Unsupervised Task Graph Generation from Instructional Video Transcripts

Lajanugen Logeswaran[1], **Sungryull Sohn**[1], **Yunseok Jang**[1,2*], **Moontae Lee**[1], **Honglak Lee**[1]
[1]LG AI Research [2]University of Michigan, Ann Arbor

Abstract

This work explores the problem of generating *task graphs* of real-world activities. Different from prior formulations, we consider a setting where text transcripts of instructional videos performing a real-world activity (e.g., making coffee) are provided and the goal is to identify the key steps relevant to the task as well as the dependency relationship between these key steps. We propose a novel task graph generation approach that combines the reasoning capabilities of instruction-tuned language models along with clustering and ranking components to generate accurate task graphs in a completely unsupervised manner. We show that the proposed approach generates more accurate task graphs compared to a supervised learning approach on tasks from the ProceL and CrossTask datasets.

1 Introduction

Tasks in the real-world are composed of multiple key steps with specific dependencies that dictate the order in which they can be performed (*e.g.*, one has to *check for breathing* before *performing CPR*). Exposing these dependencies between key steps has many downstream applications including assisting human users in troubleshooting and building artificial agents that efficiently learn and perform new tasks. However, information about tasks is typically available in unstructured and noisy form in the wild (*e.g.*, 'how to' descriptions or instructional video transcripts), presenting a major challenge in extracting structured representations.

There is a long history of work on reasoning about tasks, events and temporal ordering, broadly referred to as 'script understanding' (Chambers and Jurafsky, 2008; Regneri et al., 2010; Modi and Titov, 2014; Schank and Abelson, 2013; Pichotta and Mooney, 2016). Recent formulations of script understanding problems include generating a sequence of steps from a given task description (e.g., bake a cake) (Lyu et al., 2021; Sancheti and Rudinger, 2021; Sun et al., 2022) and generating flow graphs from goal and event descriptions (Pal et al., 2021; Sakaguchi et al., 2021). Script generation also manifests in interactive settings such as simulated embodied environments where agents are expected to reason about subgoals in order to complete tasks (Logeswaran et al., 2022; Huang et al., 2022). Many of these prior approaches either fine-tune language models on human-annotated scripts or rely on knowledge encoded in language models to generate scripts. In contrast, we attempt to use pre-trained language models as an information extraction system to perform zero-shot script inference from noisy ASR (Automatic Speech Recognition) transcriptions of instructional videos describing a task.

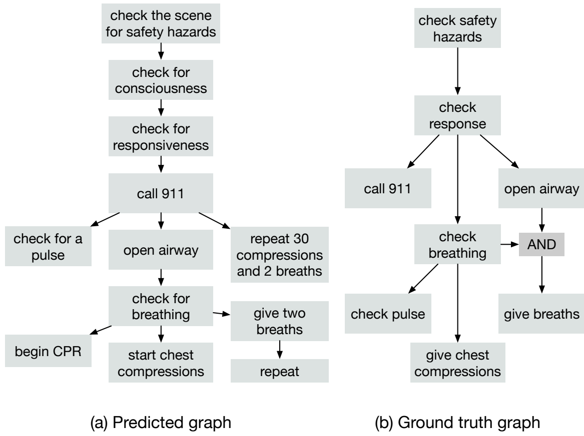


Figure 1: Predicted (a) and ground truth (b) graphs for the *perform cpr* task. Edges indicate precondition relationships. Steps with multiple preconditions are represented using an AND node.

Our focus in this work is to generate a directed graph that represents dependency relationships between the key steps relevant to a real-world task. Figure 1 (a) shows a graph predicted by our approach for *performing CPR*. An example depen-

*Work done during an internship at LG AI Research

Findings of the Association for Computational Linguistics: ACL 2023, pages 3392–3406
July 9-14, 2023

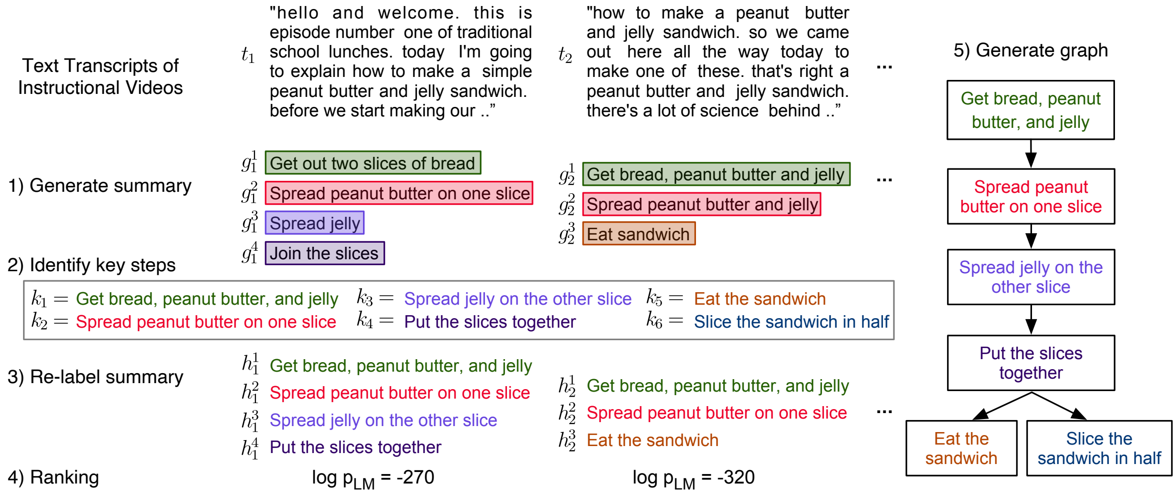


Figure 2: **Overview of our Task Graph Generation Pipeline.** Given multiple text transcripts of a task, we 1) Summarize the steps described in the transcript, 2) Identify the key steps, 3) Re-label summary steps with key steps, 4) Rank key step sequences using a language model and 5) Consolidate top-k sequences to generate a task graph for the given task.

dency that can be read from the graph is that checking for safety hazards has to have happened before any other step (i.e., it is a *precondition* that needs to be satisfied). In this paper, we will use the term *task graph* to refer to such dependency graphs.

