Supplementary Document for ParSEL: Parameterized Shape Editing with Language

ADITYA GANESHAN, Brown University, USA
RYAN Y HUANG, Brown University, USA
XIANGHAO XU, Brown University, USA
R. KENNY JONES, Brown University, USA
DANIEL RITCHIE, Brown University, USA

CCS Concepts: • **Computing methodologies** → **Computer graphics**; **Neural networks**; **Natural language generation**.

Additional Key Words and Phrases: shape editing, parametric editing, large language models, computer algebra systems, neuro-symbolic, program synthesis

1 INTRODUCTION

In this document, we present additional details regarding our system. First, we provide a brief overview of the videos included in the supplemental material. Next, we describe our editing system ParSEL in Section 3, including the pre-processing and post-processing steps. Section 4 provides additional details regarding Analytical Edit Propagation, and Section 5 provides additional details regarding our LLM prompting workflow. Finally, Section 6 provides details regarding the experiments presented in the paper and Section 7 presents common failure cases.

2 QUALITATIVE VIDEOS

We provide the following videos in the supplemental materials:

(1) The set of videos used in our perceptual study, located in `./videos/perceptual_study`.
(2) A video titled `real_time_shape_variations.mp4`, which demonstrates how analytical edit propagation enables real-time exploration of shape variations. This is contrasted with a prior approach that uses online-edit propagation, resulting in a laggy user experience.
(3) A video titled `live_proxydural_modeling.mp4`, which shows our system being used to create a *proxydural* model of a 3D asset.

3 PARAMETERIZED SHAPE EDITING DETAILS

3.1 Shape Abstraction

3.1.1 Inter-part Relations. As discussed in the main paper, our system considers two types of inter-part relations, namely Attachment relations and Symmetry relations. We consider multiple subtypes for each of these relations, which are presented in Table 1.

First, we use three types of Symmetry relations: ReflectionSymmetry, RotationSymmetry, and TranslationSymmetry. Each of these relations introduces constraints between the parts based on the parameters of the relation, allowing us to detect when a relation is broken.

Secondly, we model 4 types of inter-part Attachment relations, namely PointAttachment, LineAttachment, FaceAttachment and VolumeAttachment. These attachments constrains the relative movement between parts, with the constraints being more restrictive as we move from Point to Volume. LineAttachments are typically useful for modelling attachment between parts which 'widen' together, such as a chair's seat and its back. FaceAttachments are useful for modelling attachment between parts which *radially* scale together, such as a TV's display and its Frame. Finally, VolumeAttachments is suitable to model the attachment between parts which must always be edited in the same fashion (i.e. the only way to satisfy VolumeAttachments is by having the same edit on both the parts. While these attachment relations behave in different ways, they are all modelled using only point attachments, with line attachment in practice converting into two attachments, face into 4 co-planar points attachments and volume into 8 attachment points. Note that technically face attachments and volume attachments can be modelled with just 3 and 4 points attachments respectively as well.

The most important detail is that all these relations, the symmetry relations as well as the attachment relations, are automatically derived, and this process is described in section 3.3.

3.1.2 Virtual Part Hierarchy. The parts of man-made objects are often hierarchical in nature - parts such as 'back' can often be composed of multiple sub-parts such as 'back-support-bars' 'back-top-bar' etc. However, performing edit propagation while considering such hierarchy becomes increasing complicated and lead to undesirable edits (such as uniform scaling of a set of parts when instead translating a few sub-parts while scaling others is preferable). Prior methods [12] do offer some solutions however, we avoid modeling hierarchy so that we can avert introducing additional complexity to analytical edit propagation. This can however be explored in the future.

Therefore, all parts in the system are considered to be at the leaf-level with no hierarchy. We only allow hierarchies in one cases, namely in the presence of RotationSymmetry or TranslationSymmetry relations. When such relations are present, we instantiate a *virtual* parent part containing all the instances. This allows use to edit symmetry relations with edit operations such as ChangeCount and ChangeDelta. When we create such *virtual* parts, we also create Attachment relations between the virtual part and the other parts.

Authors' addresses: Aditya Ganeshan, adityaganeshan@gmail.com, Brown University, USA; Ryan Y Huang, ryan_y_huang@brown.edu, Brown University, USA; Xianghao Xu, xianghao_xu@brown.edu, Brown University, USA; R. Kenny Jones, russell_jones@brown.edu, Brown University, USA; Daniel Ritchie, daniel_ritchie@brown.edu, Brown University, USA.

