADE execution produces a valid positive-volume solid. On 347 natural Text2CAD repair inputs, TopoRecover achieves 78.4% plan-conditioned repair success, compared with 64.3% for the strongest adapted baseline, and obtains the highest recovered-history score. In a matched localization study holding the patcher, fallback, validation, and budget limits fixed, topology-based localization improves repair success from 47.6% to 78.4%, with gains concentrated on executable programs that violate their plans. These results identify localization as a key bottleneck in plan-conditioned CAD repair: targeting earlier plan-inconsistent constructions improves repair success without changing the patching procedure. [1]

1 INTRODUCTION

Computer-Aided Design (CAD) plays a crucial role in modern engineering and manufacturing, translating design intent into precise, editable models that support product development and fabrication (Shah & Mäntylä, 1995). CAD programs construct geometry through a sequence of commands: sketches define closed planar regions, or profiles, which later feature operations use to build solids (Zhang et al., 2025). Unlike static shape representations, these construction histories retain editable parameters and dependencies between modeling operations, enabling localized modification and reuse (Yuan et al., 2025). Because later operations depend on geometry created earlier, a single structural error can affect subsequent operations. Recent models generate these programs from text and point clouds, or support multiple input modalities (Khan et al., 2024; Rukhovich et al., 2025; Kolodiazhnyi et al., 2025). As CAD generation becomes more capable, reliable repair becomes essential for retaining both executable geometry and the requested design.

Generated CAD programs may contain invalid operations, incorrect feature references, or geometry that does not match the requested design (Fan et al., 2026a). Recent work addresses these problems through visual verification, program review, and iterative editing (Alrashedy et al., 2025; Chen et al., 2025; Yuan et al., 2026; Wang et al., 2025). Tool-augmented systems further inspect the evolving CAD model and use geometric feedback to guide subsequent actions (Mallis et al., 2025). For sequential CAD programs, a central question is how to determine which commands need revision while retaining the construction that is already correct.

Execution feedback identifies a rejected operation, while visual feedback reveals discrepancies in the resulting shape (Alrashedy et al., 2025; Chen et al., 2025). These signals alone do not establish

[1] The codebase and dataset (anonymous repositories) are available at GitHub.

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.

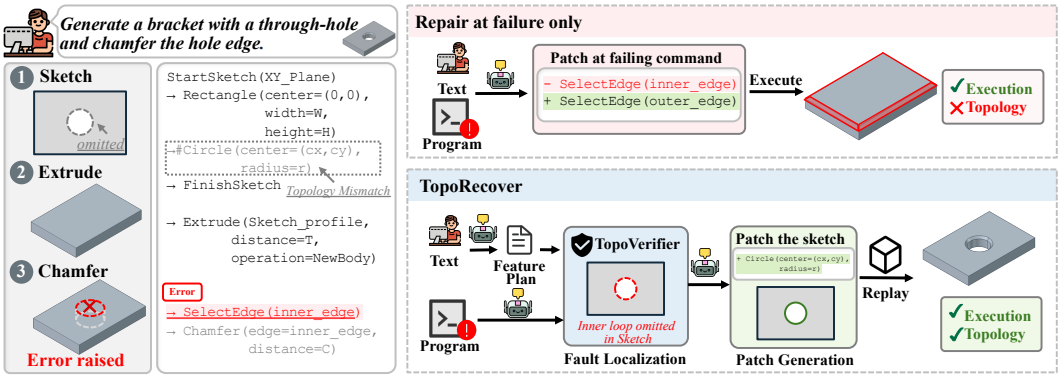

Figure 1: Motivating example: the hole is lost before anything fails.

which earlier accepted construction must be changed. In Figure 1, a sketch with an omitted hole still extrudes successfully, but a later chamfer fails when it requests the missing hole edge. The failure-local repair changes the selected edge to an existing outer edge, removing the execution error without restoring the hole. Even when a review model recognizes the missing hole, it must still connect this discrepancy to the earlier sketch-building commands that executed successfully. Repairing only the rejected operation starts too late, while regenerating the entire sequence can unnecessarily discard valid parameters, references, and construction history. This motivates us to localize where the construction first diverges from the required topology before generating a patch.

We introduce TopoRecover, a framework that localizes and repairs errors in generated CAD command sequences. Its second input is a feature plan, a structured description of the required sketch profiles and the extrusions that use them, fixed before repair begins. TopoVerifier interprets the sequence as a state machine: each command validates and updates the construction state, while completed profiles and their source commands are recorded for subsequent comparison. Comparing these saved profiles with the plan reveals missing or surplus boundaries; their recorded source locations identify the command blocks to edit. In Figure 1, this comparison identifies the sketch-building block, where inserting the missing circle restores the hole without changing the outer boundary. An LLM proposes local changes using the feature plan, observed geometry, and any supplied dimensions, after which execution repair resolves remaining references and unfinished operations. Full command replay, plan checks, and independent CAD execution determine whether the repaired program can be returned.

We evaluate TopoRecover on our repair dataset processed from Text2CAD (Khan et al., 2024), including both execution failures and executable programs that violate their plans. TopoRecover achieves 78.4% plan-conditioned repair success, compared with 64.3% for the strongest adapted repair baseline, and the highest failure-aware command-preservation score. A matched localization experiment holds the patcher, fallback, validation, and budget limits fixed: selecting plan-inconsistent profile blocks raises repair success from 47.6% to 78.4% over runtime-based selectors. The additional repairs occur on executable, plan-inconsistent programs, showing why an execution error alone is an insufficient localization signal. A separate geometry study reports the highest point estimates for failure-aware volume IoU and solid-topology agreement, although the small cohort does not establish a statistically significant IoU advantage.

The key contributions of this paper are as follows:

- We identify delayed topology divergence as a key failure mode in CAD repair and localize the earliest committed transaction that becomes inconsistent with the target structure.
- We develop TopoRecover to reopen the localized transaction, reconcile the intended topology, and complete the downstream history while preserving unaffected construction and validating repairs through verifier replay and independent OpenCASCADE execution.
- We evaluate TopoRecover on our repair dataset, controlled localization benchmarks, held-out generators, unseen structure families, and additional feature operators, showing consistent gains in repair success and history retention and confirming that incorrect localization remains a bottleneck even with strong downstream completion.

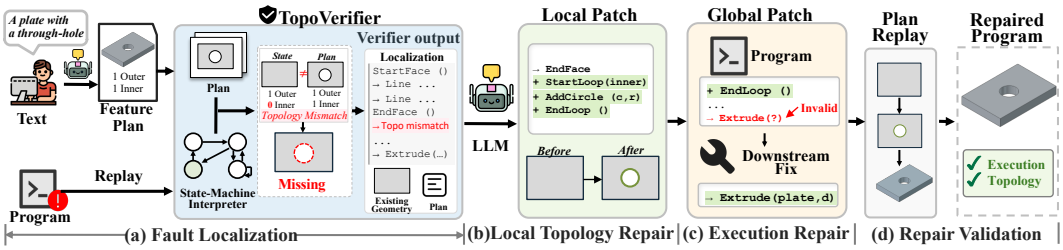

Figure 2: An overview of our proposed TopoRecover.

2 Problem Formulation

CAD program. Let $S = (a_1, \ldots, a_T)$ be a sequence of T CAD commands, each specifying an operation and its arguments. This command sequence is the program to repair; it also records how the model was constructed. A sketch is a two-dimensional drawing, and a loop is a closed curve within that drawing. A profile is the planar region enclosed by an outer loop, with any inner loops defining holes. In this representation, `StartFace` begins a profile under construction, `EndFace` completes it, and `RegisterProfile` saves it for later use. For example, a rectangular loop and a circular inner loop define the cross-section of a plate with a through-hole. Saving the profile gives it an identifier, which an extrusion command uses to select the region to extend into a solid.

Feature plan. The second input, the feature plan F, is a structured list of intended profiles and extrusions, held fixed during repair. A profile entry gives its identifier and required enclosing and hole boundaries; an extrusion entry gives its profile reference, operation (add, cut, or intersect), and any supplied depth. For the plate example, these structural requirements are one profile with one hole and an additive extrusion to depth d. They do not prescribe the drawing commands or fully determine geometry. Shape and dimension constraints, when available, come from the accompanying design brief or explicit geometry fields; unspecified parameters remain choices for repair. In the natural-generation study, a planning LLM produces F from the brief before Text2CAD generates S; F remains fixed during repair. The separate geometry study uses reference-derived structural requirements, but neither setting exposes target commands or reference coordinates (Appendix C.1).

Repair task. Given S and F, the output is a repaired command sequence S' that passes command verification, executes to a valid positive-volume solid, and satisfies the supported input requirements. Repair is needed when execution fails, when the constructed features disagree with the plan, or both. The repair should retain geometry, parameters, and commands that do not need to change. A case remains unresolved if no accepted output is found within the repair budget. Table 11 specifies the supported operators and parameters.

3 Topology-Aware Repair

3.1 Overview

TopoRecover separates deciding where to repair from generating the repair itself. A saved profile can pass local validity checks while already deviating from the intended design; a later execution failure need not identify this earlier discrepancy. Our key idea is to compare saved profiles with the feature plan and trace each mismatch to the commands that produced it. Figure 2 follows this principle: TopoVerifier uses a state-machine interpreter with construction and source records to identify the mismatched profile blocks. The LLM then proposes geometrically constrained edits within these blocks, turning a structural diagnosis into a local program change. Execution repair then completes any remaining operations, and validation checks both agreement with the plan and independent CAD execution.

3.2 FAULT LOCALIZATION

TopoVerifier is a deterministic state-machine interpreter augmented with construction and source records. It interprets CAD commands while retaining both what they construct and where that construction originates in the sequence. These records connect an earlier saved profile to a repairable command block, even when the block executes successfully and the error becomes visible only later.

State-machine interpretation. The interpreter reads S one command at a time, maintaining the state $s_t = (q_t, G_t, P_t)$ after each accepted command.

The control context q_t tracks open sketches, profiles, and loops, including whether a completed profile awaits registration. The construction record G_t stores curve endpoints, completed boundaries, saved profiles, and extrusion references; the source record P_t tracks opening indices and associates saved profiles with their source intervals. Starting from s_0 with empty records and no open construction, each accepted command induces a transition

$$s_t = \delta(s_{t-1}, a_t, t), \tag{1}$$

where δ checks the command's context, geometry, and references before updating the records, and t supplies its source index. For example, `AddLine` checks connectivity and self-intersection before extending the active curve, and `EndLoop` checks closure before storing its boundary. Figure 3 shows the sketch-building control phases; they are only one part of the interpreter state. Drawing a curve updates G_t even when the control phase remains unchanged. If a precondition fails, replay stops and returns the rejected command and reason together with the last accepted state.

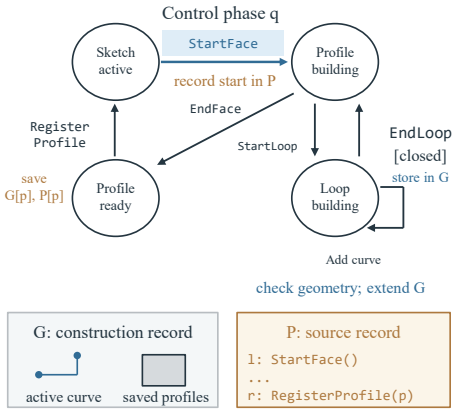

Figure 3: Sketch-building phases and record updates. Commands check preconditions and update construction record G and source record P, which links saved profiles to their source intervals.