More formally, consider a real-world *task* τ. We assume that multiple text *transcripts* $t_1, \ldots, t_n$ describing how this task is performed are available.[1] We assume that having access to such multiple transcripts helps robustly identify the dependencies between key steps so that an accurate task graph can be generated. For instance, if step y frequently follows step x, it is highly likely that step x needs to happen before step y (i.e., is a *precondition*). Our goal is to generate a *task graph* for the given task τ which models these dependencies. In particular, this involves (i) Identifying the *key steps* $K = \{k_1, \ldots, k_m\}$ relevant to performing the task and (ii) Generating a graph with nodes k_i and edges representing precondition relationships.

Our contributions in this work are as follows.

- We propose an unsupervised task graph generation approach that uses pretrained language models to infer key steps and their dependencies from multiple text descriptions of a real-world activity.
- We propose ranking and filtering mechanisms to improve the quality of generated task graphs.
- We demonstrate the effectiveness of the proposed approach compared to strong supervised and unsupervised baselines on two datasets.

[1]Each transcript is a text document derived from an instructional video using Automatic Speech Recognition.

2 Approach

Our approach to task graph generation consists of multiple steps, illustrated in Figure 2. First, we use an instruction-tuned language model to generate a summary of steps (in free-form text) from a transcript (Section 2.1). Given these *summary step sequences* generated from multiple such transcripts for the task, we identify the key steps relevant to the task using a clustering approach (Section 2.2). We then re-label *summary step sequences* using the identified key steps to obtain *key step sequences* (Section 2.3) and rank them using a language model (Section 2.4). Finally, we generate a task graph from the key step sequences (Section 2.5).

2.1 Generating Summary Steps

The first step of our pipeline extracts a summary of steps $g_i = (g_i^1, g_i^2, \ldots)$ for performing the task described in each transcript t_i. We use an instruction-tuned language model for this purpose. We prompt the model with a transcript, followed by a query such as *'Based on this description list down the key steps for making coffee using short phrases.'* and let the model generate a completion. We use the 'Davinci' version of the InstructGPT (Ouyang et al., 2022) model in our experiments. We observed that the model consistently generates the steps in the format '1. `<step 1>`\n 2. `<step 2>`\n ..', occasionally using bullet points instead of numbers. The sentences g_i^j on each line are extracted and treated as the summary steps identified from the transcript. Appendix B shows example summary step sequences generated by InstructGPT.

2.2 Identifying Key Steps Relevant to the Task

Given summary step sequences $g_1, \ldots, g_n$ generated in the previous step, we seek to identify correspondences between steps in different summaries and capture the salient steps that appear frequently. We use a clustering approach for this purpose. Sentences g_i^j are represented as embeddings using a sentence encoder (We use the MiniLMv2 encoder from the SentenceTransformers library (Reimers and Gurevych, 2019; Wolf et al., 2019), which was identified as the best sentence embedding method for semantic search/retrieval). We obtain high-confidence clusters by identifying *max cliques* – clusters of sentences that are similar (determined by a threshold - cosine similarity ≥ 0.9) to each other, and retain cliques with more than 5 sentences. We noticed that this often yields multiple clusters that represent the same key step. For instance, the steps '*fill the moka pot with water*' and '*fill the bottom chamber with water*' represent the same key step of filling water, but are placed in different clusters. Identifying such redundant clusters based on sentence similarity alone is difficult. We define the notion of *sequence overlap* between two clusters – how often a sentence from one cluster and a sentence from the other cluster appear in the same summary step sequence. Intuitively, if two clusters have high inter-cluster similarity and low *sequence overlap*, it is likely that they represent the same key step, and we merge the clusters. The resulting clusters obtained are treated as the key steps $k_1, \ldots, k_m$.[2] Appendix C shows example clusters discovered for different tasks.

2.3 Re-labeling Summary Step Sequences

We re-label each summary step sequence g (subscript i dropped for brevity) with the identified key steps $k_1, \ldots, k_m$ to produce a *key step sequence* h using the greedy algorithm described in Algorithm 1. The algorithm sequentially picks the most similar[3] candidate summary step and cluster pair (g^a, k_b) at each step, assuming each key step only appears once in the sequence. The process terminates when the highest cosine similarity drops below zero.

[2]We use *cluster* and *key step* interchangeably. For visualization purposes we represent a cluster using a random sentence in the cluster.

[3]defined as the maximum cosine similarity between g^a and any sentence in cluster k_b i.e., $\max_{s \in k_b} cos(g^a, s)$

Algorithm 1: Key Step Sequence Inference

Input $g = (g^1, g^2, \ldots)$ ▷ Summary step sequence
Input $K = \{k_1, k_2, \ldots\}$ ▷ Key steps
For each summary step identify most similar sentence from each cluster:
$C_{ij} \leftarrow \max_{s \in k_j} cos(g^i, s)$
$H_{ij} \leftarrow \arg\max_{s \in k_j} cos(g^i, s)$
$S \leftarrow \{\}$ ▷ Predicted alignments
while $\max_{i,j} C_{ij} > 0$ **do**
  $a, b \leftarrow \arg\max_{i,j} C_{ij}$
  $S \leftarrow S \cup \{(a, b)\}$
  $C_{aj} \leftarrow 0, C_{ib} \leftarrow 0 \quad \forall i, j$
Sort $(a_i, b_i) \in S$ so that $a_1, a_2, \ldots$ are in increasing order
Output $h = (H_{a_1 b_1}, H_{a_2 b_2}, \ldots)$ ▷ Key step sequence

---

2.4 Ranking

One shortcoming of the labeling algorithm described in the previous section is that it does not take the sequential nature of steps into account. To alleviate this issue, we use a language model to identify and filter the most promising key step sequences. Specifically, we use $\log p_{\text{LM}}(h|\text{prompt})$ as a measure of the quality of the key step sequence h, where we compute the likelihood of h given a prompt similar to the prompt in Section 2.1 under a pre-trained language model. Multiple labelings are ranked based on this measure using a GPT2-XL model (Radford et al., 2019) and the top-k are chosen as confident predictions for graph generation ($k = 75\%$ in our experiments).[4]

2.5 Task Graph Generation

We use an off-the-shelf algorithm (Sohn et al., 2020; Jang et al., 2023) which is based on Inductive Logic Programming (ILP) for constructing a graph from key step sequences $h_1, \ldots, h_n$. Inuitively, the algorithm identifies a set of preconditions (which key step must precede another key step due to a causal relationship) most consistent with the key step sequences. Details of the algorithm can be found in Appendix A.

