Supplementary material for
Action-guided generation of 3D functionality segmentation data

Jaime Corsetti[1,2] Francesco Giuliari[1] Davide Boscaini[1] Pedro Hermosilla[3] Andrea Pilzer[4]
Guofeng Mei[1] Alexandros Delitzas[5,6] Francis Engelmann[7,8] Fabio Poiesi[1]

[1]Fondazione Bruno Kessler [2]University of Trento [3]TU Wien [4]NVIDIA
[5]ETH Zurich [6]MPI for Informatics [7]Stanford University [8]USI Lugano

jcorsetti@fbk.eu

In this document, we report additional qualitative results and details about SynthFun3D. The Sections are organized as follows. In Sec. 1, we provide details on how we generated ground-truth data to train Gemma3 [3] to point at functional elements. In Sec. 2, we present additional qualitative results from SynthFun3D. We report additional examples of generated scenes (Sec. 2.1), and of objects retrieved by our strategy (Sec. 2.2) Finally, in Sec. 3, we report the prompts we used in the retrieval pipeline.

1. Training on downstream task

In the following Sections, we detail the process used to obtain the ground-truth points, which are used to train a VLM on the downstream task.

1.1. Generating pointing data from SceneFun3D

To train the VLM used in our experiments on the SceneFun3D dataset, we use high-resolution RGB-D videos (1920×1440), associated camera poses, and annotated 3D point clouds provided for each task description `D`. We preprocess the video data by sampling every three frames to mitigate data redundancy.

Next, we select only frames where the 3D mask of the functional element is visible and the camera distance is within two meters; this constraint ensures sufficient spatial resolution of the target object. We project the 3D mask on the point cloud onto the 2D image plane of frames that satisfies these conditions, and compute the mask centroid to serve as the pointing ground truth. When multiple masks are present (e.g., a drawer with multiple knobs), the centroid for each is selected. Finally, to prevent truncation artifacts, we discard samples where the calculated centroids fall within 200 pixels of the image boundary.

For each validated training sample, we construct a visual instruction tuning instance in a conversational format. The input prompt conditions the model on the task description `D`, explicitly requesting a structured JSON output. Specifically, the dialogue is formatted as follows:

> **User:** Point to the object part I need to interact in order to "`<task description>`". Return the points using JSON. Use the following format: a list of dicts with the key "point_2d" and the value a list of two integers. The values have to be normalized from 0 to 1000.
> **Assistant:** `[{"point_2d": [x, y]}]`

where $[x, y]$ represents the ground-truth centroid coordinates. This strict formatting ensures the model outputs machine-readable coordinates suitable for downstream evaluation.

1.2. Generating pointing data with SynthFun3D

We leverage SynthFun3D to synthesize environments populated by assets with part-level annotations, explicitly tailored for task-oriented interactions. Aligning with the SceneFun3D training distribution, we condition our scene generation on task descriptions `D` to ensure semantic coherence between the scene context and the functional object. We simulate video sequences by generating random camera trajectories that orbit the target object, maintaining the functional element as the camera's focal point. Consistently with the SceneFun3D preprocessing pipeline, we sample every three frames to avoid using redundant images. We further filter the data based on object visibility: frames are discarded if the functional element's mask occupancy is less than 0.01% (insufficient visibility) or greater than 25% (excessive proximity/occlusion) of the total image area. From the remaining candidates, we select the top-5 frames per video with the highest visible surface area of the target object. Finally, we compute the centroid of each target mask to serve as the pointing ground-truth and construct visual instruction tuning instances following the protocol defined in the previous section. We replicate the same procedure for the frames augmented with synthetic materials.

1.3. Breakdown of the generation costs

The generation cost for a single scenario in our dataset is dominated by the usage of the LLM, which in our case runs on an NVIDIA A100 GPU with 80 Gb of memory. Current GPUs providers (e.g., AWS), provides such hardware for about 4.10 USD per hours. Accounting for possible generation failures in our pipeline (e.g., no solution found in the Depth-First-Search algorithm), we estimate a rate of 12 scenes per hour generated by SynthFun3D, so that each scene costs 0.34 USD. The cost of video rendering is much smaller, as it does not requires such specialized hardware, therefore we consider 1 USD per scene as a reasonable upper bound for the cost of our synthetic data.

