RESCAD: LANGUAGE-GROUNDED CAD ASSEMBLY VIA INTERFACE RESOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Advances in CAD generation and retrieval make individual components increasingly available, but independently produced parts often lack the assembly semantics required to connect them. We study language-grounded CAD assembly: given a natural-language request and candidate STEP solids for each requested component, the goal is to select the components, ground their connections to boundary-representation (B-rep) faces or edges, and construct a CAD-kernel-verified assembly without predefined assembly interfaces. The central challenge is that multiple component combinations and connection locations can satisfy the same local geometric requirements. We introduce ResCAD, a framework that separates feasibility, grounding, and execution. Its key principle is to use learning to resolve ambiguity left by geometric constraints rather than replace geometric reasoning. A frozen language model compiles the request into a connection graph and interface requirements; deterministic B-rep filtering removes incompatible hypotheses; language-conditioned B-rep rankers resolve component and interface choices; and a CAD kernel computes poses and verifies the resulting assembly. On 114 unseen Fusion 360 assemblies, geometric filtering increases component-assignment recall within 25 proposals from 30.70% to 89.47%. Under the same candidates and feasible set, ResCAD achieves 71.05% complete-connection recall within 25 proposals, compared with 35.09% for an adapted JoinABLe-style scorer. On 50 end-to-end requests with geometrically unique interfaces, ResCAD recovers 82% exact assemblies. These results show that geometric pruning and language-conditioned interface grounding play complementary roles in converting independently produced CAD geometry into verified assemblies. [1]

1 INTRODUCTION

Computer-Aided Design (CAD) is central to modern engineering and manufacturing (Shah & Mäntylä, 1995), and recent language-driven methods can increasingly generate individual CAD components from natural-language descriptions and other high-level specifications (Khan et al., 2024; Li et al., 2025a; Guan et al., 2026; Yuan et al., 2026; Zhang et al., 2025b). However, engineering artifacts are typically assemblies rather than isolated components. Constructing them requires not only selecting the right components, but also identifying where they should connect and ensuring that these local connections form a valid assembly (Jones et al., 2021; Chen et al., 2026).

Recent CAD generation research has begun to move beyond isolated components toward assemblies that account for how parts connect and function together. Methods such as CADKnitter (Le et al., 2025) and ArtiCAD (Shui et al., 2026) incorporate geometric or functional constraints into generation, while SPADA (Zheng et al., 2026) and AssemCAD (Dong et al., 2026) study verifiable assembly generation. Figure 1 (right) highlights two distinct settings: AssemCAD generates parts with explicit ports, whereas JoinABLe predicts mates for given parts (Willis et al., 2022). In practice, independently generated or reused components may lack assembly-specific connection annotations. Turning this available geometry into usable assemblies therefore requires grounding the language-specified component and interface choices, rather than generating annotated parts or assuming that component selection is already given.

[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.

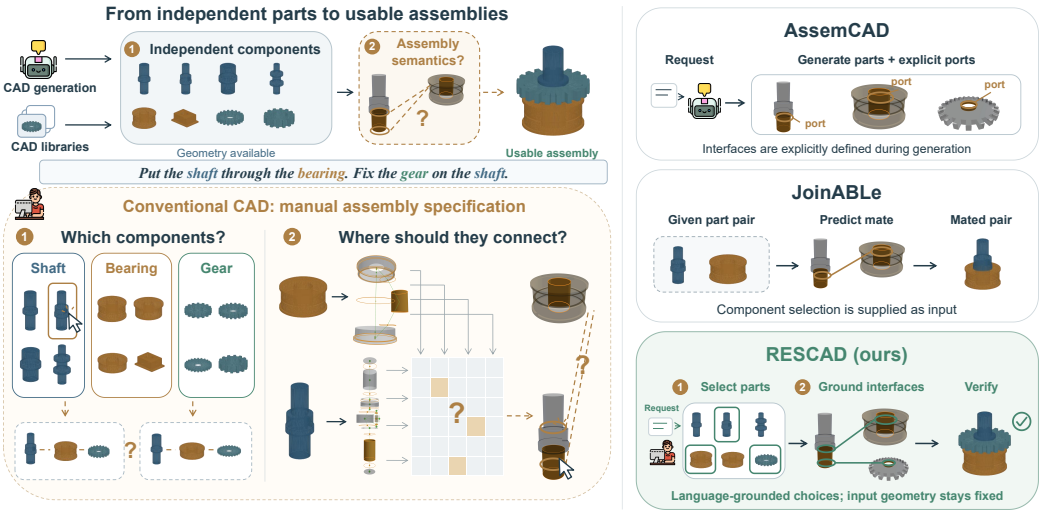


Figure 1: Language-grounded CAD assembly from independent components. The task requires selecting components, grounding requested connections to B-rep interfaces, and verifying the resulting assembly.

Figure 1 illustrates the central difficulty through the request, "*Put the shaft through the bearing, and fix the gear on the shaft*". The system receives multiple unposed shafts, bearings, and gears; in conventional CAD, a designer selects the components and specifies their mating faces or edges (Jones et al., 2021; Willis et al., 2022). These two choices correspond to (1) and (2) on the left of the figure: several component combinations may satisfy coarse geometric requirements, and even a selected shaft may expose multiple plausible seats for the gear. Language describes the requested relations without directly identifying boundary-representation (B-rep) entities, while geometric compatibility alone cannot distinguish the intended connections among these alternatives.

This motivates geometry-constrained interface resolution: learning resolves ambiguity left by geometric constraints rather than replacing geometric checks. We distinguish three questions: feasibility, whether candidates satisfy the specified local geometric requirements; grounding, which components and interfaces realize the language-specified relations; and execution, whether the selected connections can coexist in a globally realizable assembly.

We introduce ResCAD, a framework for language-grounded CAD assembly via geometry-constrained interface resolution. A frozen language model compiles the request into a connection graph and interface requirements; deterministic B-rep rules prune locally incompatible component-interface hypotheses; learned rankers select the intended components and interfaces among the remaining alternatives; and a CAD kernel executes and verifies the resulting assembly.

We compare ResCAD with adapted CAD assembly methods and direct LLMs, achieving 82% exact assembly recovery on 50 requests with geometrically unique interfaces. On a separate 114-request Fusion test with interface ambiguity, complete-connection recall within 25 proposals reaches 71.05%, versus 35.09% for adapted JoinABLe under the same filter and candidates. Ablations isolate the benefits of geometric pruning and language-conditioned interface ranking.

The key contributions of this paper are as follows.

- We formulate language-grounded CAD assembly as resolving language-specified relations on independently produced CAD components without predefined assembly interfaces.
- We propose geometry-constrained interface resolution, combining deterministic B-rep feasibility filtering, learned language-conditioned component and interface ranking, and CAD-kernel verification.
- We show that geometry and learning play complementary roles, with the largest learned gains arising in interface resolution after geometric pruning.

2 Related Work

CAD Generation and Assembly. CAD generation has been studied through construction sequences, programmatic representations, and direct B-rep modeling (Wu et al., 2021; Willis et al., 2021; Lambourne et al., 2021). Recent language- and multimodal-conditioned methods further generate or reconstruct editable CAD programs from text (Khan et al., 2024; Guan et al., 2026; Zhang et al., 2025b; Li et al., 2026), images (Wang et al., 2025; Li et al., 2025c), and point clouds (Liu et al., 2026a). Despite this progress, these methods primarily generate or reconstruct individual components and do not resolve how independently produced parts should be selected and connected. Assembly-oriented work has begun to model part hierarchies and inter-part constraints (Gong et al., 2026), retrieves parts from language (Liu et al., 2026b), predict mates or actions for a given set of parts (Jones et al., 2021; Willis et al., 2022; Chen et al., 2026), or synthesize assembly-aware components and ports under constraint or test-driven loops (Zheng et al., 2026; Dong et al., 2026). In contrast, we assemble independently produced STEP components by selecting the requested parts and grounding language-specified relations to latent interfaces in their existing B-rep geometry.

3D Assembling. 3D assembly estimates rigid poses for given parts or fragments. Correspondence-based approaches match fracture surfaces (Lu et al., 2023),while scalable and diffusion formulations address multi-part ambiguity (Zhang et al., 2024; Scarpellini et al., 2024; Li et al., 2025b). Point-flow and anchor-based methods instead generate assembled point configurations before recovering rigid transforms (Sun et al., 2026; Zhao et al., 2025). Hierarchical, symmetry-aware models improve global coherence (Zhang et al., 2026) and instruction-conditioned methods infer assembly order and poses from manuals (Zhang et al., 2025a). Assembly completion can also retrieve missing parts from a candidate toolkit (Wang et al., 2023), and holistic generative priors can synthesize missing geometry (Jiang et al., 2026). These methods primarily operate on meshes or point clouds and focus on recovering part poses or complete shapes. In contrast, language-grounded CAD assembly must ground requested relations to exact B-rep interfaces and verify their joint CAD-kernel execution.

3 CAD Assembly: The Problem

We consider a CAD assembly as a set of CAD components connected by assembly relations. Each component is represented as a B-rep solid in its own local coordinate system. An assembly relation specifies how two components should connect or move relative to each other, but must ultimately be realized on concrete local geometry. We call the B-rep face or edge selected on each component an assembly interface. A set of grounded relations forms a valid assembly only if all relations can be realized simultaneously by a CAD kernel without unintended solid intersections. This reflects our setting, where components are unposed and contain neither predefined assembly ports nor stable face identifiers.

Language-grounded CAD assembly. Given a natural-language request q and a collection of available CAD components $\mathcal{C}$, the task is to recover the assembly grounding

$$Y = (G, \mathbf{c}, \boldsymbol{\iota}),$$

where $G = (V, E)$ describes the component entities and assembly relations expressed by q, $\mathbf{c}$ assigns each entity to a concrete CAD component from $\mathcal{C}$, and $\boldsymbol{\iota}$ grounds every relation to a pair of B-rep interfaces on the selected components. Thus, the task jointly determines what should be connected, which components should be used, and where each connection should be realized. The request specifies these decisions only semantically; it does not directly provide component identities, B-rep entity indices, or rigid transformations. A CAD executor then realizes the grounding,

$$\mathcal{A} = \mathrm{Exec}(Y),$$

by computing a consistent assembly configuration and verifying the requested relations and inter-component collisions. The goal is therefore to recover the intended grounding $Y^\star$ from q such that $\mathrm{Exec}(Y^\star)$ yields a valid CAD assembly.

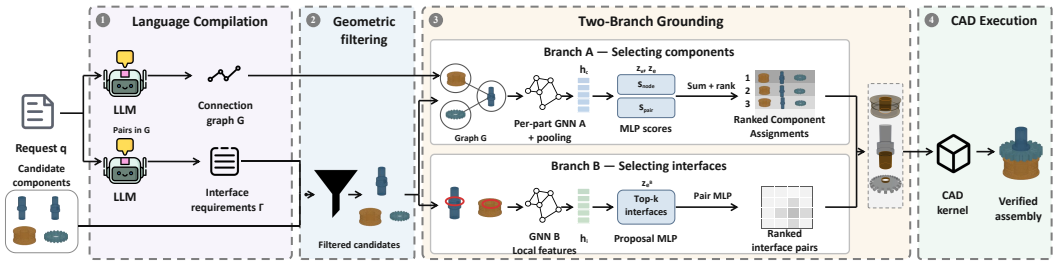


Figure 2: Overview of the RESCAD framework.