3 Experiments

Data We use ProceL (Elhamifar and Naing, 2019) and CrossTask (Zhukov et al., 2019) datasets in our experiments. We experiment with five tasks from each dataset (considering API costs). Task in these two datasets have respectively 13 and 7 key steps on average. We use 60 instances for each task, where each instance is an instructional video along with its text transcript. The transcripts have

[4]We use an open source language model considering API costs. Further, GPT2-XL led to decent ranking performance.

Model	ProceL (Accuracy ↑)						CrossTask (Accuracy ↑)					
	(a)	(b)	(c)	(d)	(e)	Avg	(f)	(g)	(h)	(i)	(j)	Avg
① Proscript (Sakaguchi et al., 2021)	65.0	51.8	46.9	52.1	53.6	53.9	52.3	**89.6**	57.0	**62.5**	61.4	64.6
② ASR → Labels → Graph	52.5	57.1	78.1	**59.4**	53.6	60.1	54.5	72.9	**71.1**	56.2	54.5	61.8
③ ASR → VPs → Labels → Graph	52.5	53.6	56.2	**59.4**	53.6	55.1	54.5	72.9	**71.1**	56.2	59.1	62.8
④ ASR → GPT → Labels → Graph	68.8	69.6	87.5	53.1	55.4	66.9	**75.0**	72.9	61.7	**62.5**	**65.9**	67.6
⑤ ASR → GPT → Labels → Rank → Graph	**76.2**	**80.4**	**90.6**	51.0	**62.5**	**72.1**	**75.0**	85.4	58.6	56.2	63.6	**67.8**
⑥ Ground-truth labels → Graph	82.5	83.9	78.1	78.1	96.4	83.8	79.5	89.6	68.0	68.8	72.7	75.7

Table 1: Graph prediction accuracy on ProceL and CrossTask datasets. The tasks are (a) make PBJ sandwich (b) change iphone battery (c) perform CPR (d) set up chromecast (e) tie tie (f) change tire (g) make latte (h) make pancakes (i) add oil to car (j) grill steak. Baselines ②, ③, ④ differ in the inputs used for key step labeling (i.e. Algorithm 1) – they respectively use the ASR sentences, verb phrases extracted from the ASR and summary steps generated by GPT (Section 2.1). ⑤ is our proposed approach which includes top-k filtering (Section 2.3). Best numbers for each task are boldfaced and second best are underlined. ⑥ shows graph generation performance with ground truth key step sequences.

565 tokens on average.

Setup The datasets come with key steps annotations (i.e., K) for each task and *key step sequence* annotations for each transcript. Our approach is unsupervised and does not make use of these annotations. However, for evaluation purposes, we consider two settings. The first setting assumes ground truth K and evaluates the performance of the full pipeline ignoring the clustering component (since key steps are known). In the above setting, we use ground truth human annotated graphs from Jang et al. (2023) for evaluation. In the second setting, we use K inferred from Section 2.2 and perform qualitative comparisons with ground truth graphs. Note that we did not use *key step sequence* annotations from the datasets in either setting.

Baselines. We compare our approach against the following baselines. Proscript (Sakaguchi et al., 2021) is a language model fine-tuned on manually curated script data. Given a task description and a set of key steps, Proscript generates a partial order of the key steps. In addition, we consider several variations of our approach as baselines in Table 1.

3.1 Results

Known Key Steps Table 1 compares our approach with baselines on graph prediction accuracy. First, we observe that our unsupervised approach performs better than the proScript baseline which was explicitly trained on script data. Second, using GPT generated summaries for labeling (④) performs better than directly labeling the ASR sentences (②) or verb phrases (VPs) extracted from the ASR (③). This baseline is inspired by prior work (Alayrac et al., 2016; Shen et al., 2021) which extract verb phrases from transcripts and attempt to identify salient actions using filtering/alignment mechanisms. These approaches are susceptible to noise in the text data and are further limited by the assumption about each step being represented by a short verb phrase (extracted using syntax templates). In contrast, we exploit large language models in order to extract key phrases from the transcript. Third, we observe that ranking and filtering key step sequences using a language model (⑤) further improves performance, with a significant improvement for ProceL. Finally, our approach comes closest to graphs generated from human annotated key step sequences in the datasets (⑥).[5]

Unknown Key Steps Next, we consider the full pipeline where key steps are identified automatically. Since ground truth reference task graphs are unavailable in this case we perform a qualitative comparison of graphs generated using our approach and the ground truth, human annotated graph. Figures 1 and 2 show predicted graphs for the tasks *perform cpr* and *make pbj sandwich*, respectively.

We observe that the predicted graph for *perform cpr* is more detailed and fine-grained than the ground truth graph and captures many of the ground truth precondition relationships. On the other hand, the graph for *make pbj sandwich* is less fine-grained compared to the ground truth (Figure 6 of Appendix D). For instance, the ground truth annotations distinguish between *putting jelly on the bread* and *spreading jelly on the bread*, whereas

[5]Performance for ground truth labels is lower than 100% due to noise in the human annotations, which is particularly prominent in CrossTask.

Model	(a)	(b)	(c)	(d)	(e)	Avg
FLAN-T5	60.0	64.3	87.5	49.0	57.1	63.6
InstructGPT	**76.2**	**80.4**	**90.6**	**51.0**	**62.5**	**71.1**

Table 2: Graph prediction accuracy on the ProceL dataset when different models are used for summary step generation (Section 2.1). The tasks are (a) make PBJ sandwich (b) change iphone battery (c) perform CPR (d) set up chromecast (e) tie tie. The models are FLAN-T5 (Chung et al., 2022) and InstructGPT (Ouyang et al., 2022).

our approach treats them as a single step. In addition, spreading peanut butter and spreading jelly are independent of each other and have no sequential dependency. However, the predicted graph fails to capture this and assumes that the former is a precondition for the latter. Appendix D shows more examples of predicted graphs.

3.2 Ablations

Summary Step Sequence Generation We perform an ablation to study the effect of the model used to generate summary step sequences from transcripts. We replace the InstructGPT model (Ouyang et al., 2022) with a FLAN-T5 model (Chung et al., 2022) and evaluate graph prediction performance. We find that InstructGPT consistently outperforms FLAN-T5 across all the tasks (Table 2). In addition, we found that plain language models (not fine-tuned with instructions) struggled to produce usable summaries. This shows that models trained with instructions and human-preference data are better at producing task graphs from transcripts compared to other forms of supervision such as language modeling and supervised multi-task training with NLP tasks.

Ranking Language Model We perform an ablation to understand the impact of the choice of language model for the ranking process in Section 2.4. We present the average performance on tasks in the ProceL dataset with different language model choices in Table 3. First, we find that performance does not degrade much when switching to a smaller model in the GPT2 family. Second, we notice that scale alone does not guarantee better ranking performance as the larger GPT-J model (Wang and Komatsuzaki, 2021) is inferior to the GPT2 models. These findings suggest that the choice of pre-training data influences the script knowledge present in a model and can be more important than model scale.

Language Model	Parameter Count	Performance
GPT2-Medium	345M	70.96
GPT2-XL	1.5B	**72.14**
GPT-J	6B	68.80

Table 3: Graph prediction accuracy on the ProceL dataset when different language models are used for ranking (Section 2.4). The models are GPT2-Medium (Radford et al., 2019), GPT2-XL (Radford et al., 2019) and GPT-J (Wang and Komatsuzaki, 2021).

4 Conclusion

This work presented an unsupervised approach to generate task graphs from text transcripts of instructional videos. Our framework exploits multiple text transcripts which describe a task in order to robustly identify the key steps relevant to a task and the depencies between these steps. We demonstrated the effectiveness of our approach compared to supervised and unsupervised baselines on instructional video transcripts from the ProceL and CrossTask datasets.

5 Limitations

Our summary generation and clustering steps have some key limitations. First, there is an API cost involved in using instruction-tuned models such as InstructGPT. It is thus preferred to use publicly available large language models, which have the additional advantage of results being reproducible by other researchers. With the recent release of large models, this is a promising avenue for further investigation. Second, it is difficult to control the granularity of steps predicted by the summary generation step as they are obtained through free-form text generation. Third, the clustering approach involves hyperparameters that need to be manually set such as the number of clusters. We believe these limitations are best addressed by tighter coupling between the different components (i.e., summary generation and clustering components which inform each other).

References

Jean-Baptiste Alayrac, Piotr Bojanowski, Nishant Agrawal, Josef Sivic, Ivan Laptev, and Simon Lacoste-Julien. 2016. Unsupervised Learning from Narrated Instruction Videos. In *CVPR*.

Leo Breiman. 1984. *Classification and Regression Trees*. Routledge.

Nathanael Chambers and Dan Jurafsky. 2008. Unsupervised Learning of Narrative Event Chains. In *ACL*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling Instruction-finetuned Language Models. *arXiv:2210.11416*.

Ehsan Elhamifar and Zwe Naing. 2019. Unsupervised Procedure Learning via Joint Dynamic Summarization. In *ICCV*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language Models as Zero-shot Planners: Extracting Actionable Knowledge for Embodied Agents. *arXiv:2201.07207*.

Yunseok Jang, Sungryull Sohn, Lajanugen Logeswaran, Tiange Luo, Moontae Lee, and Honglak Lee. 2023. Multimodal Subtask Graph Generation from Instructional Videos. *arXiv:2302.08672*.

Lajanugen Logeswaran, Yao Fu, Moontae Lee, and Honglak Lee. 2022. Few-shot Subgoal Planning with Language Models. In *NAACL*.

Qing Lyu, Li Zhang, and Chris Callison-Burch. 2021. Goal-Oriented Script Construction. In *INLG*.

Ashutosh Modi and Ivan Titov. 2014. Inducing Neural Models of Script Knowledge. In *CoNLL*.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. *arXiv:2203.02155*.

Kuntal Kumar Pal, Kazuaki Kashihara, Pratyay Banerjee, Swaroop Mishra, Ruoyu Wang, and Chitta Baral. 2021. Constructing Flow Graphs from Procedural Cybersecurity Texts. *arXiv:2105.14357*.

Karl Pichotta and Raymond Mooney. 2016. Learning Statistical Scripts with LSTM Recurrent Neural Networks. In *AAAI*.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI blog*.

Michaela Regneri, Alexander Koller, and Manfred Pinkal. 2010. Learning Script Knowledge with Web Experiments. In *ACL*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP*.

Keisuke Sakaguchi, Chandra Bhagavatula, Ronan Le Bras, Niket Tandon, Peter Clark, and Yejin Choi. 2021. proScript: Partially Ordered Scripts Generation. In *Findings of EMNLP*.

Abhilasha Sancheti and Rachel Rudinger. 2021. What do Large Language Models Learn about Scripts? *arXiv:2112.13834*.

Roger C Schank and Robert P Abelson. 2013. *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. Psychology Press.

Yuhan Shen, Lu Wang, and Ehsan Elhamifar. 2021. Learning to Segment Actions from Visual and Language Instructions via Differentiable Weak Sequence Alignment. In *CVPR*.