Linking saved structure to source commands. The interpreter's source record connects each saved profile to the commands that constructed it. An accepted `StartFace` records the opening index ℓ_p; after `EndFace` completes the profile, `RegisterProfile` saves it in G at index r_p. The interpreter thus retains both its source interval and its observed boundary counts:

$$P_{r_p}[p] = [\ell_p, r_p], \qquad \phi(p) = \big(n_{\text{outer}}(p), n_{\text{inner}}(p)\big), \tag{2}$$

where n_{outer} and n_{inner} count the enclosing and hole boundaries stored in G. The interval includes both endpoints, from `StartFace` through `RegisterProfile`. This source association is recorded as commands are interpreted, rather than inferred from a later execution error. In the supported language, downstream commands can reference a saved profile but cannot alter its boundaries, so changing that structure requires editing its earlier construction block.

Localizing a disagreement with the plan. The interpreter checks whether each construction step is legal; comparing its saved profiles with the plan determines whether the requested structure was correctly built. Let k be the last accepted command index and F_p the plan entry corresponding to saved profile p. We then compare its observed boundary counts with F_p and retrieve the exact source interval of each mismatch:

$$\mathcal{R}(S, F) = \{P_k[p] \mid p \text{ is saved in } G_k,\ \phi(p) \not\models F_p\}\,. \tag{3}$$

Here $\not\models$ denotes a difference in the required outer or inner boundary count; a saved profile without a plan entry is treated as surplus. Correspondence pairs profiles sorted by identifier with the ordered plan entries, assuming that this ordering is preserved. Only saved profiles are compared, avoiding premature diagnosis of a boundary still being drawn. Figure 2(a) shows the resulting inference: a missing hole in a saved profile selects its earlier source block, even if every command in that block was accepted. Thus, $\mathcal{R}$ specifies where the intended structure must be restored, while any rejected command separately identifies the remaining execution failure. The same localization applies when execution completes with incorrect topology.

A running example. In Figure 2, the generated program constructs a plate whose base region should contain both an outer boundary and an inner boundary representing a hole. The generated commands omit the inner boundary, but the resulting region is still locally valid and is successfully registered. Replay therefore does not report an execution error at that point. However, the committed structure disagrees with the target specification, which requires the additional inner boundary. Because TopoVerifier has retained the source commands that produced this region, the mismatch is traced back to that earlier construction block rather than to the later downstream error. This block is then exposed as the repair region for the next stage.

3.3 Local Topology Repair

Localization turns a plan discrepancy into a scoped generation problem: realize the required structural change within an identified profile block. For each mismatched profile p, the observed and required boundary counts determine how many inner boundaries must be added or removed. The existing outer boundary is preserved.

A frozen LLM proposes a structured local edit from the localized commands, observed boundaries, plan requirement, and available public context C_p:

$$\Delta_p = \mathrm{LLM}\big(S_{[\ell_p, r_p]},\, \phi(p),\, F_p,\, C_p\big),$$

where C_p contains any supplied design brief and geometric constraints. The model selects inner boundaries to retain or remove and proposes parameters for new circles or rectangles. These decisions are converted into command insertions and deletions within the selected block; retained curves, profile identifiers, and unrelated commands remain unchanged. New boundaries must fit inside the outer boundary, avoid retained holes, and satisfy supplied dimensions (Appendix B.1). We merge the proposed edits into the original program and replay the modified prefix to verify structural consistency, rejecting any out-of-scope modifications, invalid geometry or commands, and unresolved profile mismatches. Once a profile block is accepted, it is treated as structurally resolved and remains fixed throughout the subsequent execution-repair stage.

In Figure 2, the localized block constructs a plate without its required hole. The LLM proposes the missing circular boundary, which is inserted before `EndFace` while preserving the existing outline.

3.4 Execution Repair

After the diagnosed profiles have been corrected, downstream commands may still be incomplete or use invalid references. We replay the updated program to repair these remaining execution errors while keeping the accepted profile blocks fixed. A bounded compiler enumerates typed completions around the active construction or rejected command. These completions can rebind an invalid reference, close an unfinished construction, register a completed region, or add a required extrusion. Existing parameters are reused whenever possible. Among admissible candidates, we prefer later repair boundaries and fewer command changes, which preserves compatible downstream construction within the search budget.

3.5 Repair Validation

We return a repaired program only after full replay confirms that all commands terminate and the resulting construction satisfies the target structural and supported parameter requirements. The program is then executed independently through CadQuery/OpenCASCADE and must produce a valid positive-volume B-rep. This additional execution check prevents us from accepting a structurally consistent program that is invalid in the CAD kernel, while the plan check prevents us from accepting a valid solid with the wrong intended structure. If the localized repair path fails or exhausts its budget, we attempt one bounded compiler fallback from the original input under the same validation criteria. Otherwise, the case remains unresolved rather than relaxing the target requirements.

Table 1: End-to-end repair, geometric accuracy, and solid-topology comparison. Bold and underline indicate the best and second-best values, respectively; * denotes adapted systems.

Method	Plan-conditioned repair					Geometry				Solid topology
	Rep.↑	Exe.↑	PM.↑	His.↑	Rec.↑	n_g	CD ↓	HD ↓	IoU ↑	$T^{\text{all}}_{\text{corr}}$ ↑
CADReview/ReCAD*	41.2	71.5	43.8	0.657	0.271	17	0.122	0.427	<u>0.207</u>	<u>54.5</u>
CADCodeVerify*	<u>64.3</u>	**81.8**	<u>68.0</u>	<u>0.668</u>	<u>0.429</u>	14	0.119	0.415	0.157	31.8
CADDesigner*	57.6	<u>78.7</u>	62.0	0.651	0.375	15	**0.111**	**0.370**	0.190	50.0
LLM generation from plan	11.8	11.8	11.8	0.422	0.050	6	4.986	17.155	1.29×10^{-5}	27.3
LLM feedback regeneration	23.1	23.1	23.1	**0.724**	0.167	9	0.197	0.744	0.049	31.8
TopoRecover	**78.4**	78.4	**78.4**	0.655	**0.513**	22	<u>0.114</u>	<u>0.411</u>	**0.270**	**90.9**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets. Our benchmark pairs selected Text2CAD-generated sequences (Khan et al., 2024) with frozen LLM-generated feature plans; a source-disjoint cohort provides geometry references. Appendix C gives selection procedures and sample counts; Appendix D documents plan construction, verification, and supplementary reference-derived-plan results.

Baselines. CADReview/ReCAD (Chen et al., 2025) supplies an editor with diagnoses of erroneous blocks. CADCodeVerify (Alrashedy et al., 2025) iteratively refines programs through binary requirement checks. CADDesigner (Fan et al., 2026b) rewrites programs using the brief, plan, and execution feedback. LLM generation from plan generates programs from the brief and frozen plan in one round. LLM feedback regeneration additionally uses the failed program and public execution and requirement feedback for up to two rounds. The adaptations use Qwen3-30B-A3B; regeneration shares TopoRecover's frozen Qwen2.5-7B-Instruct, without localized repair regions. The comparison is not compute-matched (Appendices B.3 and E).

Metrics. Repair success (Rep.), our primary metric, requires verifier termination, plan-topology agreement, and a valid positive-volume solid. Execution validity (Exe.) checks only the last condition using OpenCASCADE (OCC), not repair success. Plan match (PM.) measures agreement with the required construction topology. History (His.) averages exact-command preservation over successful repairs; recovered history (Rec.) gives failures zero (Appendix C.2). Geometry uses normalized surface Chamfer distance (CD), Hausdorff distance (HD), and volumetric intersection over union (IoU). CD and HD use the n_g measurable outputs; IoU retains all 22 executable references, assigning failures zero. Solid-topology agreement $T^{\text{all}}_{\text{corr}}$ compares boundary-mesh Euler characteristics on the same 22 references, counting unmeasurable outputs as incorrect; it is a topology proxy, not an equivalence test. Appendix C.3 reports conditional Euler error and common-subset agreement; paired intervals use 10,000 program-level bootstrap samples.

4.2 MAIN RESULTS

Table 1 compares repair, preservation, and reference-based geometry and topology.

Repair and preservation. To assess repair, we validate outputs against the frozen plan and execute them independently in OpenCASCADE. Table 1 shows **78.4% repair success for TopoRecover**, exceeding the strongest adapted baseline by 14.1 points (paired 95% CI: 8.6–19.6). Furthermore, its **highest recovered-history score (0.513)** shows greater construction retention across the cohort. Feedback regeneration preserves more history among successful outputs, but its lower coverage yields only 0.167 recovered history. Figure 4 explains the distinction from execution validity: CADCodeVerify reaches 81.8% executability but 64.3% repair, leaving 61 executable outputs below the joint criterion. In contrast, TopoRecover accepts only jointly validated candidates, giving 78.4% on both measures: fewer executable outputs, but more successful repairs.

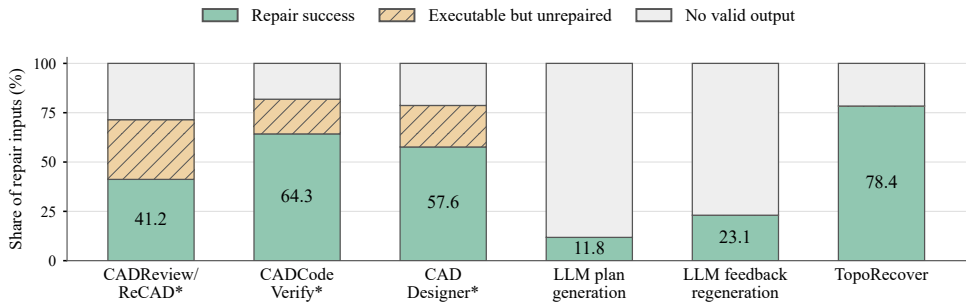

Figure 4: Outcomes on all 347 repair inputs. Executable but unrepaired outputs pass the CAD kernel without meeting the joint repair criterion.

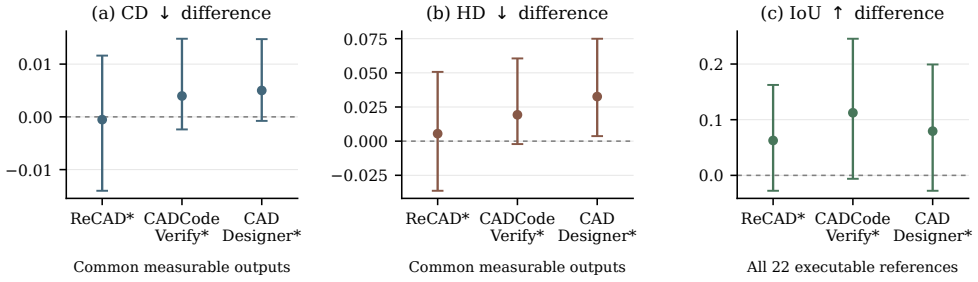

Figure 5: Paired geometric differences between TopoRecover and adapted baselines, with 95% bootstrap intervals. IoU includes output failures.

To compare localized repair with regeneration, we fix the 7B backbone, brief, and plan. Feedback raises generation success from 11.8% to 23.1%, still below **TopoRecover's 78.4%** (Table 1). Together with recovered history, this supports preserving existing construction; Section 4.3 isolates region selection under fixed budgets.

Geometry and solid topology. To assess fidelity, we evaluate selected outputs on the separate reference-geometry cohort. Table 1 shows the **highest failure-aware IoU (0.270) and Euler agreement (90.9%)** for TopoRecover, with measurable outputs for all 22 valid references. Furthermore, its conditional CD and HD rank second, behind CADDesigner. However, the paired IoU intervals in Figure 5 include zero: coverage and topology agreement are strong, but geometric superiority remains unestablished on this small cohort.

4.3 Ablation Studies

We isolate two key decisions concerning where edits are permitted in the repair process and how the verified fallback mechanism responds when LLM-generated proposals are rejected.