4 RESCAD: GEOMETRY-CONSTRAINED INTERFACE RESOLUTION

4.1 OVERVIEW

Our central idea is to separate *feasibility*, *grounding*, and *execution*. Figure 2 follows one request from candidate parts to an assembled STEP model. A frozen LLM first identifies the requested connections and their interface requirements (Section 4.2). A geometric filter removes candidates lacking the required interfaces. Two separately parameterized B-rep GNNs then resolve the remaining choices: branch A selects components, and branch B selects their local connection interfaces (Section 4.3). Finally, the CAD kernel computes part poses and checks the complete assembly (Section 4.4). This division lets geometry enforce explicit requirements and learning distinguish the remaining alternatives using language. Only the two grounding branches are trained; the language compiler, geometric filter, and CAD executor are fixed.

4.2 LANGUAGE COMPILATION

Given a natural-language request q and the available component entities V with their supplied names or descriptions, RESCAD uses a frozen language model to compile the request into

$$(G, \Gamma) = \mathrm{Compile}_{\mathrm{LM}}(q, V), \tag{1}$$

where $G = (V, E)$ specifies the requested assembly relations and $\Gamma = \{\gamma_e\}_{e \in E}$ describes how these relations should be realized geometrically. The resulting G determines which component entities must be connected, while Γ provides interface requirements for resolving the corresponding components and interfaces in the next stage.

For each relation $e = (u, v) \in E$, the compiler produces

$$\gamma_e = (\gamma_{e,u}, \gamma_{e,v}, m_e), \tag{2}$$

where $\gamma_{e,u}$ and $\gamma_{e,v}$ describe the local geometry referenced on the two connected components, and m_e specifies the requested relation or motion. Each endpoint description is expressed using intrinsic B-rep properties, including entity kind, analytic geometry, relative size, neighboring geometry, and repeated-feature status. Properties not specified by the request are left unconstrained.

The compiler operates only on the request and supplied entity names or descriptions; it does not observe candidate geometry, B-rep indices, source joints, or target assembly annotations. We additionally convert each γ_e into a canonical textual description used to condition the learned rankers. Fixed prompts, structured outputs, and deterministic decoding are used throughout; full prompt and output-schema details are provided in Appendix B.

4.3 TWO-BRANCH GROUNDING

The two branches in Figure 2 resolve different choices: branch A ranks complete component assignments, and branch B ranks the interfaces that connect those components. Both take the candidate B-reps and compiled descriptions as input, but use separately parameterized graph neural networks (GNNs). For each component c, we construct a face-adjacency graph $\mathcal{B}(c)$: nodes carry intrinsic surface features, and edges describe shared boundaries and angles between neighboring

faces. This geometric graph is distinct from the requested connection graph G, which specifies which components must connect. Let $\mathcal{I}(c)$ denote its candidate face or edge interfaces, grouping primitives with indistinguishable intrinsic geometry and neighborhoods. No global poses or face identifiers are used as encoder features.

Geometric filtering. Before ranking, we identify which interfaces match the endpoint properties in Γ. A component pair is retained for relation $e = (u, v)$ when each component has at least one matching interface:

$$F_e(c_u, c_v) = \mathbb{I}[\exists i \in \mathcal{I}(c_u) : i \models \gamma_{e,u} \ \wedge \ \exists j \in \mathcal{I}(c_v) : j \models \gamma_{e,v}] \,. \tag{3}$$

Here, $\models$ means agreement with every specified property, such as surface type or neighboring geometry; unspecified properties impose no restriction. This inexpensive test checks the two endpoints separately. Their actual fit and collisions are checked during CAD execution. A complete assignment must pass this test for every requested connection.

Branch A: selecting components. GNN A encodes each candidate graph $\mathcal{B}(c)$ using three message-passing layers. Starting from a learned embedding $h_f^{(0)}$ of each face's intrinsic features, it aggregates information from adjacent faces $\mathcal{N}(f)$:

$$\begin{aligned} m_f^{(\ell)} &= \mathrm{mean}_{g \in \mathcal{N}(f)} \, M_\ell(h_g^{(\ell)}, h_f^{(\ell)}, b_{gf}), \\ h_f^{(\ell+1)} &= U_\ell(h_f^{(\ell)}, m_f^{(\ell)}). \end{aligned} \tag{4}$$

Here, b_{gf} describes the shared boundary, M_ℓ computes a learned message, and U_ℓ updates the receiving face using the aggregated messages. After three layers, we pool face embeddings within geometric groups and across the component to obtain its geometric and topological summary h_c.

We next score how well each candidate and connected candidate pair match the request. Let z_v encode the input-provided name or description of component entity v, and let z_e encode the canonical relation description γ_e. Two multilayer perceptrons (MLPs) output scalar matching scores:

$$\begin{aligned} s_{\mathrm{node}}(c, v) &= f_{\mathrm{node}}(h_c, z_v), \\ s_{\mathrm{pair}}(a, b, e) &= f_{\mathrm{pair}}(h_a, h_b, z_e). \end{aligned} \tag{5}$$

The first scores a candidate against the component description; the second scores two candidates against their requested relation. For a complete assignment $\mathbf{c}$, we add the node scores and the pair scores only along connections in G:

$$S_{\mathrm{comp}}(\mathbf{c}) = \sum_{v \in V} s_{\mathrm{node}}(c_v, v) + \sum_{e=(u,v) \in E} s_{\mathrm{pair}}(c_u, c_v, e). \tag{6}$$

For example, a shaft–bearing–gear assignment receives three node scores plus the shaft–bearing and shaft–gear relation scores, using the same shaft candidate in both relations. Beam search excludes pairs failing Eq. 3 and retains up to 25 high-scoring complete assignments, producing the ranked component combinations shown in branch A.

Branch B: selecting interfaces. GNN B reads the same candidate graphs $\mathcal{B}(c)$ using Eq. 4 with independent parameters. Unlike GNN A, it retains local group embeddings rather than pooling them into a whole-component vector. For a face interface i, we combine its group embedding with encoded primitive features; edge interfaces are encoded from their intrinsic edge features. The resulting h_i represents a connection location rather than the entire solid.

For relation e, branch B encodes the full canonical description with its own text projection, yielding z_e^B shared by both endpoints. A proposal MLP ranks individual interfaces and retains up to 16 per candidate component; a pair MLP then ranks cross-component combinations:

$$\begin{aligned} s_{\mathrm{prop}}(i \mid e) &= f_{\mathrm{prop}}(h_i, z_e^B), \\ S_{\mathrm{int}}(i, j \mid e) &= f_{\mathrm{int}}(h_i, h_j, z_e^B). \end{aligned} \tag{7}$$

This ranking distinguishes local alternatives, such as different seats on the same shaft, using the requested relation. For each assignment retained by branch A, we take the eight highest-scoring interface pairs per connection as alternatives for CAD execution. When the endpoint constraints uniquely identify both interfaces, we use that pair directly. Branch B supplies the interface choices $\boldsymbol{\iota}$ without changing A's component scores, completing $Y = (G, \mathbf{c}, \boldsymbol{\iota})$.

Training. Reference assemblies provide the target components and interfaces. We train the GNNs, text projections, and scoring MLPs, collectively parameterized by θ, to minimize

$$\min_{\theta} \mathbb{E}_{\mathcal{D}}[\mathcal{L}_{\text{node}} + \mathcal{L}_{\text{pair}} + \lambda_{\text{int}}(\mathcal{L}_{\text{prop}} + \mathcal{L}_{\text{int}}) + \lambda_{\text{rank}}\mathcal{L}_{\text{rank}} + \mathcal{L}_{\text{aux}}], \tag{8}$$

over training set $\mathcal{D}$. The four cross-entropy terms supervise component choices, connected component pairs, interface proposals, and interface pairs, respectively. $\mathcal{L}_{\text{rank}}$ favors the reference assignment over one-component substitutions, while $\mathcal{L}_{\text{aux}}$ supervises support and motion types where applicable. Only the two grounding branches are trained; compilation, filtering, discrete search, and CAD execution are outside backpropagation. Appendix C details encoder updates, scoring inputs, and losses.

4.4 CAD Execution

Given the recovered grounding $Y = (G, \mathbf{c}, \boldsymbol{\iota})$, we use a CAD kernel to realize the selected relations on the original STEP solids and verify whether they form a globally consistent assembly.

Local realization. For each relation $e = (u, v)$, the selected interface pair ι_e defines local geometric frames on the two components. Using the relation type m_e, we enumerate rigid transformations consistent with the required alignment and motion constraints. Candidate transformations are evaluated directly on the original B-rep solids and rejected if the requested interfaces cannot be realized without unintended positive-volume intersection.

Global composition and verification. We anchor one component and propagate locally valid transformations through the assembly graph. Partial configurations are rejected if newly placed components collide with existing geometry; cyclic relations must remain consistent. After placing all components, the CAD kernel re-checks every requested relation and component pair, returning $\mathcal{A} = \text{Exec}(Y)$ only if all relations hold and no unintended collision remains.

Inference. We consider retained component assignments in descending score order and evaluate their ranked interface alternatives with the executor. Invalid assemblies are discarded, and search continues until a valid assembly is found or the fixed inference budget is exhausted.

5 Experiments

5.1 Experimental setup

Dataset. We derive requests and unposed STEP candidates from Fusion 360 Gallery (Willis et al., 2021). Training and development contain 480/50 requests and 1,720/181 connections; the primary test contains 114 unseen assemblies with 409 connections. Additional tests comprise 50 matched end-to-end requests, two 50-request sets with eight candidates per entity, and 37 requests with 577 Text-to-CadQuery components (Xie & Ju, 2025). Test STEP files are disjoint from training and other test sets.

Baselines. JoinABLe* (Willis et al., 2022) learns cross-part B-rep scores for interface matching. AssemCAD* (Dong et al., 2026) combines LLM component ranking with typed-interface matching. SPADA* (Zheng et al., 2026) refines predicted interfaces and poses using geometric feedback. All use matched requests, candidates, and metrics; Appendix D details the adaptations. Geometry-only and learned ablations isolate our modules.

Models. Direct baselines use DeepSeek-V4-Pro (DeepSeek-AI, 2026), Kimi K3 (Team et al., 2026), GPT-5.6 Sol (OpenAI, 2026), and Gemini 3.8 Flash (Google DeepMind, 2026) with a shared two-call selection/pose protocol. The frozen compiler and language-based adapters use DeepSeek-V4-Pro; Appendix D records API identifiers and decoding settings.

Metrics. *Exact assembly* requires every intended component, interface, and relation in a collision-free, kernel-accepted assembly. Assignment R@25 measures component-assignment recall; Complete-connection R@25 also requires intended interfaces, alignments, and motions; Verified R@25 additionally requires CAD execution. Interface Top-k measures local interface retrieval.

Table 1: Assembly results on 50 matched requests. Common CAD metrics measure output geometry; All parts and Exact measure recovery of the requested assembly.

	CAD geometry metrics						Task success	
				Shape Acc.				
Method	Valid ↑	CD† ↓	mIoU ↑	@1% ↑	F@1% ↑	Coll.-free ↑	All parts ↑	**Exact** ↑
Related methods								
JoinABLe*	**100.0**	31.29	24.20	24.0	24.67	**100.0**	0.0	0.0
AssemCAD*	**100.0**	**12.50**	71.27	67.0	71.98	**100.0**	16.0	16.0
SPADA*	96.0	87.96	21.54	21.5	21.57	92.0	16.0	0.0
General-purpose LLMs								
DeepSeek-V4-Pro	96.0	87.96	21.54	21.5	21.57	92.0	16.0	0.0
Kimi K3	72.0	101.70	14.02	14.0	14.04	64.0	10.0	0.0
GPT-5.6 Sol	78.0	15.61	24.40	18.5	25.20	0.0	0.0	0.0
Gemini 3.8 Flash	18.0	50.35	4.51	4.5	4.55	14.0	6.0	0.0
Our method and ablations								
Rules + executor	98.0	56.56	43.65	41.0	44.16	98.0	10.0	10.0
ResCAD w/o interface rule	96.0	15.07	73.54	73.0	73.78	96.0	**82.0**	44.0
ResCAD	96.0	18.35	**84.00**	**84.0**	**84.00**	96.0	**82.0**	**82.0**

*Adapted to our task; see Appendix D. †CD is conditional on scoreable pairs; all other columns use all 50 requests. Values are percentages.

Following prior CAD evaluation (Chen et al., 2026; Jiang et al., 2026; Wang et al., 2026), we also report STEP validity, collision freedom, mean volume IoU, Chamfer Distance, Shape Accuracy, and F-score.

Implementation details. Each B-rep GNN has three width-64 message-passing layers; text uses 128-dimensional hashed word/character-trigram features and learned MLP projections. We train with AdamW (learning rate 3×10^{-4}; weight decay 10^{-4}). Inference retains 25 component assignments, up to 16 interfaces per component, and eight interface pairs per connection. We report three-seed medians with fixed test-time checkpoints; Appendix C details training and adaptation.

5.2 End-to-End Evaluation

To test recovery of the requested assembly, we evaluate all methods on 50 matched requests and unposed STEP candidates with a shared structured frontend. Search-based methods execute up to 25 proposals; direct LLMs return one assembly; SPADA* receives two geometric-feedback rounds. We re-import product-structured STEP outputs, counting missing outputs as failures. Unique constraint-compatible interfaces isolate component selection and executable composition. To distinguish geometric validity from request recovery, Table 1 reports both: ResCAD achieves 82% Exact versus 16% for AssemCAD* and 0% for JoinABLe* and SPADA*, although these baselines are 92–100% collision-free. Furthermore, direct LLMs achieve 18–96% STEP validity but 0% Exact; GPT-5.6 Sol has lower CD than ResCAD (15.61% versus 18.35%) but no collision-free output. Thus, surface similarity alone does not establish assembly correctness. Additionally, learned ranking of geometry-determined interfaces lowers Exact from 82% to 44% while component selection stays at 82%, supporting deterministic selection of unique interfaces and learning for ambiguity. Overall, **success requires correct grounding and joint geometric realizability**.

5.3 Component Analysis

To isolate filtering and learning, we compare four variants on 114 Fusion requests with a fixed 25-assignment budget. *Learned, no filter* omits endpoint filtering; *Learned, filter at test* filters that frozen model only at inference; ResCAD also trains with the filter. JoinABLe* uses a geometry-based learned scorer with the same filter and candidates, controlling for pruning and search budget.

Figure 3 shows that test-time filtering raises assignment R@25 from 30.70% to 89.47% and complete-connection R@25 from 25.44% to 71.93%. Training with the filter gives similar results, attributing the main gain to search-space reduction. Furthermore, JoinABLe* reaches 84.21% assignment recall

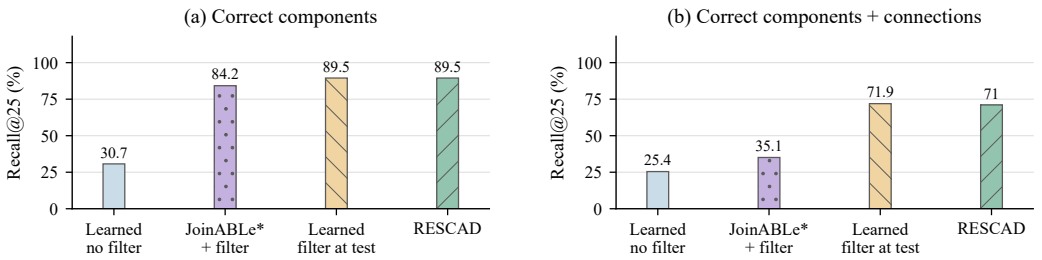


Figure 3: Component and complete-connection recall on 114 requests (three-seed medians).

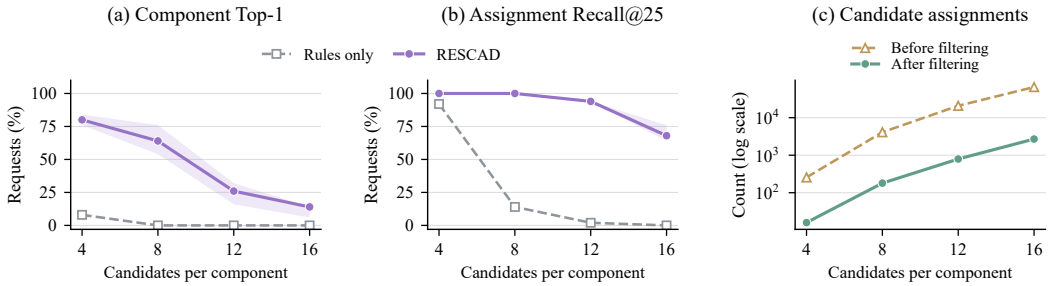


Figure 4: Candidate scaling on 50 requests: seed medians (lines) and three-seed ranges (shading).

but only 35.09% complete-connection recall versus 71.05% for RESCAD, showing that compatible parts still leave interface ambiguity.

To isolate interface scoring, we fix candidates and filtering on a separate 88-assembly test. Learned scoring improves complete-connection R@25 by 9.09 points (paired 95% interval: [3.41,17.05]) and conditional interface Top-8 by 15.76 points [8.37,23.51] (Appendix Table 9). Additionally, replacing parsed relations with raw-text embeddings lowers complete-connection R@25 from 71.05% to 42.98%. Together, **language-conditioned local ranking improves recovery beyond geometric pruning**.

5.4 CANDIDATE SCALING

To test scaling with candidate count, we vary nested pools from 4 to 8, 12, and 16 per component, fixing 50 four-component requests, three checkpoints, and a 25-assignment budget. The eight-candidate pool is unchanged; smaller pools retain the target and three fixed-order distractors; larger pools add distinct STEP candidates from other requests at the same component position. Only competing candidates change: models are not retrained, and CAD execution is not re-evaluated.

To assess ranking under larger candidate pools, Figure 4 reports assignment recall: RESCAD reaches 100% and 94% assignment R@25 at 8 and 12 candidates, versus 14% and 2% for rules-only ranking. Furthermore, it retains 68% recall at 16 candidates, where rules-only recall is zero and filtering reduces the median search space from 65,536 to 2,694 assignments. Overall, **learned ranking outperforms rules over 4–16 candidates**, but larger pools reduce recall under a fixed budget.

5.5 CROSS-SOURCE TRANSFER

To test transfer to generated components, we use 37 requests with 577 valid Text-to-CadQuery parts sharing compatible connection diameters but differing in requested shapes. We compare the frozen Fusion model, rules-only ranking, component-ranker adaptation, and adaptation of both rankers. Adaptation uses separate generated-part training/development pools, with the filter, executor, and 25-proposal budget fixed (Appendix F).

To separate component and interface adaptation, Figure 5 reports that component-only adaptation raises assignment R@25 from 48.65% to 100%, but verified recall reaches only 13.51%. Furthermore,

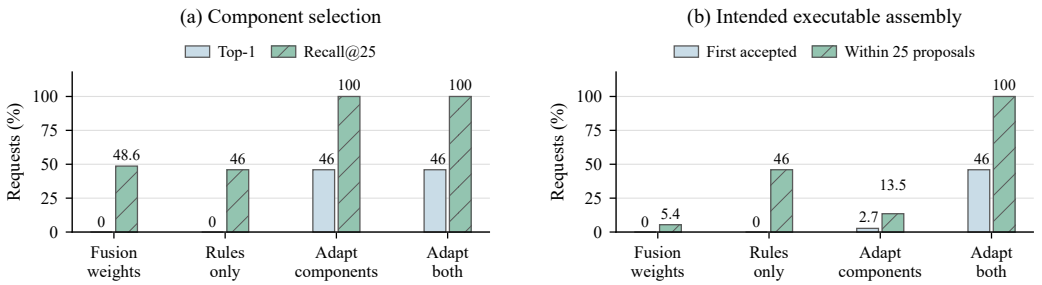


Figure 5: Transfer to 37 generated-part requests: component selection and executable assembly.

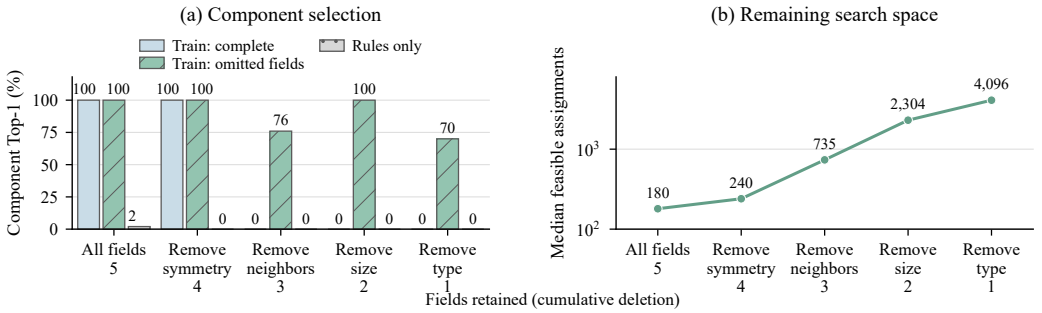


Figure 6: Robustness to omitted description fields on 50 requests (three-seed medians).

adapting the interface branch raises verified recall to 100% under the same budgets, although the first accepted assembly is intended on only 45.95% of requests. Overall, **adapting both rankers supports cross-source recovery**; geometric acceptance alone cannot resolve ranking ambiguity.

5.6 Robustness Analysis

We test underspecified requests by cumulatively removing repeated-feature status, neighboring surfaces, relative size, and analytic type from 50 held-out requests. Unspecified properties remain unconstrained, and targets stay feasible. We compare training with omitted fields against complete-description training, fixing geometry, labels, and training updates.

Figure 6 shows that removing fields expands the median feasible set from 180 to 4,096 assignments. Complete-description training falls to 0% all-component Top-1 once neighboring surfaces are omitted. In contrast, omitted-field training retains 76%, 100%, and 70% with three, two, and one fields; different competing candidates explain the non-monotonic trend. **Overall, omitted-field training improves robustness without requiring the filter to guess missing properties.**

6 Conclusion and Future Work

We studied language-grounded assembly of independently produced CAD components without predefined interfaces. ResCAD combines language-derived constraints, geometric filtering, learned component and interface resolution, and CAD-kernel verification. Our results show that feasibility, correct grounding, and global realizability require complementary mechanisms; learning contributes most when multiple feasible interpretations remain. These findings suggest that explicit geometric reasoning and learned semantic resolution are most effective when used together, rather than treating either as a complete substitute for the other.

Our setting assumes candidate pools containing the required components and a finite set of assembly relations. Future work includes open-domain retrieval, free-form component discovery, richer kinematic reasoning, and more scalable search over larger and more ambiguous candidate spaces.

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 data and outputs were checked through the documented validation procedures and CAD-kernel 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 uses research CAD datasets, natural-language assembly requests, and model-generated outputs, and does not involve human-subject experiments. The study is intended for research on automated CAD assembly. Generated assemblies are verified using geometric constraints and CAD-kernel execution within the experimental pipeline and should not be interpreted as independently certified engineering designs.

REPRODUCIBILITY STATEMENT

The codebase and dataset are provided through the anonymous repositories linked in the paper. The appendix documents the fixed language-model prompts, structured output schemas, decoding procedure, training and inference configurations, geometric filtering and CAD execution procedures, baseline adaptations, and train–development–test separation. These materials are provided to support reproduction of both the proposed method and the reported experimental results.

REFERENCES

Jiali Chen, DingBa Fu, Xusen Hei, Yuhang Liu, Yiyang Chen, Jiayuan Xie, Wenqi Fan, and Yi Cai. Cadmate: Generating cad assembly plan with geometric chain-of-thought and spatial physical rewards. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18324–18348, 2026.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Yurui Dong, Shu Zou, Siqi Li, Nianchen Deng, Hongbin Zhou, Xuemeng Yang, Pinlong Cai, Licheng Wen, Xinyu Cai, and Botian Shi. Assemcad: Production-ready cad assembly generation from natural language. *arXiv preprint arXiv:2607.05123*, 2026.

Shengjie Gong, Wenjie Peng, Hongyuan Chen, Gangyu Zhang, Yunqing Hu, Huiyuan Zhang, Shuangping Huang, and Tianshui Chen. Learning hierarchical and geometry-aware graph representations for text-to-CAD. In *The Fourteenth International Conference on Learning Representations*, 2026.

Google DeepMind. Gemini 3.8 Flash model card, 2026. URL `https://deepmind.google/models/model-cards/gemini-3-8-flash/`.

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.

Zeyu Jiang, Sihang Li, Siqi Tan, Chenyang Xu, Juexiao Zhang, Julia Galway-Witham, Xue Wang, Scott A Williams, Radu Iovita, Chen Feng, et al. Crag: Can 3d generative models help 3d assembly? *arXiv preprint arXiv:2602.22629*, 2026.

Benjamin Jones, Dalton Hildreth, Duowen Chen, Ilya Baran, Vladimir G Kim, and Adriana Schulz. Automate: A dataset and learning approach for automatic mating of cad assemblies. *ACM Transactions on Graphics (TOG)*, 40(6):1–18, 2021.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential CAD designs from beginner-to-expert level text prompts. In *Advances in Neural Information Processing Systems*, volume 37, pp. 7552–7579, 2024.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *2019 IEEE/CVF conference on computer vision and pattern Recognition (CVPR)*, pp. 9593–9603. Ieee, 2019.

Joseph G. Lambourne, Karl D. D. Willis, Pradeep Kumar Jayaraman, Aditya Sanghi, Peter Meltzer, and Hooman Shayani. Brepnet: A topological message passing system for solid models. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 12773–12782. Computer Vision Foundation / IEEE, 2021.

Tri Le, Khang Nguyen, Baoru Huang, Tung D Ta, and Anh Nguyen. Cadknitter: Compositional cad generation from text and geometry guidance. *arXiv preprint arXiv:2512.11199*, 2025.

Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. Cad-llama: leveraging large language models for computer-aided design parametric 3d model generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18563–18573. IEEE, 2025a.

Jiahao Li, Qingwang Zhang, Qiuyu Chen, Guozhan Qiu, Yunzhong Lou, and Xiangdong Zhou. Towards high-fidelity CAD generation via LLM-driven program generation and text-based b-rep primitive grounding. In *Forty-third International Conference on Machine Learning*, 2026.

Sihang Li, Zeyu Jiang, Grace Chen, Chenyang Xu, Siqi Tan, Xue Wang, Irving Fang, Kristof Zyskowski, Shannon P McPherron, Radu Iovita, et al. Garf: Learning generalizable 3d reassembly for real-world fractures. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5711–5721. IEEE, 2025b.

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, 2025c.

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, 2026a.

Yunqing Liu, Nan Zhang, and Zhiming Tan. Error notebook-guided, training-free part retrieval in 3d cad assemblies via vision-language models. In *International Conference on Learning Representations*, volume 2026, pp. 135254–135273, 2026b.

Jiaxin Lu, Yifan Sun, and Qixing Huang. Jigsaw: Learning to assemble multiple fractured objects. *Advances in Neural Information Processing Systems*, 36:14969–14986, 2023.

OpenAI. GPT-5.6 system card, 2026. URL `https://deploymentsafety.openai.com/gpt-5-6`.

Dacheng Qi, Chenyu Wang, Jingwei Xu, Tianzhe Chu, Zibo Zhao, Wen Liu, Wenrui Ding, Yi Ma, and Shenghua Gao. Pointer-cad: Unifying b-rep and command sequences via pointer-based edges & faces selection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17377–17387, June 2026.

Gianluca Scarpellini, Stefano Fiorini, Francesco Giuliari, Pietro Moreiro, and Alessio Del Bue. Diffassemble: A unified graph-diffusion model for 2d and 3d reassembly. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 28098–28108, June 2024.

Jami J. Shah and Martti Mäntylä. *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. John Wiley & Sons, 1995.

Yuan Shui, Yandong Guan, Zhanwei Zhang, Juncheng Hu, Jing Zhang, Dong Xu, and Qian Yu. Articad: Articulated cad assembly design via multi-agent code generation. *arXiv preprint arXiv:2604.10992*, 2026.

Tao Sun, Liyuan Zhu, Shengyu Huang, Shuran Song, and Iro Armeni. Rectified point flow: Generic point cloud pose estimation. *Advances in Neural Information Processing Systems*, 38:168960–168989, 2026.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Liang Wang, Heng Meng, Zekai Xiang, Jin Liu, Pingyi Zhou, Litao Chen, and Yongqiang Tang. Text2cad-bench: A benchmark for llm-based text-to-parametric cad generation. *arXiv preprint arXiv:2605.18430*, 2026.

Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. Cad-gpt: Synthesising cad construction sequence with spatial reasoning-enhanced multimodal llms. In *Proceedings of the AAAI conference on artificial intelligence*, volume 39, pp. 7880–7888, 2025.

Weihao Wang, Rufeng Zhang, Mingyu You, Hongjun Zhou, and Bin He. 3d assembly completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 2663–2671, 2023.

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.

Karl DD Willis, Pradeep Kumar Jayaraman, Hang Chu, Yunsheng Tian, Yifei Li, Daniele Grandi, Aditya Sanghi, Linh Tran, Joseph G Lambourne, Armando Solar-Lezama, et al. Joinable: Learning bottom-up assembly of parametric cad joints. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15828–15839. IEEE, 2022.

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.

Haoyang Xie and Feng Ju. Text-to-cadquery: A new paradigm for cad generation with scalable large model capabilities. *arXiv preprint arXiv:2505.06507*, 2025.

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.

Jiahao Zhang, Anoop Cherian, Cristian Rodriguez, Weijian Deng, and Stephen Gould. Manual-pa: Learning 3d part assembly from instruction diagrams. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6304–6314. IEEE, 2025a.

Ruiyuan Zhang, Jiaxiang Liu, Zexi Li, Hao Dong, Jie Fu, and Chao Wu. Scalable geometric fracture assembly via co-creation space among assemblers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 7269–7277, 2024.

Xinyi Zhang, Bingyang Wei, Ruixuan Yu, and Jian Sun. Coarse-to-fine 3d part assembly via semantic super-parts and symmetry-aware pose estimation. *Advances in Neural Information Processing Systems*, 38:171393–171420, 2026.

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, 2025b.

Wang Zhao, Yan-Pei Cao, Jiale Xu, Yuejiang Dong, and Ying Shan. Assembler: Scalable 3d part assembly via anchor point diffusion. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, pp. 1–11, 2025.

Keyou Zheng, Xuyang Su, and Jiewu Leng. SPADA: A verifiable test-driven agent for controllable parametric CAD assembly generation. In *Forty-third International Conference on Machine Learning*, 2026.

Part I

Appendix

Table of Contents

A Geometric Properties Extracted from Language **15**

B Language Parsing Details **15**

B.1 Inputs, outputs, and decoding 15

B.2 Fixed system prompts 16

B.3 A recorded input–output example 16

B.4 Evaluation and reuse of saved predictions 19

C Training and Inference Details **20**

D Comparison with Assembly Baselines **22**

E Train, Development, and Test Separation **23**

F Details for Text-to-CadQuery Parts **23**

G Additional Interface-Selection Results **25**

H Illustrative Assembly Tasks **27**

I Contact-Complete Execution Details **27**

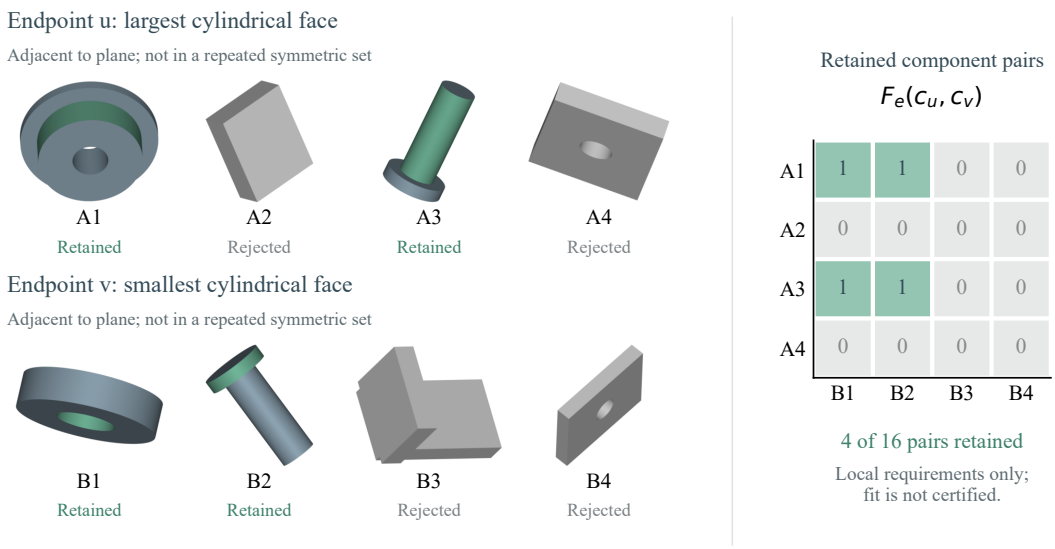


Figure 7: Endpoint-wise geometric filtering on a recorded candidate pool. Green faces match the requirements; the matrix shows the retained component pairs.

A Geometric Properties Extracted from Language

For each side of a requested connection, the parser returns five fields derived from standard B-rep entities (Lambourne et al., 2021; Koch et al., 2019). They specify face or edge, analytic geometry type, size relative to similar entities on the same part, neighboring surface types, and whether the entity belongs to a repeated symmetric group. Relative size takes one of `only`, `smallest`, `small`, `medium`, `large`, or `largest`. The geometry types include planar, cylindrical, conical, spherical, toroidal, and spline faces, as well as circular, linear, elliptical, and spline edges. The current executor constructs local frames for planes, cylinders, circles, and lines. Other combinations return unsupported.

A filtering example. To illustrate endpoint-wise filtering, Figure 7 shows all candidates for one recorded connection and applies the same B-rep predicate used in evaluation. Two candidates at each endpoint contain a matching interface, leaving four of the 16 component pairs. The highlighted faces satisfy the local requirements; this does not certify pairwise fit or a valid complete assembly.

B Language Parsing Details

This section specifies the prompts, inputs, and saved outputs used to obtain the connection graph G and geometric requirements Γ. The two language tasks are whole-request graph prediction and per-connection attribute parsing. The implementation uses the keys `role` and `port` for a requested component and its endpoint description, respectively; these are serialization names, not annotated ports on the input CAD parts.

B.1 Inputs, Outputs, and Decoding

Whole-request compiler. We provide a complete request q and the available component names in a JSON user message. The model returns the component list and a list of connections, each with two component identifiers, a motion label, and endpoint descriptions. Thus, the connections in G are inferred from the request, rather than supplied as reference edges. This prompt also asks for geometric attributes; the combined evaluation below uses the separately recorded per-connection parser for those attributes.

Per-connection parser. We provide a component pair and the instruction describing its connection. The model returns the requested motion and five fields for each endpoint: entity kind, analytic geometry type, relative size, neighboring surface types, and repeated-feature status. If the instruction does not describe both endpoints, the prompt requests `unknown` with two `null` endpoint records. The pair is an input to this parsing task, so its field accuracy measures attribute extraction, not graph discovery.

Model and decoding. Both tasks use DeepSeek-V4-Pro with the recorded API identifier `deepseek-v4-pro`, temperature 0, JSON-object response format, and `thinking.type=disabled`. We send a fixed system prompt and one JSON user message, without few-shot demonstrations. Requests may be batched inside an `items` array; a logical parsing stage therefore need not correspond to one API call per assembly. For the 409 held-out connections, the saved attribute run uses 52 batches of at most eight items. Model outputs are stored and reused without changing the prompts or selecting a response using assembly accuracy.

Validation and failure handling. The graph validator checks request coverage, the allowed component set, unique non-self connections, connectedness, and motion labels. The endpoint validator checks field names and allowed values; a size is a within-component category, not a numerical mating tolerance. The graph loader can preserve a valid topology while marking an invalid endpoint description unknown, whereas the attribute parser rejects malformed endpoint records. A missing graph gives no assembly proposal; if an edge parse is missing, the combination step keeps its endpoints unknown and retains the graph-predicted motion. The final graph results include four single-request retries after size-field validation failures, using the same model and prompt; they are not a first-attempt success rate.

B.2 Fixed system prompts

Figures 8 and 9 reproduce the two implemented `system_prompt_v1` functions. The instruction text and response schemas are unchanged; headings, bullets, and whitespace are added only for readability. The graph prompt uses `available_roles`, whereas the attribute prompt receives `role_a` and `role_b` for one connection. The displayed schemas retain the type placeholders used in the prompts rather than showing example JSON responses.

B.3 A recorded input–output example

We use a saved three-component request to illustrate the serialization. The component identifiers `role_00`, `role_01`, and `role_02` have the descriptions `Component6:1`, `Component19:1`, and `Component18:1`. These identify candidate groups; the language model does not receive the candidates within those groups. The complete request q is:

> Build one assembly from the available role pools. Use the largest cylinder face adjacent to plane on role_02 and the medium cylinder face adjacent to plane on role_01. Connect role_02 to role_01 so it can rotate about their shared axis without sliding. Use the largest cylinder face adjacent to plane on role_01 and the largest cylinder face adjacent to plane and torus on role_00. Connect role_01 to role_00 so it can rotate about their shared axis without sliding.

Graph input. The user-message template is shown below. The `request` placeholder is replaced by the complete q quoted above; the other values are taken from the saved input.


```
{
  "schema_version": "linkcad_request_graph_inputs.v1",
  "items": [
    {
      "query_id": "linkcad_bcd7872609ff38dd036f",
      "available_roles": [
        {"description":"Component6:1","role_id":"role_00"},
        {"description":"Component19:1","role_id":"role_01"},
        {"description":"Component18:1","role_id":"role_02"}
```

System prompt for assembly-graph compilation

You compile complete natural-language CAD assembly requests into role-level connection graphs. Return one JSON object and no prose.

Input and task

- Each input item provides `query_id`, `available_roles`, and request.
- Copy every available `role_id` exactly into roles.
- Infer which role pairs must connect from the request.
- For each connection, extract `requested_mobility` and the two local port descriptions.
- Do not invent or omit roles, and do not use candidate IDs, primitive indices, file names, paths, hashes, or source-joint IDs.

Allowed values

- Allowed `primitive_kind`: face, edge.
- Allowed `geometry_type`: plane, cylinder, cone, sphere, torus, spline, other, circle, line, ellipse, `bspline_curve`, `other_curve`.
- Allowed `size_class`: only, smallest, small, medium, large, largest.
- Allowed `adjacent_surface_types`: plane, cylinder, cone, sphere, torus, spline, other; emit zero to two values.
- Allowed `symmetry_class`: single, repeated.
- Allowed `requested_mobility`: fixed, revolute, prismatic, cylindrical, planar, ball, `pin_slot`.

Missing descriptions

If the request identifies a connection but does not describe both ports, set `port_contract_status` to unknown and `port_a` and `port_b` to null.

Output format

Return exactly

```
{"items": [{
  "query_id": str,
  "roles": [str],
  "connections": [{
    "role_a": str, "role_b": str,
    "port_contract_status": "specified"|"unknown",
    "port_a": object|null, "port_b": object|null,
    "requested_mobility": str
  }]
}]}
```

Each port object has exactly `primitive_kind`, `geometry_type`, `size_class`, `adjacent_surface_types`, `symmetry_class`.

Figure 8: System prompt for converting a complete request into a connection graph.

```
        ],
        "request": "<q: the complete request quoted above>"
      }
    ]
  }
```

Saved graph output. The following is the complete normalized prediction for this request, not a newly generated or manually corrected response. The `schema_version` fields inside endpoint records are added by the validator.

```
{
  "query_id": "linkcad_bcd7872609ff38dd036f",
  "roles": ["role_00","role_01","role_02"],
  "connections": [
    {"role_a": "role_02", "role_b": "role_01",
```

System prompt for connection attributes

You compile natural-language CAD assembly requests into a small symbolic port vocabulary. Return one JSON object and no prose.

Input and task

- Preserve every `query_id`, `edge_id`, `role_a`, and `role_b` exactly.
- For each item, extract `requested_mobility` and the two local port descriptions.

Allowed values

- Allowed `primitive_kind`: face, edge.
- Allowed `geometry_type`: plane, cylinder, cone, sphere, torus, spline, other, circle, line, ellipse, `bspline_curve`, `other_curve`.
- Allowed `size_class`: only, smallest, small, medium, large, largest.
- Allowed `adjacent_surface_types`: plane, cylinder, cone, sphere, torus, spline, other; emit zero to two values.
- Allowed `symmetry_class`: single, repeated.
- Allowed `requested_mobility`: fixed, revolute, prismatic, cylindrical, planar, ball, `pin_slot`.

Missing descriptions and constraints

If the text does not describe both ports, set `port_contract_status` to unknown and `port_a` and `port_b` to null; still extract mobility when it is unambiguous. Do not invent candidate IDs, primitive indices, file names, paths, hashes, or source-joint IDs.

Output format

Output exactly

```
{"items": [{
  "query_id": str, "edge_id": str,
  "role_a": str, "role_b": str,
  "port_contract_status": "specified"|"unknown",
  "port_a": object|null, "port_b": object|null,
  "requested_mobility": str
}]}
```

where each port object has exactly `primitive_kind`, `geometry_type`, `size_class`, `adjacent_surface_types`, `symmetry_class`.

Figure 9: System prompt for extracting geometric descriptions and motion for one connection.


```
 "requested_mobility": "revolute",
 "port_contract_status": "specified",
 "port_a": {
   "primitive_kind": "face", "geometry_type": "cylinder",
   "size_class": "largest", "symmetry_class": "single",
   "adjacent_surface_types": ["plane"],
   "schema_version": "linkcad_port_contract.v1"},
 "port_b": {
   "primitive_kind": "face", "geometry_type": "cylinder",
   "size_class": "medium", "symmetry_class": "single",
   "adjacent_surface_types": ["plane"],
   "schema_version": "linkcad_port_contract.v1"}
},
{"role_a": "role_01", "role_b": "role_00",
 "requested_mobility": "revolute",
 "port_contract_status": "specified",
 "port_a": {
   "primitive_kind": "face", "geometry_type": "cylinder",
   "size_class": "largest", "symmetry_class": "single",
   "adjacent_surface_types": ["plane"],
   "schema_version": "linkcad_port_contract.v1"},
```

Table 2: **Accuracy of the frozen language frontend.** "Geometry correct" requires all fields for both sides, or a correct unknown answer; "connection correct" also requires the requested motion.

Split	Connections	Geometry correct	Motion correct	Complete requests
Development	181	181 (100.0)	180 (99.45)	49/50 (98.0)
Held-out Fusion	409	409 (100.0)	409 (100.0)	114/114 (100.0)


```
      "port_b": {
        "primitive_kind": "face", "geometry_type": "cylinder",
        "size_class": "largest", "symmetry_class": "single",
        "adjacent_surface_types": ["plane","torus"],
        "schema_version": "linkcad_port_contract.v1"}
      }
    ]
}
```


The two connections give $E = \{(2, 1), (1, 0)\}$ over the three component indices, and both request revolute motion. For connection $(2, 1)$, the endpoints are a largest cylindrical face on component 2 and a medium cylindrical face on component 1, each adjacent to a plane. For $(1, 0)$, both faces are largest cylinders, but the endpoint on component 0 must also have a neighboring toroidal surface. This local distinction shows how the same component can participate in two connections through different interfaces. The output describes the required geometry; it does not yet identify a face in any particular STEP candidate.

Recorded attribute-parser inputs. For this same request, the separately saved parser uses the following two instructions, keyed by the component pairs $(2, 1)$ and $(1, 0)$:

> Connect role_02 through its largest cylinder face next to plane surfaces that occurs once, and connect role_01 through its medium cylinder face next to plane surfaces that occurs once. After assembly they must rotate about their shared axis without sliding.
>
> Make role_01 and role_00 meet at the following local features: its largest cylinder face next to plane surfaces that occurs once for role_01, its largest cylinder face next to plane and torus surfaces that occurs once for role_00. Their permitted relative motion is to rotate about their shared axis without sliding.

Each is placed in a user-message record with `query_id`, `edge_id`, `role_a`, `role_b`, and `instruction`, inside an `items` array. The corresponding edge identifiers are `edge_00` and `edge_01`. For this example, both saved attribute outputs agree with the endpoint and motion values in the graph output above. The attribute serializer calls the two endpoint records `port_contract_a` and `port_contract_b` after validation. We use these records as $\gamma_{e,u}$ and $\gamma_{e,v}$, and render them as a fixed textual description for the rankers.

B.4 Evaluation and reuse of saved predictions

The graph and attribute runs were collected separately. The former sees complete assembly requests; the latter sees controlled descriptions of individual connections, which need not be literal substrings of those complete requests. These descriptions can state endpoint properties, such as whether a feature occurs once, more explicitly than the whole-request wording. For the reported combined evaluation, we construct G from the predicted graph and look up saved attribute predictions by request identifier and unordered component pair, swapping endpoints when their orientation differs. Only graph-predicted pairs are retained. Consequently, the results support graph prediction and attribute parsing under these recorded inputs, rather than an additional fresh two-call evaluation on identical clause wording.

The graph compiler recovers all 114 held-out graphs and all 409 edges, giving edge precision, recall, and F1 of 100%. Table 2 shows that the second call returns all required fields for the 114 held-out requests. On development data, one connection has the wrong motion label, while the geometric

fields are correct. These controlled requests explicitly name the available roles, so the graph result measures multi-edge composition rather than open-vocabulary role discovery.

C Training and Inference Details

Both B-rep encoders use three message-passing layers with hidden dimension 64. For each candidate, the interface ranker keeps at most 16 symmetry-equivalent groups of faces or edges. The search retains 25 complete part assignments and eight interface-pair alternatives for each required connection. We train three optimizer seeds (1701, 1702, and 1703) with the same data order policy, loss weights, epochs, and learning rate. We select these settings on development data and keep all three checkpoints fixed before constructing each held-out set.

Loss computation. In Eq. 8, each cross-entropy is the negative log-softmax probability of the reference choice. We average $\mathcal{L}_{\mathrm{node}}$ over component entities and $\mathcal{L}_{\mathrm{pair}}$ over requested connections, using all available candidates for the former and geometrically retained candidate pairs for the latter. Interface supervision uses the reference components: $\mathcal{L}_{\mathrm{prop}}$ averages endpoint losses over their full interface sets, while $\mathcal{L}_{\mathrm{int}}$ averages pair losses over retained proposal combinations only when both reference interfaces are present. The discrete top-16 indices are not differentiated; proposal supervision remains available when a reference interface misses the shortlist. The ranking loss averages $\log(1+\exp(1+S_{\mathrm{comp}}(\mathbf{c}^-)-S_{\mathrm{comp}}(\mathbf{c}^\star)))$ over one-component substitutions $\mathbf{c}^-$ of the reference assignment $\mathbf{c}^\star$. The training implementation defaults to $\lambda_{\mathrm{int}}=0.5$ and $\lambda_{\mathrm{rank}}=0.25$. The auxiliary loss averages support-type cross-entropy where labels are available; motion-type cross-entropy is omitted when the request supplies motion, as in our compiled-request setting.

Message passing and readout. In Eq. 4, an input MLP projects intrinsic face features to the initial node embedding. The message function M_ℓ concatenates the sender embedding, receiver embedding, and shared-boundary features, including the dihedral angle, and applies a two-layer MLP with SiLU activation. Messages are averaged over neighbors; an isolated face receives a zero message. The update is implemented as

$$U_\ell(h,m)=\mathrm{LayerNorm}\Big(h+\widetilde{U}_\ell([h;m])\Big), \tag{9}$$

where $\widetilde{U}_\ell$ is a two-layer MLP and $[\,;\,]$ denotes concatenation. Each layer has its own parameters, and the two branches do not share encoder parameters. We attention-pool face embeddings within groups defined by intrinsic geometry and neighborhoods. Branch A concatenates the mean and maximum of these group embeddings and projects them to h_c. Branch B adds each face-group embedding to an MLP projection of its primitive features and applies layer normalization; edge-interface features are encoded by the primitive MLP without face-group addition.

Language-conditioned scoring. Text uses fixed 128-dimensional signed-hash word and character-trigram features followed by learned MLP projections; branch B has a separate text projection. The scoring functions in Eq. 5 and 7 use two feature constructions:

$$\begin{aligned}\phi(a,z)&=[a;z;a\odot z;|a-z|],\\ \psi(a,b,z)&=[a;b;z;a\odot b;|a-b|;a\odot z;b\odot z],\end{aligned} \tag{10}$$

where $\odot$ is elementwise multiplication. The node and interface-proposal MLPs receive $\phi(h_c,z_v)$ and $\phi(h_i,z_e^B)$, respectively; the component-pair and interface-pair MLPs receive $\psi(h_a,h_b,z_e)$ and $\psi(h_i,h_j,z_e^B)$. Each outputs a scalar logit; these are ranking scores, not calibrated probabilities of geometric validity. Interface scores are computed across candidate component pairs and indexed by the assignments retained by branch A, rather than recomputed for each complete assignment. The component-pair geometric mask is applied to logits after the neural forward pass, so it restricts search but does not skip the corresponding neural scoring computations. To train for incomplete instructions, we present the same 50 development requests at five fixed information levels without changing their candidate geometry or labels. We train for three epochs over these 250 views. The matched control sees five copies of each complete-description view, giving both models approximately the same number of updates. We fix both models before generating the 50-query test set.

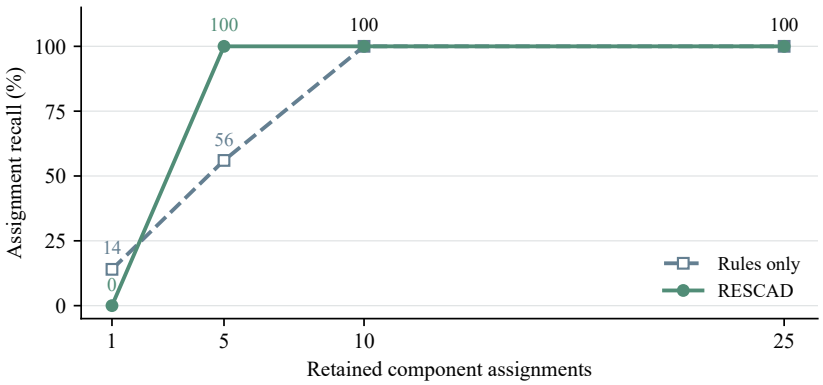


Figure 10: Assignment recall versus retention budget on the four-candidate test set (Table 4). These are ranking results before CAD execution.

Table 3: **Results with incomplete interface descriptions.** "Remaining assignments" is the median number of complete assignments after rule-based filtering; the intended assignment remains in every row. Learned values are three-seed medians.

Fields	Information retained	Remaining	Rules T1	Complete T1	Missing-field T1	Missing-field R@25
5	kind, type, size, adjacency, symmetry	180	2	100	100	100
4	kind, type, size, adjacency	240	0	100	100	100
3	kind, type, size	735	0	0	76	100
2	kind, type	2,304	0	0	100	100
1	kind only	4,096	0	0	70	100

Table 3 gives the numerical values behind Figure 6. Rule-based ordering never exceeds 2% Top-1 because many candidates share the remaining properties. Training with missing fields keeps the intended assignment within the first 25 for every request and gives 70–100% Top-1 across all information levels.

Table 4: **Effect of the number of retained part assignments.** Values are assignment recall on a held-out four-candidate test set; RESCAD is the three-seed median.

Method	R@1	R@5	R@10	R@25
Rules only	14	56	100	100
RESCAD	0	100	100	100

Figure 10 visualizes the same budget sweep: the learned advantage appears at five retained assignments, not at Top-1. Table 4 shows that the learned ranker places the intended assignment within the first five choices for every request, whereas rule-based ordering requires ten. The learned ranker's R@1 is zero before execution because a geometrically invalid choice receives the highest score. The kernel check removes that choice, after which the learned scores recover the intended candidate (Table 5).

Table 5: **How kernel checks change the top-ranked result.** Every method uses the same 25 assignments and the same local kernel-check budget on 50 held-out queries; learned values are three-seed medians.

Method	Assignment R@25	All-part T1	Connection T1
Rules only	100	14	14
Rules + kernel check	100	0	0
RESCAD ranking	100	0	0
RESCAD + kernel check	100	**90**	**48**

Table 7: **Direct LLM configurations.** Each model receives the same 50 requests and makes up to two calls per request.

Model	API	Identifier	Thinking / effort	Temp.
DeepSeek-V4-Pro	DeepSeek	`deepseek-v4-pro`	disabled	0
Kimi K3	OFox	`moonshotai/kimi-k3`	disabled	0.6
GPT-5.6 Sol	OFox	`openai/gpt-5.6-sol`	low	0
Gemini 3.8 Flash	OFox	`google/gemini-3.8-flash`	low	0

D Comparison with Assembly Baselines

Table 6: **Inputs and outputs of related assembly systems.** Methods that receive one fixed part pair predict its connection but do not select parts from a candidate set.

Method	Native input	Ports?	Selects?	Native output
AutoMate / JoinABLe	one selected/fixed pair	no	no	joint / pose
Pointer-CAD (Qi et al., 2026)	text + evolving B-rep	no	no	next operation + entity
CADMate	fixed parts + target image	no	no	pose sequence
SPADA	text	generated	no	assembly code + tests
CRAG	observed pieces	no	no	missing shape + poses
AssemCAD	typed generated parts	yes	no	named-port connections
ResCAD	multiple STEP per role	no	yes	interfaces + assembly

Adapting related methods. The superscript * in Table 1 denotes an adaptation to our candidate-set task; these rows transfer published modeling patterns and do not reproduce native checkpoints or native benchmark results. All methods receive the complete candidate pools and the same structured connection requests. JoinABLe* uses symmetric cross-part B-rep scoring, is retrained on our 480 training requests, and shares the language-derived candidate filter and assembly executor. For AssemCAD*, we extract typed interfaces from the candidate B-reps and use DeepSeek-V4-Pro to rank candidates for each role. The adapter retains constraint-compatible assignments, fills their interfaces by geometric matching, and executes the resulting programs. SPADA* starts from the historical DeepSeek-V4-Pro selection and direct-pose prediction, then receives two rounds of interface-distance, axis-angle, axial-gap, and collision feedback. It may replace interface IDs and world poses but cannot change the selected parts or inspect the reference. The search-based adapters share a 25-proposal upper bound; the direct LLMs make one prediction without pose search or geometric correction. The shared frontend and geometric processing are part of these adapted systems, so this comparison measures their final assembly outputs under the stated configuration. We keep CADMate and CRAG out of the numeric table because their native inputs expose a target image or the correct part set.

The SPADA* audit confirms that its equality with the historical DeepSeek-V4-Pro direct-pose result is not a transcription error. Of 48 valid initial predictions, 47 are repeated after repair; the remaining prediction changes without changing any reported outcome. The language model therefore fails to convert scalar test residuals into coordinated multi-part transforms under this fixed budget.

LLM models and input protocol. The request compiler, language-based adapters, and DeepSeek baseline use the original DeepSeek API; Kimi, GPT, and Gemini use the OFox API. Table 7 specifies the identifiers and inference settings. The three OFox models use a 16,384-token output cap per call; DeepSeek retains its original provider configuration. We preserve each requested model identifier, returned model identifier, and response with the experiment records. Each direct baseline first ranks all candidates from the same public B-rep summaries and connection descriptions. These summaries include sizes, surface and edge counts, and counts of interfaces matching the shared request constraints. We select its first-ranked candidate for every role without filtering assignments. The second call receives the selected parts' local geometric frames and predicts interface IDs and one world transform per part. Within the fixed 2×10^{-3} input tolerance, we convert each approximately rigid rotation matrix to its closest proper rotation by singular value decomposition, leaving the predicted translation, parts, and interfaces unchanged. We apply this numerical conversion uniformly to all direct LLMs and SPADA*; matrices outside the input tolerance remain invalid. The evaluator then executes each assembly once, with no interface snapping, pose search, or collision repair.

Geometry metrics and denominators. We re-import each exported STEP and count it as valid when it contains nonempty, valid, positive-volume solids; inter-part collision is checked separately. After one anchor alignment, we compute volume IoU as intersection volume divided by union volume for each corresponding role, then average over roles and requests. Shape Accuracy is the fraction of roles whose mean bidirectional, unsquared surface distance falls below 1% of the reference assembly's bounding-box diagonal. F-score uses the same threshold for bidirectional surface precision and recall. CD averages the two directed mean surface distances, averages over roles, and reports the result as a percentage of the reference assembly's bounding-box diagonal. These role-wise definitions give each part equal weight and do not align parts independently. Rates, IoU, Shape Accuracy, and F-score use all 50 requests, with missing outputs and unavailable references contributing zero to the reference-based scores. Two reference assemblies could not be exported, leaving at most 48 scoreable pairs for CD; we report that count separately from output validity. In seed order 1701/1702/1703, the CD counts are 48/24/48 for JoinABLe*, 48/40/48 for ResCAD without the interface rule, and 48/48/48 for ResCAD. DeepSeek-V4-Pro and SPADA* each have 46 scoreable pairs; Kimi, GPT, and Gemini have 34, 37, and 9, respectively. Rules and AssemCAD* each have 48 scoreable pairs.

E Train, Development, and Test Separation

Fusion assemblies. The 88-request interface set excludes the 480 training assemblies, 50 development assemblies, and 150 assemblies used in an earlier part-selection study. The 114-request main test set also excludes all 88 interface-test assemblies and contains only sources with 21–100 visible instances. We do not reuse an evaluated assembly in a later test set.

Generated parts. We use separate CadQuery sets to choose the search procedure, choose the kernel-check budget, and report final results. The reported four-candidate execution set contains 50 requests, 150 connections, and 800 STEP files. The larger-candidate set contains 50 requests and 1,600 STEP files; each request has 4,096 possible assignments and a median of 180 after geometric filtering. The incomplete-description and language-model comparisons each use another independently generated 50-request, 1,600-STEP set. No generated test file shares identical STEP bytes with training, development, Fusion, or another reported test set.

Keeping test labels separate. For every held-out set, we choose the model checkpoints, geometric rules, candidate summaries, field-deletion order, beam sizes, and kernel budgets on development data. We then generate all predictions and kernel outputs before revealing the intended parts and interfaces. The test labels are used once to compute the reported metrics and are never used to update a model or rerun prediction. This procedure also applies to the two general-purpose language-model baselines. They receive surface and adjacency counts, sizes of repeated interface groups, geometric properties, and continuous-feature summaries, but no rendering, file identity, or target label. All nine outputs that fail the required candidate format remain in the denominator.

F Details for Text-to-CadQuery Parts

We generate 16 candidates for each of 40 Text-to-CadQuery requests. The generator returns 618 valid single-solid B-reps, 21 duplicate candidates, and one failed generation. Two requests lose the intended candidate, and one has fewer than six valid candidates for a role. We evaluate the remaining 37 requests, which contain 577 candidates and no identity overlap with earlier data. Within each request, every candidate has the same cylindrical connection diameter. The language instead distinguishes eight ring styles and eight shaft styles, so geometric compatibility alone cannot reveal the intended part. To make the input pool and data construction concrete, Figures 11 and 12 show one complete generated candidate pool and the recorded screening counts. The gallery renders original generator outputs before interface-diameter calibration and rigid posing; thumbnails are independently scaled for visibility and are not assembly predictions. The screening plot separates candidate-generation outcomes from request-level exclusions, leaving the 37-request, 577-candidate evaluation set described above.

We adapt the text-conditioned part ranker for 48 epochs, updating 94,722 parameters. We then adapt the interface text and scoring layers for 12 epochs, updating 78,209 parameters while keeping the B-rep encoder, geometric filter, and executor fixed. We use 256 labeled views from four generated-part

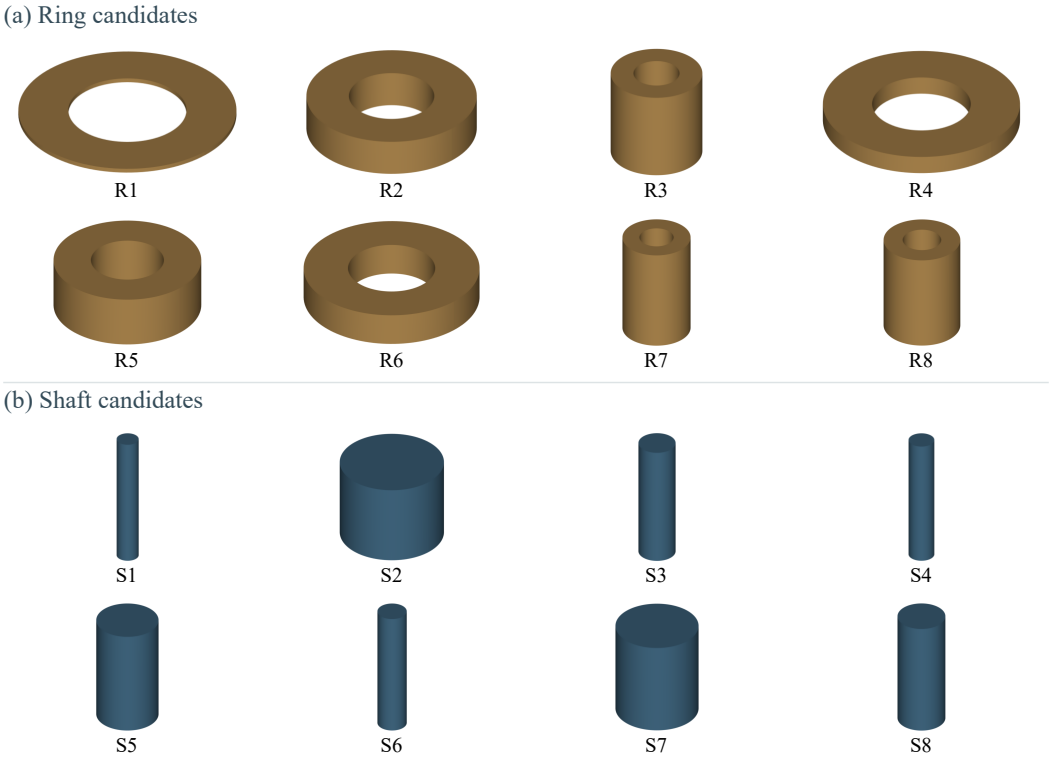


Figure 11: All 16 generated candidates for one request: eight rings and eight shafts. Original STEP geometry is shown before calibration; thumbnails are not to scale.

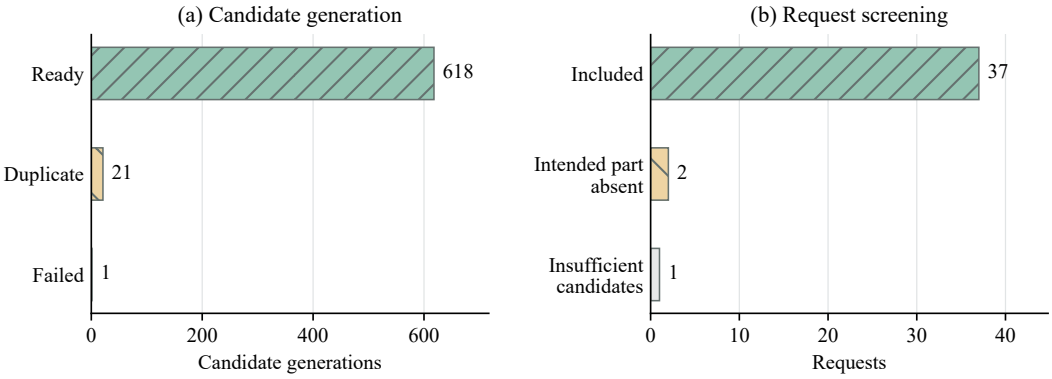


Figure 12: Recorded generation and screening outcomes: 640 candidate-generation attempts and 40 requests yield 37 evaluable requests containing 577 candidates.

training pools and choose the epoch counts and learning rate of 3×10^{-4} on four disjoint development pools. Across three seeds, development all-part Top-1 is 47.98%, 56.45%, and 50.81%; interface Top-1 conditioned on the intended part pair is 100% for every seed.

Figure 5 separates the contribution of the two rankers. Adapting the part ranker raises assignment R@25 from 48.65% to 100%, but verified R@25 remains 13.51% because the interface is often wrong. Adapting both rankers raises verified R@25 to 100% and first-verified accuracy to 45.95%. The remaining single-output errors come from part ordering: whenever the intended part pair is available, the adapted interface ranker retrieves the intended interface and the executor realizes it within the evaluation budget.

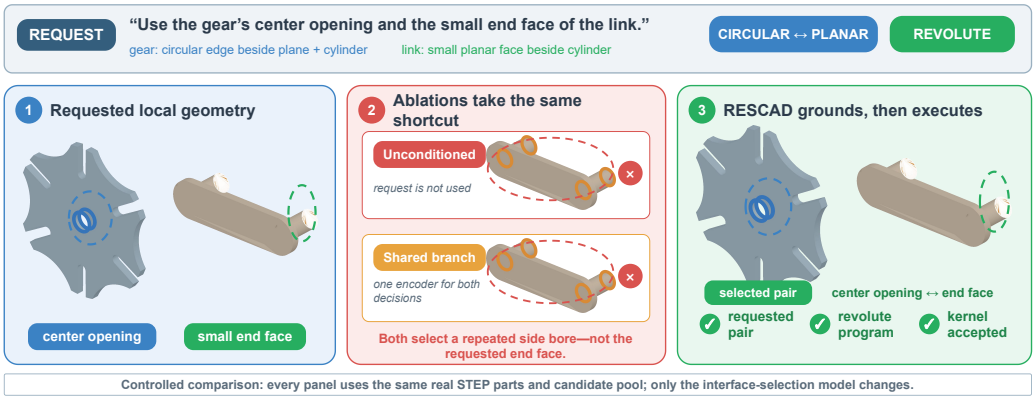


Figure 13: **Language changes the selected interface on real STEP geometry.** The candidate pool is identical for all methods. The models without a dedicated language-conditioned interface ranker select plausible repeated features. RESCAD uses the request to select the intended circular edge and planar face, after which Open CASCADE executes and accepts the revolute connection.