Sungryull Sohn, Hyunjae Woo, Jongwook Choi, and Honglak Lee. 2020. Meta Reinforcement Learning with Autonomous Inference of Subtask Dependencies. In *ICLR*.

Chenkai Sun, Tie Xu, ChengXiang Zhai, et al. 2022. Incorporating Task-specific Concept Knowledge into Script Learning. *arXiv:2209.00068*.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's Transformers: State-of-the-art Natural Language Processing. *arXiv:1910.03771*.

Dimitri Zhukov, Jean-Baptiste Alayrac, Ramazan Gokberk Cinbis, David Fouhey, Ivan Laptev, and Josef Sivic. 2019. Cross-task Weakly Supervised Learning from Instructional Videos. In *CVPR*.

A Task Graph Generation

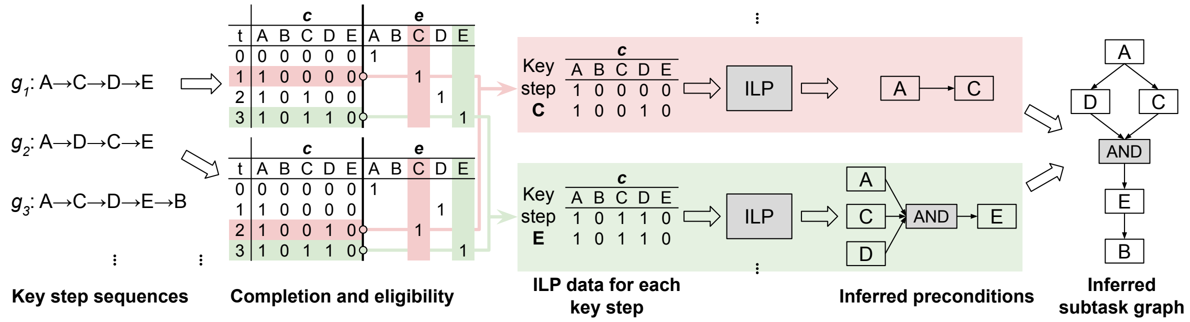


Figure 3: Procedure of predicting a task graph from key step sequences.

We present details about the graph inference algorithm below.

Graph Representation Precondition describes the causal relationship between key steps relevant to a task and imposes a constraint on the order in which the key steps can be performed. Formally, the precondition is defined as a logical expression that combines the key steps using AND and OR logic operations, which means *all* or *any* of certain key steps should be completed, respectively. The precondition can be represented in disjunctive normal form (DNF) where multiple AND terms are combined with OR operations. These preconditions can be compactly represented in the form of a graph. The arguments of AND and OR operations in a precondition become the parents of corresponding AND and OR nodes in the graph, respectively. For example, in Figure 1, the precondition of step `give breaths` is OR(AND(`check breathing`, `open airway`)). Note that we omit AND and OR nodes with only one argument in the graph visualization for simplicity.

Graph Inference Given the set of known key steps $k_1, \ldots, k_m$ (*i.e.*, vertices of the graph), graph inference aims to infer the preconditions (*i.e.*, edges of the graph). We first define the notions of *completion* and *eligibility* of key steps at a given point in time while the task is being performed. We define the *completion* vector $\mathbf{c} \in \{0,1\}^m$ as a binary vector where $c[p] \in \{0,1\}; p \in \{1,\ldots,m\}$ represents whether key step k_p was performed in the past. Similarly we define the *eligibility* vector $\mathbf{e} \in \{0,1\}^m$ as a binary vector where $e[p]$ represents whether key step k_p is eligible to be performed (i.e., it's precondition is satisfied). The completion and eligibility status of key steps will change over time as different key steps are performed to complete the task. The precondition inference problem can be formulated as learning a function $\mathbf{e} = f_G(\mathbf{c})$. In other words, precondition inference amounts to predicting the eligibility status of a key step given the completion status of all key steps.

Given a key step sequence $h = (h^1, h^2, \ldots)$, we convert it into a sequence of completion and eligibility vectors $((\mathbf{c}^1, \mathbf{e}^1), (\mathbf{c}^2, \mathbf{e}^2), \ldots)$ as described next. We define the completion status $c^i[p]$ and eligibility status $e^i[p]$ of key step k_p as follows. If key step k_p was completed in the past (i.e., there exists $j \le i$ s.t. $h^j \in k_p$), $c^i[p]$ is defined as 1 and 0 otherwise. On the other hand, key step k_p is considered eligible if $h^i \in k_p$ and it's eligibility is considered unknown otherwise. Cases where eligibility is unknown are ignored by the algorithm. See Figure 3 for an illustration of this conversion process.

Given $\{(\mathbf{c}^j, \mathbf{e}^j)\}$ as training data, Sohn et al. (2020); Jang et al. (2023) proposed an Inductive Logic Programming (ILP) algorithm which finds the graph G that maximizes the data likelihood (Equation (1))

$$\widehat{G} = \arg\max_G \prod_j p(\mathbf{e}^j|\mathbf{c}^j, G) = \arg\max_G \sum_j \mathbb{I}[\mathbf{e}^j = f_G(\mathbf{c}^j)] \tag{1}$$

where $\mathbb{I}[\cdot]$ is the element-wise indicator function and f_G is the precondition function defined by the graph G, which predicts whether key steps are eligible from the key step completion vector $\mathbf{c}$. The precondition function f_G^p for key step k_p (i.e., $e[p] = f_G^p(\mathbf{c})$) is modeled as a binary decision tree where each branching node chooses the best key step to predict whether the key step k_p is eligible or not based on Gini impurity (Breiman, 1984). The precondition functions $f_G^1, \ldots, f_G^p$ learned for each key step k_p

induce a partial graph, which are consolidated to build the overall graph. See Figure 3 for an illustration of the process.