2. Qualitative results

In this Section, we show additional qualitative results obtained with SynthFun3D. In Sec. 2.1, we show additional examples of scene generation. For each scene, we report an example frame with the functional mask, the original RGB, and the material-augmented RGB. In Sec. 2.2, we show some examples of retrieval performed by our method, and compare it with the results obtained by the Holodeck [5] strategy using the same prompt.

2.1. Frames generation results

We show in Fig. 1 additional results of setups generated with SynthFun3D, along with examples of frames and ground-truth functional masks. We observe that SynthFun3D is capable of producing reasonable and realistic layouts while maintaining the spatial requirements described in the prompts. For example, in the first row the cabinet is correctly placed on the wall opposite to the one of the bed, while on the second row the cabinet is placed exactly under the painting and near the piano. These two examples have challenging prompts that requires retrieval of very specific functional object configurations: the "bottom-left" drawer in the first row implies a cabinet with a grid-like structure of drawers, similar to the second row ("top, second drawer from the right"). These examples show the capability of our retrieval pipeline in retrieving objects with fine-grained characteristics, and their relative mask as well (see column c). Similarly, the third row shows a bedroom setup in which the cabinet is correctly placed to the left of the bed, and the top-right handle is selected as ground-truth mask. The bottom row shows a simple case in thich the prompt simply request to open a window, in which the window handle is highlighted. Columns (d-e) show instead two material-augmented version of the same frame shown in column (b). Although the material appear clearly synthetic, our quantitative results show that this augmentation is instrumental in providing more training data for the downstream task. Note also that generating new videos is the less time-consuming part of the SynthFun3D pipeline, and therefore this simple strategy allows cheap generation of training-ready data, without requiring specific hardware.

2.2. Retrieval results

We report in Fig. 2 some examples of objects and masks obtained with our retrieval strategy, compared to the objects obtained with the same prompts with Holodeck [5] default strategy. Our strategy can accurately retrieve both objects and the segmentation mask of a specific part, while Holodeck is limited to objects only. We can observe that SynthFun3D objects are retrieved with a good degree of accuracy, in particular in Fig. 2(b-c-d). In Fig. 2(a), our method does not retrieve a "blue storage chest" as requested, but instead retrieves a desk with the same functionality, and the retrieved mask is correct as it refers to the bottom drawer. We attribute this type of errors to two main factors. The first is the limited span of PartNet-Mobility [4], which comprises many different object types, but the variety in appearance and colors is limited. The second is the fact that our method focuses on the functionality of objects, and therefore will consider objects with a relatively low similarity score in the vector search, but with the requested functionalities.

3. Prompts

As our retrieval strategy makes extensive use of the metadata provided by PartNet-Mobility [4], in Sec. 3.1 we discuss the metadata structure and how it was used in SynthFun3D. We report the prompts used to guide the retrieval step of SynthFun3D in Sec. 3.2.

3.1. Partnet-Mobility annotations

Fig. 3 shows an example of the metadata provided by PartNet-Mobility, which is essential to the design of our retrieval strategy. Each asset is composed by a set of parts, each associated with a label, and we manually select a set of label names to be used as functional elements for the whole dataset (*e.g.*, "*handle*","*knob*","*switch*"). Fig. 3 shows in red the mask and label of each functional element as an example asset. Additionally, given an asset, PartNet-Mobility provides a hierarchy in which the parts are organized. To enrich the representation of each asset provided to the LLM, together with the label of each functional element, we also consider the label of its parent part (i.e., the part of higher level in the hierarchy). Specifically, for each functional element, we provide to the LLM the label of the functional element concatenated with the label of its parent part. Fig. 3 shows in green the mask and labels for each parent part of the functional elements. In this example, all functional elements are labeled "*handles*", but while the top two handles are on drawers, the bottom two are on cabinet doors. By concatenating the labels of the parent parts, we obtain two "*cabinet door handles*" and two "*drawer handles*".

Figure 1. Qualitative examples of synthetic data generated by SynthFun3D from the action description shown on top of each row. In particular, we show: (a) a view of the layout, (b) a rendered RGB image, (c) the functional element mask, (d-e) variants augmented with different colors and materials.

On the right-hand side of Fig. 3 we report a representation of the data provided to the LLM for this asset: along with the label names, we report the X and Y axis of the normalized centroids of the functional elements. As for the object placement, we define left and right from the point of view of a person standing in front of the object.

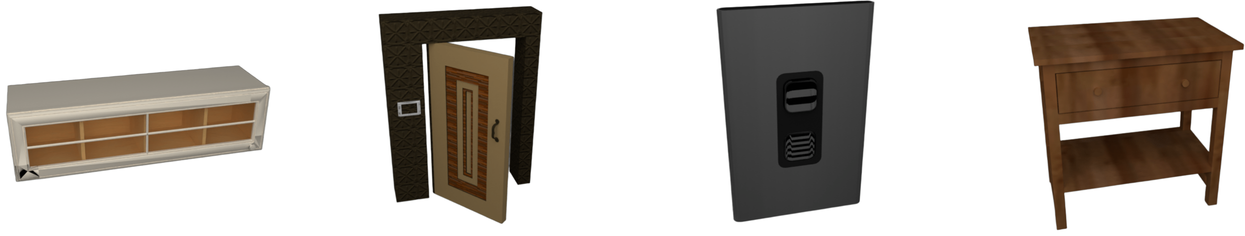
3.2. Retrieval prompts

In Fig. 4 we show the prompt used to parse the task description `D`, as described in Sec. 3.3 of the main paper. The LLM is instructed to extract the object name (that correspond to `O`), the layout prompt `L`, the object type and the context-free prompt. The object type can be *door*, *window* or *other*. This division is necessary as we follow the structure of Holodeck, which separates the databases to perform retrieval among structural objects (*door*, *window*) and standard furniture objects (*other*). The context-free prompt instead is a sightly modified version of `D`, in which we instruct the LLM to remove any reference to the positioning of the object in the scene. E.g., "*Open the bottom drawer of the nightstand next to the bed*" should become "*Open the bottom drawer of the nightstand*". In practice, we use the context-free prompt in the next steps of the retrieval, as we found that it produces better results than using the default task description `D`. In Fig. 4, <*prompt*> represents the task description `D`.

In Fig. 5 we show the prompt used to extract the requirement used in the requirement-based filtering described in Sec. 3.4 of the main paper. The objective of this step is to extract a requirement, expressed as number of occurrences of a functional element, that can be used to filter the previously retrieved assets. Additionally, we ask the LLM to provide the name of the object part, which is the functional element name `F`. We input the LLM with a set of examples, and ask it to output the reasoning along with the requirement string. The <*funclist*> is a list of functional element names, obtained from the ones contained in all the assets previously retrieved, while <*prompt*> is the context-free prompt.

In Fig. 6 we show the prompt used to select the final object

Holodeck [5]



SynthFun3D (Ours)

(a) *Open the bottom drawer of the blue storage chest* (b) *Close the door* (c) *Turn on the ceiling light* (d) *Open the bottom drawer*

Figure 2. Examples of object retrieval obtained with SynthFun3D. Top row: object obtained with Holodeck [5] default strategy, which ensembles vector search with embeddings from CLIP [1] and SentenceBert[2]. Middle row: objects retrieved with the same prompts as Holodeck, with SynthFun3D. Bottom row: part mask retrieved with SynthFun3D. Under each column we report the context-free prompt derived from the task description `D`.

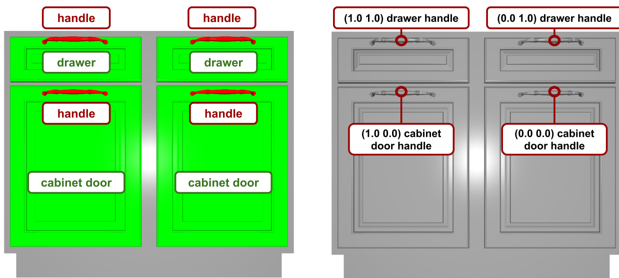


Figure 3. Example of metadata provided for an asset in PartNet-Mobility [4]. Left: functional elements masks and their labels (in red), and the parents in the hierarchy of the functional elements along with their labels (in green). Right: representation of the asset provided to the LLM in the retrieval phase, consisting in the normalized coordinates of the centroid and the label name for each functional element.

and the mask of `F`, which is the final step of the retrieval strategy detailed in Sec. 3.4 of the main paper. As preamble, we provide a list of examples of typical functional elements configurations, along with the description of the frame of reference to be used. In the prompt template, *<object>* and *<func>* are the target object name `O` and the functional element name `F` respectively, while *<prompt>* is the context-free prompt.

References

[1] Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning Transferable Visual Models from Natural Language Supervision. In: International Conference on Machine Learning (ICML) (2021) 4

[2] Reimers, N., Gurevych, I.: Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks (2019) 4

```
You have to provide a textual description of the layout of a room (called the layout_prompt), given a functional
     prompt, that described an action to be carried out in the room. This action involves using small interactive
     elements of an object to accomplish a task, for example opening a drawer, turning on a tv, or turning on the
     room light. The answer should be concise, and only describe the characteristics and relationships of the object.
     Additional objects can be added, as long as they are consistent with the room type.  It's very important to
     describe the layout of the object mentioned in the prompt, but other objects are optional. Additionally, you
     have to provide the name of the object that contains the small interactive element that allows to carry out the
     action. Additionally, you have to provide the object type, which can be "door", "window", or "other".
     Additionally, you have to provide a context-free version of the functional prompt. This should exclude any
     information that describe the position of the object in the scene ("next to the TV", "in the living room").
     Format the output in the following YAML format:

```yaml
layout_prompt: the textual description of the room layout
context_free_prompt: the context-free version of the functional prompt
object_name: the name of the object
object_type: can be "door","window" or "other"
```

A few examples. If the functional prompt is "Open the fourth drawer of the cabinet next to the TV", a correct output
     would be the following:
```yaml
layout_prompt: A living room with a TV and a cabinet. The cabinet is next to the TV and has multiple drawers.
context_free_prompt: Open the fourth drawer of the cabinet
object_name: cabinet
object_type: other
```

For "Open the bedroom door", a correct output would be the following:
```yaml
layout_prompt: A bedroom with a bed, a door, a nightstand
context_free_prompt: Open the bedroom door
object_name: door
object_type: door
```

For "Open the window next to the wardrobe", a correct output would be the following:
```yaml
layout_prompt: A room with a window, a wardrobe, and a bed
context_free_prompt: Open the window
object_name: window
object_type: window
```

For "Turn on the bedroom light", a correct output would be the following:
```yaml
layout_prompt: A room with a bed, a ceiling light, and a light switch
context_free_prompt: Turn on the light
object_name: light switch
object_type: other
```

Here is the functional prompt: <prompt>
```

Figure 4. Prompt used to parse the task description D as described in Sec. 3.3 of the main paper.

[3] Team, G.: Gemma 3 technical report (2025), https://arxiv.org/abs/2503.19786 1

[4] Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., Yi, L., Chang, A.X., Guibas, L.J., Su, H.: SAPIEN: A SimulAted Part-Based Interactive ENvironment. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2020) 2, 4

[5] Yang, Y., Sun, F.Y., Weihs, L., VanderBilt, E., Herrasti, A., Han, W., Wu, J., Haber, N., Krishna, R., Liu, L., et al.: Holodeck: Language Guided Generation of 3D Embodied Ai Environments. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2024) 2, 4

```
You are an expert robotic manipulation system. You have access to a set of 3D object assets. Each object possess a set of functional elements, expressed in a way such as "handle: 5", meaning that this object has 5 instances of a part named "handle". A functional element is the part of an object that can be physically interacted with (e.g., grabbed, pushed, pulled). You will be given a prompt that describes an action to be carried out on an object, and you must determine the requirement, expressed as quantity of a certain object part, that is consistent with the request. You should answer in the following structured output format:

object: this is the name of the object on which the operation is to be performed
object_part: this is the name of the interactive part of the object that is relevant to the operation
object_requirement_description: this should answer the question "How many <object_part> should this <object> have to satisfy the described prompt?"
object_requirement: this is the requirement expressed in the form "element <symbol> <N>", where <symbol> is a mathematical symbol such as >, <, >=, <=, and where N is an integer

The prompt may not explicitly mention the functional element, but you should infer it from the action to be performed. Additionally, the prompt may contain a requirement on the position of the object part, such as "the leftmost drawer", or "the right handle". When this happens, you should assume that more objects part are present (e.g., "the top handle" implies at least 2 handles, one on the top and one on the bottom). Conversely, when the prompt does not contain any positional requirement, you should assume that a single functional element is present, to avoid ambiguity. Here are a few examples:

For "open the third drawer of the cabinet from the bottom", a correct output would be the following:

object: cabinet
object_part: handle
object_requirement_description: The cabinet should have at least 3 handles to satisfy the request of opening the third drawer.
object_requirement: handle >= 3

For "Regulate the temperature on the oven", a correct output would be the following:

object: oven
object_part: knob
object_requirement_description: The oven should have exactly one knob to satisfy the request of regulating the temperature.
object_requirement: knob = 1

For "Open the top left drawer of the nightstand", a correct output would be the following:

object: nightstand
object_part: handle
object_requirement_description: The existence of at least 4 handles is required to satisfy the request of opening the top left drawer.
object_requirement: handle >= 4

Only answer with the requirement, do not add any additional text. In this case, the possible functional part names are only the following: <funclist>.
Only values from this list may appear in `object_part` and in `object_requirement` fields.
The current prompt is: <prompt>
```