Where to repair. To isolate localization, we vary only the region selector, fixing the patcher, validation, fallback, and budgets (Appendix B.2). Figure 6 shows that **topology-based regions raise repair from 47.6% to 78.4%** over runtime selectors; the paired gain over runtime dependency is 30.8 points (95% CI: 25.9–35.7). Furthermore, recovered history rises from 0.228 to 0.513. Panel (b) attributes the gain to executable, plan-inconsistent inputs; selectors tie on runtime-rejected inputs after fallback, leaving an advantage on delayed runtime failures unestablished.

How to handle rejected proposals. To assess fallback, we compare the unchanged LLM path, bounded compiler, and fixed-order combination with localization fixed. Figure 7(a) shows that **fallback raises repair from 64.0% to 78.4%**, adding 50 successes without losing an accepted LLM repair. Additionally, the combined method has the highest recovered history (panel b) and improves failure-aware IoU and Euler agreement over the LLM path (panels c–d). The compiler alone has slightly higher IoU but lower repair coverage, so the combination does not dominate every

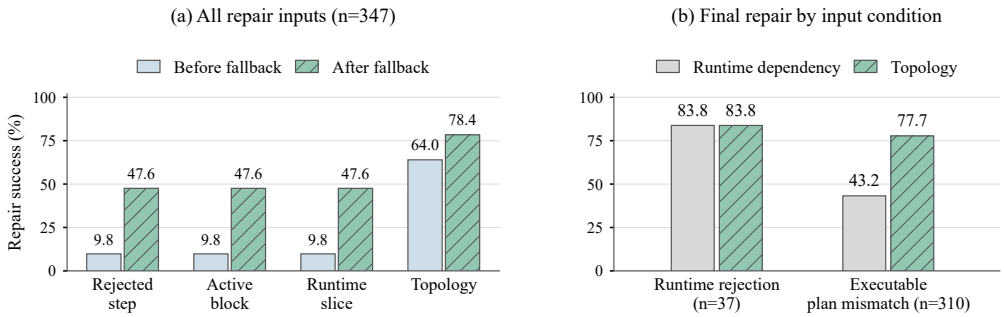

Figure 6: Matched localization with the deployed patcher and identical fallback. Gains concentrate on executable programs that violate the plan.

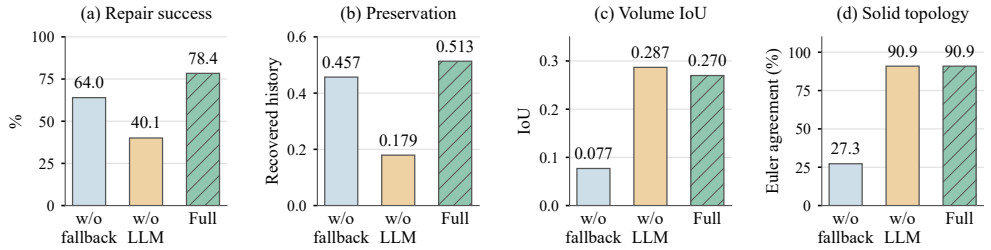

Figure 7: Component ablations. Repair and preservation use 347 cases; IoU and failure-aware Euler agreement use 22 valid references.

metric. This trade-off indicates that geometric accuracy on recovered cases and the ability to repair a broader set of failures reflect distinct aspects of recovery quality. Together, the ablations show complementary benefits: localization identifies necessary edits, while fallback recovers cases left unresolved by the LLM.

4.4 Case Study

To illustrate localized repair, Figure 8 compares a missing-hole case and its repairs at a common viewpoint and scale. CADCodeVerify leaves the executable cylinder unchanged; ReCAD adds a hole but makes the body rectangular, whereas CADDesigner and TopoRecover restore the hole while preserving the outer dimensions. The aligned program excerpts explain the correction: TopoVerifier traces the missing hole to lines 2–7.

The LLM inserts three loop-construction commands before `EndFace`, retaining all ten original commands, including the outer circle and extrusion. Thus, **an earlier sketch edit corrects topology without rebuilding the body**; Appendix G provides further program, baseline, and reference-geometry comparisons.

5 Related Work

Traditional CAD Modeling. Parametric CAD represents 3D shapes through editable construction histories composed of sketches, profiles, feature references, and geometric operations (Willis et al., 2021; Wu et al., 2021; Xu et al., 2022). Classical CAD research studies feature dependencies, geometric consistency, and history repair (Yang & Han, 2006), while recent neural approaches reconstruct or generate CAD representations and boundary models (Liu et al., 2024; Xu et al., 2024; Li et al., 2025b;a). These methods focus on constructing or recovering valid CAD representations. TopoRecover instead studies repair of generated command histories: it identifies construction blocks whose committed states disagree with the intended design and edits those blocks while preserving compatible history.

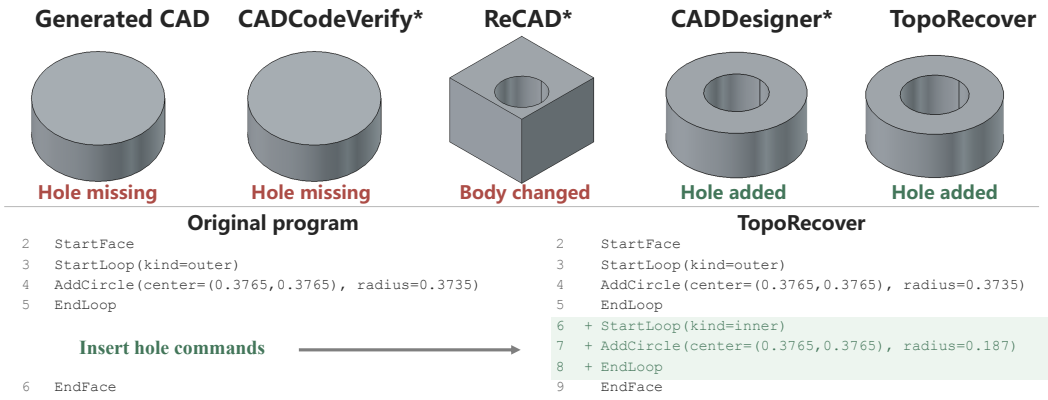

Figure 8: Missing-hole repair results with CAD outputs and corresponding program excerpts.

LLM-based CAD Modeling and Parametric CAD Generation. Recent systems generate editable CAD programs from geometry or language using sequential, hierarchical, and language-model-based representations (Khan et al., 2024; Zhang et al., 2025; Guan et al., 2026; Xu et al., 2026). Point-cloud and multimodal reconstruction further extend CAD generation to broader input modalities (Rukhovich et al., 2025; Liu et al., 2026; Kolodiazhnyi et al., 2025). Other works improve generated programs through execution feedback, geometric or visual verification, and agentic refinement (Alrashedy et al., 2025; Chen et al., 2025; Tsuji et al., 2025; Ma et al., 2026; Fan et al., 2026b;a), CAD-tool interaction (Mallis et al., 2025), or solver-aided constraints for 2D CAD (Jones et al., 2025). These systems improve generation quality through feedback or constrained revision, but their edit scope is typically determined by the observed failure or refinement procedure. TopoRecover instead compares persistent construction states with a supplied feature plan and traces mismatches back to the commands that created them, enabling repair localization even when those commands executed successfully.

Automated Program Repair. Automated program repair commonly separates fault localization—determining where an edit should be made—from patch synthesis and validation. Recent repair systems combine localization or code-context retrieval with patch generation (Xia et al., 2025; Zhang et al., 2024), while execution feedback and tool interaction are used to revise or validate generated candidates (Chen et al., 2024; Bouzenia et al., 2025). Related verifier-guided methods use execution or structural constraints to reject, rerank, or supervise generated programs and reasoning steps (Wang et al., 2018; Scholak et al., 2021; Ni et al., 2023; Ugare et al.; Lightman et al., 2024; Wang et al., 2024). However, these mechanisms generally identify invalid outputs or failing executions rather than locating which committed state in a persistent program first diverged from the intended structure. TopoRecover introduces a CAD-specific localization signal: disagreement between saved construction states and the feature plan, together with provenance linking that disagreement to editable source commands. This separates the decision of where to repair from how to generate the patch.

6 Conclusion

We introduced TopoRecover, a topology-aware framework for CAD program repair that localizes the first committed topology divergence rather than repairing only at the first visible failure. By replaying CAD topology states, TopoRecover identifies where repair should begin, restores the intended structure, and preserves unaffected construction history. Experiments on natural Text2CAD failures, controlled localization benchmarks, held-out generators, unseen structure families, and additional feature operators show that correct repair localization is critical: when topology diverges earlier, downstream patching alone is insufficient. These results highlight repair localization as a key bottleneck in stateful CAD generation.

AI USE STATEMENT

In this work, we used generative AI tools to assist in designing and providing feedback on the research methodology and experiments, implementing methods, and cleaning and reformatting datasets. We have not used generative AI tools for translation or result interpretation, and the remaining required-disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to edit software code and improve the readability of the manuscript. We have reviewed all AI-assisted work. AI-assisted code was executed and tested by the authors, and experimental outputs were validated through verifier replay, structural checks, and independent CAD execution. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work evaluates automated repair of CAD programs using research datasets and model-generated CAD programs and does not involve human-subject experiments. The proposed repair pipeline validates generated outputs through structural checks and CAD execution. The resulting CAD programs are research outputs and should undergo appropriate engineering validation before use in safety-critical or production settings.

REPRODUCIBILITY STATEMENT

The codebase and dataset are provided through the anonymous repositories linked in the paper. The appendix documents the evaluation inputs and their provenance, feature-plan construction, repair and validation procedures, baseline adaptations, model-call and budget settings, evaluation metrics, and controlled localization experiments. Together with the released artifacts, these details are provided to support reproduction of the reported repair and localization results.

REFERENCES

Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Haider Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. Generating cad code with vision-language models for 3d designs. In *International Conference on Learning Representations*, volume 2025, pp. 52236–52262, 2025.

Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. Repairagent: An autonomous, llm-based agent for program repair. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 2188–2200. IEEE, 2025.

Jiali Chen, Xusen Hei, HongFei Liu, Yuancheng Wei, Zikun Deng, Jiayuan Xie, Yi Cai, and Qing Li. Cadreview: Automatically reviewing cad programs with error detection and correction. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9909–9927, 2025.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, volume 2024, pp. 8746–8825, 2024.

Fengxiao Fan, Jingzhe Ni, Fan Sang, Xiaolong Yin, Yu Liu, Ruofeng Tong, Min Tang, and Peng Du. Tracecad: Trace-guided repair for agentic cad generation. *arXiv preprint arXiv:2608.03062*, 2026a.

Fengxiao Fan, Jingzhe Ni, Xiaolong Yin, Sirui Wang, Xingyu Lu, Qiang Zou, Ruofeng Tong, Min Tang, and Peng Du. Caddesigner: Conceptual cad model generation with a general-purpose agent. *Computer-Aided Design*, pp. 104087, 2026b.

Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. Cad-coder: Text-to-cad generation with chain-of-thought and geometric reward. *Advances in Neural Information Processing Systems*, 38:59765–59789, 2026.

Benjamin T Jones, Zihan Zhang, Felix Hähnlein, Wojciech Matusik, Maaz Ahmad, Vladimir Kim, and Adriana Schulz. A solver-aided hierarchical language for llm-driven cad design. In *Computer Graphics Forum*, volume 44, pp. e70250. Wiley Online Library, 2025.

Mohammad S Khan, Sankalp Sinha, Talha U Sheikh, Didier Stricker, Sk A Ali, and Muhammad Z Afzal. Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts. *Advances in Neural Information Processing Systems*, 37:7552–7579, 2024.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multi-modal cad reconstruction with online reinforcement learning. *arXiv e-prints*, pp. arXiv–2505, 2025.

Jing Li, Yihang Fu, and Falai Chen. Dtgbrepgen: A novel b-rep generative model through decoupling topology and geometry. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21438–21447. IEEE, 2025a.

Yuan Li, Cheng Lin, Yuan Liu, Xiaoxiao Long, Chenxu Zhang, Ningna Wang, Xin Li, Wenping Wang, and Xiaohu Guo. Caddreamer: Cad object generation from single-view images. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21448–21457. IEEE, 2025b.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Yang Liu, Daxuan Ren, Yijie Ding, Jianmin Zheng, and Fang Deng. Learning cad modeling sequences via projection and part awareness. *Advances in Neural Information Processing Systems*, 38:98243–98275, 2026.

Yujia Liu, Anton Obukhov, Jan Dirk Wegner, and Konrad Schindler. Point2cad: Reverse engineering cad models from 3d point clouds. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3763–3772. IEEE, 2024.

Weijian Ma, Shizhao Sun, Ruiyu Wang, and Jiang Bian. Cadmorph: Geometry-driven parametric cad editing via a plan–generate–verify loop. *Advances in Neural Information Processing Systems*, 38:100279–100297, 2026.

Dimitrios Mallis, Ahmet Serda Karadeniz, Sebastian Cavada, Danila Rukhovich, Niki Foteinopoulou, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-assistant: Tool-augmented vllms as generic cad task solvers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7284–7294. IEEE, 2025.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. Lever: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning*, pp. 26106–26128. PMLR, 2023.

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

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 9895–9901, 2021.

Jami J Shah and Martti Mäntylä. *Parametric and feature-based CAD/CAM: concepts, techniques, and applications*. John Wiley & Sons, 1995.

Chikaha Tsuji, Enrique Flores Medina, Harshit Gupta, and Md Ferdous Alam. Gencad-self-repairing: Feasibility enhancement for 3d cad generation. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 89206, pp. V02AT02A055. American Society of Mechanical Engineers, 2025.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. Syncode: Llm generation with grammar augmentation, 2024. *URL https://arxiv. org/abs/2403.01632*.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*, 2018.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. Text-to-cad generation through infusing visual feedback in large language models. *arXiv preprint arXiv:2501.19054*, 2025.

Karl DD Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics (TOG)*, 40(4): 1–24, 2021.

Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6752–6762. IEEE, 2021.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying llm-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 2025.

Jimin Xu, Tianbao Wang, Tao Jin, and Zhou Zhao. Hiercad: Hierarchical text-to-cad design via structure alignment and parameter grounding. *arXiv preprint arXiv:2607.11339*, 2026.

Xiang Xu, Karl DD Willis, Joseph G Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks. *arXiv preprint arXiv:2207.04632*, 2022.

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

Jeongsam Yang and Soonhung Han. Repairing cad model errors based on the design history. *Computer-Aided Design*, 38(6):627–640, 2006.

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. *arXiv preprint arXiv:2502.03997*, 2025.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1592–1604, 2024.

Zhanwei Zhang, Shizhao Sun, Wenxiao Wang, Deng Cai, and Jiang Bian. Flexcad: Unified and versatile controllable cad generation with fine-tuned large language models. In *International Conference on Learning Representations*, volume 2025, pp. 3204–3227, 2025.

Part I

Appendix

Table of Contents

A Verifier State and Localization **14**
A.1 State Updates and Source Attribution 14

B Implementation and Evaluation Protocol **15**
B.1 LLM Patch Interface 15
B.2 Matched Localization Protocol and Results 17
B.3 Full-program LLM baselines 18

C Evaluation Inputs and Measurement **18**
C.1 Feature Plans and Input Provenance 18
C.2 Metric Definitions and Aggregation 19
C.3 Final-Solid Topology 20

D Plan Construction and Supplementary Evaluation **20**
D.1 LLM-Generated Plans 20
D.2 Reference-Derived Plans 21
D.3 Supplementary Shared-7B Comparison 21

E Adapted Repair Baselines **22**

F Additional Quantitative Analyses **23**
F.1 Which Cases Are Repaired? 23
F.2 Performance Across Input Lengths 24
F.3 Successful Repair with Command Preservation 24
F.4 Per-Case Geometric Accuracy 25
F.5 Model Usage under Matched Localization 25

G Additional Qualitative Results **25**
G.1 Local Edits with Preserved Boundaries 25
G.2 Repair with Reference-Consistent Geometry 26
G.3 Comparison with Adapted Repair Methods 27

H Supported Constraints **27**

I Limitations and Evidence Scope **28**

J Reproducibility **29**

Table 2: Verifier updates and the source information retained for repair.

Command group	State update	Repair evidence
Open sketch, face, loop	Enter the corresponding active context	Opening command and nesting context
Add line, arc, circle	Accumulate the current boundary and check geometry	Original curve commands and parameters
Close loop, face	Check closure and retain completed boundaries	Boundary roles and their source intervals
Register profile	Store the completed profile under its identifier	Full source interval and observed boundary roles
Extrude	Check profile reference and operation preconditions	Referenced profile and rejected command, if any
End	Check completion of the command sequence	Pending construction or termination error

A Verifier State and Localization

TopoVerifier is a deterministic state machine, not a learned classifier. It maintains active sketch, face and loop contexts; curve endpoints and closure; committed profile identifiers and boundary roles; feature references and operation types; and termination status. A command updates this state only after its local preconditions pass. Replay stops at the first rejected command, while retaining the committed profile records accumulated before that point. Each record includes the source interval from the opening face command through profile registration. Comparing its boundary-role multiset against the corresponding plan entry yields both a structural discrepancy and the source commands available for patching.

Alignment and multiple discrepancies. The diagnostic routine sorts committed profile records by identifier and pairs them positionally with ordered plan entries. This convention assumes that profile identifiers and plan order remain aligned in the supported representation. Missing, duplicated, or reordered profiles can invalidate that assumption; there is no learned correspondence search or guarantee of correct alignment for arbitrary equivalent programs. All mismatched intervals and their observed and required roles are recorded before patching. The implementation creates a separate request for each discrepant profile and merges the returned retention decisions and new-loop parameters into a joint patch. The committed-profile interface supports up to eight discrepant profiles, 16 loop additions or removals, and 128 newly serialized actions. Source offsets are indexed in the original sequence, and insertions for one profile are combined before application. Each edit must fit within a diagnosed interval. An accepted patch must resolve every diagnosed committed-profile mismatch on full prefix replay. The earliest mismatch is a diagnostic timestamp, not a claim of uniquely minimal edit scope.

Scope of structural verification. Boundary-role counts and profile references describe construction structure, not complete B-rep topology. Equal role counts can coexist with misplaced holes, different nesting, or different final-solid topology. Public parameter checks and independent kernel execution catch some additional errors, but are not a proof of geometric equivalence. The final-solid Euler metric is evaluated separately from the plan-matching predicate.

A.1 State Updates and Source Attribution

Table 2 details the state transitions summarized in Figure 2(a). The verifier checks the active construction as commands arrive and retains completed profiles after their sketch has closed. Consequently, the location of a later rejected operation and the source of an earlier plan discrepancy are separate outputs of replay. An unfinished loop is checked for local validity, but its temporary absence from the committed profile registry is not treated as a missing planned hole. Plan comparisons are made against completed records.

Source intervals remain attached to the completed records instead of being inferred from the error message. For example, a registered profile containing only an outline can disagree with a plan requiring an outline and a hole even though registration succeeds. A later unresolved extrusion

reference is a separate execution error: the missing hole does not, by itself, make every extrusion invalid. The profile edit and reference correction therefore address distinct requirements. This distinction prevents a plan mismatch from being misrepresented as a guaranteed kernel failure.

B Implementation and Evaluation Protocol

Frozen inference and fallback. The localized LLM path uses Qwen2.5-7B-Instruct without an adapter or training, greedy grammar-constrained decoding, seed zero, an 8,192-token input limit and a 2,048-token output limit, for at most six orchestration rounds. Public parameter binding precedes patching; supplied supported geometry and extrusion depths are checked after completion. A compliant LLM-path output is returned unchanged. Otherwise, the compiler runs once from the original input, with replay horizon six, rollback limit 32 and replacement limit 240 actions. It initializes wholly new geometry in the observed source frame and binds supported explicit public parameters without editing protected observed loops. Both outputs must satisfy verifier replay, the topology specification, public parameter checks and independent OCC execution; neither path sees a reference shape. Inputs rejected as contradictory or unsupported public parameter evidence remain unresolved.

The committed-profile path shown in Figure 2 is supplemented by bounded handlers for unfinished sketches and remaining feature references. For malformed delimiters, replay identifies the active face/sketch window and lists the commands that may be moved, inserted, or deleted. The LLM chooses those edits; numeric geometry and protected extrusions remain unchanged, and deleting geometric commands requires evidence of a redundant construction. Every accepted edit must make replay progress before subsequent repair is attempted. These handlers share the frozen orchestration budget and acceptance checks; they are not unrestricted whole-program generation.

B.1 LLM Patch Interface

Each committed-profile request contains the public brief and feature plan, the localized commands, existing hole identifiers and geometry, and the required numbers of retained and added holes. The model selects which existing holes to keep and supplies parameters for new circles or rectangles. Unselected existing holes are removed; retained curves are copied exactly. New loops are inserted before the closing face command, leaving the profile identifier and protected commands unchanged. Responses are merged using original source offsets, then checked jointly against all required profiles. Selecting a legal edit location alone does not make the patch acceptable.

Geometry and parameter representation. World coordinates are used for explicit dimensions. The alternative profile-relative frame uses the lower-left corner of the observed outline's bounding box as its origin, with width and height defining the two coordinate scales. Circle radius is scaled by the shorter box side. The box defines coordinates, not a guarantee of containment in the actual outline. All additions are checked against the observed boundary and retained holes.

For a circle whose center and radius are unspecified, the current patcher also exposes triangles lying inside the observed outline. The LLM selects a triangle, three positive integer weights between one and nine, and a radius fraction between 0.1 and 0.9. The weighted vertices determine the center; the radius fraction interpolates between the minimum supported radius and 90% of the available clearance to the outline and other holes. An obstructed center or insufficient clearance causes rejection. This decoder supplies a geometric parameterization: it neither searches for a substitute proposal nor certifies agreement with an unknown reference shape. Specified dimensions use the explicit coordinate interface instead.

Feedback and budget. An unsuccessful first profile proposal receives the measured violation, such as incorrect retained-hole count, insufficient area, boundary contact, or overlap. One feedback proposal is permitted per profile. For ambiguous retention, a separate evidence-selection request identifies the relevant brief sentences before patching; that call counts toward the same orchestration budget. Malformed output, edits outside the selected profile, unresolved discrepancies, and exhausted budgets remain explicit failures of the LLM path. Table 3 lists the limits used by the current implementation.

Table 3: Fixed limits of the current repair system. Per-profile limits share the overall orchestration ceiling.

Component	Limit or policy
Backbone	Frozen Qwen2.5-7B-Instruct, no adapter or training
Decoding	Greedy grammar-constrained decoding; seed 0
Per request	8,192 input tokens; 2,048 output tokens; no input truncation
Orchestration	At most six rounds, including evidence selection and feedback
Committed-profile patching	Eight discrepant profiles; two proposals per profile
Merged profile edits	At most 16 loop additions/removals and 128 new commands
Structural repair	Active window of at most 128 commands; at most 32 edits
Verified fallback	One invocation; horizon 6, rollback 32, replacement 240 commands
Selection	Return the accepted LLM output; invoke fallback otherwise

Public parameter binding. Structured parameter evidence in the public brief is matched to plan features by explicit feature identifier, profile reference, type, and operation. Supported length units are converted before comparison. Missing depths may be filled from consistent evidence; conflicting supplied depths or unresolved units are rejected. This binding does not extract arbitrary dimensions from unrestricted prose or recover parameters from reference CAD. The final parameter checks apply equally to the LLM candidate and the fallback candidate.

Evaluation configuration. Development uses 45 cases: 18 analytic refusal fixtures and 27 adaptive regression cases. The end-to-end comparison retains all 347 repair cases and all 23 geometry cases, including the one unexecutable reference excluded only from reference-based geometry metrics. The input cohorts were also used in preceding development studies; the reported comparison fixes the implementation, inputs and output-selection rules before measurement. All final outputs are fixed before measuring reference-dependent metrics; no selection or method revision uses these results. Exact archived LLM responses are reused after input and source-identity checks: 573 logical calls and 54,876 generated tokens, with no new model inference. TopoRecover attempts the compiler fallback on 142 of the 370 cases.

Component attribution. The hybrid adds 50 repairs to the unchanged LLM path, with no lost successes (paired repair gain 14.4 percentage points, 95% CI 10.7–18.2). The fallback-only ablation uses the same bounded repair routine and public parameter checks as TopoRecover and achieves 40.1% joint repair. Its 191 extrusion-depth mismatches reflect incomplete parameter realization in this implementation, so TopoRecover's advantage over this control does not demonstrate that symbolic repair is inherently weaker. The hybrid does not improve geometry over this compiler: the failure-aware IoU difference is -0.0169 (95% CI $[-0.0499, 0.0087]$), with identical Euler agreement. No main-cohort input supplies the explicit profile-geometry field supported by the new circle/rectangle interface; main-table geometry therefore does not validate that interface's transfer.

Unresolved cases. The hybrid retains 75 repair failures: 58 explicit extrusion-depth mismatches, 13 inputs outside its positive-depth convention, and four kernel failures. The 13 convention failures include signed and zero depths and do not imply that these source designs are universally invalid CAD. No case is removed to improve the repair rate.

Baseline comparison and uncertainty. All 1,110 archived final-output/action pairs for the three adapted baselines agree with the evaluated sequences. Their backbone and budget differ from the hybrid, and full historical prompts and brief-sharing flags are unavailable; this is not a matched-compute or proven identical-prompt comparison. Repair gains over ReCAD, CADCodeVerify and CADDesigner are 37.2, 14.1 and 20.7 percentage points, respectively, with paired 95% intervals $[30.8, 43.5]$, $[8.6, 19.6]$ and $[15.0, 26.8]$. Exact McNemar tests remain significant after Holm correction over the six repair/Euler comparisons. Bootstrap intervals use 10,000 resamples of source-program groups and are descriptive rather than simultaneous intervals. The three failure-aware IoU gain intervals all include zero.

Table 4: Matched profile-region selection with the deployed patcher. All arms share validation, fallback, and budget limits.

Region selector	Before fallback (%)	Final repair (%)	Recovered history	Fallback (%)
Rejected command	9.8	47.6	0.228	90.2
Active block	9.8	47.6	0.228	90.2
Runtime dependency	9.8	47.6	0.228	90.2
Topology discrepancy	64.0	78.4	0.513	36.0

Table 5: Geometry and final-solid topology for the matched localization arms, using the same 22 valid references.

Region selector	n_g	CD ↓	HD ↓	IoU ↑	$T_{\text{corr}}^{\text{all}}$ (%) ↑
Rejected command	22	0.1121	0.4052	0.2867	90.9
Active block	22	0.1121	0.4052	0.2867	90.9
Runtime dependency	22	0.1121	0.4052	0.2867	90.9
Topology discrepancy	22	0.1145	0.4112	0.2698	90.9

B.2 Matched Localization Protocol and Results

The intervention changes which committed profiles may enter the production LLM patcher. Rejected-command selection exposes the first verifier-rejected command; active-block selection exposes its open construction block. Runtime-dependency selection follows a rejected extrusion to its registered profile, using the active block when the reference is unresolved. Topology selection uses the committed-profile discrepancies of the full method. When a selected interval intersects a committed profile, it expands to that complete profile, the patcher's atomic repair unit. No rejection means no committed-profile request for a runtime-only selector; the protocol does not invent a failure at the final command.

For each accessible profile, all prompt fields, geometry constraints, model weights, grammar, and feedback rules are unchanged. The final prefix check still requires every plan discrepancy to be resolved, including those outside the selected region. The structural, profile-retention, and execution handlers remain active in every arm, and all use the same fallback from the original public input. Consequently, a runtime selector can still repair some inputs through these unchanged handlers or fallback. Budget ceilings are matched; actual calls differ when a selector produces no eligible profile request. This is a matched intervention on committed-profile access, not a comparison of arbitrary patch generators or learned localization models.

All four arms retain the 347 repair and 23 geometry inputs, yielding 1,480 frozen outputs. The topology arm reproduces the archived production output on every input. Complete matching requests reuse verified frozen responses; four new requests, including feedback, run with the same Qwen2.5-7B-Instruct checkpoint on a V100. Missing cache entries are never counted as repair failures. All outputs are fixed before opening the private geometry references.

Table 4 shows the overall repair and preservation results. Relative to runtime dependency, topology-based access adds 107 successful repairs without losing a successful repair. The 310 verifier-accepted inputs account for this gain: final repair increases from 43.2% to 77.7%. On the 37 verifier-rejected inputs, all selectors reach 83.8% after fallback, with no accepted pre-fallback output. These strata distinguish the value of detecting semantic discrepancies from evidence of improved causal localization for runtime errors. The logical call counts on the repair cohort are 183, 187, 187, and 544 for rejected command, active block, runtime dependency, and topology selection, respectively.

Table 5 reports the separate geometry cohort without pooling its denominator with repair. Topology-based selection preserves Euler agreement but does not improve geometric fidelity over the fallback-dominated runtime selectors. The measured contribution in this experiment is repair coverage and preserved command structure, not a uniform gain in all metrics.

B.3 FULL-PROGRAM LLM BASELINES

Both generation baselines use frozen Qwen2.5-7B-Instruct and grammar-constrained decoding over the same typed commands as TopoRecover. LLM generation from plan receives only the public brief and frozen feature plan, with one call capped at 12,288 output tokens. LLM feedback regeneration also receives the original failed sequence, the current candidate, and public execution and requirement feedback. It makes at most two calls of 6,144 output tokens each, stopping at the first accepted candidate. The grammar enforces syntax but does not mask incorrect references, ordering, or dimensions. Each candidate must satisfy the public topology and parameter checks and independent CAD execution; no compiler completes or repairs its output. Neither baseline receives the source spans of earlier topology mismatches or reference geometry.

Both protocols were fixed after an interface smoke test on four development cases disjoint from the comparison cohorts. All 370 inputs were evaluated for each method, and all 740 final outputs were frozen before reference-based measurement. Plan generation used 370 calls and 182,349 generated tokens; feedback regeneration used 656 calls and 266,540 tokens. These baselines control the backbone and public input source, not the orchestration or computation budget. The adapted 30B repair systems have archived two-round, four-candidate budgets; their full historical prompts are unavailable, so identical prompt information cannot be established.

C EVALUATION INPUTS AND MEASUREMENT

Plan-conditioned repair. The source study froze 500 official Text2CAD briefs, 125 at each complexity level, excluding 840 prompt identifiers used in earlier source studies. Planning precedes CAD generation. The funnel contains 488 parseable plans, 487 replayable programs and 347 programs needing structural or execution repair. Of these, 310 violate only the plan topology, five also fail execution, and 32 fail execution alone. Repair receives the frozen upstream plan, never reference CAD or target commands. These counts define a conditional repair population, not an overall generation success rate over all 500 briefs. TopoRecover repairs 272 of the 347 inputs.

C.1 FEATURE PLANS AND INPUT PROVENANCE

The feature plan lists ordered sketch profiles and the features that consume them. Each profile has an identifier, sketch plane, and boundary-role list; each extrusion has an identifier, profile reference, operation, and an optional depth. A boundary role distinguishes the outer outline from a hole boundary. These fields specify construction requirements without providing the correct command sequence. The topology projection retains profile boundary roles, extrusion-to-profile references, operation types, and aggregate counts. Depths remain in the public parameter channel rather than the topology projection.

For example, the following illustrative plan requests a plate profile with a hole and an additive extrusion:

```
profiles: [{profile_id: plate, sketch_plane: XY,
  loop_roles: [outer, inner]}]
features: [{feature_id: body, type: extrude,
  profile_id: plate, operation: add, depth: 10}]
```

No correct hole center or radius is specified by these fields. The repair must preserve the observed outline and satisfy any additional supplied dimensions; otherwise the new hole's geometry is underdetermined. The plan is frozen before the source program is generated, so repair cannot revise it after observing a failure. The input archive records the source identifier, plan, projected requirements, and generated commands separately. This establishes the availability of an upstream repair target, not the correctness of every generated plan.

Table 6 separates the input sources and denominators used by the current method. The reference-derived geometry requirements are never described as generator-authored plans.

Geometry cohort. The geometry subset contains 23 cases selected from a separate 181-case natural-error cohort before repair outcomes were inspected. Its source identifiers do not overlap

Table 6: Inputs used by the current method and the provenance of their requirements.

Population	Size	Requirement source	Use
Development	45	Public development fixtures	Interface and regression checks
Pre-generation briefs	500	Generator-authored plans	Planning and generation funnel
Natural repair inputs	347	Frozen upstream plans	End-to-end repair and preservation
Geometry-compatible inputs	23	Reference-derived topology	Separate geometry comparison
Executable references	22	Reference solids	Geometry and final-solid topology metrics

those of the plan-conditioned cohort; structure-family disjointness is not guaranteed. The topology plan for this cohort is reference-derived, unlike the upstream plans in the 347-case study. One reference fails CAD execution, leaving 22 valid references for geometric and solid-topology metrics. The 23 cases are exactly those for which both generated and reference native-to-DSL conversions preserve geometry. Eligibility is determined by conversion fidelity before repair outcomes, rather than by which repairs succeed. Geometric distances on the other conversions would conflate repair errors with representation loss.

Measurement and uncertainty. All selected programs are fixed before reference-based evaluation. CD and HD use method-specific measurable output/reference pairs. Failure-aware volume IoU and Euler agreement use all 22 valid references and count output failures as zero. The paired geometric plot resamples source-program groups 10,000 times. Because conditional distances can compare different successful subsets, they are reported alongside measurable-output counts and failure-aware metrics.

C.2 Metric Definitions and Aggregation

Execution, plan match, and joint repair. Execution validity is the fraction of outputs that the independent CAD executor accepts as valid positive-volume solids. Plan match additionally requires a terminated verifier replay and equality of the projected construction requirements. Joint repair requires both predicates. The system's public parameter checks constrain candidate acceptance; they are not inferred from reference geometry during evaluation. An absent output contributes zero to execution, plan match, and joint repair, rather than being dropped from their denominators.

History preservation. The archived metric named action-LCS matches exact command strings with Python's `SequenceMatcher`, with `autojunk=False`. Let m_i be the number of source commands in its ordered matching blocks and n_i the original sequence length; the per-case score is $h_i = m_i/n_i$. This implementation is a matching-block measure, not a dynamic-programming computation of the longest common subsequence. All systems use the same implementation. History averages h_i over successful repairs, whereas recovered history averages $s_i h_i$ over all repair inputs, where s_i indicates joint repair success. Thus returning a nearly unchanged but incorrect program does not earn recovered-history credit. Equivalent command reformulations can nevertheless receive different scores because matching is textual.

Surface distances. We sample 4,096 surface points from each measurable output and reference using the shared evaluator. Distances are computed in the original coordinate system and normalized by the reference bounding-box diagonal d; shapes are not independently rescaled or aligned to improve agreement. For output points A, reference points B, and $D(a, B) = \min_{b\in B} \|a - b\|_2$, the reported distances are

$$\mathrm{CD}(A, B) = \frac{1}{2d}\left(\frac{1}{|A|}\sum_{a\in A} D(a, B) + \frac{1}{|B|}\sum_{b\in B} D(b, A)\right), \tag{4}$$

$$\mathrm{HD}(A, B) = \frac{1}{d}\max\left\{\max_{a\in A} D(a, B), \max_{b\in B} D(b, A)\right\}. \tag{5}$$

These use unsquared Euclidean distances. Table 1 reports their means over each method's measurable pairs and lists that count as n_g.

Table 7: Conditional solid-topology comparison on the four references measurable by all six systems. This small intersection complements the all-reference metric in Table 1.

Method	$T_{\mathrm{err}} \downarrow$	T_{corr} (%) $\uparrow$
CADReview/ReCAD*	0.000	100.0
CADCodeVerify*	1.000	50.0
CADDesigner*	0.000	100.0
LLM generation from plan	0.000	100.0
LLM feedback regeneration	0.000	100.0
TopoRecover	0.000	100.0

Volume and statistical comparisons. Volume IoU uses the kernel intersection volume divided by the union volume, rather than voxel or rendered-image overlap. For the failure-aware mean, every executable reference remains in the denominator and an unavailable output contributes zero. Bootstrap comparisons resample source-program groups 10,000 times while keeping all paired method outputs together. For binary repair outcomes, exact McNemar tests use the two discordant counts; Holm correction covers the stated family of six repair/Euler comparisons. Conditional distance estimates, failure-aware means, and their denominators answer different questions and are not pooled.

C.3 Final-Solid Topology

We independently execute each fixed output with OCC and tessellate its final boundary after normalizing the bounding-box diagonal to one. The Euler characteristic is $\chi = V - E + F$ after merging coincident seam vertices to eight decimal places and removing unreferenced vertices; we do not repair the mesh. An output is measurable only if it has valid positive volume, is watertight at both meshing resolutions, and has the same χ at both resolutions. The linear/angular tolerances are $(10^{-3}, 0.1)$ and $(5{\times}10^{-4}, 0.05)$. On the common measurable subset, $T_{\mathrm{err}} = \mathrm{mean}\,|\chi_{\mathrm{out}} - \chi_{\mathrm{ref}}|$ and T_{corr} counts exact equality. We also report $T_{\mathrm{corr}}^{\mathrm{all}}$ over every measurable reference, assigning failed or unmeasurable outputs zero credit. After including both LLM generation baselines, the common subset contains four cases; the reference denominator remains 22 because one of the 23 reference histories fails execution. Table 7 is consequently a conditional diagnostic with limited coverage, not the primary topology comparison. TopoRecover agrees with the reference Euler characteristic on 20 of the 22 references overall; the other two outputs satisfy their plan but have a different boundary Euler characteristic. Equal Euler characteristics need not imply equivalent component or genus structure, so these metrics complement geometry and plan agreement rather than certify complete topological equivalence. Reference geometry is used only for evaluation, never for patch construction or candidate selection.

D Plan Construction and Supplementary Evaluation

We distinguish two sources of repair requirements: an upstream planning model and a reference CAD construction. The main repair comparison uses the first source, with the selection procedure in Appendix C. The second source provides the separate supplementary experiment below; it does not replace the plans or results in the main repair comparison. Both specify requirements in the supported sketch-and-extrusion representation and keep the plan fixed during repair.

D.1 LLM-Generated Plans

A Qwen2.5-7B-Instruct planner reads a Text2CAD brief and produces a structured feature plan before CAD generation. The plan records ordered profile identifiers, boundary roles, feature references, operations, and any proposed extrusion depths. We parse the plan as structured data, check its schema and references, and project its structural fields into executable requirement checks. We then replay the generated command sequence and select inputs that fail execution or disagree with those requirements. This procedure selects programs requiring repair relative to the supplied plan. It does not establish that every plan is the unique or correct interpretation of the brief.

Table 8: Two sources of fixed repair requirements and the scope of their verification.

	LLM-generated plan	Reference-derived plan
Source	Natural-language design brief	Decoded native reference construction
Construction	Structured prediction, parsing, and projection	Profile and feature traversal with geometric checks
Verification	Schema, source alignment, and a separate consistency audit	Native execution, boundary checks, and command replay
Interpretation	Requirements proposed by the planner	Supported structure of the chosen reference
Use here	Main repair and localization studies	Supplementary reference-conditioned study

Consistency checks. The archived audit checks source-brief alignment and structural projection for all 488 parseable plans; neither check found a bookkeeping discrepancy. For the 347 selected repair inputs, a comparison with archived converted reference constructions found structural agreement in 50 cases, disagreement in 289, and unavailable reference comparisons in eight. These are construction-agreement counts, not semantic plan-accuracy labels: alternative valid constructions and conversion differences can also cause disagreement. A non-representative, agent-assisted review of 12 briefs identified three explicit brief–plan conflicts and five plans containing requirements not supported by the brief. These qualitative checks are not independent human annotations or an estimate of population accuracy. Consequently, the main repair percentage measures execution and frozen-plan satisfaction rather than independently established compliance with every original brief.

D.2 Reference-Derived Plans

We retrieve each available native reference construction by its Text2CAD source identifier and decode it using the official structured representation. The extractor traverses profiles and extrusions to record boundary roles, profile-to-feature references, and operation types. It uses the decoded CAD objects rather than regular expressions over text descriptions. Geometric checks verify closed, valid profile boundaries and the containment and separation of hole boundaries. References must also execute successfully in both the native pipeline and the supported command representation. Unsupported constructions are excluded with recorded reasons.

The public plan retains structural requirements but omits reference commands, sketch coordinates, and extrusion depths. These checks establish agreement with the supported structure of the chosen reference construction. They do not establish equivalence to every valid realization of a brief or preservation of all native CAD geometry. We separately audit native-to-command conversion fidelity before making reference-geometry claims.

Table 8 summarizes the distinction between the two sources. An upstream plan supplies a deployment-time target without requiring a reference CAD model; reference extraction supplies a reproducible target for evaluating agreement with a known construction. This is a comparison of requirement provenance, not a matched experiment measuring the causal effect of replacing one planner with another.

D.3 Supplementary Shared-7B Comparison

Population and inputs. Starting from all 487 archived evaluable Text2CAD generations, reference verification retains 453 constructions and excludes 34 unsupported or invalid references. Comparing the generated commands with the verified requirements identifies 242 repair inputs; the other 211 already satisfy the execution and structural checks. This selection uses reference verification and the original generated programs, without consulting repair outputs. Every method receives the same structural plan, and repair methods also receive the original command sequence. Plan-only generation receives no original program. The original free-text brief is replaced by a common instruction to satisfy the supplied construction requirements and preserve compatible input geometry. No target dimensions are provided. Because the source generations were produced under upstream LLM plans, this experiment is a retrospective reference-conditioned evaluation rather than fresh generation under the extracted plans.

Table 9: Supplementary shared-7B comparison with reference-derived plans. * denotes adaptations; n_g gives measurable geometry coverage.

Method	RS (%)↑	Exe. (%)↑	CD↓	HD↓	IoU↑	T_{corr} (%)↑	n_g
CADReview/ReCAD*	14.88	82.23	0.1911	0.6751	0.1956	54.63	199
CADCodeVerify*	<u>43.39</u>	<u>85.95</u>	0.1796	0.6608	<u>0.2042</u>	<u>60.79</u>	208
CADDesigner*	11.57	81.40	0.1961	0.7028	0.1888	52.42	197
LLM generation from plan	2.48	2.48	10.4286	39.5235	1.88×10^{-6}	1.32	6
LLM feedback regeneration	18.18	18.18	0.2834	1.0968	0.0537	11.89	44
TopoRecover	**100.00**	**100.00**	0.1795	0.6296	**0.2303**	**82.38**	242

Protocol and metrics. All six systems use Qwen2.5-7B-Instruct, with the frozen repair implementation and separately recorded method budgets. The main method preserves accepted local LLM proposals and applies its public-geometry-constrained fallback when needed. The reported run reuses archived model proposals and freezes final programs before independent CAD measurement; it is a supplementary re-evaluation, not a newly blind test. We evaluate every selected input for every system, yielding 1,452 method–case measurements. No repair proposal is selected using reference geometric scores.

Repair success requires terminated verifier replay, satisfaction of the supplied structural requirements and any supported supplied parameters, and valid positive-volume CAD execution. Because these plans supply no target geometric parameters, the parameter check imposes no additional target-dimension requirement in this experiment. CD and HD use each method's measurable output/reference pairs, whose count is n_g. IoU uses all 242 inputs, assigning unsuccessful outputs zero. Euler agreement uses the 227 references with stable boundary-mesh measurements, counting invalid or unmeasurable candidate topology as a nonmatch. These geometry and topology scores compare outputs with references in the converted command representation. Only two repair inputs pass the stricter native-geometry fidelity audit, so the full-cohort scores do not establish fidelity to native CAD geometry.

Results and interpretation. Table 9 reports higher plan-conditioned repair coverage, failure-aware IoU, and Euler agreement for TopoRecover than for the tested adaptations and regeneration systems. The conditional CD/HD means use different measurable populations, so we do not rank them as paired comparisons. All TopoRecover outputs satisfy the executable-program and supplied structural checks; the lower reference IoU and Euler agreement show that passing these checks does not imply complete shape reconstruction. An additional input-relative audit finds unanchored solids in ten of its 234 assessable outputs, where an unanchored solid has no positive-volume overlap with the executable input or its valid prefix. This is a regression signal requiring case-level interpretation, not a universal semantic error label or an extra repair-success criterion. The supplementary and main repair percentages use different requirement sources, input populations, and baseline backbones; their numerical difference cannot be attributed to plan construction alone.

E Adapted Repair Baselines

We adapt three recent repair strategies to the same typed histories with pre-generation plans. CADReview/ReCAD (Chen et al., 2025) first predicts erroneous action blocks and error types, then passes only that generated review to a separate editor; CADCodeVerify (Alrashedy et al., 2025) iterates over binary requirement checks without a supplied repair location; CADDesigner (Fan et al., 2026b) revises the full history from the brief, feature plan, and execution feedback. All three use Qwen3-30B-A3B, two rounds with four candidates per round, and a public candidate selector. Their final histories align with the evaluation archive, but full historical prompt equality with the 7B hybrid cannot be established (Appendix B). They receive neither t_{goal}, the target-mismatch map, target actions or geometry, a reference CAD, nor an oracle patch; these are method adaptations under a shared representation, not official reproductions.

Table 10 distinguishes the comparison interfaces from the original systems' native image or code representations. The public candidate selector used by the adapted systems does not consult a ref-

Table 10: Interfaces of the compared systems. The starred methods are adaptations to the typed CAD representation.

System	Repair or generation signal	Output interface
CADReview/ReCAD*	Model-generated block review and error categories	A separate editor revises the program from the review
CADCodeVerify*	Iterative binary requirement checks	Revise candidates without a supplied earlier repair location
CADDesigner*	Brief, feature plan, and execution feedback	Full-program revision
LLM generation from plan	Public brief and frozen feature plan	One complete new command sequence
LLM feedback regeneration	Brief, plan, failed program, and candidate feedback	Up to two complete generated sequences
TopoRecover	Verifier-localized profiles and public geometric constraints	Local LLM edits, execution completion, and verified fallback

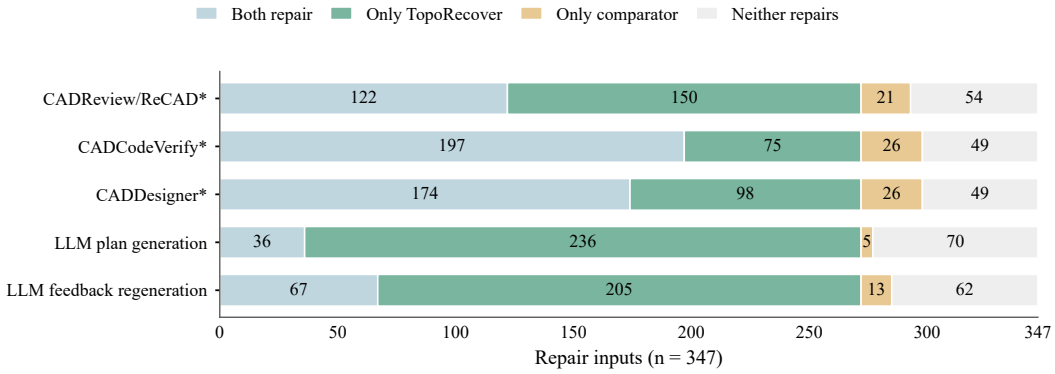

Figure 9: Pairwise repair overlap on the same 347 inputs. Each row compares TopoRecover with the indicated method; segment labels are case counts and each row sums to 347. Asterisks denote adapted systems.

erence solid or an oracle patch. The 30B backbone and its two-round, four-candidate budget are common to the three adaptations, whereas the two new generation baselines share TopoRecover's 7B backbone. For reproducibility, the artifact retains the evaluated final sequence for every method and case, including unsuccessful outputs. These differences are why the main table is an end-to-end comparison and the region-selection study holds the deployed patcher fixed separately.

F Additional Quantitative Analyses

We examine the frozen per-case outputs behind Table 1 and the matched localization study, without generating new candidates or changing the evaluation protocol. The following analyses use all 347 repair inputs or, separately, all 22 executable references in the geometry cohort. They complement the aggregate results with case overlap, input-length stratification, command preservation, geometric variation, and realized model usage.

F.1 Which Cases Are Repaired?

To determine whether methods repair the same inputs, we partition each pairwise comparison into shared successes, method-specific successes, and shared failures. Figure 9 shows that TopoRecover repairs 75 cases that CADCodeVerify does not, while CADCodeVerify repairs 26 cases that TopoRecover does not. Furthermore, the two methods share 197 successful repairs and leave 49 inputs unresolved. **The higher aggregate success rate reflects a net gain in repaired cases, not success on every case handled by a baseline.**

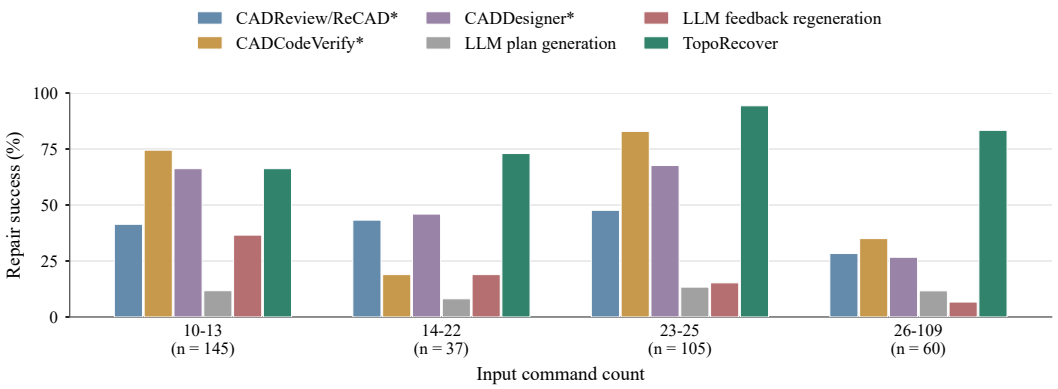

Figure 10: Repair success by input command count. Bin boundaries depend only on the input-length distribution, not on repair outcomes; all 347 inputs are included, with group sizes shown below the axis. Asterisks denote adapted systems.

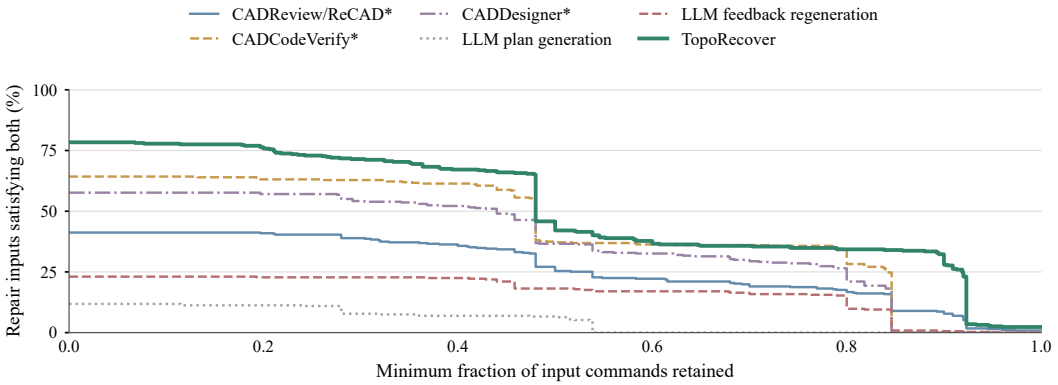

Figure 11: Successful repair subject to a command-retention threshold. The denominator is always 347, rather than the number of successful repairs. The left endpoint is each method's repair success rate; increasing the threshold requires more original commands to be preserved. Asterisks denote adapted systems.

F.2 Performance Across Input Lengths

To examine variation with program length, we group inputs using the quartiles of their command counts, keeping equal-length programs together. This input-only rule yields boundaries at 13, 22, and 25 commands; tied lengths explain the unequal group sizes in Figure 10. TopoRecover achieves the highest repair rate in three groups, including 83.3% on the 60 inputs with more than 25 commands, compared with 35.0% for CADCodeVerify. However, CADCodeVerify performs better on the shortest group, at 74.5% versus 66.2%. These descriptive differences do not isolate a causal effect of length, since the groups can also differ in construction requirements and error types.

F.3 Successful Repair with Command Preservation

To examine preservation beyond its mean, we measure the fraction of all repair inputs that are successfully repaired while retaining at least a given fraction of their original commands. Retention uses the same exact-command matching-block score as the history metric, and unsuccessful repairs never contribute to the curves in Figure 11. At a 90% retention threshold, TopoRecover satisfies both conditions on 112 inputs (32.3%), compared with 30 (8.6%) for CADReview/ReCAD. At a 75% threshold, CADCodeVerify instead has a small advantage, with 124 qualifying repairs versus 121 for TopoRecover. **The distribution reveals substantial high-preservation repairs without implying uniform superiority at every retention threshold.**

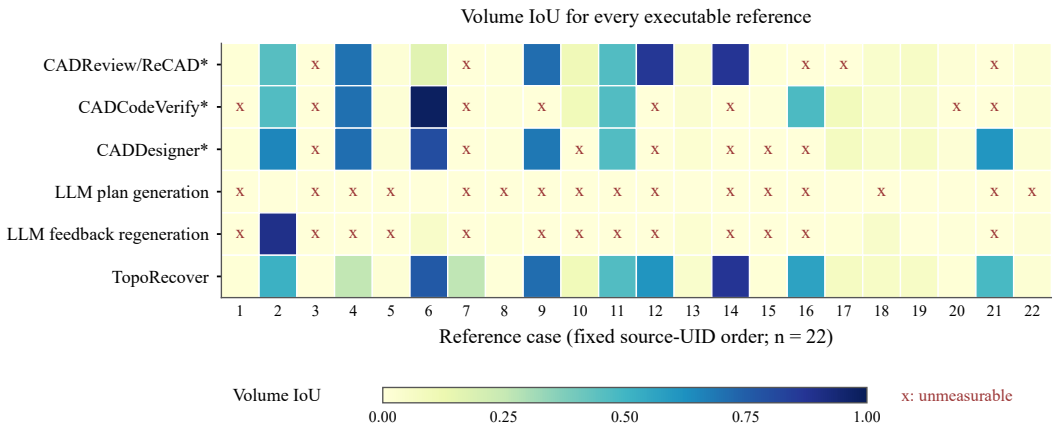

Figure 12: Volume IoU on all 22 executable reference models, ordered independently of method outcomes. Crosses mark unmeasurable outputs, which contribute zero to the reported IoU; the remaining reference in the 23-case cohort is excluded because the reference itself is not executable. Asterisks denote adapted systems.

F.4 Per-Case Geometric Accuracy

To expose variation hidden by the average IoU, Figure 12 shows every executable reference in a fixed source-UID order shared by all methods. TopoRecover produces measurable geometry for all 22 references, whereas the three adapted baselines cover 14–17. Nevertheless, its IoU remains low on several cases, and individual baselines achieve higher IoU on some references. Thus, broader output coverage and higher average IoU do not imply exact geometric recovery for every input. The qualitative examples in Appendix G illustrate both preserved geometry and remaining discrepancies.

F.5 Model Usage under Matched Localization

To distinguish a fixed repair budget from its realized use, we count logical LLM calls in the four region-selection arms described in Appendix B.2. Figure 13 shows 1.57 calls per input for topology-based localization, compared with 0.53–0.54 for the runtime-based selectors. The corresponding numbers of zero-call inputs are 25 and 199; meanwhile, fallback is attempted on 125 and 313 inputs, respectively. These counts show that changing the selector changes how often the shared patcher is invoked, even though its configuration and budget ceilings remain fixed. They measure logical requests, including reused frozen responses, rather than new inference requests or wall-clock cost. **The localization study therefore controls the available budget, not the realized number of model calls.**

G Additional Qualitative Results

We render five selected successful repairs to illustrate local hole insertion, correction of feature operations, and removal of surplus construction. All models are obtained by executing the archived command sequences; no repair output is regenerated or edited for these figures. Models within each row share a camera and world-to-image scale, although scales differ across rows. The examples are selected for interpretable edits and, where a reference is shown, close agreement with that evaluation reference; they do not estimate the frequency of these outcomes. Reference models are shown only for evaluation and were not available to the repair procedure. The reference-based examples come from the supplementary shared-7B study in Appendix D.3, not the main repair cohort.

G.1 Local Edits with Preserved Boundaries

Figure 14 shows two executable inputs from the repair cohort whose saved profiles lack a required central hole. In both cases, localization selects commands 2–10, and the accepted LLM patch inserts an inner loop before `EndFace`. The four original outline segments are retained, and the extrusion

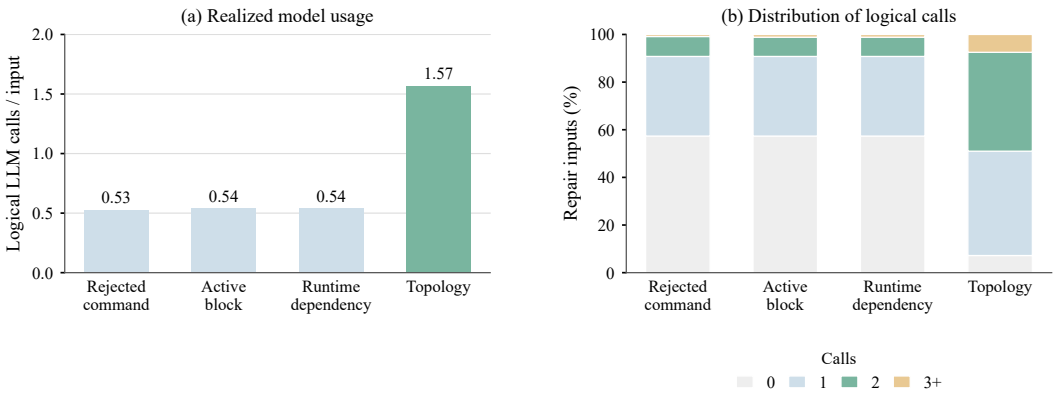

Figure 13: Realized model usage in the matched localization study on all 347 repair inputs. Left: mean logical LLM calls per input. Right: the distribution of call counts under the same patcher, fallback, validation, and budget ceilings. These are usage diagnostics, not an equal-compute comparison.