Graph Prediction Accuracy. *Accuracy* (Equation (2)) measures how often the output (*i.e.*, eligibility) of the predicted and the ground-truth preconditions agree (Sohn et al., 2020). f_G^{p*} is the ground-truth precondition of the key step k_p.

$$\text{Accuracy} = \frac{1}{m}\sum_{p=1}^{m} P\left(f_G^{p}(\mathbf{c}) = f_G^{p*}(\mathbf{c})\right) \tag{2}$$

B InstructGPT Summary Step Generations

We present *summary step sequences* generated by InstructGPT for the *setup chromecast* task below, conditioned on text transcripts from the dataset.

1. Go to Chromecast.com/setup
2. Connect Chromecast to HDMI port
3. Connect USB power cord to TV or power outlet
4. Open Google Home App
5. Follow on-screen instructions

1. Plug in the Chromecast to the TV.
2. Connect the Chromecast to the Wi-Fi network.
3. Use the Chromecast App to select what to cast.

1. Plug in the USB cable to the Chromecast.
2. Connect the Chromecast to the HDMI port on the TV.
3. Change the TV's input to the HDMI port that the Chromecast is connected to.
4. Download the Chromecast App.
5. Set up the Chromecast using the App.
6. Choose the Wi-Fi network.
7. Enter the Wi-Fi password.
8. Cast from the computer by using the Chromecast extension in Google Chrome.
9. Cast from the smartphone or tablet by using a compatible App.

We present *summary step sequences* generated by InstructGPT for the *change iphone battery* task below, conditioned on text transcripts from the dataset.

1. Turn off the phone
2. Remove the bottom screws
3. Lift up the screen
4. Remove the metal plate
5. Unclip the battery connector
6. Pry up the battery
7. Replace the battery
8. Replace the metal plate
9. Line up the screen
10. Snap the screen into place

1. Unscrew the two pentalobe screws beside the Lightning jack.
2. Use a mini suction cup and place it right above the home button.
3. Use a guitar pick to gently rock back and forth until the screen starts lifting.
4. Unscrew the battery cover and remove the shield.
5. Unplug the existing battery by going under the metal flap with a flat edge.
6. Remove the adhesive that keeps the battery in place.
7. Place the new battery in the chassis and plug it in.
8. Place the battery cover back on and screw it in.
9. Lock the top edge of the screen in place.
10. Screw the bottom screws in place.

1. Turn off the iPhone.
2. Remove the screws from the bottom of the phone.
3. Remove the screen from the phone.
4. Remove the battery connector.
5. Remove the adhesive strips from the old battery.
6. Attach the new adhesive strips to the new battery.
7. Place the new battery in the phone.
8. Reconnect the screen to the phone.
9. Replace the screws.
10. Turn on the phone.

C Key-steps identified

We show clusters/key steps identified by the clustering algorithm for the *setup chromecast* task below.

1. - Connect the Chromecast to the Wi-Fi network
   - Connect to the same Wi-Fi network
   - Enter Wi-Fi password to connect Chromecast to Wi-Fi network
   - Join the Chromecast to the Wi-Fi network
   - Connect the Chromecast device to the Wi-Fi network
   - Connect the Chromecast to a Wi-Fi network
   - Connect to the Chromecast's Wi-Fi network
   - Connect the Chromecast to your Wi-Fi network

2. - Plug in the Chromecast to the TV
   - Plug in the Chromecast device to the TV
   - Connect the Chromecast to the TV

3. - Download the Google Home app
   - Download the Google Home application
   - Download the Google Home App

4. - Plug Chromecast into HDMI port and USB port on TV
   - Plug Chromecast into HDMI port on TV
   - Plug Chromecast into HDMI port and USB port for power
   - Plug Chromecast into the HDMI port on your TV
   - Plug Chromecast into power and HDMI port on TV
   - Plug in the Chromecast device to the HDMI port and USB port for power

5. - Go to Chromecast.com/setup
   - Go to chromecast.com/setup
   - Go to chromecast.com/setup on an Android device
   - Go to google.com/chromecast/setup
   - Go to google.com/chromecast/setup in Chrome browser

6. - Follow on-screen instructions to set up Chromecast
   - Follow the instructions on the app to set up Chromecast
   - Follow the prompts to set up the Chromecast
   - Follow the prompts to set up Chromecast

7. - Install the Chromecast App on your phone or tablet
   - Open the Google Home app on your phone or tablet
   - Install the Chromecast app on the phone
   - Install the Chromecast App on your Android device
   - Install the Chromecast App on a computer or mobile device

8. - Download Chromecast App
   - Download Chromecast app
   - Download the Chromecast App
   - Download the Chromecast app

We show clusters/key steps identified by the clustering algorithm for the *change iphone battery* task below.

1.
   - remove the bottom two screws from the phone
   - Remove the screws at the bottom of the iphone
   - Remove the two pentalobe screws at the bottom of the phone
   - remove the two screws on the bottom of the iphone
   - Remove the two screws at the bottom of the iPhone

2.
   - remove battery
   - remove the battery
   - Remove battery
   - Remove the battery
   - Lift up the battery to remove it

3.
   - put in the new battery
   - Install the new battery
   - stick the new battery in
   - Insert the new battery
   - Put in new battery

4.
   - Pry up the frame of the screen with a pry tool
   - use a suction cup and sharp blade to pry open the screen case
   - use a suction cup and pry tool to remove the screen
   - use a pry tool to snap the latches and remove the screen
   - pry up very gently to separate the screen from the frame

5.
   - Turn off the phone
   - Turn off phone
   - Turn off the iPhone

6.
   - Remove the adhesive strips from the old battery
   - remove the adhesive from underneath the battery
   - use the fine tip curved tweezers to peel up the edges of the two adhesive strips at the bottom of the battery
   - remove the adhesive strips holding the battery in place

7.
   - Replace the screws
   - replace screws
   - Replace screws

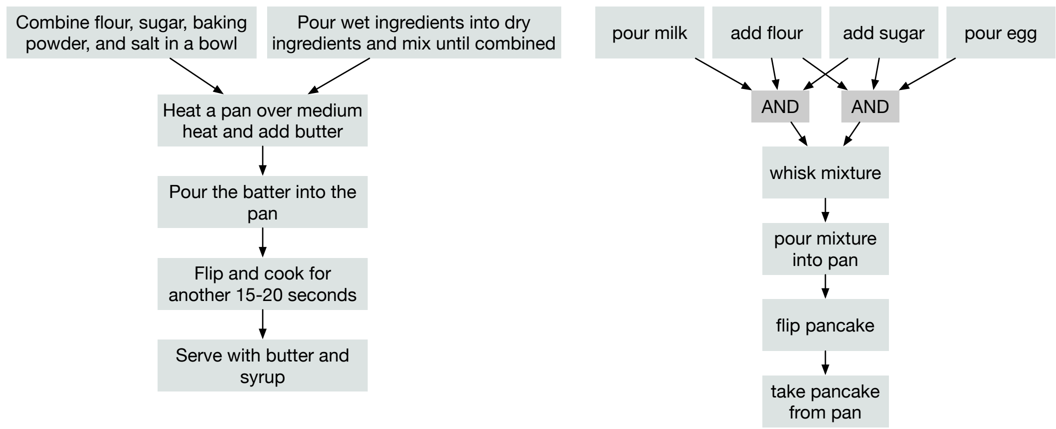
8.
   - Lift up the screen with a suction cup
   - use the suction cup to pull the screen up gently
   - use a suction cup to pull up the screen
   - Use a suction cup to slightly lift the screen
   - Use a suction cup to apply upward pressure on the screen

9.
   - Remove the metal bracket and the two screws holding down the battery cable
   - remove the protective metal cover of the battery connector
   - Remove the two screws in the battery connector cover
   - remove the two screws on the shield that's covering the battery connector
   - unscrew the metal bracket holding the battery connector in place

10.
   - unscrew the four screws that cover the connectors for the screen
   - remove the cover plate that covers the screen connectors
   - Carefully dislodge the three connector tabs and set the screen aside
   - remove the metal cover and gently pry off the connectors of the screen one by one
   - Pull back the screen and remove the four screws securing the metal connector cover

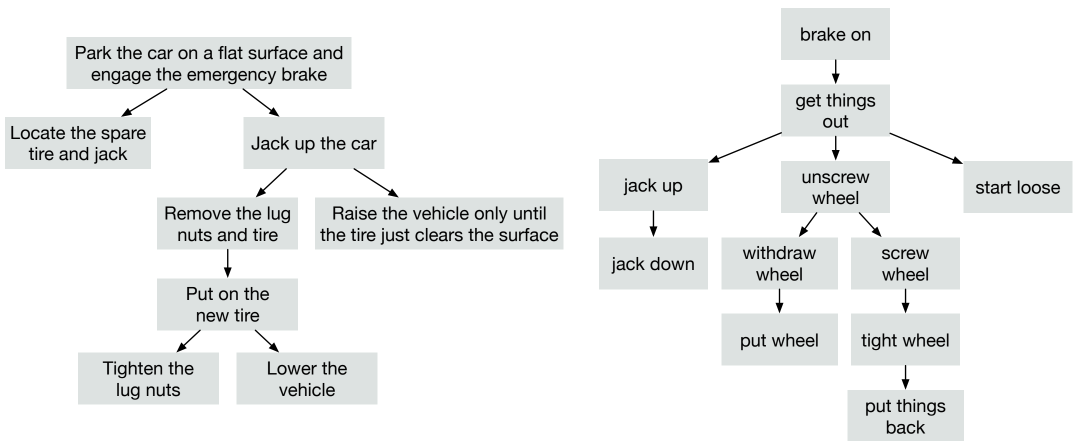
D Generated graphs

We include generated graphs for other ProceL and CrossTask tasks below.



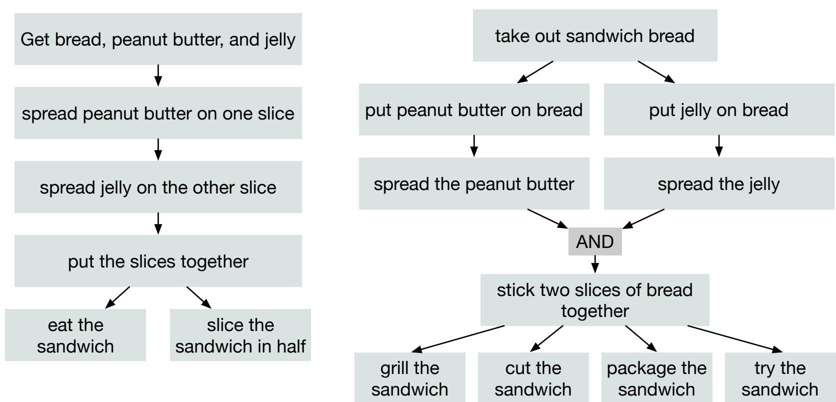
(a) Predicted graph (b) Ground truth graph

Figure 4: Predicted (a) and ground truth (b) graphs for the *make pancakes* task.



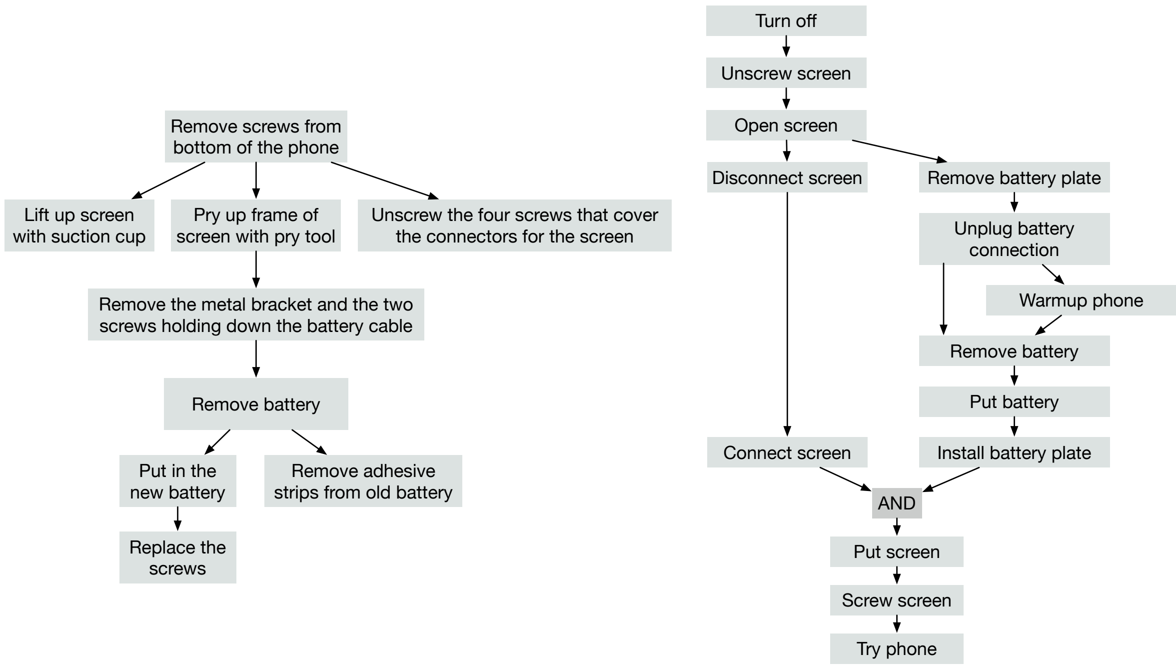
(a) Predicted graph (b) Ground truth graph

Figure 5: Predicted (a) and ground truth (b) graphs for the *change tire* task.



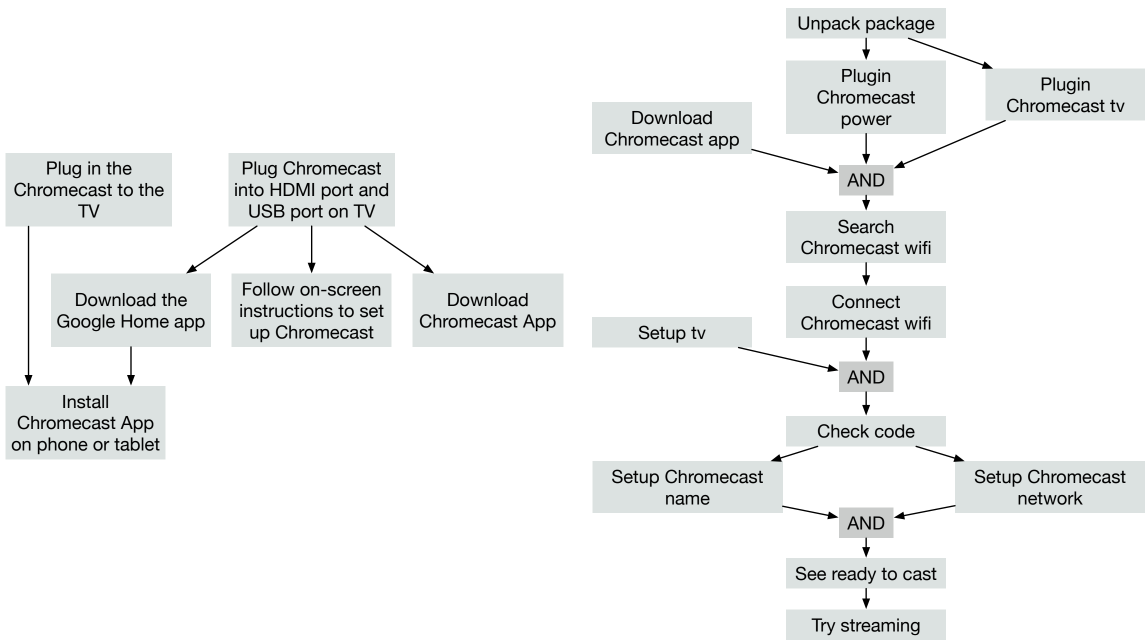
(a) Predicted graph (b) Ground truth graph

Figure 6: Predicted (a) and ground truth (b) graphs for the *make PBJ sandwich* task.



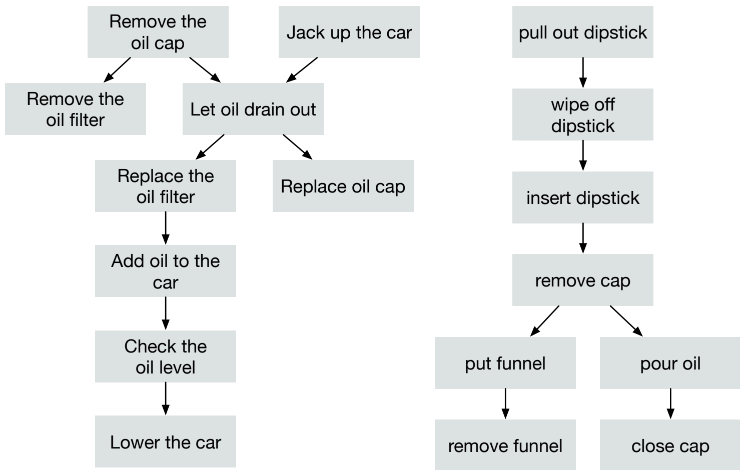
(a) Predicted graph (b) Ground truth graph

Figure 7: Predicted (a) and ground truth (b) graphs for the *change iphone battery* task.



(a) Predicted graph (b) Ground truth graph

Figure 8: Predicted (a) and ground truth (b) graphs for the *setup chromecast* task.



(a) Predicted graph (b) Ground truth graph

Figure 9: Predicted (a) and ground truth (b) graphs for the *add oil to your car* task.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
5

☐ A2. Did you discuss any potential risks of your work?
Not applicable. Left blank.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

3

☑ B1. Did you cite the creators of artifacts you used?
3

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Left blank.

☒ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Left blank.

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Left blank.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
3

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
3

C ☑ Did you run computational experiments?

3

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Left blank.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Left blank.

☒ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Left blank.

☒ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Left blank.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.