Appendix for "EgoTV 📺: Egocentric Task Verification from Natural Language Task Descriptions"

This appendix is organized as follows:
8. *EgoTV dataset*: Generation details, additional statistics.
9. *CrossTask Verification (CTV) dataset*: Details on generation process and evaluation.
10. *NSG*: Semantic parsing approaches, additional analysis and ablations, performance on CTV.
11. *Baselines*: Details on VLM and Oracle baselines.

8. EgoTV Dataset

8.1. Task-video Generation using PDDL Planner

To generate EgoTV tasks, we encode the final state of the objects achieved by an EgoTV task as the "goal state" for the Planning Domain Definition Language (PDDL) planner. Note that the ordering constraints of the tasks aren't captured when the tasks are encoded as goal states for the PDDL planner in this manner. For instance, the tasks of *clean_then_heat(apple)* and *clean_and_heat(apple)* would have the same PDDL goal states. Consequently, we enforce ordering constraints for a given task using "pre-conditions" in PDDL. In the above example task of *clean_then_heat(apple)*, the *clean* sub-task would thus be the pre-condition for the *heat* sub-task. Apart from the tasks and object states, the agent state and environment dynamics – e.g., the *heat* sub-task changes the state of the target object to be hot – are also encoded in PDDL.

For each task, a random kitchen scene is picked and the agent is spawned in the corresponding AI2-THOR scene. The planner leverages the initial state and action definitions to generate a sequence of sub-tasks required to achieve the goal. Rather than simply selecting the best plan, which may not always reflect human-like behavior, we aim to mimic the less-than-optimal decision-making of humans by randomly selecting a plan from the top-k plans. This approach enables the inclusion of sub-tasks that may not be strictly necessary for achieving the goal. Furthermore, the partial-ordered nature of tasks enables different plan generations to achieve the same task, thus promoting diversity. We use the Fast Forward (FF) planner [23] to generate plans.

Apart from sequencing the sub-tasks appropriately for achieving a given task, the planner also ensures that the agent can navigate to the correct locations for each sub-task. For instance, to execute the *clean* sub-task, an agent typically requires using the sink in the kitchen and hence must navigate there. The generated plans thus consist of navigation and object interaction actions.

8.2. EgoTV Task-description Generation

The task descriptions corresponding to negative samples, where the task videos are not entailed by their descriptions, are created by either altering the sequence of sub-tasks in the positive template or by replacing some of them with alternative sub-tasks picked randomly from the remaining repertoire of sub-tasks (see Figure 1 where *heat* is replaced with *cool*). To ensure the practicality of an assistive agent that aids a human, we maintain the target object in the negative samples but vary the sub-tasks, as negative task descriptions on the sub-task level are more relevant than on the object level. For abstraction, we (i) omit the low-level details like *clean in the sinkbasin, cool in the fridge*; (ii) (some) task-oriented descriptions are changed to goal-oriented descriptions (*apple is heated and cleaned* $\mapsto$ *hot, clean apple*).

An example template of task descriptions corresponding to the task *cool_then_clean* is ['{obj} is cooled in a Fridge, then cleaned in a SinkBasin', '{obj} is cleaned in a SinkBasin after cooling in a Fridge', '{obj} is cooled in a Fridge before cleaning in a SinkBasin']. While EgoTV already incorporates some diversity in task descriptions in this manner, we note that inclusion of more diverse free-form language descriptions in the dataset (possibly collected through crowdsourcing) would be a valuable future enhancement.

8.3. Dataset Analysis and Statistics

See Figure 6 for a comparison of video lengths and task description lengths across different splits. It can be observed that the Novel Tasks split has the longest videos (≈ 1.6 minutes) and task descriptions (≈ 12 words) owing to its compositional tasks. Additionally, the Abstraction split has the shortest task description (≈ 5 words), even for longer videos due to abstraction. We include all tasks of EgoTV in Table 8 at the end of the supplement. We also perform a detailed analysis of each split (Figure 11).

