MULTIMODAL SUBTASK GRAPH GENERATION FROM INSTRUCTIONAL VIDEOS

Yunseok Jang[*†], **Sungryull Sohn**[*‡], **Lajanugen Logeswaran**[‡], **Tiange Luo**[†], **Moontae Lee**[‡], **Honglak Lee**[†‡]
[†]{yunseokj, tiangel, honglak}@umich.edu
[‡]{srsohn, llajan, moontae, honglak}@lgresearch.ai

ABSTRACT

Real-world tasks consist of multiple inter-dependent subtasks (*e.g.*, a dirty pan needs to be washed before cooking). In this work, we aim to model the causal dependencies between such subtasks from instructional videos describing the task. This is a challenging problem since complete information about the world is often inaccessible from videos, which demands robust learning mechanisms to understand the causal structure of events. We present Multimodal Subtask Graph Generation (MSG^2), an approach that constructs a *Subtask Graph* defining the dependency between subtasks relevant to a task from noisy web videos. Graphs generated by our multimodal approach are closer to human-annotated graphs compared to prior approaches. MSG^2 further performs the downstream task of next subtask prediction 85% and 30% more accurately than recent video transformer models in the ProceL and CrossTask datasets, respectively.

1 INTRODUCTION

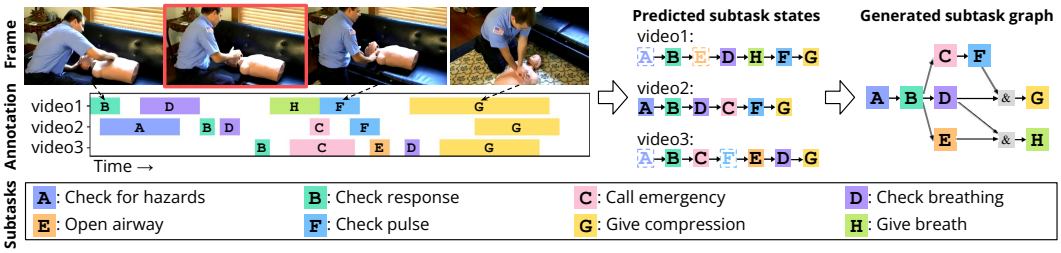


Figure 1: Given video instances describing a task such as *perform CPR*, along with noisy subtask annotations, we attempt to extract causal dependencies between the subtasks. Data noise presents a major challenge in identifying these dependencies – (i) subtask A is omitted in videos 1 and 3 (ii) even though subtask E is present in video 1 (frame highlighted in red outline) it is missing from the annotations. We predict such missing steps (dotted boxes) by leveraging visual and text signals (Section 3.1) and generate a subtask graph based on the updated subtask sequences (Section 3.2).

We aim to infer subtask dependencies from multi-modal video-text data and generate a concise graph representation of the subtasks that shows these dependencies. In particular, we focus on instructional videos describing how to perform real-world activities such as *jump-starting a car* or *replacing an iPhone's battery*. We consider a subtask graph representation that is more expressive at modeling subtask dependencies (*e.g.*, AND nodes allow modeling multiple preconditions) compared to representations considered in the literature such as partial order graphs or dataflow graphs.

Instructional videos present a unique set of challenges in understanding task structure due to data noise, as in Figure 1. To tackle these challenges, we present Multimodal Subtask Graph Generation (MSG^2) in order to robustly learn task structure from online real-world instructional videos. As identifying dependencies between the subtasks is challenging due to the data noise, MSG^2 first present a multimodal subtask state prediction module that makes use of video and text data to infer the missing subtasks (Section 3.1). We then adopt a complexity-regularized inductive logic