Relation	Instantiation	Constraint
ReflectionSymmetry	`ref_sym(`$\mathbb{H}$`, origin=`$\mathbf{o}$`, normal=`$\mathbf{n}$`)`	$\|\|\mathbf{H}_j - \mathbf{H}_i - 2\left((\mathbf{H}_i - \mathbf{o}) \cdot \mathbf{n}\right)\mathbf{n}\|\|_\infty < \delta \,\forall\, \mathbf{H}_i, \mathbf{H}_j \in \mathbb{H}$
RotationSymmetry	`rot_sym(`$\mathbb{H}$`, point=`$\mathbf{o}$`, rot_mat=`$\mathbf{R}$`)`	$\|\|\mathbf{H}_j - \mathbf{o} + \mathbf{R}^n(\mathbf{H}_i - \mathbf{o})\|\|_\infty < \delta \,\forall\, \mathbf{H}_i, \mathbf{H}_j \in \mathbb{H}, n = i - j$
TranslationSymmetry	`trans_sym(`$\mathbb{H}$`, delta=`$\mathbf{d}$`)`	$\|\|\mathbf{H}_j - (\mathbf{H}_i + \mathbf{d})\|\|_\infty < \delta \,\forall\, \mathbf{H}_i, \mathbf{H}_j \in \mathbb{H}, n = i - j$
PointAttachment	`point_attach(`$\mathbb{A}$`, `$\mathbb{B}$`)`	$\|\|M_a\mathbf{H}_i - M_b\mathbf{H}_j\|\|_\infty < \delta \,\forall\, a_n \in \mathbb{A}, b_n \in \mathbb{B}$
LineAttachment	`line_attach(`$\mathbb{A}$`, `$\mathbb{B}$`)`	$\|\|M_{a_n}\mathbf{H}_i - M_{b_n}\mathbf{H}_j\|\|_\infty < \delta \,\forall\, a_n \in \mathbb{A}, b_n \in \mathbb{B}, \{n \in \mathbb{Z} \mid 1 \le n \le 2\}$
FaceAttachment	`face_attach(`$\mathbb{A}$`, `$\mathbb{B}$`)`	$\|\|M_{a_n}\mathbf{H}_i - M_{b_n}\mathbf{H}_j\|\|_\infty < \delta \,\forall\, a_n \in \mathbb{A}, b_n \in \mathbb{B}, \{n \in \mathbb{Z} \mid 1 \le n \le 4\}$
VolumeAttachment	`vol_attach(`$\mathbb{A}$`, `$\mathbb{B}$`)`	$\|\|M_{a_n}\mathbf{H}_i - M_{b_n}\mathbf{H}_j\|\|_\infty < \delta \,\forall\, a_n \in \mathbb{A}, b_n \in \mathbb{B}, \{n \in \mathbb{Z} \mid 1 \le n \le 8\}$

Table 1. **Inter-Part Relations**: We enlist the different inter-part relations supported in our structured shape representation. In the first column, we annotate the type of each relation. The second column depicts the pseudo-code used to initialize these relations, providing reference for the relation parameters. Finally, the last column depicts the constraints that are enforced as a consequence of each relation. Note that for the Attachment relations, M_i denotes the harmonic coordinates [2] for the corresponding point i.

Specifically, when considering virtual part, we either allow edits on the virtual part - this allows modelling edits on all the instances together as well the relation edits (ChangeCount and ChangeDelta). To edit parts within a symmetry relation, such as when scaling only the central back slat, we automatically turn off the virtual part (and its attachment) reducing the shape back to a hierarchy-less graph.

3.2 Parameterized Shape Editing DSL

As described in the main paper, our DSL provides atomic operators to perform parameterized edits on parts as well as relations in the input 3D asset. We enumerate these operators in Table 2. Furthermore, we also present the algebraic form conferred to the operand by these operators.

As shown in the table, our DSL includes two relation editing operation, namely ChangeCount and ChangeDelta. These edits affect the number of instances, or the spacing between them in TranslationSymmetry and RotationSymmetry relations. We apply these operators in two modes, (i) a *Top-Down* mode, where the relation configurations (such as count and spacing) are derived from the parent part (i.e. the a virtual part containing all the instances in the symmetry group). This allows edits where the scaling of the parent virtual part affects the count or spacing between the instances. and (ii) a *Bottom-Up* mode, where the parent virtual part's shape is derived from the relation configuration. To achieve this functionality, we first over-parameterize the relations, for instance, *TranslationSymmetry* is represented by 5 variables its *start-point*, *mid-point*, *end-point*, *count* and *delta-vector*. This allows us to flexibly explore edits on any of these features (such as changing the count be shifting the *end-point*). Secondly, we replicate these points, marking them as points of the *virtual parent*, and create virtual Attachment relation between the points and their virtual parent counterparts. For instance, we create an Attachment relation between the *start-point* of the relation and the *start-point* in the virtual parent part. This allows us to *analytically* consider how the parent should be updated as the relation is updated, or vice-versa. Similarly, RotationSymmetry relations are also over parameterized by symbolic expression for its *arch-length*, *radius*, *count*, *angle* which are then used to communicate edits between the parent virtual part, and the relation. Note that we currently only support editing of closed loop RotationSymmetry groups.

3.3 Pre-Processing Input 3D Assets

Given a 3D mesh as input, we utilize a prior work [11] to infer the reflection, translation and rotation symmetries in the shape. This approach performs ICP between mesh parts with the same label to detect these relations. We alter the algorithm slightly to fit our needs. Specifically, we only allow translation and rotation symmetry on part-sets with more than 2 parts, and only consider axis aligned reflection symmetry relations. We also avoid considering hierarchies of such symmetry groups (for instance, translation symmetry of rotation symmetries) as such hierarchies tend to occur less frequently in the man-made objects we considered.

To create the attachment relations we utilize the overlaps between parts. First, we ascertain if two parts have an attachment relation by checking if points sampled on one part are contained in the other part's bounding box (and vice versa). Once an attachment is detected, we must now decide the type of attachment between the parts. Attachments are classified as PointAttachments, LineAttachment, FaceAttachment or VolumeAttachment based on the number of shared planes of intrinsic symmetry between the part and their intersection region.

This is computed by the following process:

(1) First, we generate a point cloud of the intersection region by sampling points on one part's surface and rejecting those which are outside the other part's bounding box. This process is repeated for both the parts.
(2) Next, for each part as calculated the *intersection-shared symmetry plane count*. We consider the number of intrinsic symmetry planes of its bounding box about which the intersection point cloud is symmetric as well.
(3) The symmetry order of the attachment is assigned as the minimum of the *intersection-shared symmetry plane count* of both the parts.