9. CrossTask Verification (CTV) Dataset

9.1. Dataset construction

We leverage the CrossTask dataset and its action step annotations to construct our CrossTask Verification dataset. Each video contains task descriptions that are obtained by concatenating the sequence of *action steps* annotations available in the original CrossTask dataset. For ease of

	Reasoning		Dataset Characteristics				Grounding	
	compos-itional	causal	language	egocentric	real-world	diagnostic tools	objects, relations	actions
CLEVRER [74]	✓	✓	✓	✗	✗	✓	✓	✗
Next-QA [69]	✗	✓	✓	✗	✓	✓	✓	✓
AGQA [18]	✓	✗	✓	✗	✓	✓	✓	✓
Activity Net-QA [21]	✗	✗	✓	✗	✓	✗	✓	✓
STAR [68]	✓	✓	✓	✗	✓	✓	✓	✓
CoPhy [5]	✗	✓	✗	✗	✗	✓	✓	✗
Social-IQ [78]	✗	✓	✓	✗	✓	✓	✗	✗
Causal-VidQA [33]	✗	✓	✓	✗	✓	✓	✓	✓
Charades [58]	✗	✗	✓	✗	✓	✓	✓	✓
CATER [16]	✓	✗	✗	✗	✗	✓	✓	✓
EPIC-KITCHENS [10]	✓	✓	✓	✓	✓	✓	✓	✓
Ego-4D [17]	✗	✓	✓	✓	✓	✓	✓	✓
VIOLIN [38]	✗	✓	✓	✗	✓	✗	✓	✓
Change-It [60]	✗	✓	✗	✓	✓	✗	✓	✓
Cross-Task [82]	✓	✓	✓	✗	✓	✗	✓	✓
EgoTV	✓	✓	✓	✓	✗	✓	✓	✓

Table 5. **EgoTV vs. existing video-language datasets.** EgoTV benchmark enables reasoning (compositional, causal, and temporal); has unique dataset characteristics along with diagnostics; requires grounding of objects, relations, and actions.

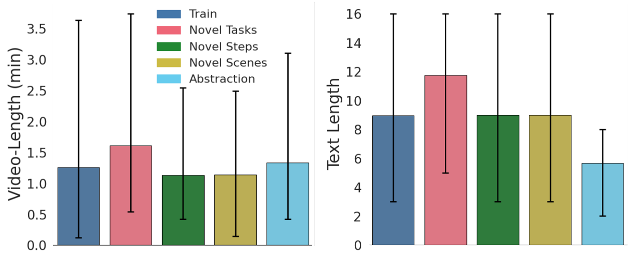

Figure 6. **Additional EgoTV dataset statistics.** Comparison of video length (in minutes) and task description length (number of words) across different splits using error plots. Novel Tasks split has the longest videos and task descriptions. Abstraction split has the shortest task descriptions.

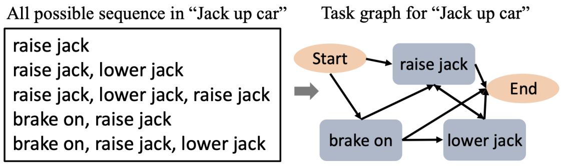
All possible sequence in "Jack up car"
raise jack
raise jack, lower jack
raise jack, lower jack, raise jack
brake on, raise jack
brake on, raise jack, lower jack
Task graph for "Jack up car"
Start
raise jack
End
brake on
lower jack

Figure 7. **Task graphs are used to create impossible sequences for generating negative task descriptions in the CTV dataset.**

experimentation, we only consider the top-4 frequent action steps for each task in CrossTask. Consequently, CTV dataset videos are constructed by selecting the segments corresponding to these frequent actions per video and are thus shorter than the original CrossTask videos.

We follow a process similar to EgoTV for generating **negative descriptions**: (i) *replacing action steps from other tasks*: where we substitute an action step in the sequence with an action step in another task. (ii) *replacing action steps from the same task*: instead of replacing steps from another task, we reuse the unused action steps, which were not the top-4 frequent steps in the same task. These action steps are closer in semantics and serve as hard negative. (iii) *replacing action step sequence order to an impossible order*: we first list out all possible action step orders in CrossTask following [42] as shown in Figure 7. Then, we construct an action step sequence that doesn't exist in CrossTask for the given task, e.g., *lower jack, upper jack, break on* for the task of *fixing the car*. We assume that these sequence orders are impossible since they were not observed in any videos for a given task.

Apart from generating negative NL task descriptions, we also generate **negative videos** in two ways: (i) *shuffling the video order corresponding to action step sequence*: this mimics the case where the action steps were not executed in the correct order. (ii) *dropping a video segment corresponding to an action step*: this corresponds to the situations where an action step is missed while executing the task, potentially resulting in a failure in accomplishing the task. Thus, this is an important use case for task verification.

10. NSG details

10.1. NSG Training details

When training NSG, we do not update the weights of the CLIP feature extractors (Sec. 5.3) due to GPU memory limitations. We use a batch of N = 64 samples, where we sample the video at 2.5 FPS. We set a window size $k = 20$ frames for segmentation in NSG (Sec. 5.4), each window representing an 8-second video segment. We use a train-validation split of 80-20 and use the validation performance as an indicator of convergence. We minimize the binary cross-entropy loss in Eq. 2 with Adam [28] and a learning rate of 1e-3. Each model is trained on 8 V100 GPUs for 50 epochs for two days.

10.2. NSG Semantic Parsing

We test two semantic parsing methods: (i) Finetuning language models to generate graphs from NL descriptions; (ii) Few-shot prompting of large language models.

10.2.1 Finetuning Language Models

Recently, it has been shown that pre-trained language models can be leveraged for graph [53] and plan generation [72] from NL. We use a similar framework to train a T5-small transformer [51] to generate partial-ordered plans. For this, we use a subset of the training data (in particular the positive task descriptions for which we have gold-label graphs annotations) and annotate them with their partial-ordered plans in the form of directed acyclic graphs (DAG) – $G(V, E)$, where vertices $n_i \in V$ represent sub-tasks, and edges $e_{ij} \in E$ are ordering constraints that indicate n_i must precede n_j (i.e. $n_i \rightarrow n_j$). To train the text generation transformer, we represent the output graph in DOT language. The corresponding DOT representation for the graph in Figure 4 is given as: `Step 0: StateQuery(apple,hot),` `Step 1: StateQuery(apple,clean),` `Step 2: StateQuery(apple,sliced),` `Step 3: RelationQuery(apple,plate,in),` `Step 0 → Step 1, Step 0 → Step 2,` `Step 1 → Step 3, Step 2 → Step 3`

Ablations on Plan Generation framework: We assess the correctness of the graphs generated by the trained T5-transformer model on the positive task descriptions of our test splits. The evaluation metric is Graph Edit Distance (GED) [1] which computes the distance between two graphs (G_1 and G_2) given as:

$$GED(G_1, G_2) = \min_{G_1 \xrightarrow{d_1,\ldots,d_k} G_2} \sum_{i=1}^{k} cost(d_i)$$

where, $d_1, \ldots d_k$ are graph edit operations (insertion, deletion, replacement of a vertex or an edge) from G_1 to G_2. With the exception of the abstraction split[6], the GED for all test splits was observed to be ≈ 0.03 (GED $\downarrow$). Here, $\downarrow$ signifies that a lower GED score is better, with the lowest value being $= 0$. Note, that although it is possible for the response to contain syntax errors, e.g., invalid names or queries with invalid arguments, the output can be improved by leveraging the DSL grammar to avoid invalid arguments.

10.2.2 Prompting Language Models

We also experimented with prompting. As a proof-of-concept, we show our example prompts in Table 10.4. We use ChatGPT [46] with few-shot prompting. The prompt is displayed in gray, the queries in blue, and the generated output in green. We observed that the use of Chain-of-Thought [66] improved the output.

10.3. Integer Programming for Alignment in NSG

The constrained optimization problem defined in Eqs. 3, without the ordering constraint, can also be formulated as an Integer Programming problem [67] where variables Z_{jt} can only take integer values in $\{0, 1\}$ (i.e. with the additional constraint Z $\in \{0, 1\}^{N \times S}$). Notably, the proposed DP solution adheres to the constraints Eqs. 3a 3b 3c. The first part of the DP (highlighted in green in Figure 4) pairs the current query q_j with the current segment s_t, then tries to align the rest of the queries with the remaining segments to meet the requirement of one query per segment, as specified in Eq. 3a. The second part of the solution (denoted by the red box) skips over the current segment s_t and tries to align the same queries with the remaining segments until a pairing is found (i.e. $Z_{jt} = 1$). Thus, together with the base case of Z $= \mathbb{I}$ if $N = S$, it satisfies Eq. 3b. Furthermore, since the DP processes the queries and segments in a specific order (topological sorting for queries and temporal order of video segments), it also meets the ordering constraint requirement specified in Eq. 3c.

10.4. NSG Analysis

- Table 6 demonstrates the robustness of NSG to changes in the window size k.
- We trained two different NSG models to compare the impact of query types with (i) `(State+Relation)Query`, (ii) `ActionQuery`. Figure 9 reveals that a combination of `(State+Relation)Query` is effective at detecting sub-tasks in segments with high recall, particularly for *slice* and *put*. We also note that the NSG model struggles to detect the *slice* sub-task in the Novel Steps split.

---

[6] For abstraction, we observed syntax errors, e.g., missing/incorrect arguments for queries in the generated graphs like `RelationQuery(apple, slice)`, instead of `RelationQuery(apple, knife, slice)`. However, the selection of the correct query module and the partial grounding of the correct arguments lead to a significant improvement over baselines.

Table 10.4: Semantic Parsing by Prompting ChatGPT

```
Q. slice of apple is heated in microwave, then placed in a plate
A. First the apple is sliced, hence Step 1:  slice apple.  Then the apple is heated,
hence Step 2:  heat apple.  Then the apple is placed in a plate, hence Step 3:  place
apple in plate.  Hence, the order of the steps are:  slice apple < heat apple < place
apple in plate

Q. potato is cleaned in sinkbasin and sliced, then cooled in a fridge
A. First the potato is cleaned and sliced in any order, hence Step 1:  clean potato;
Step 2:  slice potato.  Then the potato is cooled in a Fridge, hence Step 3:  cool
potato.  Hence, the order of the steps are:  clean potato and slice potato < cool
potato

Q. hot, sliced, clean tomato
A. First the tomato is heated and sliced and cleaned in any order, hence Step 1:  heat
tomato; Step 2:  slice tomato; Step 3:  clean tomato.  Hence, the order of the steps
are:  heat tomato and slice tomato and clean tomato

Q. apple is heated and cleaned in sinkbasin, then cooled and sliced
A. First the apple is heated and cleaned in a sink basin in any order, hence Step 1:
heat apple; Step 2:  clean apple.  Then the apple is cooled and sliced in any order,
hence Step 3:  cool apple; Step 4:  slice apple.  Hence, the order of the steps are:
heat apple and clean apple < cool apple and slice apple.

Q. sliced apple is heated in microwave, then cleaned in a sinkbasin
A. First the apple is sliced, hence Step 1:  slice apple.  Then the apple is heated
in a microwave, hence Step 2:  heat apple.  Then the apple is cleaned in a sink basin,
hence Step 3:  clean apple.  Hence, the order of the steps are:  slice apple < heat
apple < clean apple

Q. apple is heated in microwave after cooling and cleaning
First the apple is cooled and cleaned in any order, hence Step 1:  cool apple; Step
2:  clean apple.  Then the apple is heated in a microwave, hence Step 3:  heat apple.
Hence, the order of the steps are:  cool apple and clean apple < heat apple.
```

- We evaluated the NSG model against the best-performing baseline (VIOLIN-ResNet) across two axes: task complexity and ordering. Figure 8 shows that NSG's performance is robust to complexity and ordering variations.
- Since NSG only utilizes the features from aligned video segments and ignores the remaining segments, we conducted experiments to investigate the role of context (from the discarded segments) in EgoTV. Specifically, we trained an extra BiLSTM layer to encode bidirectional context from adjacent segments on top of the CLIP segment features. From Table 6, we observed improved performance across all splits, except for Novel Tasks. We attribute this to a potential loss of compositional and temporal comprehension of the segment features caused by the inclusion of additional context information.

NSG	Novel Tasks	Novel Steps	Novel Scenes	Abstraction
$k = 12$	90.6	50.3	81.3	82.5
$\boldsymbol{k = 20}$ (default)	90.0	64.7	84.9	80.4
$k = 32$	89.2	54.9	81.0	82.3
NSG-BiLSTM	73.3	70.2	90.0	87.2

Table 6. **NSG ablations.** [Block 1]: NSG with different window sizes $k = \{12, 20, 32\}$. The best-performing NSG model with $k = 20$ is reported in Table 3. [Block 2]: NSG with BiLSTM to encode additional context from neighboring segments.

10.5. NSG on Real-world Data

NSG for CTV. In Section 5.1, we described two symbolic operations to process task description. In CTV, all of the action steps refer to a certain action. Hence, we apply the `ActionQuery` as in Table 2 to encode all action steps. In Section 5.2, we described (1) how we processed task description into a graph and (2) how we generated all possible sequences from the graph. Since the action sequences are already given in CTV, we can skip the above two steps in Section 5.2 and directly feed the sequence in our Query Encoder and Video Aligner models.

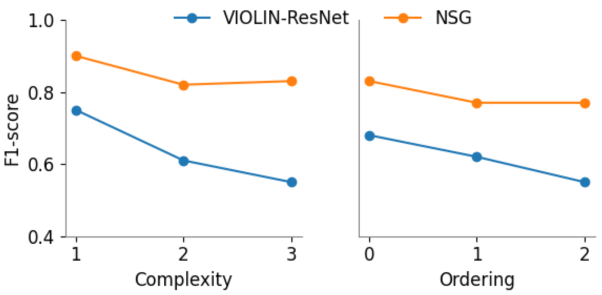

Figure 8. **NSG maintains consistent performance as task complexity and ordering difficulty increases.** F1-score of NSG vs. best-performing baseline for EgoTV tasks with varying complexity and ordering are shown.

Model	Visual	Text	Fusion	**F1**
VIOLIN-I3D [38]	I3D	BERT	Y	34.7
MIL-NCE [44]	S3D	Word2Vec		43.4
VideoCLIP [36]	Tx	Tx	Y	49.7
CoCa [76]	Tx	Tx	Y	70.9
NSG (ours)	CLIP	CLIP	Y	**76.3**

Table 7. **Performance on CTV's test split, which mirrors the Novel Tasks split from EgoTV**

Performance Evaluation. To evaluate how NSG enables task verification in the real world, we compare the performance of NSG against selected baselines described in Sec. 6, which were either applied to the previous CrossTask evaluation (MIL-NCE[44], VideoCLIP[36]) or had competitive performance (CoCa[76], VIOLIN[38]) on EgoTV. Note that the methods for CrossTask are not directly applicable to CTV since CTV focuses on task verification (predicting if the task is accomplished) instead of temporal localization (localizing action temporally). Table 7 shows that the baseline models VideoCLIP, CoCa perform better than VIOLIN on CTV as compared to EgoTV. This indicates that CTV is potentially able to better harness the gains from the large-scale VL pretraining in these models compared to EgoTV. Despite these gains, NSG outperforms the baselines by a significant margin.

11. Baseline descriptions and details

11.1. VLM Baselines

Majority of VLMs can be characterized based on their approach of extracting and fusing features from vision and text modalities. VLMs for video tasks use either a video encoder or an image encoder with temporal aggregation using sequence model, e.g., transformers [51] or LSTM [22] to obtain the video features. The vision and text encoders can be jointly trained using either a) contrastive loss that aligns both modalities in a shared latent space e.g., CLIP [50], MIL-NCE [44], b) masked token prediction losses on the generated text [32] aka captioning loss, or c) combination of captioning and contrastive losses [76]. The vision-text features from encoders can either be fused (multimodal fusion) using attention-based mechanisms or by computing cross-modal similarity scores. We investigate 6 VLMs that span the space of these characteristics for EgoTV.

CLIP4Clip [39] uses CLIP-based [50] text and image encoders. Parameterized (e.g., LSTM-based) or non-parameterized (e.g., mean-pooling) aggregation of the resultant image features allows video representation using a single feature vector, without any explicit fusion. **CLIP Hitchhiker [3]** uses a similar encoder structure as CLIP4Clip [39] and performs weighted-mean pooling of frame embeddings using text-visual similarity scores. **CoCa [76]** uses image-text (dual) encoder-decoder architecture trained using contrastive and captioning loss. The frame-level features are pooled via attentional pooling [30] to model the temporal sequence of the video and then fused with the text features for downstream tasks. **MIL-NCE [44]** learns to encode video and text into a single vector using separate encoders (S3D[71], word2vec[45]) learned from scratch using the proposed multi-instance contrastive loss. **VideoCLIP [36]** is built on top of MIL-NCE [44]. It adds additional transformer layers for video and text encoders, trained using contrastive loss. The resultant embeddings can be fused together for downstream tasks. **VIOLIN [38]** uses pre-trained image/video (e.g., ResNet [19], I3D [7]) and text encoders (e.g., GloVe [48], BERT [12]) separately and fuses the resultant representations from each modality using bi-directional attention [55]. For each of the above VLMs, we freeze the pretrained feature extractors and finetune a fully-connected probe layer, along with the temporal aggregation layers where appropriate (CLIP4Clip-LSTM, VIOLIN), using EgoTV's train split.

11.2. Text2Text Baseline Model

In this baseline, we generate video captions from ground-truth objects and sub-task labels and calculate a text similarity score (cosine similarity) between the video caption and the task description using a pretrained RoBERTa model [81] by using a manually set threshold. Videos are split into segments, with each caption containing the segment index (for temporal grounding), scene location (kitchen), (*top-k*) objects in the scene, and the activity (subtask). Refer to Table 10 for examples. This baseline mirrors Socratic Models [79] which generalizes in zero-shot by leveraging the multimodal capabilities from several pretrained models. For instance, the objects from each segment can be captured using open-vocabulary VLMs [35, 80, 76], while the activities for each segment can be detected using [44, 36] by zero-shot classification with text-to-video feature similarity. Similarly, a final summarized video caption for the whole video can be generated using an LLM. No-

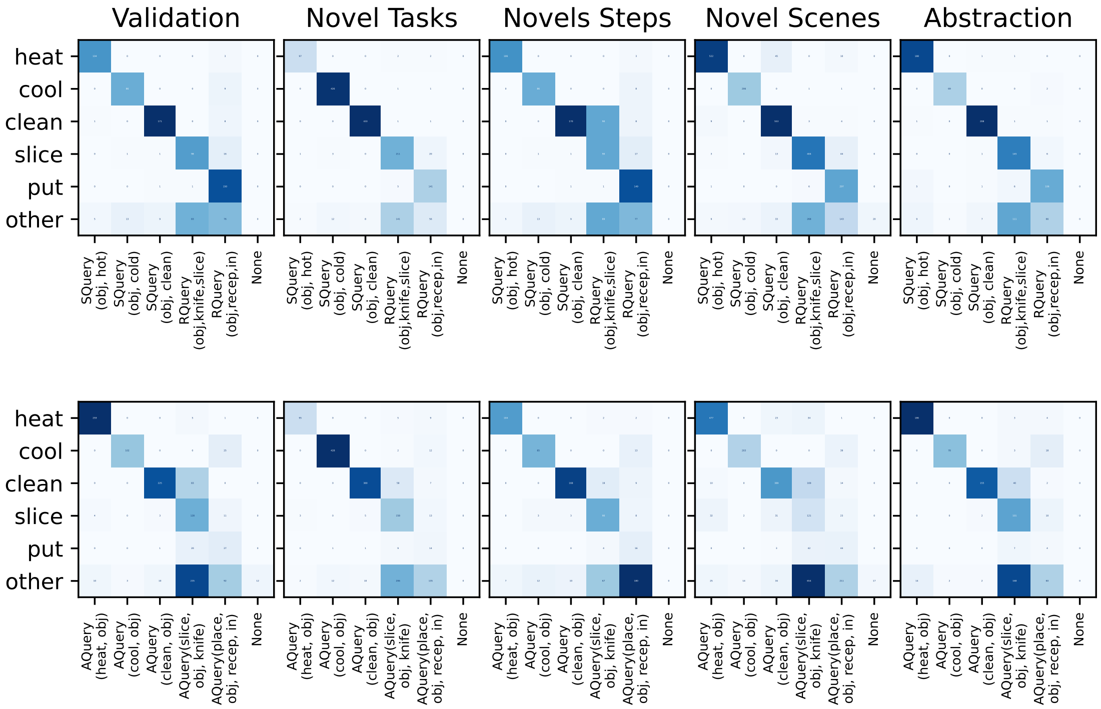

Figure 9. **Effect of query types on NSG performance.** Confusion matrices for two different query models across train/test splits of EgoTV. The Y-axis represents the ground-truth sub-task for a segment, and the X-axis denotes the aligned query for that segment. [Row 1]: Here, SQuery and RQuery denote `StateQuery` & `RelationQuery`, respectively. [Row 2]: Here, AQuery denotes `ActionQuery`. It can be observed that `{State+Relation}Query` performs better than `ActionQuery`.

table, despite having ground-truth textual representations of objects and sub-tasks on a scene-by-scene basis, the Text2text baseline model fails to generalize. We attribute this to two reasons: (1) The pretrained RoBERTa model has limited capacity to capture (out-of-domain) word-level sub-task orderings to determine entailment in EgoTV, and (2) Text2Text lacks visual inputs and might suffer from lack of inferring relationships between objects.

Text2Text Baseline Examples

Text Description: `apple is cooled in a Fridge and cleaned in a SinkBasin`
Video Caption:
`Segment: 1. Location: kitchen. Objects: countertop, apple. Activity: go to countertop`
`Segment: 2. Location: kitchen. Objects: fridge, apple. Activity: go to fridge`
`Segment: 3. Location: kitchen. Objects: apple, fridge. Activity: cool apple`
`Segment: 4. Location: kitchen. Objects: apple, fridge. Activity: cool apple`
`Segment: 5. Location: kitchen. Objects: sink, apple. Activity: go to sink`
`Segment: 6. Location: kitchen. Objects: apple, sink. Activity: clean apple`
Task Verified: `True`

Text Description: `lettuce is picked, cooled in a Fridge, and sliced in a SinkBasin`
Video Caption:
`Segment: 1. Location: kitchen. Objects: sink, lettuce. Activity: go to sink`
`Segment: 2. Location: kitchen. Objects: sink, lettuce. Activity: go to sink`
`Segment: 3. Location: kitchen. Objects: lettuce, sink. Activity: clean lettuce`
`Segment: 4. Location: kitchen. Objects: fridge, lettuce. Activity: go to fridge`
`Segment: 5. Location: kitchen. Objects: lettuce, fridge. Activity: cool lettuce`
Task Verified: `False`

Figure 10. Text2Text Baseline Examples.

Complex\Order	0	1	2
1	clean_simple, cool_simple, heat_simple pick_simple, place_simple, slice_simple		
2	clean_and_cool, clean_and_heat clean_and_place, clean_and_slice cool_and_place, heat_and_place slice_and_cool, slice_and_heat slice_and_place	clean_then_cool, clean_then_heat clean_then_place, clean_then_slice cool_then_clean, cool_then_place cool_then_slice, heat_then_clean heat_then_place, heat_then_slice slice_then_clean, slice_then_cool slice_then_heat, slice_then_place	
3	slice_and_clean_and_place, cool_and_clean_and_place cool_and_slice_and_place, heat_and_clean_and_place slice_and_heat_and_place, slice_and_heat_and_clean cool_and_slice_and_clean		clean_then_cool_then_place, clean_then_cool_then_slice clean_then_heat_then_place, clean_then_heat_then_slice clean_then_slice_then_cool, clean_then_slice_then_heat cool_then_clean_then_place, cool_then_clean_then_slice cool_then_slice_then_clean, heat_then_clean_then_place heat_then_clean_then_slice, heat_then_slice_then_clean slice_then_clean_then_cool, slice_then_clean_then_heat slice_then_clean_then_place, slice_then_cool_then_clean slice_then_cool_then_place, slice_then_heat_then_clean slice_then_heat_then_place, clean_and_cool_then_place clean_and_cool_then_slice, clean_and_heat_then_place clean_and_heat_then_slice, clean_and_slice_then_cool clean_and_slice_then_heat, clean_then_cool_and_slice clean_then_heat_and_slice, cool_and_slice_then_clean cool_then_clean_and_slice, heat_and_slice_then_clean heat_then_clean_and_slice, slice_and_clean_then_place slice_and_cool_then_place, slice_and_heat_then_place slice_then_clean_and_cool, slice_then_clean_and_heat clean_then_cool_and_place, clean_then_heat_and_place clean_then_slice_and_place, slice_then_cool_and_place slice_then_heat_and_place, slice_then_clean_and_place heat_then_clean_and_place, heat_then_slice_and_place cool_then_clean_and_place, cool_then_slice_and_place

Table 8. **List of tasks in EgoTV dataset** arranged according to complexity (rows) and ordering (columns) in the tasks. A total of 82 tasks were considered for the dataset generation, split into train and novel composition sets. Blue denotes novel composition tasks.

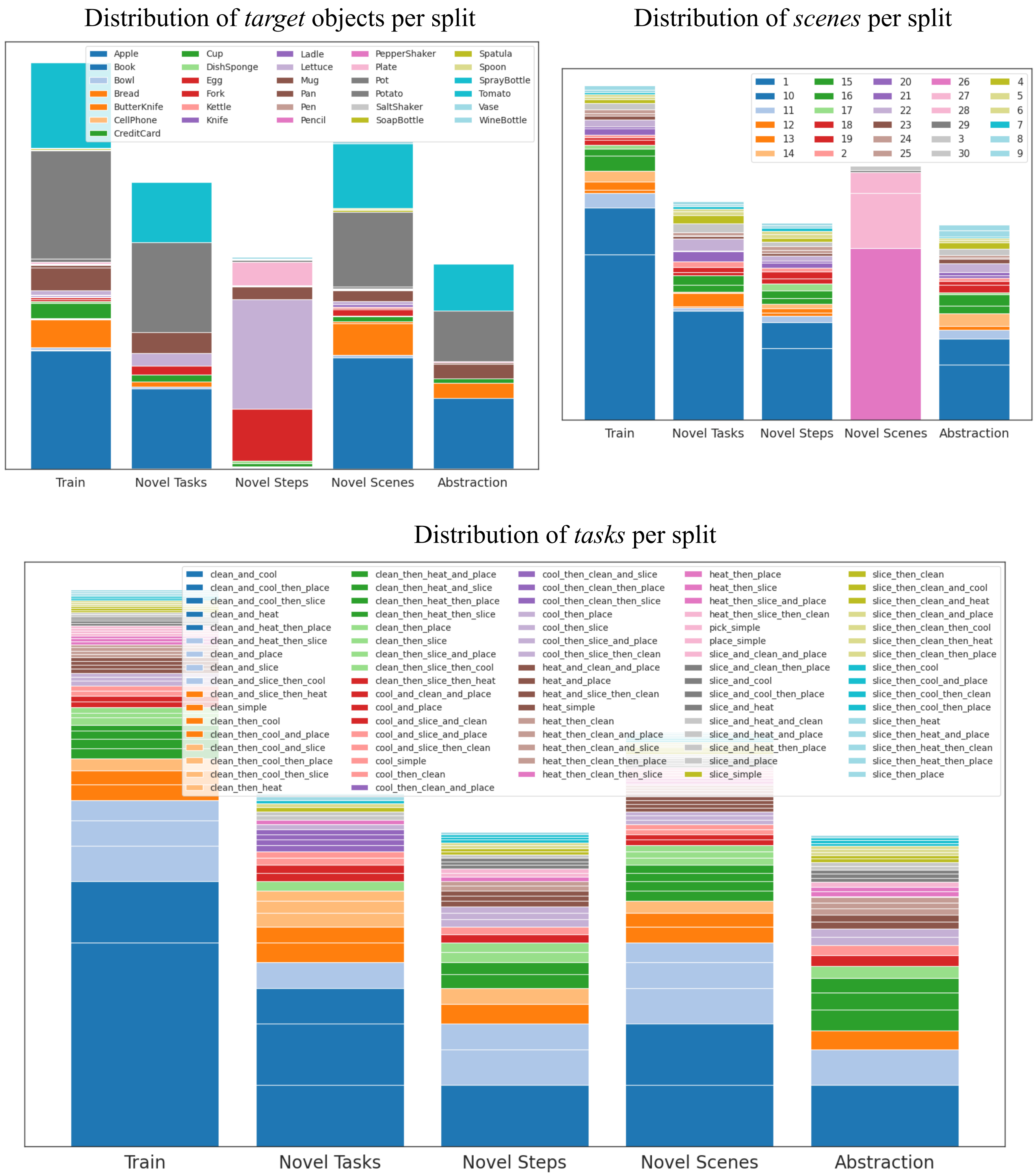

Figure 11. EgoTV dataset analysis. Distribution of *target* objects [row 1, left], *kitchen-scenes* [row 1, right], and *sub-tasks* [row 2] in each split. The Y-axis is in the log scale.