[*]Equal contribution

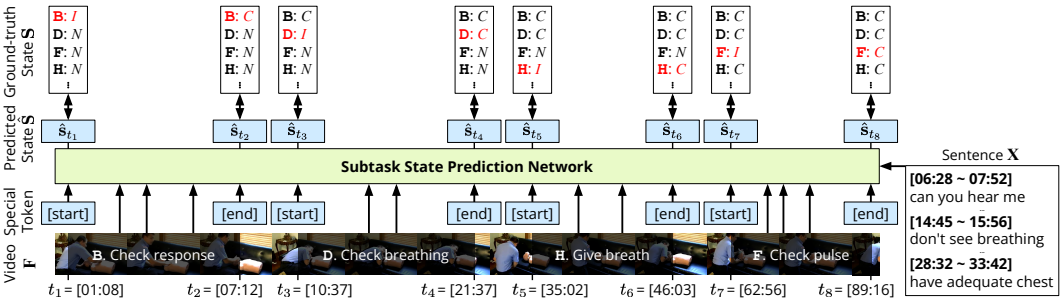


Figure 2: **Architecture of subtask state prediction module.** We denote subtask state labels as *N* (*not started*), *I* (*in progress*), *C* (*completed*), and labels in red represent ongoing subtask. We train the model to predict each subtask's state and use it to predict missing subtask labels in the data, which in turn contributes to more accurate graph generation.

programming (ILP) method to generate a graph from the noise-reduced subtask state predictions (Section 3.2). Lastly, we demonstrate the utility of MSG^2 by employing predicted subtask graphs in a downstream task of predicting the next subtask in a video.

2 Problem Setting

Our work builds on the subtask graph framework (Sohn et al., 2018; 2020), which describes the causal dependency structure of a complex task τ consisting of N_τ subtasks. Each subtask has a **precondition** that must be satisfied before the subtask can be completed. For instance, the precondition of subtask `H` in Figure 1 is $f_{\texttt{H}} = \&(\texttt{D}, \texttt{E})$ (*i.e.*, subtasks `D` and `E` must be completed before performing `H`). The **subtask graph** visualizes the preconditions $f_1, \ldots, f_{N_\tau}$ of the subtasks.

Problem. We are given a set of instructional videos $\mathbf{V}_\tau$ (*i.e.*, task instances) describing a task τ (*e.g.*, *perform CPR*, *jump car*). Each video consists of video frame $\mathbf{F} = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_T)$, text transcript $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$, and subtask state labels $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_T)$, which are time-aligned with respect to the video frames $\mathbf{F}$. Specifically, the state of the n^{th} subtask at frame t can have the following values: $s_t[n] \in \{$*not started*, *in progress*, *completed*$\}$. Our goal is to predict the subtask graph G of task τ by extracting accurate subtask state from the given videos $\mathbf{V}_\tau$.

3 Method

3.1 Subtask State Prediction from Noisy Annotations

Based on the intuition that visual and textual data provide complementary information, we train a network by providing both visual and text signals to predict which of the three states ($\hat{s}_t[n] \in \{$*not started*, *in progress*, *completed*$\}$) a subtask n is at any given time t. These predictions are used as inputs for subtask graph generation as described in Section 3.2.

Architecture. Given visual information $\mathbf{F}$ and sentences from corresponding text transcript $\mathbf{X}$, we first obtain frame embeddings $\mathbf{F}^e$ and sentence embeddings $\mathbf{X}^e$ using a pre-trained CLIP (Radford et al., 2021) model. We enrich the obtained visual representations with information from the text transcript through a Multi-Head Attention (Vaswani et al., 2017), followed by a residual layer. We subsample frames within each labeled subtask segment and predict subtask states at the beginning and the end of each subtask (instead of predicting for every time step, which can be noisy) by appending special delimiter symbols ([start] and [end], respectively) as shown in Figure 2.

Training Objective for Subtask State Prediction. We consider an ordinal regression loss (Niu et al., 2016; Cao et al., 2020) to model subtask states, noting that a subtask cannot directly transition from *not started* to *completed* without going through the intermediate *in progress* state. Our objective is based on two binary classification losses corresponding to the two decision boundaries (*i.e.*, $\hat{s}_t[n] = -b$ and $\hat{s}_t[n] = b$) as shown in Equation (1). σ denotes the sigmoid function, CE is the binary cross-entropy loss and the expectation is taken over time-steps t corresponding to the special delimiter tokens and subtasks n that appear in the video.

$$\mathcal{L}_{\text{ssp}} = \mathbb{E}_{t,n}[\text{CE}(\sigma(\hat{s}_t[n] + b), \mathbb{I}[s_t[n] \neq \textit{not started}]) + \text{CE}(\sigma(\hat{s}_t[n] - b), \mathbb{I}[s_t[n] = \textit{completed}])] \quad (1)$$