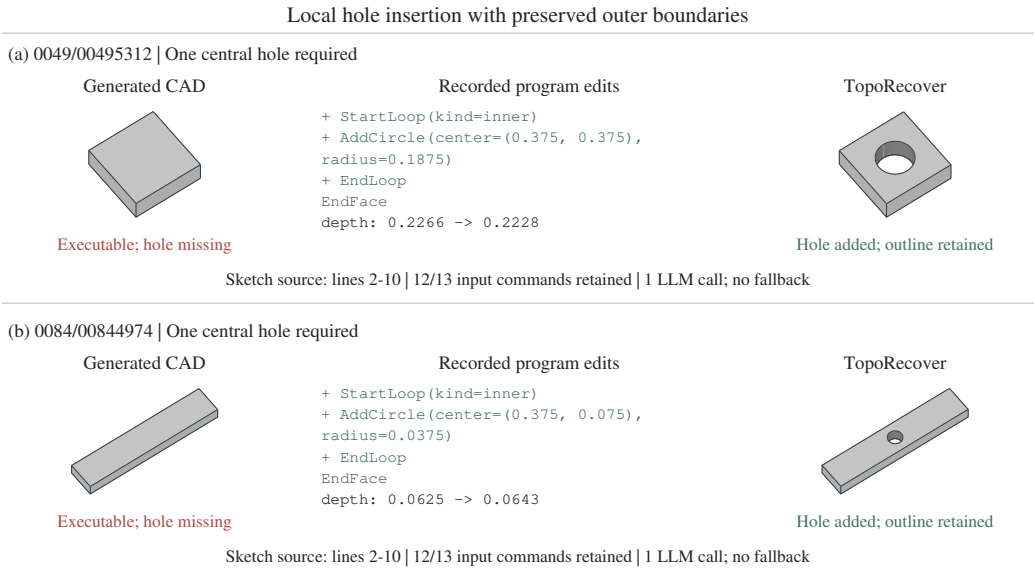

Figure 14: Local repair on a plate and a narrow bar. The central excerpts show inserted loop commands and the depth adjustment; unchanged commands are omitted. Green denotes insertions. Both inputs execute but lack the hole required by their fixed upstream plans.

depth is updated to the supplied plan value. Each repair uses one LLM call without fallback and retains 12 of the 13 input commands. The different aspect ratios show how the inserted hole is placed within the existing geometry; its radius is a proposed parameter, not a recovered reference dimension.

G.2 Repair with Reference-Consistent Geometry

To illustrate successful repair beyond a structural pass label, Figure 15 compares two frozen outputs directly with their evaluation references. In panel (a), the generated program executes but incorrectly adds material where two cuts are required. TopoRecover changes only the operation types at commands 24 and 36, retaining all profile geometry and the other 35 commands. The resulting solid recovers both cutouts and has a measured reference IoU of 1.000; this correction requires no LLM call. In panel (b), the program contains an additional sketch and extrusion that are absent

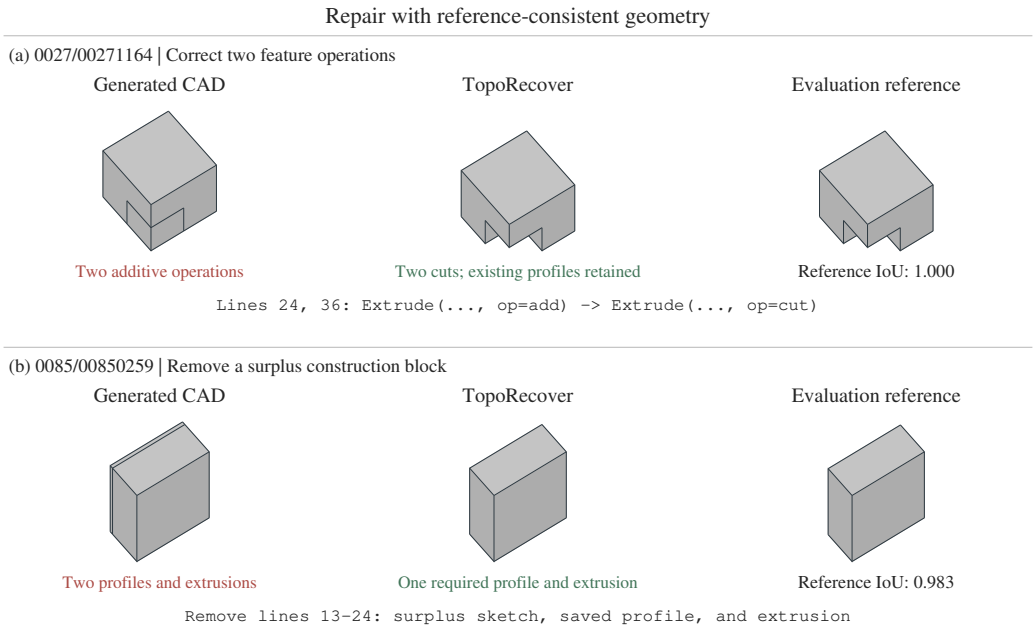

Figure 15: Selected successful repairs from the supplementary reference-conditioned study. Panel (a) changes two additions into cuts; panel (b) removes a surplus construction block. The repaired solids closely match the evaluation references in the converted command representation. Models share a camera and scale within each row, and the displayed command edits come from the frozen outputs. Selection is qualitative, not representative sampling.

from the required construction. The repair removes commands 13–24 while leaving the first profile and its extrusion unchanged, yielding a valid solid with reference IoU 0.983 after one LLM call. **These cases show how correcting the responsible operations or construction block can recover the intended shape while retaining compatible geometry.** Both outputs pass the execution and supplied-requirement checks. The references here use the converted command representation described in Appendix D.3; these examples do not certify fidelity to the original native CAD files.

G.3 Comparison with Adapted Repair Methods

To compare the complete systems on the same input, Figure 16 shows all three adapted baselines alongside TopoRecover and the evaluation reference. The generated program constructs a hollow cylinder, whereas the supplied requirement specifies a solid cylinder without an inner boundary. TopoRecover removes the inner-loop commands while retaining the outer circle and extrusion depth, producing a solid close to the reference with IoU 0.920. CADDesigner retains the unwanted hole (IoU 0.188), while ReCAD removes it but increases the outer radius from 0.2101 to 0.3500 (IoU 0.388). CADCodeVerify also removes the hole and achieves the same IoU as TopoRecover. **This example illustrates successful structural repair with preserved geometry, rather than a case where every competing method fails.** ReCAD passes the plan-conditioned repair criterion despite its geometric discrepancy, which is why both repair status and reference IoU are reported.

H Supported Constraints

Supplied geometry and parameters constrain repair; unspecified coordinates remain choices rather than recovered design intent. LLM proposals use the observed coordinate frame and preserve protected boundaries. When fallback creates new geometry, it also initializes it in the observed frame and binds supported public parameters. Neither path selects an output using reference geometry. Table 11 distinguishes these checks from a general-purpose constraint solver. The main repair co-

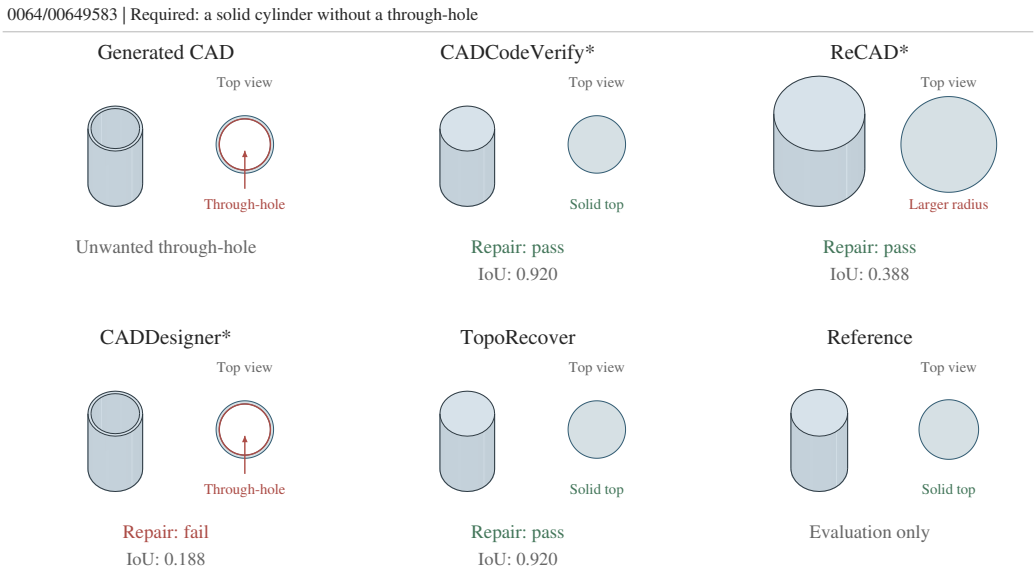

Figure 16: Selected six-way comparison from the supplementary shared-7B evaluation: generated CAD, all three adapted baselines (*), TopoRecover, and the converted-command evaluation reference. The same camera and world scale expose the retained hole in CADDesigner and the enlarged radius in ReCAD. TopoRecover and CADCodeVerify both produce geometry close to the reference. Repair status tests execution and supplied requirements, not exact reference reconstruction; IoU measures reference overlap.

Table 11: Supported checks and their limits. Passing them does not certify unrestricted CAD equivalence.

Input requirement	Check or repair interface	Boundary
Profile boundaries	Required boundary roles; localized loop edits	Requires aligned profile identifiers/order
Feature references	Registered profiles, extrusion references and operation types	Sketch-and-extrusion representation
Hole or outline dimensions	Complete XY circles and axis-aligned rectangles	Explicit fields absent in the main repair cohort
Extrusion depth	Bind and check supplied positive depths	Signed/zero-depth conventions unsupported
Geometric validity	Independent positive-volume OCC execution	No guarantee of intended shape or full topology

hort supplies no explicit profile-geometry fields, so its result does not establish the transfer of the circle/rectangle parameter interface.

I Limitations and Evidence Scope

Attribution and generalization. The adapted systems differ in backbone, budget and incompletely archived prompt details. The main table supports an end-to-end comparison with those systems; attributing the gain to localization alone requires a matched component comparison. The matched intervention in Appendix B.2 isolates access to committed-profile repair under fixed components and budget ceilings. Its gains occur on executable plan-inconsistent inputs; it does not establish a runtime-failure localization advantage, and actual model-call counts differ across selectors. The geometric and repair cohorts answer different questions and are reported separately.

Geometry and scope. Only 22 executable references support the geometric comparison, and paired IoU intervals include zero. Boundary Euler agreement is not full topology equivalence. Exact action LCS can penalize semantically equivalent command sequences and alternative parameterizations. The executor covers sketch-and-extrusion programs through CadQuery/OpenCASCADE; the chamfer in Figure 1 illustrates delayed failure conceptually rather than a tested repair operator. A wrong upstream plan remains a wrong repair target; inferring or correcting design intent is outside the present task. The fixed-order fallback does not compare two accepted candidates for quality, and a locally valid patch may still fail downstream execution.

J REPRODUCIBILITY

The artifact retains public inputs, code snapshots, frozen model responses, final command sequences, per-case checks and result manifests. The main table contains six systems evaluated on separate 347-case repair and 23-case geometry cohorts. Only reference-based measurements exclude the one invalid reference. Component ablations use the same acceptance checks as the current method. No method is selected or retuned using the reported reference metrics.