We permit one exception - if a part is contained within another part, we mark the attachement between them with the highest

Edit Operators	Instantiation	Algebraic Form
Translate	translate($\mathbf{H}_i$, dir=$\mathbf{n}$, amt=x)	$\mathbf{H}_i(x) = \mathbf{H}_i^0 + x \cdot \mathbf{n}$
Rotate	rotate($\mathbf{H}_i$, orig=$\mathbf{o}$, axis=$\mathbf{n}$, amt=x)	$\mathbf{R}(x) = \cos(x)\mathbf{I} + \sin(x)[\mathbf{n}]_\times + (1-\cos(x))\mathbf{n}\mathbf{n}^T$ $\mathbf{H}_i(x) = \mathbf{o} + \mathbf{R}(x)(\mathbf{H}_i^0 - \mathbf{o})$
Scale1D	scale_1D($\mathbf{H}_i$, orig=$\mathbf{o}$, dir=$\mathbf{n}$, amt=x)	$\mathbf{H}_i(x) = \mathbf{o} + x(\mathbf{n} \cdot (\mathbf{H}_i^0 - \mathbf{o}))\mathbf{n} + (\mathbf{H}_i^0 - \mathbf{o})$
Scale2D	scale_2D($\mathbf{H}_i$, orig=$\mathbf{o}$, normal=$\mathbf{n}$, amt=x)	$\mathbf{H}_i^c = \mathbf{H}_i^0 - \mathbf{o}$ $\mathbf{H}_i^p = \mathbf{H}_i^c - (\mathbf{H}_i^c \cdot \mathbf{n})\mathbf{n}$ $\mathbf{H}_i(x) = \mathbf{o} + (1+x)\mathbf{H}_i^p + (\mathbf{H}_i^c \cdot \mathbf{n})\mathbf{n}$
Scale3D	scale_3D($\mathbf{H}_i$, orig=$\mathbf{o}$, amt=x)	$\mathbf{H}_i(x) = \mathbf{o} + (1+x)(\mathbf{H}_i^0 - \mathbf{o})$
Shear	shear($\mathbf{H}_i$, orig=$\mathbf{o}$, normal=$\mathbf{n}$, dir=$\mathbf{d}$, amt=x)	$\mathbf{H}_i^c = \mathbf{H}_i^0 - \mathbf{o}$ $\mathbf{S} = \mathbf{I} + x\mathbf{d} \otimes \mathbf{n}$ $\mathbf{H}_i(x) = \mathbf{o} + \mathbf{S}^T\mathbf{H}_i^c$
ChangeCount	change_count($SymR_i$, amt=x)	-
ChangeDelta	change_delta($SymR_i$, amt=x)	-
KeepFixed	keep_fixed($\mathbf{H}_i$)	$\mathbf{H}_i(x) = \mathbf{H}_i^0$

Table 2. **DSL commands**: We enlist the parameterized editing operators provided in our DSL. Each command is parameterized by a variable x, which controls the edit magnitude. We omit the algebraic form of the ChangeCount and ChangeDelta, the two operators which are applied on symmetry relations due to their complexity. Note that the initial 6 operators can also be applied only on part features such as a face, edge or a vertex, enabling non-affine transforms.

symmetry order (i.e. 3). The attachment relations are then directly derived based on the symmetry order, with order 0 mapping to PointAttachment and 3 mapping to VolumeAttachment.

3.3.1 Extending Part Labels. Additionally, we automatically enhance each part's label with verbal directional phrases such as "front" and "back" to capture its relative positioning among other instances with the same label. These phrases help the LLM discern which instance to edit when the edit intent itself contains directional specifications (for instance, in 'scale the *front* legs.').

To perform this task, we first compute the relative placement vectors for each part within a shared label group (such as all parts containing the label 'leg'). The relative placement vector is computed w.r.t. the center of all the parts. Then based on the alignment of these vectors with the cardinal direction we allot them terms such as 'front', 'back-right' etc. which are then used to extend the labels.

When a group has more the 2 parts, we also separately identify if they are approximate arranged in a rotational pattern, or a straight line. This is done by calculating the angle created between the parts and the center. Based on this different set of directional phrases are used. For instance, 4 legs along a line going left to right are assigned the labels 'left', 'left-center', 'right-center', 'right', whereas if they are approximately rotationally arranged, they are given the labels 'left-back', 'left-front', 'right-back', 'right-front'. Groups with more than 5 labels (such as rungs in a bed's ladder) are simply provided numeric indices as the directional phrase.

We found this approach provided correct labels for the majority of shapes considered, requiring minimal manual adjustment to the labels, particularly when parts are arranged in a non-trivial pattern. We emphasize that this is only implemented to help the LLMs and has no affect on our edit propagation algorithm. In future, we expect this technique to be replaced by methods which leverage foundation models to consider inter-group part arrangements similar to PartSLIP [5].

3.4 Post-Processing Edited 3D Assets

3D assets are edited in ParSEL with part-level deformations, and symmetry group modifications. Both of these operations can lead to the creation of holes in the 3D asset's mesh, particularly when the user sets a high edit magnitude. In Figure 1 (c) we show this in effect, where deforming the legs of a chair results in gaps between the chair legs and the seat. Therefore, after editing with the parametric editing programs, certain 3D assets require additional post-processing to improve edited assets quality.

Note that this artifact is also present in prior edit propagation methods. Prior approach [12] employ an intricate approach, where surface-based deformation approach [4] is used with virtual edges constructed between the points and the surface feature curves extracted using [8]. However, it doesn't resolve all mesh-related issues. We instead use a Surface Reconstruction based method to correct such artifacts. Given an edited 3D asset, we perform the following steps:

(1) First a dense point cloud is sampled on the deformed mesh (with 200000 points) using Poisson disk sampling.
(2) We then employ screened Poisson surface reconstruction [3] to reconstruct a 3D mesh from the point clouds.

(a) Input 3D Asset *(b) Edited 3D Asset* *(c) Mesh holes due to edits* *(d) Mesh Clean-up with SPR* *(c) Post-processed 3D Asset*

Fig. 1. **Perceptual Study Interface**: Extreme deformation under our system can cause gaps to appear in the 3D mesh. With the post-processing we describe in Section 3.4, which employs screened Poisson surface reconstruction (SPR) [3], these gaps can be fixed although with a detrimental effect on the shape's material.