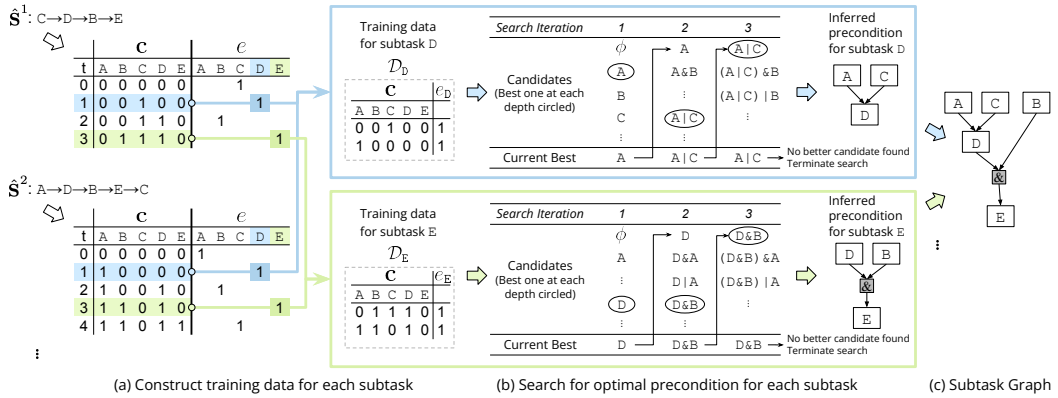


Figure 3: **Overview of subtask graph generation.** (a) Given the subtask state labels $\hat{\mathbf{S}}$, we extract training data $\mathcal{D}$ for precondition inference for each subtask. (b) We use a greedy algorithm to find the precondition for each subtask that maximizes the objective in Equation (4). At each iteration, the algorithm adds a Boolean operation and variable to construct new hypotheses. The optimal hypothesis is chosen and advanced to the next iteration. The search terminates either when there is no better solution at the current iteration or maximum number of iterations is reached. (c) All preconditions are consolidated into a single subtask graph.

3.2 Graph Generation from Subtask State Labels

Learning to Model Preconditions. Given the noise-reduced subtask state labels $\hat{\mathbf{S}}$, we infer the precondition for each subtask to construct the subtask graph. The precondition learning problem can be stated as learning a function $e_n = f_n(\mathbf{c})$ where $\mathbf{c} \in \{0,1\}^{N_\tau}$ represents the completion status of each subtask (*i.e.*, $c[i]$ denotes whether i^{th} subtask was completed), and $e_n \in \{0,1\}$ represents whether the precondition of n^{th} subtask is satisfied.

The dataset $\mathcal{D}_n$ for inferring the precondition for n^{th} subtask needs to be constructed from the noise-reduced subtask state labels $\hat{s}_t[n]$. This is non-trivial as the $\hat{s}_t[n]$ only provides partial information about the preconditions. We can infer that the precondition of the n^{th} subtask is satisfied whenever $\hat{s}_t[n] =$ *in progress*, since a subtask can only be performed if its precondition is satisfied. However, in all other cases (*i.e.*, $\hat{s}_t[n] \in$ {*not started*, *completed*}) it is unknown whether the precondition for the subtask is met. Based on the instances where the subtask state is *in progress*, we construct the training dataset as $\mathcal{D}_n = \{(\hat{\mathbf{c}}_t, 1) \mid \hat{e}_t[n] = 1\}$, where we define $\hat{c}_t[n] = \mathbb{I}(\hat{s}_t[n] =$ *completed*$)$ and $\hat{e}_t[n] = \mathbb{I}(\hat{s}_t[n] =$ *in progress*$)$. Figure 3 (a) illustrates the data construction process.

Learning Objective. Conventional ILP algorithms that optimize J_{acc} in Equation (2) (Muggleton, 1991; Sohn et al., 2020) produce a trivial solution where there is no precondition; *i.e.*, J_{acc} is maximized by simply predicting $f_n(\mathbf{c}) = 1$ for all $\mathbf{c}$ since e_n is always 1 in the data $\mathcal{D}_n$.

$$J_{\text{acc}} = P(e_n = f_n(\mathbf{c})) = \mathbb{E}_{(\mathbf{c},e_n)}[\mathbb{I}[e_n = f_n(\mathbf{c})]] \simeq \mathbb{E}_{(\mathbf{c},e_n)\in\mathcal{D}_n}[\mathbb{I}[e_n = f_n(\mathbf{c})]] \tag{2}$$

To overcome these challenges, we modify Equation (2) and maximize the *precision* as follows:

$$J_{\text{prec}} = P(e_n = 1 \mid f_n(\mathbf{c}) = 1) = \frac{\mathbb{E}_{(\mathbf{c},e_n)}[e_n \cdot f_n(\mathbf{c})]}{\mathbb{E}_{\mathbf{c}}[f_n(\mathbf{c})]} \simeq \frac{\mathbb{E}_{(\mathbf{c},e_n)\in\mathcal{D}_n}[e_n \cdot f_n(\mathbf{c})]}{\mathbb{E}_{\mathbf{c}}[f_n(\mathbf{c})]}. \tag{3}$$

Note that $J_{\text{prec}} \simeq J_{\text{acc}}/\mathbb{E}_{\mathbf{c}}[f_n(\mathbf{c})]$ since our dataset only consists of instances with $e_n = 1$. The denominator penalizes the precondition being overly *optimistic* (*i.e.*, predicting $f_n(\mathbf{c}) = 1$ more often than $f_n(\mathbf{c}) = 0$ will increase the denominator $\mathbb{E}_{\mathbf{c}}[f_n(\mathbf{c})]$), which helps mitigate the effect of positive label bias in the data (*i.e.*, having no data with $e_n = 0$). Our final optimization objective is given by Equation (4) where α is a regularization weight and $C(f_n)$ measures the complexity of the precondition f_n in terms of the number of Boolean operations. The regularization term handles the data noise since the noise often adds extra dependency between subtasks and results in overly complicated (*i.e.*, large number of Boolean operations) precondition. See Appendix B.b for details.

$$\max_{f_n} J_{\text{ours}}(f_n) = \max_{f_n} J_{\text{prec}}(f_n) - \alpha C(f_n) \tag{4}$$

Optimization. We consider a greedy search algorithm to optimize J_{ours}. Starting from the null precondition, at each iteration of the search, we construct candidate preconditions by adding a Boolean

Model	Acc ↑	SPOC ↑
ProScript	57.50	62.34
MSGI	54.23	64.62
MSGI+	73.59	72.39
MSG^2-noSSP	81.62	88.35
MSG^2 (Ours)	**83.16**	**89.91**

Table 1: **Graph generation results in ProceL.** We report the average percentage (%) across all tasks (See Appendix B.c for task-level performance).

Model	ProceL	CrossTask
STAM	29.86	40.17
ViViT	26.98	41.96
ProScript	18.86	36.78
MSGI	17.42	32.31
MSGI+	26.54	32.72
MSG^2-noSSP	48.39	53.39
MSG^2 (Ours)	**55.38**	**54.42**

Table 2: **Next subtask prediction results.** Average subtask prediction accuracy (%) across tasks on ProceL and CrossTask datasets (Please check Appendix C.c for task-level performance).

operation (*e.g.*, & and |) and a variable (*e.g.*, `A`, `B`, etc) to the best precondition of the previous iteration. We choose the candidate precondition that maximizes Equation (4) and continue until either when a maximum number of iterations is reached or no better solution is found in the current iteration. See Figure 3 (b) for an illustration of the search algorithm.

4 Experiments

4.1 Graph Generation

Baselines. MSGI (Sohn et al., 2020): An inductive logic programming (ILP) method that maximizes the objective in Equation (2). **MSGI+**: A variant of MSGI which assumes the precondition is not met until the subtask is performed (*i.e.*, $e_t[n] = 0$ if $\hat{s}_t[n] =$ *not started*) similar to prior work (Hayes and Scassellati, 2016; Xu et al., 2018)). **MSG^2-noSSP**: Generate graph (Section 3.2) from human-annotated subtask state labels $\mathbf{S}$. In other words, this is our method without the multimodal subtask state inference module in Section 3.1. **ProScript** (Sakaguchi et al., 2021): A T5 model (Raffel et al., 2020) fine-tuned to predict a partially-ordered script from scenario descriptions (*e.g.*, baking a cake) with unordered subtask descriptions.

We use T5-Large (Raffel et al., 2020) as the transformer in our state prediction module. During training we randomly omit video frames corresponding to 25% of subtasks for data augmentation. See Appendix A.b for further details and ablations regarding choice of model and architecture.

Metrics. We consider the following metrics to evaluate predicted graphs $G = (f_1, \ldots, f_{N_\tau})$. **Accuracy** (Equation (5)) measures how often predicted and ground-truth preconditions agree (Sohn et al., 2020), where f_n^* is the ground-truth precondition of the n^{th} subtask. **Strict Partial Order Consistency** (SPOC): We define a strict partial order relation R_G imposed by graph G on subtasks $a, b \in S$ as: $(a, b) \in R_G$ iff a is an ancestor of b in graph G (*i.e.*, in matrix notation, $R_G[a][b] \triangleq \mathbb{I}[a$ is an ancestor of b in $G]$). SPOC is defined as in Equation (6), where G^* is the ground-truth graph.

$$\text{Accuracy} = \frac{\sum_{n=1}^{N_\tau} \mathbb{E}_{\mathbf{c}} \left[\mathbb{I}\left[f_n(\mathbf{c}) = f_n^*(\mathbf{c})\right]\right]}{N_\tau} \quad (5) \qquad \text{SPOC} = \frac{\sum_{a \neq b} \mathbb{I}\left[R_G[a][b] = R_{G^*}[a][b]\right]}{N_\tau(N_\tau - 1)} \quad (6)$$

Results. We measure the graph generation performance over ProceL (Elhamifar and Naing, 2019) dataset (See Appendix A.a for the details about preprocessing). Table 1 summarizes the performance of different graph generation methods. First, the complexity regularization term (Equation (3)) helps our method be more robust to incomplete and noisy data compared to the MSGI and MSGI+ baselines. When using subtask states predicted by our state prediction module (MSG^2) instead of human annotations (MSG^2-noSSP), we observe consistent improvements for all metrics, which shows the benefit of predicting missing subtasks by exploiting multimodal vision-text data. See Figure A for examples of subtask state prediction. Our method also outperforms proScript (Sakaguchi et al., 2021), which relies on a web-scale pretrained text model (Raffel et al., 2020).

Qualitative Evaluation. Figure 4 shows predicted graphs for the *perform CPR* task. The MSG^2 graph (Figure 4.(c)) is closer to the ground-truth graph (Figure 4.(a)) compared to our model without subtask state prediction (Figure 4.(b)). This is because the predicted subtask states provide additional clues about subtask completion. For instance, consider the subtask `D`, which is

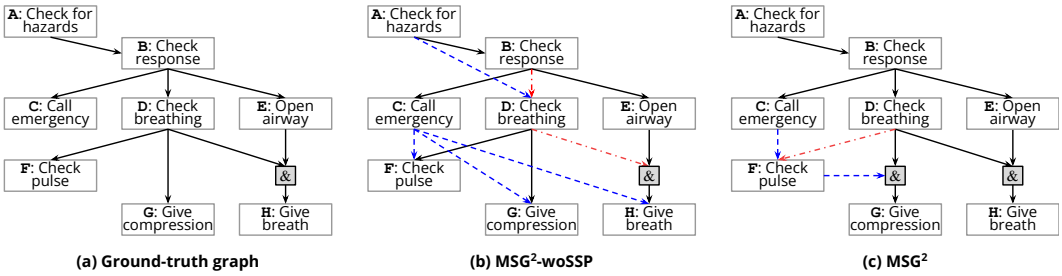


Figure 4: **Illustration of subtask graphs generated by each method for *perform CPR* task**. For the predicted graphs ((b) MSG^2 without Subtask State Prediction and (c) MSG^2), redundant edges (*i.e.*, false positive) are colored in blue and missing edges (*i.e.*, false negative) are colored in red.

a prerequisite for subtasks F, G and H in the ground truth graph. Since D is sparsely annotated in the data (does not appear in 29% of the sequences), the baseline model assumes that the presence of other subtasks (*e.g.*, C) explains subtasks F, G and H (the redundant outgoing arrows from C). By recovering some of the missing annotations for D based on visual and text signals (*e.g.*, it is clearly visible), our method resolves some of these cases (*e.g.*, C is no longer a precondition for H). However, the recovery is not perfect, and our model still mistakenly assumes that C is a precondition for F. Further improvements in multimodal subtask state modeling can help address these errors.

4.2 Next Subtask Prediction

Baselines. We first choose two open-sourced end-to-end video-based Transformer models (STAM (Sharir et al., 2021), ViViT (Arnab et al., 2021)) for comparison. To keep the input the same for all models used in this experiment, we replace 16×16 patch embeddings from each model implementation with the CLIP embedding used in our model. We train the methods to predict the next subtask from the history of previous subtasks using a multi-class subtask classification loss, where a linear projection of the final transformer hidden state is used as the subtask prediction. In addition to these end-to-end neural baselines, we also tested the graph-based methods introduced in Table 1. For all graph-based methods, we first generated the graph from the train split. We then use the generated subtask graph to predict whether the precondition is satisfied or not from the subtask completion in test split, using the GRProp policy (Sohn et al., 2018). Please check Appendix C.a for details.

Metric. For each test video, we measure the accuracy of predicting the next subtask correctly, given previous subtasks and corresponding ASR sentences.

Results. Table 2 shows the next subtask prediction performance. Compared to the end-to-end neural baselines, MSG^2 achieves 85% and 30% higher prediction performance in ProceL (Elhamifar and Naing, 2019) and CrossTask (Zhukov et al., 2019) datasets, respectively. In addition, we observe that end-to-end neural baselines are competitive with some of the graph generation baselines. This indicates that the higher quality of graphs predicted by our approach is responsible for the significant performance improvement over all baselines.

5 Conclusion

We present MSG^2, a method for generating subtask graphs from instructional videos on the web. We first predict accurate subtask state labels from noisily annotated instructional videos. We additionally propose a novel complexity-constrained ILP method to deal with noisy and incomplete data. We demonstrate that our method produces more accurate subtask graphs than baselines. In addition, our method achieves better performance than recent video transformer approaches in predicting the next subtask, demonstrating the usefulness of our subtask graph.

Acknowledgements

We thank Jae-Won Chung, Junhyug Noh, Dongsub Shim and Anthony Liu for constructive feedback on the manuscript. This work was supported in part by grants from LG AI Research and NSF CAREER IIS-1453651.

REFERENCES

A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid. ViViT: A Video Vision Transformer. In *ICCV*, 2021.

L. Breiman. *Classification and Regression Trees*. Routledge, 1984.

W. Cao, V. Mirjalili, and S. Raschka. Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation. *Pattern Recognition Letters*, 2020.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 2019.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021.

E. Elhamifar and Z. Naing. Unsupervised Procedure Learning via Joint Dynamic Summarization. In *ICCV*, 2019.

B. Hayes and B. Scassellati. Autonomously Constructing Hierarchical Task Networks for Planning and Human-Robot Collaboration. In *ICRA*, 2016.

D. P. Kingma and J. Ba. ADAM: A Method for Stochastic Optimization. In *ICLR*, 2015.

L. H. Li, M. Yatskar, D. Yin, C.-J. Hsieh, and K.-W. Chang. VisualBERT: A Simple and Performant Baseline for Vision and Language. *arXiv:1908.03557*, 2019.

A. Miech, J.-B. Alayrac, L. Smaira, I. Laptev, J. Sivic, and A. Zisserman. End-to-End Learning of Visual Representations from Uncurated Instructional Videos. In *CVPR*, 2020.

S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 1991.

Z. Niu, M. Zhou, L. Wang, X. Gao, and G. Hua. Ordinal Regression with Multiple Output CNN for Age Estimation. In *CVPR*, 2016.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*, 2021.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *JMLR*, 2020.

K. Sakaguchi, C. Bhagavatula, R. Le Bras, N. Tandon, P. Clark, and Y. Choi. proScript: Partially Ordered Scripts Generation. In *Findings of EMNLP*, 2021.

G. Sharir, A. