FunFact: Building Probabilistic Functional 3D Scene Graphs via Factor-Graph Reasoning

Supplementary Material

A. Part Filtering

To filter out spurious part detections on the background or having unreasonably large overlap with the parent objects, we implement a part filter based on the overlap ratio between the part's bounding box and the parent object's bounding box. A part detection is discarded if the intersection between the part's bounding box and the parent object's bounding box occupies:

1) less than 30% of the part's bounding box (*i.e.*, background objects are incorrectly detected as functional parts of the parent object), or
2) more than 70% of the object's bounding box (*i.e.*, the model misdetects the object itself as one of its functional parts)

B. *FunThor* Generation Process

To build *FunThor*, we follow a principled pipeline that produces part-aware geometries and dense functional-relation annotations based on AI2-THOR [23] infrastructure. In addition, we generate posed RGB-D images for each scene to support perceptual functional scene understanding.

Scene selection. In all AI2-THOR scenes, a single light switch controls the lighting of the entire environment, regardless of how many light fixtures are present or how many sub-switches the switch includes. To better reflect the real world settings, we exclude scenes where sub-switches cannot be heuristically assigned to controlled lights (*e.g.*, a dual switch controls a single light or a quad switch controls three rows of lights). From the remaining set, we intentionally select scenes with higher visual ambiguity. For example, scenes with a hanging light controlled by a light switch surrounded by multiple floor lamps. For kitchen-type scenes, we additionally filter out those containing a pan on a stove burner to support burner detection. Fig. 7 illustrates examples of excluded scenes.

Figure 7. **Examples of scene filtering for *FunThor*.** Scenes are excluded if switch-light associations are ambiguous (left) or if pans are present on stove burners (right).

Part-level annotation. We decompose and annotate relevant object CAD models from AI2-THOR–Habitat dataset [19] in Blender, separating them into semantically meaningful parts (*e.g.*, cabinet body and handle, appliance body and buttons, knobs and burners). This produces a library of part-aware assets that can be instantiated in AI2-THOR scenes.

Ground-truth object and part point clouds. Using these annotated assets, we construct ground-truth 3D point clouds for both objects and parts within AI2-THOR scenes. For each object instance, we uniformly sample points along objects' and their parts' mesh surface, resulting in an object- and part-centric point cloud and object-part hierarchy.

Functional relation annotation. We then annotate functional relations using a set of predefined annotation rules. Each rule is defined as a functional triplet in the format (first label, relation, second label). The rules are grouped into categories based on the matching strategies used, as shown below:

1. Exact matching - We annotate edges between node pairs with relation type "relation" if and only if the first node's

label exactly matches the triplet's "first label" and the second node's label exactly matches the triplet's "second label". Representative examples of rules employing this strategy include: knife slices apple; faucet fills kettle with water.

2. Proximity-based matching - For every node whose label matches the first label of the triplet, we identify the closest node matching the triplet's "second label". If such a node exists and the Euclidean distance between the node pair's centers is less than one meter, we annotate a functional relation labeled as "relation" between them. Some examples of rules using this strategy include: curtains cover/uncover windows; faucets fill sinks with water; and faucets fill bathtubs with water.
3. Part-object and part-part matching - For objects that exhibit toggleable or openable properties in AI2-THOR and possess explicit functional part annotations such as power switches or handles (see Appendix B), we annotate the corresponding node pairs with their respective functional relations. Representative rules in this category include: lever pushes down to activate toaster; handle pulls to open door.
4. Manual matching - For semantically ambiguous relations, we manually record affected node pairs and store these annotations in a JSON file for subsequent loading by the annotation pipeline. Currently, only stove knob-burner associations require manual matching.

These rules are applied consistently across scenes to produce dense and unbiased annotations for both part–object and object–object relations.

View sampling and RGB-D capture. To mimic realistic partial observations, we randomly sample reachable poses in each scene and a camera pitch angle for each pose. From these viewpoints, we capture RGB-D sequences and corresponding camera poses, with each scene containing 60 frames. These sequences form the input to our mapping and functional inference pipeline.

Visibility filtering. Not all objects are visible from the sampled camera trajectories (*e.g.*, items stored inside closed cabinets). To ensure that evaluation only considers observable entities, we first fuse the posed RGB-D sequence into a class-agnostic point cloud and then retain only those ground-truth object and part points that lie within a fixed radius of this fused cloud. Objects whose visible subset contains fewer than 10 points are discarded, and any functional relations involving discarded objects are removed. We store the remaining visible nodes and their functional edges as the final *FunThor* benchmark used in our experiments.

C. Instance Association and Merging Details

We adopt a two-stage instance association and merging strategy to consolidate object and part detections across multiple frames into a unified 3D object-part map. In the first stage, we perform spatial association by computing the Intersection over Union (IoU) between the 3D bounding boxes of newly detected instances and those of existing scene instances. If the IoU exceeds a threshold of 0.03, the detections are considered to potentially correspond to the same instance, and we proceed to the second stage of semantic verification. In this stage, we use DINOv2 with registers [7] to extract 2D features from the RGB image for each detection and compute the cosine similarity between the feature of the newly detected instance and the existing scene instance. If the similarity exceeds a threshold of 0.5, we confirm that the two instances correspond to the same object or part and merge them by aggregating their point clouds. The label of the merged instance is then updated to reflect the most frequently occurring label among all detections composing the instance. If either the spatial association or semantic verification fails, the newly detected instance is treated as a new object or part and added to the scene map. It is worth noting that we only permit merging between instances of the same category (*i.e.*, object-to-object or part-to-part) to prevent erroneous associations. Objects are merged prior to parts. When objects are merged, their contained parts are also aggregated and become parts of the merged object.

D. Inference Time Analysis

We analyze the inference time of *FunFact* on the FunGraph3D dataset, which consists of 14 scenes with an average of 203.3 RGB-D frames per scene. In total, *FunFact* processes all 14 scenes in 27479s ($\simeq$ 7.6h).

Scene reconstruction. *FunFact* processes each frame in 9.7s on average, encompassing GPT-based scene analysis, object and part detection, instance association, and merging. Although *FunFact* queries GPT once per frame for scene analysis, this step accounts for nearly 93% of the total stage runtime. Since scene analysis is performed independently for each frame, it can be straightforwardly parallelized across multiple concurrent GPT calls, which would significantly reduce the overall runtime.

Functional scene graph creation. *FunFact* processes each scene in 26.3s on average, encompassing LLM-based functional proposal generation, edge and dual graph construction, and factor graph inference.

E. Pseudo-code for Functional Scene Graph Construction

A high-level pseudo-code for building the functional scene graph using *FunFact* is provided in Algorithm 1. The process consists of three main phases: (1) generating part-object and part-part connection factor graph (Lines 3–16), (2) generating object-object connection factor graph (Lines 17–32), and (3) performing global inference to es-

timate confidence scores and construct the final functional scene graph (Lines 33–39). At the beginning, we initialize two empty lists, $\mathcal{G}_L$ (Line 1) and $\mathcal{G}_R$ (Line 2), to store EdgeGroups for part-object/part-part connections and object-object connections, respectively. For each LLM-proposed semantic functional proposal (*e.g.*, remote control operates TV), we create an EdgeGroup that encapsulates all resulting edges (*e.g.*, the edges connecting all pairs of remote controls and TVs) and their associated factor graph. The default edge confidence is set to the semantic confidence of the functional proposal. In phase 1, we iterate over each object with parts (Line 4) and obtain LLM-proposed part-object and part-part connections (Line 5). We assume proximity always holds for part-object and part-part connections, and we add a CardinalityFactor only when the proposal is believed by LLMs to be one-to-one (Line 10) (*i.e.* the functional connection is mutually exclusive, such as a stove knob only controlling one stove burner). In phase 2, we first obtain LLM-proposed object-object connections (Line 18) and build different factor graphs based on the nature of the proposed connections. If the proposal is one-to-one (Line 21), we include both ProximityFactor and CardinalityFactor; if the proposal requires proximity but is not one-to-one (Line 26), we only include ProximityFactor. Finally, in phase 3, we combine all EdgeGroups (Line 34) and their factor graphs (Line 35), and perform marginal inference (Line 36) to estimate confidence scores for all edges. These scores are then used to update the confidence of each EdgeGroup (Line 37) before constructing the final functional scene graph (Line 38).

Algorithm 1 Build Functional Scene Graph

Require: Object map $\mathcal{O}$ with contained parts info
Ensure: Functional scene graph

1: $\mathcal{G}_L \leftarrow []$ ▷ Local (part-object) based groups
2: $\mathcal{G}_R \leftarrow []$ ▷ Remote (object-object) groups

3: ▷ **Phase 1: Part-Object and Part-Part Connections**
4: **for** each object $o \in \mathcal{O}$ with parts **do**
5: $\mathcal{P} \leftarrow$ LLM-Propose-Parts(o, o.parts)
6: **for** each proposal $p \in \mathcal{P}$ **do**
7: Create EdgeGroup g from p
8: Build FactorGraph $\mathcal{F}_g$ with:
9: ProximityFactor(g) ▷ Encodes spatial closeness
10: **if** p.isOneToOne **then**
11: CardinalityFactor(g) ▷ Enforce one-to-one
12: **end if**
13: $g.\mathcal{F} \leftarrow \mathcal{F}_g$ ▷ Attach factors to g
14: Add g to $\mathcal{G}_L$ ▷ Store local group
15: **end for**
16: **end for**

17: ▷ **Phase 2: Object-Object Connections**
18: $\mathcal{R} \leftarrow$ LLM-Propose-Objects($\mathcal{O}$) ▷ LLM object connection proposals
19: **for** each proposal $r \in \mathcal{R}$ **do**
20: Create EdgeGroup g from r
21: **if** r.isOneToOne **then**
22: Build FactorGraph $\mathcal{F}_g$ with:
23: ProximityFactor(g) ▷ Encodes spatial closeness
24: CardinalityFactor(g) ▷ Enforce one-to-one
25: $g.\mathcal{F} \leftarrow \mathcal{F}_g$
26: **else if** r.requiresProximity **then**
27: Build FactorGraph $\mathcal{F}_g$ with:
28: ProximityFactor(g) ▷ Encodes spatial closeness
29: $g.\mathcal{F} \leftarrow \mathcal{F}_g$
30: **end if**
31: Add g to $\mathcal{G}_R$ ▷ Store remote group
32: **end for**

33: ▷ **Phase 3: Global Inference**
34: $\mathcal{G} \leftarrow \mathcal{G}_L \cup \mathcal{G}_R$
35: $\mathcal{F} \leftarrow \bigcup_{g \in \mathcal{G}} g.\mathcal{F}$ ▷ Global factor graph
36: $\mu \leftarrow$ MarginalInference($\mathcal{F}$)
37: UpdateConfidence($\mathcal{G}$, μ)
38: $\mathcal{S} \leftarrow$ BuildFSG($\mathcal{O}$,$\mathcal{G}$) ▷ Final Functional Scene Graph
39: **return** $\mathcal{S}$

F. Alternative VLM Backbones

We evaluate the sensitivity of *FunFact* to the choice of VLM backbone by replacing GPT-4.1 with gemini-3-flash-preview and the open-weight qwen3-vl-32b-instruct. Scene reconstruction and triplet evaluation results are reported in Tab. 4 and Tab. 5, respectively. *FunFact* performs consistently well across all three backbones, demonstrating that its strong performance stems from the pipeline design rather than reliance on any particular VLM.

G. Exclusive and Non-Exclusive Matching

When calculating recall to evaluate the mapping quality, following the evaluation protocol in [46], it is necessary to count how many ground-truth nodes are correctly mapped to predicted nodes. However, [46]'s protocol allows multiple ground-truth nodes to map to the same predicted node, which may lead to an overestimation of mapping quality. For instance, when multiple cabinets are located in close proximity, a single merged detection that spatially covers all cabinets may achieve perfect recall, as each ground-truth cabinet can map to the same predicted node. To more accurately assess mapping quality, we additionally evaluate the results under an exclusive matching constraint, where each predicted node maps to at most one ground-truth node. Table 6 presents the results under this setting. We observe a drop in recall for all methods under exclusive matching, indicating that some predicted nodes are mapped to multiple ground-truth nodes under non-exclusive matching. However, the relative performance between different meth-

Table 4. **Scene Reconstruction with Alternative VLM Backbones.** We report Recall@K (**R@3/R@10**; higher is better) for three node categories—*Objects*, *Interactive Elements*, and *Overall Nodes*—on **SceneFun3D** and **FunGraph3D**, using gemini-3-flash-preview and qwen3-vl-32b-instruct as drop-in replacements for GPT-4.1. Bold numbers mark the best result per column.

VLM Backbone	**SceneFun3D**						**FunGraph3D**					
	Objects (↑)		Inter. Elements (↑)		Overall Nodes (↑)		Objects (↑)		Inter. Elements (↑)		Overall Nodes (↑)	
	R@3	R@10	R@3	R@10	R@3	R@10	R@3	R@10	R@3	R@10	R@3	R@10
gemini-3-flash-preview	**94.3**	97.1	63.2	**74.5**	**73.5**	**82.0**	**93.2**	**95.9**	**68.8**	**78.8**	**79.0**	**85.9**
qwen3-vl-32b-instruct	92.4	**98.1**	**64.2**	72.2	**73.5**	80.8	88.4	94.5	59.4	73.3	71.6	82.2

Table 5. **Triplet Evaluation with Alternative VLM Backbones.** We report node association, edge prediction and overall triplet recall as Recall@K (R@5/R@10) on **SceneFun3D** and **FunGraph3D**, and (R@3/R@5) on **FunThor**, using gemini-3-flash-preview and qwen3-vl-32b-instruct as drop-in replacements for GPT-4.1. Bold numbers mark the best result per column.

VLM Backbone	**SceneFun3D**						**FunGraph3D**						**FunThor**					
	Node Assoc. (↑)		Edge Pred. (↑)		Overall Triplets (↑)		Node Assoc. (↑)		Edge Pred. (↑)		Overall Triplets (↑)		Node Assoc. (↑)		Edge Pred. (↑)		Overall Triplets (↑)	
	R@5	R@10	R@5	R@10	R@5	R@10	R@5	R@10	R@5	R@10	R@5	R@10	R@3	R@5	R@3	R@5	R@3	R@5
gemini-3-flash-preview	**67.2**	**72.3**	**51.1**	**68.1**	**34.4**	**49.2**	**74.3**	**79.6**	**61.5**	72.1	**45.2**	**57.4**	**70.9**	**72.3**	**76.2**	**78.5**	**54.1**	**56.8**
qwen3-vl-32b-instruct	66.2	69.7	46.5	67.7	30.8	47.2	69.6	74.8	60.6	**73.8**	42.2	55.2	68.9	70.9	72.5	78.1	50.0	55.4

Table 6. **Mapping Quality under Exclusive Matching.** We report Recall@K (**R@3/R@10**; higher is better) for three node categories—*Objects*, *Interactive Elements*, and *Overall Nodes*—on **SceneFun3D**, **FunGraph3D**, and **FunThor** under the exclusive matching constraint, where each predicted node maps to at most one ground-truth node. This provides a more accurate assessment of mapping quality compared to non-exclusive matching, which may overestimate performance when a single merged detection covers multiple ground-truth objects. All methods exhibit decreased recall under this stricter evaluation, but **FunFact** maintains superior performance across nearly all categories and datasets.

Methods	**SceneFun3D**						**FunGraph3D**						**FunThor**					
	Objects (↑)		Inter. Elements (↑)		Overall Nodes (↑)		Objects (↑)		Inter. Elements (↑)		Overall Nodes (↑)		Objects (↑)		Inter. Elements (↑)		Overall Nodes (↑)	
	R@3	R@10	R@3	R@10	R@3	R@10	R@3	R@10	R@3	R@10	R@3	R@10	R@3	R@10	R@3	R@10	R@3	R@10
OpenFunGraph	78.1	79.0	**64.4**	**78.3**	69.1	78.5	61.1	68.5	42.1	52.5	50.3	59.2	43.7	52.3	34.8	40.4	41.5	49.3
FunFact (Ours)	**90.5**	**90.5**	**64.4**	75.5	**73.2**	**80.4**	**84.9**	**89.0**	**62.9**	**75.2**	**72.1**	**81.0**	**57.2**	**61.0**	**64.5**	**68.8**	**59.1**	**63.0**

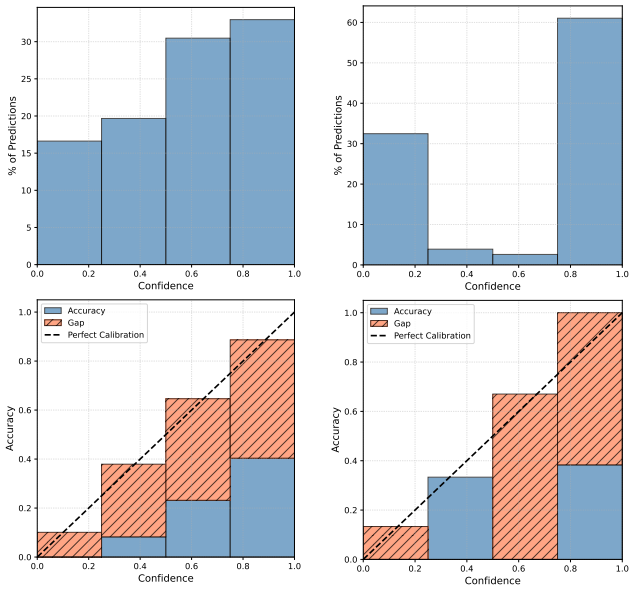
ods remains consistent, with our method outperforming the baselines by a significant margin except Recall@10 for interactive elements in SceneFun3D.

H. Confidence Histograms and Reliability Diagrams

Fig. 8 presents confidence histograms and reliability diagrams for *FunFact* and OpenFunGraph on the *FunThor* dataset, providing a visual assessment of model calibration [29]. The confidence histogram illustrates the distribution of predicted confidence scores, while the reliability diagram compares predicted confidence against actual accuracy. We follow the procedure in [13] to plot the figures. We use a bin size of 4 (*i.e.*, each bin spans a confidence interval of 0.25) to be consistent with the settings in Section 4.4. In the reliability diagram, the diagonal line represents perfect calibration, and the red bars indicate the gap between average confidence and accuracy within each bin. Since OpenFunGraph estimates confidence scores only for classes containing "outlet," "switch," "power," and "remote," we assign a confidence of 1.0 to all other predictions when constructing these diagrams.

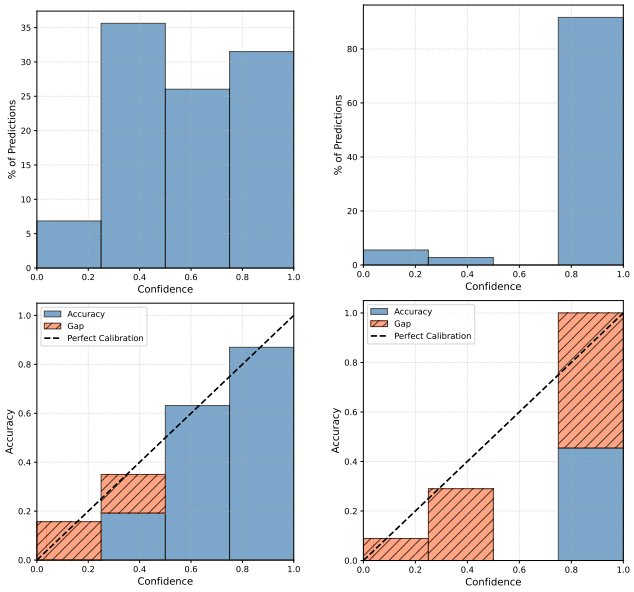
When considering all predictions, *FunFact* produces a more uniform distribution of confidence scores across the full range, whereas OpenFunGraph assigns confidence scores only to a fixed subset of classes. Both models exhibit overconfidence and achieve comparable performance in the high-confidence range. This behavior is expected, as both methods rely primarily on visual data and LLM priors for functional relation prediction, with limited visual cues available to assess the reliability of high-confidence predictions.

However, when examining ambiguous classes specifically, *FunFact* demonstrates superior calibration compared to OpenFunGraph, yielding confidence scores that better align with actual accuracy. This improvement stems from *FunFact*'s use of a factor graph to estimate confidence scores holistically across all edges, incorporating global context and inter-dependencies among multiple functional relations. For instance, when multiple switches and lights coexist in a scene, *FunFact* can reason about mutual exclusivity among different switch-light pairs, resulting in better-



(a) All predictions (Ours)

(b) All predictions (OpenFunGraph)



(c) Ambiguous classes only (Ours)

(d) Ambiguous classes only (OpenFunGraph)

Figure 8. **Calibration analysis on *FunThor* dataset.** Each subfigure shows a confidence histogram (top) and reliability diagram (bottom).

calibrated confidence scores. In contrast, OpenFunGraph makes independent predictions for each object pair without considering broader contextual information, leading to less reliable confidence estimates for ambiguous classes.

I. Additional Qualitative Results

In this section, we present qualitative results for both the mapping stage and functional scene graph construction stage of *FunFact* on the SceneFun3D (Fig. 9 and 10), FunGraph3D (Fig. 11), and *FunThor* datasets (Fig. 12). Each figure comprises four subfigures arranged from left to right and top to bottom as follows: **(a: top-left)** ground truth point cloud with node and functional relation annotations; **(b: top-right)** reconstructed object- and part-centric point cloud with predicted node labels and functional relations; **(c: bottom-left)** predicted functional scene graph with one node selected (highlighted); **(d: bottom-right)** visualization of the selected node in the reconstructed point cloud along with all functional relations associated with that node.

In subfigures (a) and (b), black node labels indicate matched nodes, while red node labels indicate unmatched nodes. Similarly, blue arrows denote matched functional relations, and red arrows denote unmatched functional relations. In subfigure (c), the selected node is highlighted in dark blue, red edges indicate object-part hierarchical relationships (always directed from object to part), and gray edges represent functional relations. In subfigure (d), the bounding box of the selected node is highlighted in blue, and all functional relations involving the selected node are visualized using green lines.

For the SceneFun3D and FunGraph3D datasets, our reconstruction quality is generally high, with most objects and parts accurately detected and labeled. Our pipeline consistently identifies more functional elements and functional relations than those annotated in the ground truth, revealing the sparse annotation limitations of existing datasets. In Fig. 9, we demonstrate some misalignment between predicted node labels and ground truth labels, where all drains are labeled as "button/knob" and the trash bin pedal is labeled as "handle" in the ground truth. Due to this misalignment, some functional relations are incorrectly marked as unmatched despite being correctly predicted. This issue highlights the limitations of existing evaluation protocols for open-vocabulary functional scene graphs.

For the *FunThor* dataset, our method effectively reconstructs objects and parts in complex indoor scenes, accurately capturing functional relations such as stove knobs operating burners and light switches controlling lights. These qualitative results demonstrate the effectiveness of *FunFact* in mapping and understanding functional scene graphs across diverse datasets.

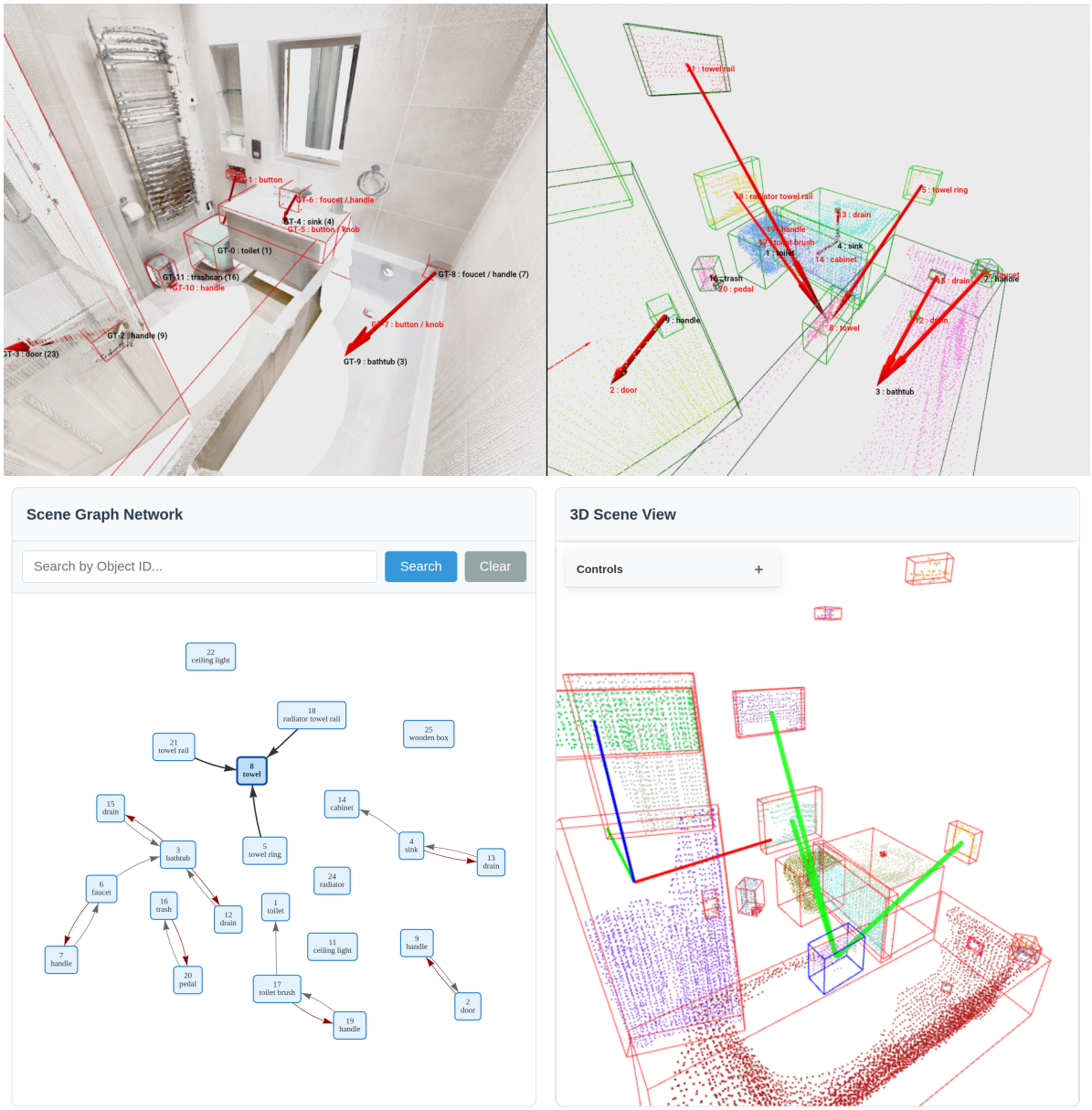


Figure 9. **Scene *dev/421063/42444511* of SceneFun3D.** See Section I for detailed subfigure descriptions. As shown in the top-right subfigure, we predict semantically correct functional relations in this bathroom scene, but none of them are matched to ground truth as indicated by the red arrows. This is due to several annotation issues in the dataset: (1) node label mismatches where "drains" are incorrectly labeled as "button/knob" and the trash bin "pedal" is labeled as "handle" in ground truth; (2) the ground truth label "foucet" is a typo and can never match our correct prediction "faucet"; (3) even when node labels match, our open-vocabulary relation prediction "handle turn or pull to open or close door" cannot be matched to the ground truth annotation "handle rotate to open or close door" because when computing semantic similarity using BERT embeddings, the ground truth relation does not rank in the top-5 most similar relations, despite clearly describing the same interaction to humans. These issues lead to zero matched relations despite semantic correctness. Beyond ground truth, our method identifies additional functional relations, including "towel ring can hold towel" and "towel radiator can dry towel" (highlighted in bottom-left graph and bottom-right point cloud viewer).

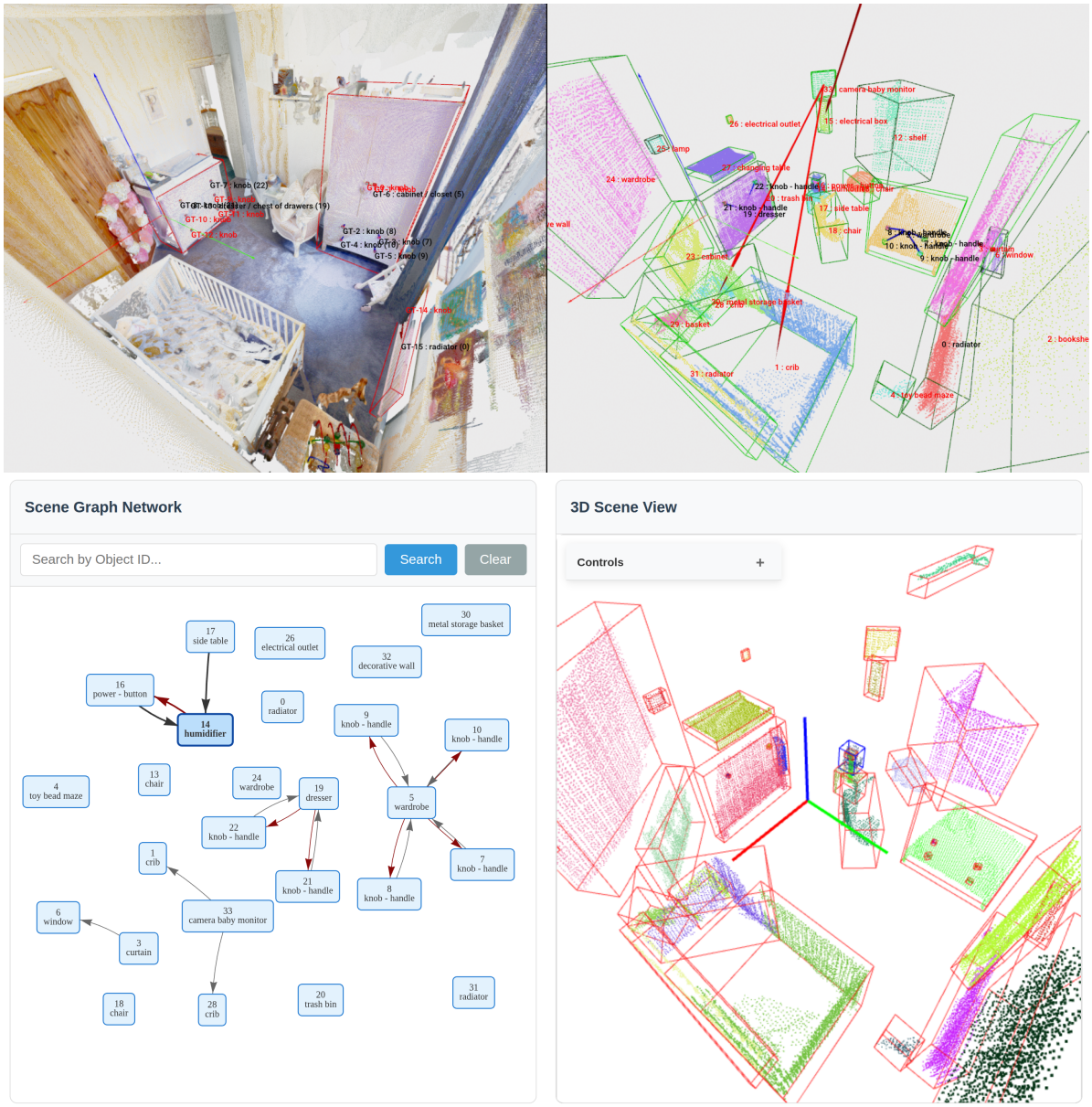


Figure 10. **Scene *dev/422007/42446017* of SceneFun3D.** See Section I for detailed subfigure descriptions. This baby bedroom scene highlights how the SceneFun3D dataset overly focuses on knob-pull interactions to open cabinets and drawers. Nearly all ground-truth annotations involve such interactions, with only one exception for radiator temperature adjustment. The dataset overlooks inter-object and intra-object functional relations that are crucial for understanding the scene. As shown in the top-right subfigure (predictions) and bottom-left subfigure (functional scene graph), our method identifies these missing relations, including "power button press to turn on/off humidifier" and "baby monitor camera watch crib", demonstrating our pipeline's ability to capture diverse functional interactions beyond the narrow focus of existing annotations.

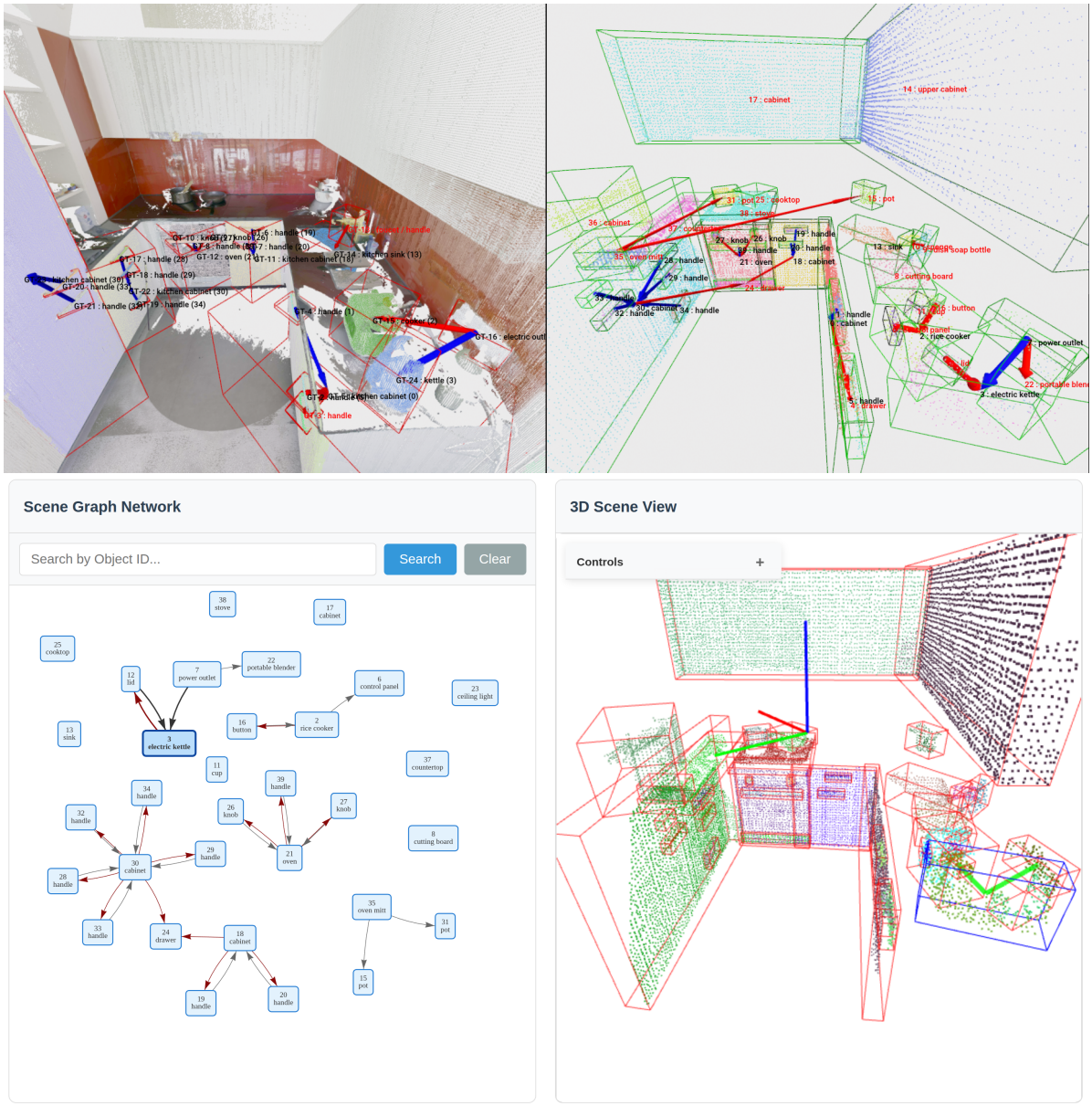


Figure 11. **Scene *4livingroom* of FunGraph3D.** See Section I for detailed subfigure descriptions. Our pipeline predicts most ground-truth functional relations, including power outlet-to-kettle connections and oven knob temperature/mode adjustments. We highlight "kettle" in the bottom two subfigures to demonstrate that our pipeline captures both inter-object relations (*e.g.*, power outlet provides power to kettle) and intra-object relations (*e.g.*, lid flip to open kettle).

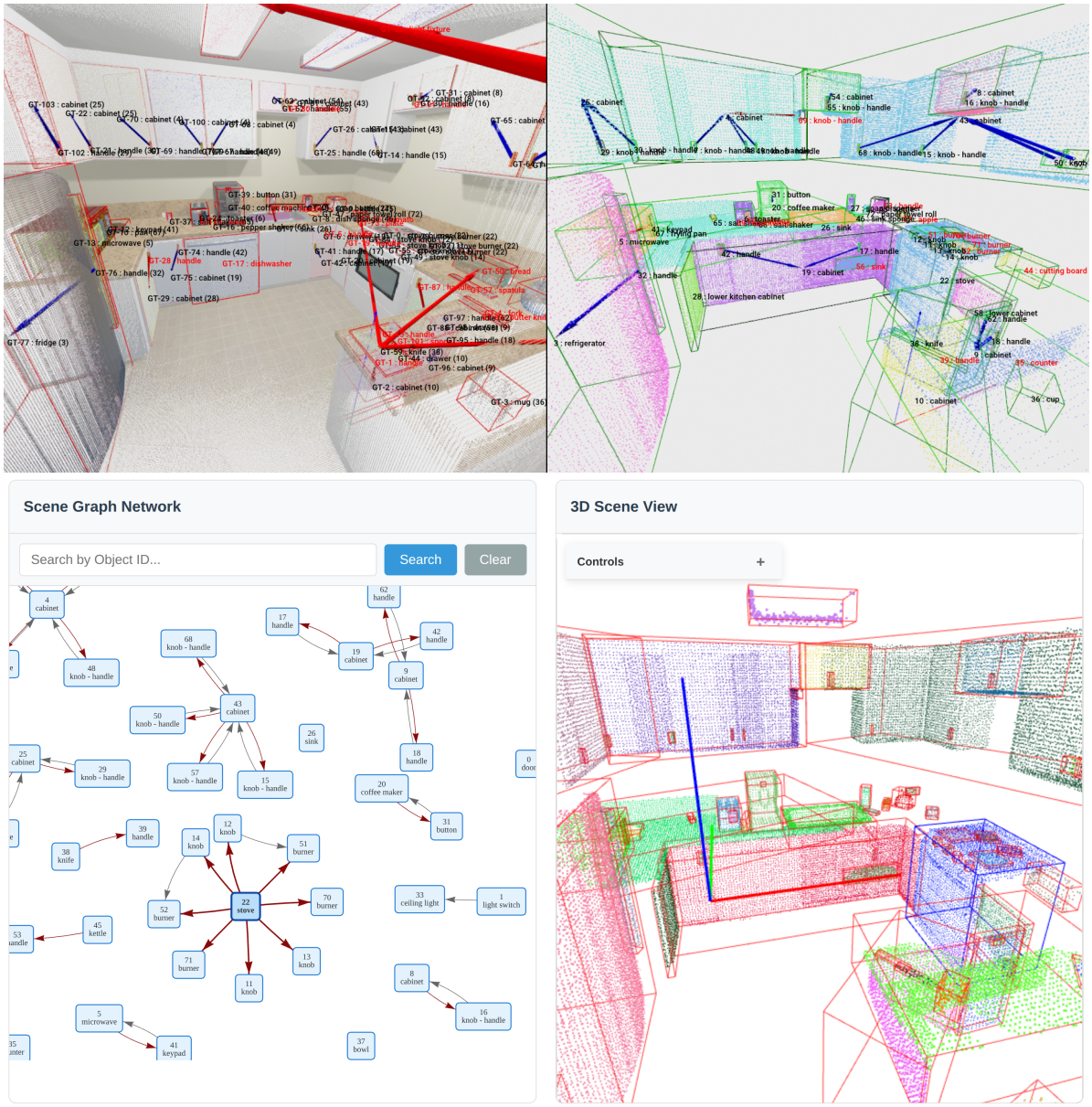


Figure 12. **Scene *FloorPlan5* of *FunThor*.** See Section I for detailed subfigure descriptions. We illustrate the dense and complete ground-truth annotations for the *FunThor* dataset (top-left). Our pipeline accurately detects functional parts of objects (*e.g.*, burners and knobs of the stove; buttons of the coffee machine). As only predicted functional relations with confidence scores above 0.5 are displayed in the functional scene graph, only the two outermost knobs are predicted to functionally connect to their respective outermost burners in the graph.

J. Potential Pipeline Adaptation for SceneFun3D dataset

As discussed in Section 4.2, SceneFun3D primarily focuses on part annotations, especially on knob and handle parts. It systematically annotates knob-shaped components (*e.g.*, sink drains and radiator valves) with the generic label "knob", leading to lexical discrepancies between open-vocabulary predictions and ground-truth labels, thereby negatively impacting both mapping and functional relation evaluation. To isolate the effects of the evaluation protocol from actual pipeline performance on this issue (*i.e.*, "knob" and "drain" are semantically far apart in CLIP embedding space), we modify the mapping pipeline to always detect knobs and handles whenever the scene analysis VLM identifies any functional parts in an object. This adaptation yields two key effects: first, it forces detection of knobs and handles independent of VLM proposals, thereby increasing recall for these part categories; second, it promotes assignment of generic "knob" and "handle" labels to merged knob- and handle-shaped components, as these labels become more prevalent during the merging process and are selected as the most common label (see Section C for merging details).

Table 7 presents results on SceneFun3D after pipeline adaptation. Both mapping and functional relation metrics improve substantially; notably, the modified pipeline surpasses OpenFunGraph on all mapping-related metrics. Functional relation recall increases significantly due to improved part detection with aligned labels. However, overall triplet Recall@5 remains below OpenFunGraph, primarily due to semantic variations in relation descriptions. For instance, the ground-truth relation "handle rotate to open or close door" is functionally equivalent to our predicted "handle turn or pull to open or close door," yet not ranked in the top-5 retrieval during evaluation. When relaxing the evaluation to consider top-10 retrieval, our method achieves an overall triplet recall of 75.9%, outperforming OpenFunGraph's 70.3%. This performance reversal highlights the sensitivity of triplet recall to top-k selection and suggests that our predictions capture correct semantics despite lexical differences.

However, enforcing knob and handle detection can introduce false positives, which indirectly affects functional relation precision. To assess the impact of this adaptation, we evaluate the modified pipeline on *FunThor* for both mapping and functional relation metrics, as shown in Table 8. The results indicate that the adaptation indeed significantly improves the recall of interactive elements and functional relations, but also leads to a substantial decrease in precision and F1 score for functional relations. Nevertheless, the modified pipeline still achieves better F1 score for functional relation than OpenFunGraph (25.1% vs. 16.0%) on *FunThor*. Given the reduced precision, however, we adopt the original pipeline for general applications, while the modified version can be beneficial for datasets with known annotation patterns.

K. VLM Prompt for Scene Analysis

We provide the complete prompt used for scene analysis in our mapping pipeline. This prompt instructs the vision-language model to identify functional objects and their interactive parts from RGB images of indoor scenes, including bounding box localization and part enumeration.

```
You are an expert at identifying functional objects
in an RGB image of an indoor scene.  Your task
is to identify all functional objects and their
interactive parts or interfaces that have potential
functional relationships with other objects in the
scene.  A functional relationship means that the
state change of one object will affect the state of
other objects, or one object can be used to change
the state of other objects.  For example, turning on
a light switch will turn on lights, turning a stove
knob will turn on a burner, and using a key can open
a specific lock or door.

## Extra Guidelines
1.  If a handle appears rounded, resembles a knob,
and is attached to a compartment to aid in opening
or closing it (such as a cabinet or drawer), label
this part as `knob handle`.
2.  For all other handles that are functional
components of an object (such as a door handle
or drawer handle, except when covered in point
1), simply use `handle` as the part name, without
specifying the handle type.


## Input Format
- A single RGB image of an indoor scene.


## Output Format
Your response should be a structured JSON object
containing a list of identified objects.  For each
object, provide:

- name:  A common name for the object (e.g.,
  'stove', 'light switch').  Avoid using 'and'
  or 'or' in the name.
- description:  A specific description that helps
  identify the object in the scene.  This can
  include:
    - Spatial information (e.g., 'a cabinet above a
      sink').
    - Feature descriptions (e.g., 'light switch
      with dual switches').
    - Or both.
- functional_parts:  A list of functional parts
  of the object (e.g., for a 'stove':  ['burner',
  'knob']).
    - If the object has no functional parts, leave
      this empty.
    - List multiple parts separately.
    - Avoid using 'and' or 'or' in the functional
      parts.
- bounding_box:  The bounding box coordinates in the
  format [x_min, y_min, x_max, y_max].
    - Coordinates should be normalized to the range
      [0, 1] relative to the image dimensions.
    - Origin (0, 0) is at the top-left corner.
    - x increases to the right, y increases
      downwards.
    - The bounding box should be tight around the
      object, excluding the background.
```

Table 7. **Performance after pipeline adaptation on SceneFun3D.** To address SceneFun3D's systematic annotation of knob-shaped and handle-shaped components (*e.g.*, drains, valves, lever) with generic "knob" and "handle" labels, we modify our pipeline to always detect knobs and handles when functional parts are identified. The modified pipeline substantially improves both mapping and functional relation metrics, surpassing OpenFunGraph on all mapping metrics and achieving competitive triplet recall, especially at R@10 (75.9% vs. 70.3%).

Methods	**Mapping**						**Functional Graph**					
	Objects (↑)		Inter. Elements (↑)		Overall Nodes (↑)		Node Assoc. (↑)		Edge Pred. (↑)		Overall Triplets (↑)	
	R@3	R@10	R@3	R@10	R@3	R@10	R@5	R@10	R@5	R@10	R@5	R@10
OpenFunGraph	81.1	87.8	71.0	79.5	73.0	82.8	68.3	73.0	**88.1**	**96.2**	**60.4**	70.3
FunFact (Modified)	**90.5**	**95.2**	**76.4**	**88.7**	**81.1**	**90.9**	**76.4**	**83.1**	75.8	91.4	57.9	**75.9**

Table 8. **Evaluating side effects of the pipeline adaptation.** To determine whether forcing knob and handle detection negatively affect overall mapping and functional prediction performance, we evaluate both the original and modified pipelines on *FunThor*. The modified pipeline significantly improves recall for interactive elements (69.5% → 87.9%) and functional relations (49.3% → 58.1%), but substantially decreases precision (31.9% → 16.0%) and F1 score (38.7% → 25.1%), confirming that the modification negatively affects the overall performance. Nevertheless, the modified pipeline still outperforms OpenFunGraph in F1 score (25.1% vs. 16.0%).

Methods	**Mapping**			**Functional Graph**		
	Recall @ 3 (↑)			Prec. [%]	Recall [%]	F1 [%]
	Objects	Inter. Elem.	Overall Nodes	(↑)	(↑)	(↑)
OpenFunGraph	54.6	41.1	51.2	23.4	12.2	16.0
FunFact (Original)	**68.2**	69.5	68.5	**31.9**	49.3	**38.7**
FunFact (Modified)	**68.2**	**87.9**	**73.1**	16.0	**58.1**	25.1

L. LLM Prompt for Local Functional Proposal

We provide the complete prompt used for generating local functional proposals between objects and their parts. This prompt guides the language model to propose plausible part-object and part-part functional connections, assign prior confidence scores, and determine whether each connection type is inherently one-to-one.

```
You are an expert in understanding functional
connections between objects and their parts in
indoor scenes.  A functional connection means that
the state change of one object will affect the
state of other objects, or one object can be used
to change the state of other objects.  For example,
turning on a light switch will turn on the lights,
turning a stove knob will turn on a burner, and
using a key can open a specific lock or door.  You
are given a JSON string describing a parent object
and its potential interactable parts.  Your task is
to:  (1) propose plausible, semantically distinct
functional connections between the parent object and
its parts, as well as among the parts themselves;
(2) provide a confidence score for each functional
connection; and (3) indicate whether each connection
is ``one-to-one'' or not.

You are required to output a list of functional
proposals in JSON format.  A functional proposal
is made of a functional connection in triplet
format <first_item> <interaction> <second_item>,
a confidence score, and an ``is_one_to_one'' flag.
You should only output semantically different
functional proposals, and do not repeat the same
type of interaction for different subjects/objects
with the same name.  For example, for a cabinet with
multiple handles, only output a single functional
proposal ``handle pull to open cabinet''.
After proposing the function connection, you
are required to give a confidence score of this
connection semantically, like how correct it sounds
based on common knowledge.  If you are confident
about your proposal based on common sense, like
``handle - pull to open - cabinet'', give a high
score above 0.9.  If you are unsure about your
proposal and need more data to validate it, but it
can still exist in reality, assign a low confidence
score of around 0.5.

If you believe the proposed functional connection is
inherently ``one-to-one'' in general, for example,
a light switch usually controls a single light,
not multiple lights, set the flag ``is_one_to_on''.
On the other hand, a cabinet may contain multiple
handles, and all of its handles can pull to open
it.  Therefore, the connection ``handle pull to open
cabinet'' is not ``one-to-one''.

The given JSON comes from a mapping pipeline, which
may contain detection errors and misannotation.  If
the parent object contains an illogical part, for
example, a faucet has a part named ``lock'', treat
the part as a detection error and do not propose any
functional connection for it.

## Pay attention
- Do not use passive voice in the ``interaction''
  field.  Use active voice instead.  Swap the first
  and second items to avoid passive voice.  For
  example, instead of ``light is turned on/off by
  switch'', use ``switch turns on/off light''.
## Input format
A JSON describing an object and its potential
interactable parts.  It has the following fields:
- object_name:  The name of the parent object.
- object_description:  A short description of the
```

```
  parent object.
- contained_parts: List of part names that the
  parent object contains.
## Output format
A JSON of a list of your functional proposals. Each
proposal should contain the following fields:
- first_item_name: Name of the first item of this
  proposal. It should be one of the part names
  from the given JSON input.
- second_item_name: Name of the second item of this
  proposal. It should be one of the part names or
  the parent object name from the given JSON input.
- interaction: A short sentence describing the
  action imposed by the first item on the second
  item (e.g., ''pull to open'', ''turns on/off'',
  ''controls'').
- confidence: Prior confidence score for this
  connection in general. The range is [0, 1].
- is_one_to_one: Whether this connection type is
  inherently one-to-one.
Your task is to propose plausible functional
proposals between the object and its parts and
among the parts themselves, give a confidence score
of your functional proposal, and indicate whether
such a connection is ''one-to-one'' or not.
```

M. LLM Prompt for Remote Functional Proposal

We provide the complete prompt used for generating object-object functional proposals. This prompt instructs the language model to identify inter-object functional connections, assign prior confidence scores, and determine the properties of each connection (one-to-one relationships and proximity requirements).

```
You are an expert at identifying functional
connections between objects in indoor scenes. A
functional connection means that the state change
of one object affects the state of other objects,
or one object can be used to change the state of
other objects. For example, turning on a light
switch turns on the lights; rotating a stove knob
activates on a burner; and using a key can unlock a
specific lock or door. You are given a JSON string
describing a list of objects. Your task is to: (1)
propose plausible, semantically distinct functional
connections between these objects; (2) provide a
confidence score for each functional connection; (3)
indicate whether each connection is ''one-to-one''
or not; and (4) indicate whether each connection is
a local and require close proximity between the two
objects.
You are required to output a list of functional
proposals in JSON format. A functional proposal
is made of a functional connection in a triplet
format <first_item> <interaction> <second_item>, a
confidence score, and an ''is_one_to_one'' flag.
You should only output semantically different
functional proposals, and do not repeat the same
type of interaction for different subjects/objects
with the same name. For example, if there are
multiple switches and lights in the object list,
only output a single functional proposal ''switch
turn on/off light''.
After proposing the function connection, you
are required to give a confidence score of this
connection semantically, like how correct it sounds
based on common knowledge. If you are confident
about your proposal based on common sense, like
''power outlet - provide power to - TV'', give a
high score above 0.9. If you are unsure about your
proposal and need more data to validate it, but it
can still exist in reality, assign a low confidence
score of around 0.5.
If you believe the proposed functional connection is
inherently ''one-to-one'' and exclusive in general,
for example, a light switch usually controls a
single light, not multiple lights, set the flag
''is_one_to_one''. On the other hand, a power
socket may provide power to multiple appliances;
therefore, the connection ''power socket provides
power to appliances'' is ''one-to-many'' and not
''one-to-one''.
If you believe that the proposed functional
connection requires close proximity between the
two objects to function properly, for example, a
''faucet pours water into sink'' requires the faucet
to be close to the sink, set the flag ''is_local''
to true. If the functional connection can work
regardless of the distance between the two objects,
for example, a ''remote control operates TV'' can
work from a distance, set the flag ''is_local'' to
false.
## Pay attention
- Do not use passive voice in the ''interaction''
  field. Use active voice instead. Swap the first
  and second items to avoid passive voice. For
  example, instead of ''light is turned on/off by
  switch'', use ''switch turns on/off light''.
## Input format
A JSON describing a list of objects:
- objects: A list of objects. Each object
  contains the following fields:
    - name: Name of the object.
    - description: A short description of the
      object.
## Output format
A JSON of a list of your functional proposals. Each
proposal should contain the following fields:
- first_item_name: Name of the first item of this
  proposal. It should be one of the object names
  from the given JSON input.
- second_item_name: Name of the second item of this
  proposal. It should be one of the object names
  from the given JSON input.
- interaction: A short sentence describing the
  action imposed by the first item on the second
  item (e.g., ''pull to open'', ''turns on/off'',
  ''controls'', ''provide power to'', etc.).
- confidence: Prior confidence score for this
  connection in general. The range is [0, 1].
- is_one_to_one: Whether this connection type is
  inherently one-to-one.
- is_local: Whether this connection is local (i.e.,
  the two objects need to be close to each other).
Your task is to propose plausible, semantically
distinct functional proposals between the given
objects, give confidence scores of your functional
proposals, indicate whether such a connection is
''one-to-one'' or not, and indicate whether the
connection is local or not.
```