(3) This mesh can often be dense, hence we perform mesh decimation using Quadric error based edge collapse.
(4) Next, we partition this mesh into part-level mesh fragments.To perform this, first we extract all the vertices of the original deformed mesh, and simply transfer the label from these vertices to the reconstructed mesh vertices using a nearest neighbor lookup. The part-labels are then used to partition the reconstructed mesh into fragments.
(5) Finally, we transfer the material from the original part to the reconstructed part mesh. We transfer the vertex UV coordinates from the original (deformed) mesh to the reconstructed mesh using nearest neighbor lookup as well.

The results of this process are shown in Figure 1 (d). We found this approach to be effective at filling small-medium gaps between the deformed parts with nominal reduction in the 3D asset's quality.

However, similar to prior approaches [12], we emphasize that this approach is only a stop-gap, and only serves as a means to demonstrate the effectiveness of analytical edit propagation. Fixing holes in edited meshes is an interesting research direction in itself, with many works [1] specifically targeting hole correction. In future, we hope to leverage such approaches to improve the output from our method to maintain the quality of the 3D asset after deformation.

4 ANALYTICAL EDIT PROPAGATION DETAILS

4.1 Pseudo-code for Analytical Edit Propagation

We present the pseudo-code for our system in Algo. 1, Algo. 2 and Algo. 3.

Algo. 1 provides the pseudo code for the overall editing process. First, the LLM is prompted to return the *relation-validity*, *seed-edit* and *type-hints*. Then, we perform Analytical Edit Propagation given the *seed-edit* and the *type-hints*.

Algo. 2 provides the pseudo code for the edit propagation framework used in our system. Note that broken relations are gathered based on the broken constraints ($\mathbb{C}$, listed in 1). The function `get_part_to_edit` returns a part that should be edited based on the broken relations. Following CSP literature, we use the Minimum Entropy Heuristic (MinE) and select the part with the most relations (since it is likely to have the least number of constraint satisfying edits).

Finally, Algo. 3 provides the pseudo code for the Analytical edit solver. Here, the function `cas_solve` deploys a Computer Algebra System Solver [6] to infers function assignments to the symbolic variable `Y` that satisfy the constraint equation `eq`.

4.2 Analytical Edit Solver

In this step, we are tasked with introducing new parameterized edits which restore and preserve the broken relations across the input range. As discussed in the paper, our strategy involves searching only over a subset of feasible PARAM assignments. The feasible PARAMS assignments are created using the hexahedron features such as its face-center, vertices, and local axis directions. To create these operators, we first collect a set of 'feasible' points (the center, face-center, corners, edge-centers), a set of 27 3D points, and a set of 'feasible' directions (the global cardinal directions as well as local cardinal directions). These feasible points and directions are then used to create the edit candidates. These edit candidates are then merged based on the equivalence between the edit expressions they create. We note that this simple sampling process creates a large set of edit operators to consider. However, since our Analytical Solving process is embarrassingly parallel, we are able to search successful edits in a reasonable duration.

4.2.1 nhbd-edits Parameterization. *nhbd-edits* address the fact that often the PARAM require to edit a part may not be derived by one of its features. Therefore, when all the candidates in $\mathbb{E}_R$ fails, we introduce additional candidates which contain PARAM assignments which are based on the features of other *edited* hexahedrons. We also introduce edit candidates with PARAM assignments based on the relative arrangement of the two parts, such as when one part is being scaled, moving the other part along the direction connecting the two parts is a potential translation edit candidate.

4.2.2 Edit Selection. For each candidate from the edit candidate set $\mathbb{E}_R$, our analytical edit solver infers plausible functional assignments to the AMOUNT parameter that create valid parameterized edits, i.e. edits which restore and preserve all the broken relations (of the part being edited). This results in a set of feasible edits, from which we must select the most suitable edit. We now introduce the criterion employed for this task.

Algorithm 1: ParSEL Overview

```
generate_program(shape, edit_request):
    shape = set_sym_relations(shape, edit_request)
    init_edit = get_init_edit(shape, edit_request)
    edit_hints = get_edit_hints(shape, edit_request)

    program = propagate(shape, init_edit, edit_hints)
    return program
```

Algorithm 2: Analytical Edit Propagation

```
propagate(shape, init_edit):
    all_edits, finish = init_edit, false
    while !finish:
        propagate(all_edits)
        broken = gather_broken_relations(shape)
        if len(broken) > 0:
            part = get_part_to_edit(broken)
            edit, unfixed = solve(broken, part)
            reject_relations(unfixed)
            all_edits.add(edit)
        else:
            finish = true
        clean_up()
    return all_edits
```

Algorithm 3: Analytical Edit Solver

```
solve(broken, part, edit_hints):
    Y = new_var()
    solution_set, scored_solutions = {}, {}
    candidates = get_candidates(part, edit_hints)
    constraints = get_constraints(part, broken)
    for edit in candidates:
        propagate(edit, amount=Y)
        for eq in constraints:
            solutions = cas_solve(eq, Y)
            solution_set.add(solutions)
    for sol in solutions:
        score = count_satisfying(sol, constraints)
        unfixed = get_broken(sol, constraints)
        scored_solutions.add((score, unfixed))
    if len(solution_set) > 0:
        edit, unfixed = get_best(scored_solutions)
    else:
        edit, unfixed = None, broken
    return edit, unfixed
```

At a high level, we select the edit which minimizes ARAP deformation energy [10], while maximally preserving the edited part's intrinsic symmetry planes. First, we remove all the edit candidates which result in a high ARAP energy. Next, each edit, based on its type and PARAM, is assigned the number of intrinsic symmetry planes of the hexahedron it retains. For instance, edits such as translation, rotation, isotropic scaling do not affect any symmetry planes - the hexahedron is symmetric about axially aligned planes passing through its center even after being edited. On the other hand, edits such as Face Scaling disrupt this symmetry, and hence retain a lower number of intrinsic symmetry planes. From these edits we select the set with the highest number of retained symmetry planes. Within this set, we simply select the edit which results in the least amount of ARAP energy.