Figure 5. Prompt used to extract the requirement on the number of functional elements, as described in Sec. 3.4 of the main paper.

````
You are an expert robotic manipulation system. You have access to a small set of <object>. Each <object> possesses a
     set of objects of type <func>. You will be given a list of <object> objects, expressed as an object_id and a
     list of 2D centroids. Each centroid is the center of one of its <func>. You will be given an action to be
     carried out on an <object>, that references one of the <func>. You must choose the instance of <object> with the
     best disposition of <func> to satisfy the request. A few rules:
- when you are asked to "open the leftmost/rightmost/left/right X of the Y", this implies that the X (and the
     corresponding functional objects) are arranged horizontally, and that there are more than one.
- when you are asked to "open the top/bottom X of the Y", this implies that the X (and the corresponding functional
     objects) are arranged vertically, and that there are more than one.
- when you are asked to "open the Nth X of the Y from the left (or right)", this implies that the X (and the
     corresponding functional objects) are arranged horizontally, and that there are at least N.
- when you are asked to "open the Nth X of the Y" (without any frame of reference) this implies that the X (and the
     corresponding functional objects) are arranged vertically, and that there are at least N.
- when you are asked to "open the top left X of the Y, this implies that there are at least 4 X, arranged in a 2x2
     grid.

The format of the input for each object is the following:


```yaml
- id: id1
  parts:
    - id: partid1
      name: partname1
      centroid: [x1, y1]
    - id: partid2
      name: partname2
      centroid: [x2, y2]
```

The field `id` specifies the unique ID of the part, while `[x, y]` are the normalized 2D coordinates of the centroid
     of the part, in the range [0,1]. The field `name` specifies the name of the part, which can be useful to
     disambiguate between different parts. You should use the name in case of ambiguity, considering that the same
     object may have multiple parts with the same name (e.g., a cabinet may have handles on both doors and drawers).
For example, if you are looking for "door handle", you should choose the part with name "door handle" or "handle",
     and not a part with name "drawer handle". If you have generic "handle", you can assume that they are the parts
     you can use. Consider the following facts about the coordinate system:
- The X coordinate represents the horizontal axis. A value close to 0 indicates a position on the right of the
     origin, while a value close to 1 indicates a position on the left of the origin.
- The Y coordinate represents the vertical axis. A value close to 0 indicates a position at the bottom of the origin,
     while a value close to 1 indicates a position at the top of the origin.
The output, should be a list, with an element for each object. Each object is structured as follows:

```yaml
- id: object_id1
reasoning: "briefly reason about this object in the list, explaining why it does or does not satisfy the request"
suitable: true/false (true if this object is a valid candidate, false otherwise)
part_id: id of the part that best satisfies the request, or None if no part satisfies the request
```
Reasoning strings should always be enclosed in double quotes. Strictly follow the output format, in particulat the
     spaces, and do not add any additional text. The current prompt is: <prompt>
````

Figure 6. Prompt used to retrieve the object and mask in the final step of the retrieval strategy of SynthFun3D, as described in Sec. 3.4 of the main paper.