G ADDITIONAL INTERFACE-SELECTION RESULTS

Table 8: **Changing only the requested interface.** Top-k asks whether the requested face or edge pair is retrieved; paired Top-8 requires correct predictions for two different requests on the same geometry. RESCAD values are three-seed medians.

Method	Top-1	Top-8	Prediction changes	Paired Top-8
Unconditioned	2.27	17.05	0.00	7.95
Shared branch	4.55	24.43	14.77	11.36
RESCAD	**7.39**	**32.95**	**50.00**	**13.64**

Table 8 tests whether language changes the selected interface while the parts remain fixed. The unconditioned model never changes its prediction, whereas RESCAD changes the top interface for 50% of paired requests and improves Top-8 from 17.05% to 32.95%. This result shows that the interface branch uses the requested local geometry rather than only the identity of the two parts.

Table 9: **Interface-selection gains on 88 unseen assemblies.** Values are percentage-point gains over the strongest matched baseline for each metric, followed by paired 95% confidence intervals.

Metric	All 88	Supported 84	Complete descriptions 76
Complete connection R@25	+9.09 [3.41,17.05]	+8.33 [3.57,16.67]	+10.53 [3.95,19.74]
Part pair + interface Top-1	+7.33 [2.61,12.61]	+7.17 [2.21,12.39]	+8.49 [3.07,13.66]
Part pair + interface Top-8	+9.91 [2.54,16.88]	+8.52 [1.24,15.38]	+9.91 [3.70,15.69]
Interface Top-8 | part hit	+15.76 [8.37,23.51]	+13.55 [6.58,20.68]	+15.22 [7.44,23.47]

Table 8 isolates paired changes on the same geometry, while Table 9 measures average gains across the full interface test set. Every confidence interval remains above zero in the supported and complete-description subsets. The largest gain occurs after the correct part pair has been found, locating the main learned contribution in interface selection.

Visual summary. To visualize the effect of language conditioning, Figure 14 replots Tables 8 and 9. The left panel shows improvements in both individual and paired-request retrieval, although paired Top-8 remains 13.64%, indicating substantial unresolved ambiguity. Furthermore, the right panel shows positive gain intervals across all four metrics on the full 88-assembly set; the largest

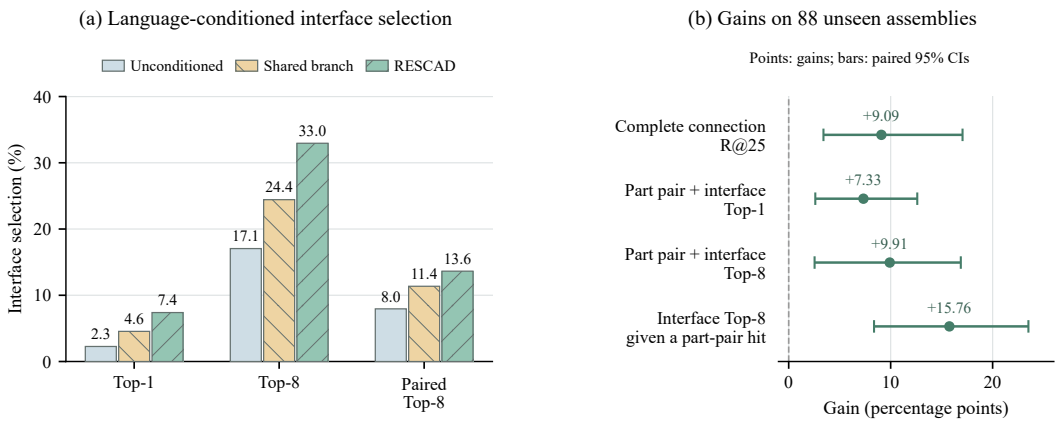


Figure 14: Interface-selection accuracy (left) and gains over the strongest matched baseline with paired 95% confidence intervals (right).

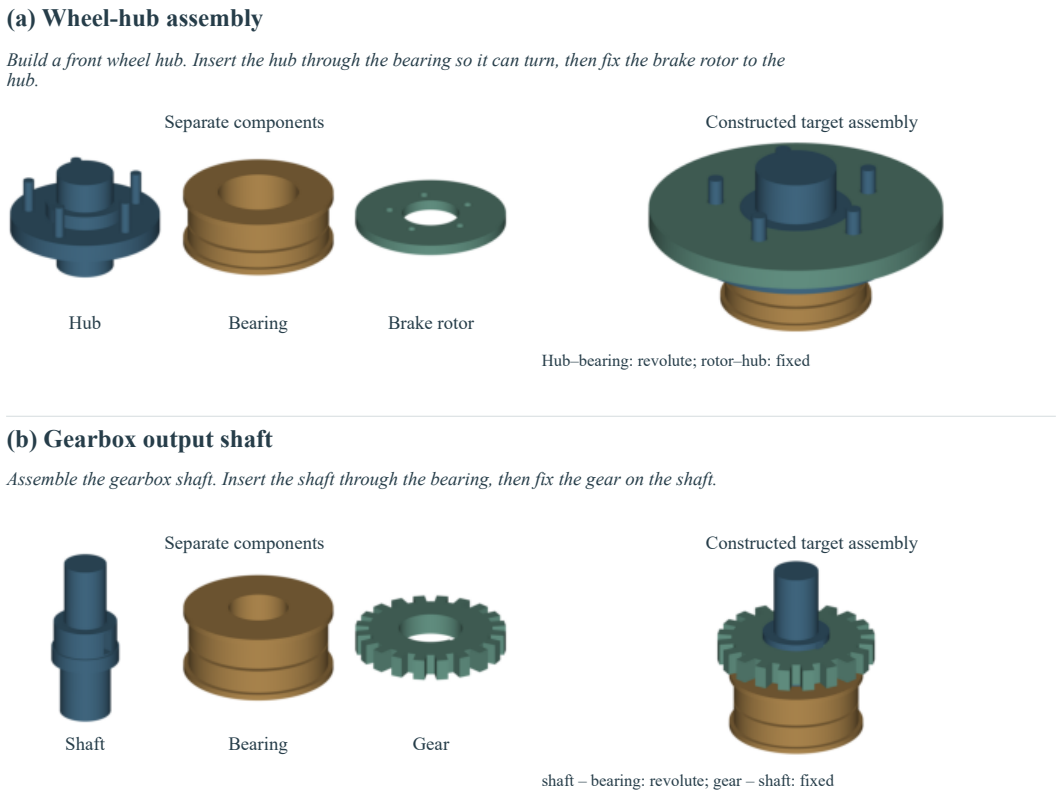
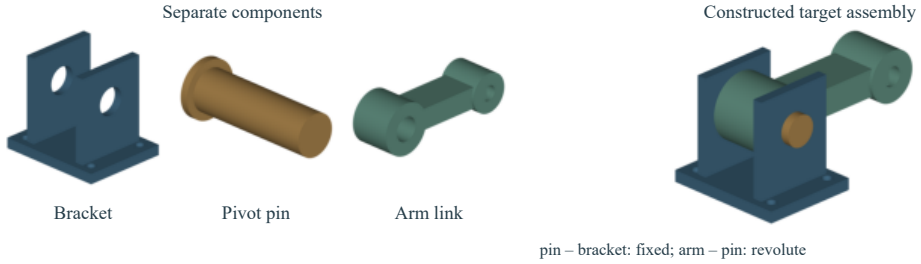


Figure 15: Illustrative wheel-hub and gearbox tasks: separate components and constructed target assemblies, not model outputs.

point estimate is for interface retrieval given a correct part pair. These plots summarize the same evaluations as the tables, not additional experiments.

(c) Robot elbow joint

Assemble the robot elbow. Put the pivot pin through the bracket and arm so the arm can rotate.



(d) Motor-fan assembly

Assemble the motor fan. Insert the shaft through the bearing housing, then fix the fan to the shaft.

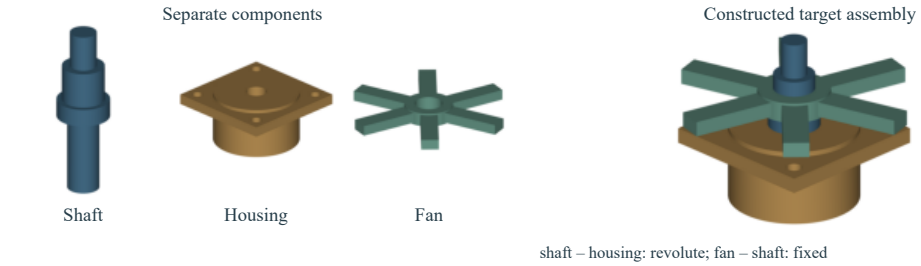


Figure 16: Illustrative robot-joint and motor-fan tasks. The constructed targets show different connection layouts, not additional evaluation results.

H Illustrative Assembly Tasks

I Contact-Complete Execution Details

The executor evaluates three conditions in order. First, the predicted interface frames must satisfy the requested alignment and motion rule. Second, the two connected solids must be within 0.1 mm while their common volume remains below 10^{-7} mm^3. Third, after all relative poses are composed, every requested connection is measured again and every unordered solid pair is checked for positive-volume intersection. This final replay prevents a locally accepted pose from being counted when composition leaves a part detached or moves another part into collision.

For axial connections, the executor aligns the two axes and searches along their remaining translation degree of freedom. If the aligned pose intersects, deterministic bisection approaches the boundary from each separated side and retains only collision-free contact poses. For planar connections, it uses the center of the trimmed face patch, not the arbitrary origin of the infinite supporting plane. When several faces or edges share the same intrinsic description, the finite pose budget is interleaved across their representatives before additional roll choices are considered.

To make the component and interface choices concrete, Figures 15 and 16 present four constructed task examples from our motivation-design materials, not additional model predictions or evaluation results. Each example pairs a language request with separate STEP components and a target arrangement; matching colors track the same component across views. The target arrangements were checked for contact between the requested component pairs and absence of positive-volume intersections. These static checks do not establish motion correctness or manufacturing validity.

Rotating assemblies. The wheel-hub example combines a hub–bearing connection with a rotor–hub attachment, whereas the gearbox example places a bearing and gear on different regions of a

stepped shaft. Both illustrate why identifying the participating components is not enough: the request must also be associated with specific connection regions.

Articulated and hub-mounted components. The robot example places a pivot pin through a bracket and arm, while the motor example places a fan on a shaft supported by a housing. These contrasting layouts make the distinction between *which components* and *where to connect* visible beyond a single motivating geometry.