We note that selecting directly based on ARAP energy works in most cases, however considering symmetry planes helps in a few cases. This is mainly because when editing shapes, the user-desired edit may not necessarily match the min arap energy candidate - for example, most edits contain stretching which despite higher arap-energy cost can actually be the preferred edit in many cases. The consideration of both ARAP energy and intrinsic symmetry planes however results in a criterion closer to human preference.

5 PROMPTING DETAILS

We provide the prompts we utilize for our system in the supplemental material. In total, our system employ 4 prompts, one each for inferring the *seed-edit* and *type-hints*, and two for *relation validity*, one customized for ReflectionSymmetry and the other for RotationSymmetry and TranslationSymmetry. We found that having two separate prompts improved the LLM's ability to correctly infer the *relation-validity*. Further, we employ two additional prompts, one for each of the demonstrated applications.

We follow a common pattern for all the prompts, providing the following sequence on instructions: (i) the task overview, (ii) the output specification, (iii) examples, (iv) verbal shape description and edit request (v) steps for performing the task and (vi) guidelines. That last detail plays a crucial role in improving the LLM's ability to perform the tasks. Figure 2 depicts the prompt provided to the LLM to infer the *seed-edit*. Note that to conserve space, we provide only snippets of various sections (as marked by the ellipses).

6 EXPERIMENTAL DETAILS

6.1 Dataset

We construct our datasets by sourcing shapes from PartNet [7] and CoMPaT3D++ [9]. We provide more details of the dataset here.

First, we present the statistics of the parsed structured shape representation derived from the mesh models. Table 3 reports these statistics. We note that our *dev-set* containing shapes from PartNet contains a wider array of shapes sizes than the *test-set* created with CoMPaT3D++. Furthermore, since shapes in PartNet are often labeled at a finer level, more symmetry relations such as translation and rotation are detected in them. In contrast, CoMPaT3D++ presents a wider array of shape categories. Finally, we observe that CoMPaT3D++ models are closer to real-world 3D assets, and as a result they (1) Often have more intricate Attachment relations, and (2) Tend to use closed surface parts, which consequently helps us avoid the post-processing pipeline discussed in 3.4.

We have included all the edit request used for the both the datasets in the supplementary materials. Furthermore, the provided perceptual study videos also show the shapes with their corresponding edit requests. As can be seen in the edit requests, the use of LLMs allows

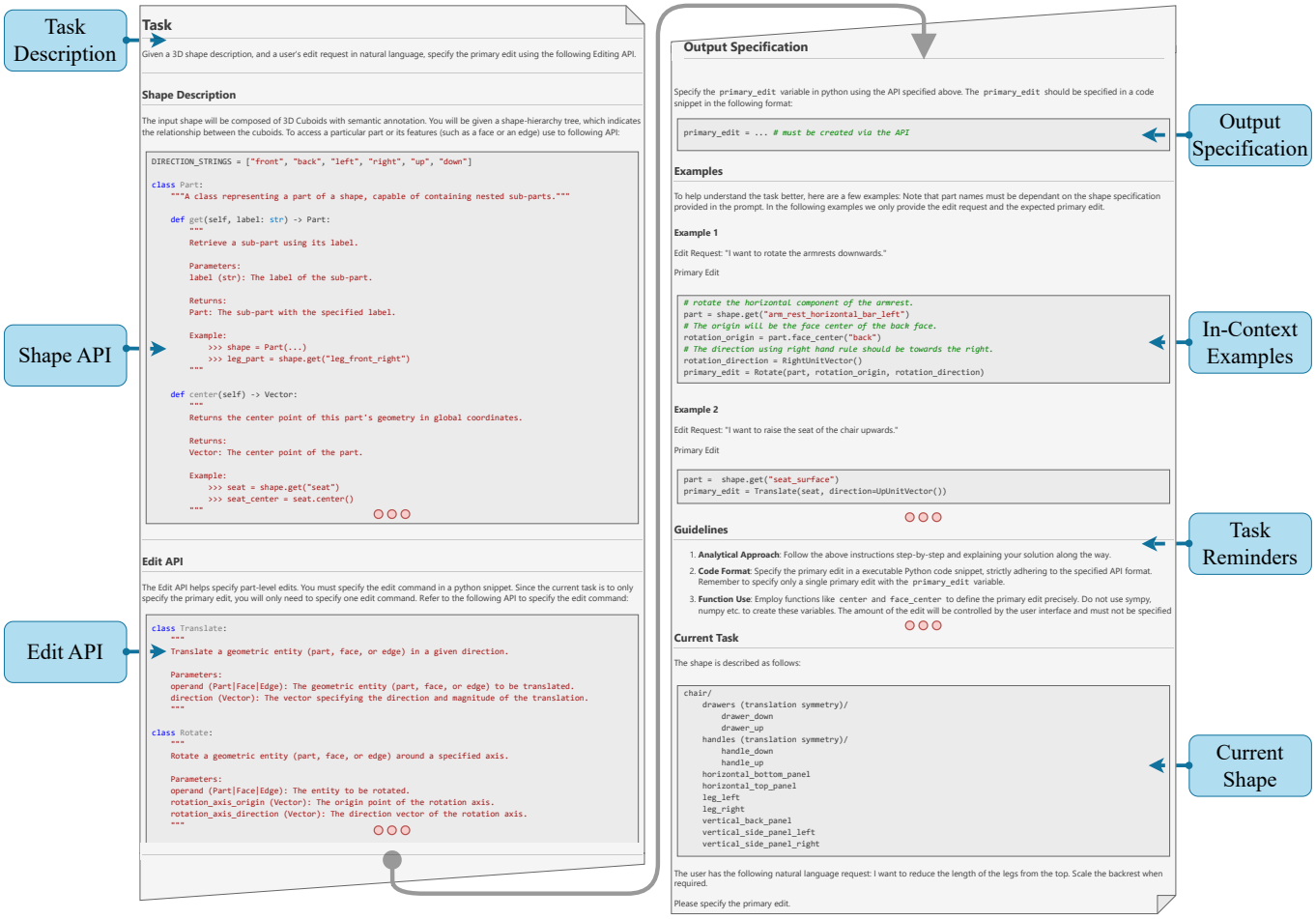

Fig. 2. Overview of the structured prompt provided to the LLM for inferring the *seed-edit*. The prompt follows a systematic approach, including task description, shape API, edit API, output specification, in-context examples, and task reminders. To conserve space, only snippets of various sections are shown, as indicated by the ellipses.

Dataset	N. Shapes	N. Parts			N. Attachment Relations			N. Symmetry Relations		
		Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
PartNet Lamp	7	5	11.57	30	4	7.00	12	0	1.71	4
PartNet Chair	15	7	16.40	31	2	18.93	42	1	5.60	11
PartNet Table	9	5	12.89	20	3	14.44	41	1	6.22	16
PartNet Storage Furniture	13	12	21.67	48	17	51.83	117	3	8.50	18
PartNet Bed	7	17	44.25	81	31	77.25	110	7	24.88	53
PartNet (*dev-set*)	51	5	19.5	81	2	31.82	117	0	7.96	42
CoMPaT3D++ (*test-set*)	50	3	12.5	23	1	19.54	72	0	4.8	14

Table 3. **Dataset Statistics**: We provide statistics of the two datasets we use. We note that our *dev-set*, explores a wider range of structural complexity, whereas our *test-set* explores a wider range of semantic variations (as it contains shape from 21 different categories).

us to support broader, intricate, nuanced and longer edit requests as well.

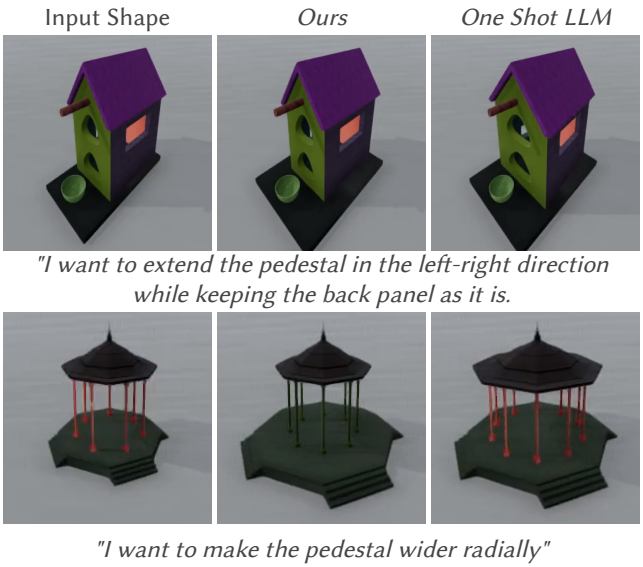

Fig. 3. **Perceptual Study Analysis**: In a few cases, as shown above, users preferred *One Shot LLM* over *Ours*. We note that this preference is not caused by a our model's failure to infer a valid program, but a misalignment between the system interpretation (perform secondary adjustments) and the participant's interpretation (do not perform any secondary adjustments).

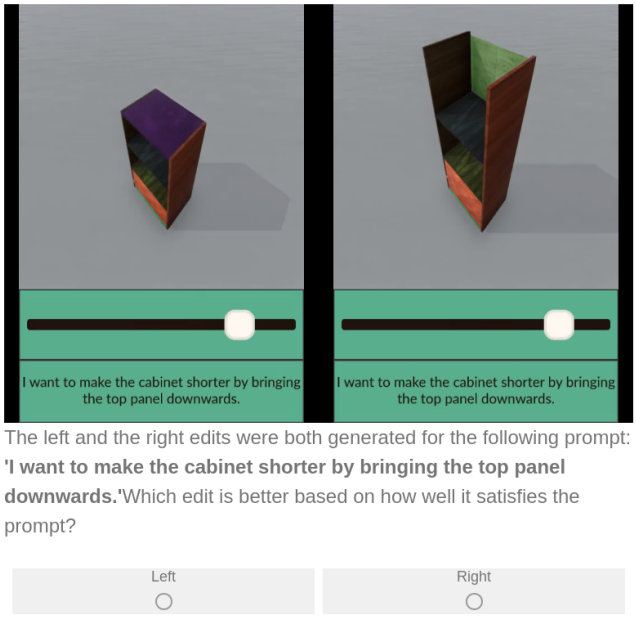

Fig. 4. **Perceptual Study Interface**: We provide a screenshot of the questionnaire provided to the participants in our two-alternatives forced-choice perceptual study. Note that the videos used for this study are provided in the supplemental material.

6.2 Program Quality Metrics

We introduce multiple metrics in the paper to analyse the different methods from various perspectives. First, we present the detail of the metrics used for measuring the quality of inferred programs. These metrics compare the inferred programs against the 'GT' program annotated by a Human-Solver inference process. These metrics are used to measure quality across three criterion, namely:

6.2.1 Programmatic ($\mathcal{J}(prog)$). This metric assesses how closely the inferred programs match the GT programs. First, we convert each program statement (or edit operator) into a string signature consisting of only the type of edit operation, and its operand. Then, this metric is computed by measuring the Jaccard similarity between the set of edit signatures derived from the inferred program and the GT program.

6.2.2 Geometric ($\mathcal{D}(geo)$). The metric measures the geometric distance between the shape edited with the inferred programs and the shape edited with the GT program. First, for both the programs, we set the user-controlled parameter x to a fixed value (0.35). Then, for each part, we compute the L^2 distance between the hexahedron vertices deformed by the inferred program, and the corresponding ones deformed by the GT program. This metric is computed by summing up these L^2 distances across the program.

6.2.3 Structural (%Rel). This metric quantifies the percentage of inter-part relations whose state (broken vs. maintained) matches the relation's state under the ground truth (GT) program. As all our relations can be represented as parameterized constraints as well, we simply check and annotate each relation's state under the GT program, and the inferred program. This metric is then computed by measuring the fraction of relations for which the annotations match. Note that we only consider relations which involve at least one edited part.

6.3 LLM Accuracy Metrics

Next, we elaborate on the metrics introduced for judging the quality of the LLM's output.

6.3.1 $Acc(SE)$. This metric measures the seed edit accuracy. Similar to the process conducted for measuring $\mathcal{J}(prog)$, we derive the string edit signature of the seed edit inferred by the LLM. If this seed-edit signature is present in the GT program's edit signature set, then we accept the seed edit to be an accurately predicted seed edit (i.e. $Acc(SE) = 1$).

6.3.2 $Acc(R)$. This metric measures relation validity accuracy. During the Human-Solver inference process, we record the relations that the human user disables to enable the edit. $Acc(R)$ is then computed by comparing the LLM's inference for each relation to the state set by the human user.

6.3.3 $\mathcal{J}(T)$. This metric measures the type hint accuracy. This metric is computed by computing the Jaccard similarity between the type hints set by the expert user during the Human-Solver inference process and the type hints set by the LLM. We find that expert user often set only a minimal set of required type-hints, i.e. the expert user elides type-hints for parts where the solver inferred edit (without type-hints) will match the edit type specified by the

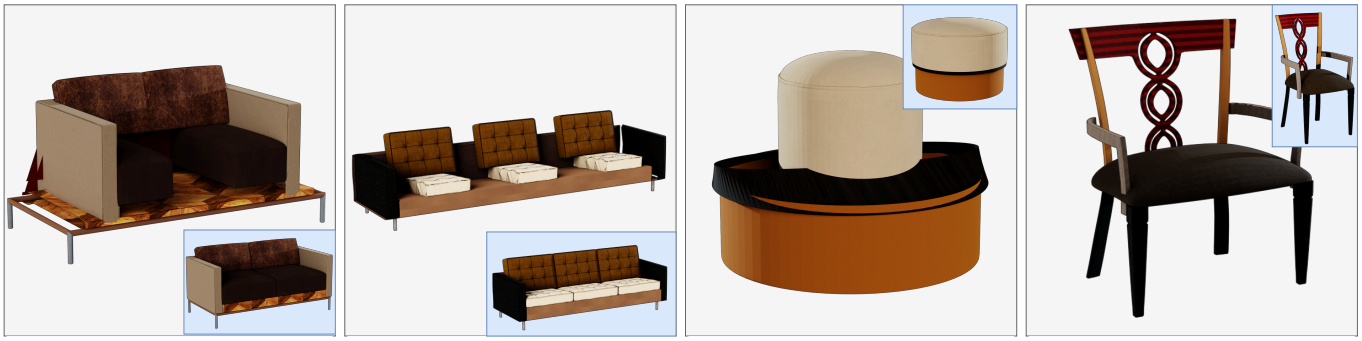

Fig. 5. Examples of failure cases in our system. (a) Propagation failure due to incorrect relation propagation, leading to a structurally unsound shape. (b) Propagation failure caused by unsupported operations, such as scaling with retained translation symmetry. (c) Propagation failure due to numerical precision errors, resulting in part clipping. (d) LLM parsing failure where the shape was edited but did not follow the edit request which specifies that the front legs must remain fixed. The corresponding natural language edit request provided to the LLM was "Scale the backrest ... while keeping the front legs as they are".

	$\mathcal{J}(Prog)(\uparrow)$	$\mathcal{D}(Geo)(\downarrow)$	%Rel(↑)
	1.0	0.0	100%
- *nhbd*	0.85	0.12	95.02%
- *breaking*	0.77	1.44	89.48%
Naive	0.76	4.32	87.9%

Table 4. Quality of programs inferred by the Solver: Removing *nhbd-edits* results in higher geometric distance ($\mathcal{D}(Geo)$), while removing *breaking-edits* leads to more structural distance (%Rel). The naive approach that removes both of these options is the least effective.

edit request. In contrast, the LLM often tends to provide more type hints. We therefore discount the entries in the LLM inferred type hints which, though not present in the expert user's annotation, are not wrong, i.e. setting these type-hints does not change the solver's output.

6.3.4 Match. This metric measures the fraction of input pairs where all LLM-inferred quantities match human annotations. *Match* is set as 1 if and only if all the other metrics $Acc(SE)$, $Acc(R)$ and $\mathcal{J}(T)$ are measured to be 1.

6.4 Perceptual Study Details

The two-alternatives forced-choice perceptual study presented in the main paper is conducted with 56 participants, collecting 1642 total judgements. These comparisons are conducted between the programs inferred by the different methods (*One Shot LLM*, *Ours Seed Only* and *Ours*) on the 50 (shape, edit request) pairs in our *test-set*. We notice that the program inferred by *Ours* matches those inferred by the *One Shot LLM* in 4 of the 50 pairs, and matches the program inferred by the *Ours Seed Only* in 14 of the the 50 pairs. *Note that this is not a drawback of our method.* Certain edit request only require a single *seed-edit* (for instance 'I want to increase the number of legs of a round table') making the program inferred by the three methods match. Similarly, certain edits require no *type-hints* or *relation validity* settings (for instance "widen the chair") making the program inferred by *Ours* and *Ours Seed Only* match. We remove these cases from our perceptual study to remove the noise that could potentially arise from including these comparisons.

Figure 4 shows a screenshot of the visual interface the participants is provided with. We further emphasize that all the comparison videos, and the anonymized participant preferences are also included in the supplemental materials.

6.4.1 Additional Analysis. We analyse the (shape, edit-request) input pairs for which a majority of participants preferred the *One Shot LLM* over *Ours*. This occurred in 5 of the 50 input pairs. We present two of them in Figure 3. We note that this preference is not caused by any failure of *Ours* method to infer a valid program, but rather due to a misalignment between the system interpretation (always perform secondary adjustments unless specifically asked not to) and the participant's interpretation (do not perform any secondary adjustments unless *really* required).

6.5 AEP Solver Analysis with 'GT' Annotation

Our main draft reports AEP solver ablation where the two features (use of *nhbd-edits* and *breaking-edits*) are removed to measure their effect on the solver's performance. To emphasize the importance of these two features, we also present this analysis with the expert-user's 'GT' annotations as input. We record the *seed-edit*, *type-hints* and *relation-validity* set by the expert-user, and infer the corresponding editing programs with the different variants of the solver. This result is presented in table 4. The metrics show that these features indeed play a crucial role in the success of the solver, and naively employ the solver can result in a lot of failure cases.

7 FAILURE CASES

Despite the robustness of our system, we identify some common failure modes. These failures primarily fall into two categories: *Propagation Failures* and *LLM Parsing Failures*.

7.0.1 Propagation Failures. These failures occur when, during the analytical edit propagation, the system fails to infer part-level editing operations that align with the edit intent. This can happen in three common ways:

(1) *Breaking Relation Errors*: When none of the discovered analytical edits can restore and preserve all the relations of the edited part, a *minimally* relation breaking edit is selected. When multiple edit candidates break a minimal number of relations, the optimal edit is selected using the criterion described in Section 4.2.2. Edit propagation can fail when this heuristic fails. As shown in Figure 5 (a), this results in edited shapes that are structurally unsound.
(2) *Unsupported Operation Errors*: Edit propagation also fails when the desirable editing operations is not yet supported by our system. An example of such a failure is depicted in Figure 5 (b). As the couch is extended, it is desirable to widen each of the seats, while also translating then accordingly. Our system fails to perform this edit as simultaneously translating and scaling each instance in a translation symmetry is not yet supported.
(3) *Numerical Precision Errors*: Occasionally, propagation fails due to numerical precision issues. In Figure 5 (c), a 1D scaling operation is mistakenly accepted as a feasible edit, leading to clipping between the object parts.

7.0.2 LLM Parsing Failures. These occur when the LLM fails to correctly parse and execute the input request. For example, in Figure 5 (d), although the resulting shape is structurally acceptable, the edit does not adhere to the editing request, which specifies that the front legs must remain fixed. This demonstrates the occasional failure of LLMs to properly infer the all the quantities. In this case, the LLM fails to set the type-hint for the front legs.

REFERENCES

[1] Xiaoyuan Guo, Jun Xiao, and Ying Wang. 2018. A survey on algorithms of hole filling in 3D surface reconstruction. *Vis. Comput.* 34, 1 (jan 2018), 93–103. https://doi.org/10.1007/s00371-016-1316-y
[2] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph.* 26, 3 (jul 2007), 71–es. https://doi.org/10.1145/1276377.1276466
[3] Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Trans. Graph.* 32, 3, Article 29 (jul 2013), 13 pages. https://doi.org/10.1145/2487228.2487237
[4] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. 2005. Linear Rotation-invariant Coordinates for Meshes. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)* 24, 3 (2005), 479–487.
[5] Minghua Liu, Yinhao Zhu, Hong Cai, Shizhong Han, Zhan Ling, Fatih Porikli, and Hao Su. 2023. Partslip: Low-shot part segmentation for 3d point clouds via pretrained image-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 21736–21746.
[6] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (Jan. 2017), e103. https://doi.org/10.7717/peerj-cs.103
[7] Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. 2019. PartNet: A Large-Scale Benchmark for Fine-Grained and Hierarchical Part-Level 3D Object Understanding. In *CVPR*.
[8] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. 2004. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.* 23, 3 (aug 2004), 609–612. https://doi.org/10.1145/1015706.1015768
[9] Habib Slim, Xiang Li, Yuchen Li, Mahmoud Ahmed, Mohamed Ayman, Ujjwal Upadhyay, Ahmed Abdelreheem, Arpit Prajapati, Suhail Pothigara, Peter Wonka, and Mohamed Elhoseiny. 2023. 3DCoMPaT++: An improved Large-scale 3D Vision Dataset for Compositional Recognition. (2023).
[10] Olga Sorkine and Marc Alexa. 2007. As-Rigid-As-Possible Surface Modeling. In *Proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing*. 109–116.
[11] Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiquan Cheng, and Yueshan Xiong. 2011. Symmetry Hierarchy of Man-Made Objects. *Comput. Graph. Forum* (2011).
[12] Youyi Zheng, Hongbo Fu, Daniel Cohen-Or, Oscar Kin-Chung Au, and Chiew-Lan Tai. 2011. Component-wise Controllers for Structure-Preserving Shape Manipulation. *Computer Graphics Forum* 30, 2 (2011), 563–572. https://doi.org/10.1111/j.1467-8659.2011.01880.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.01880.x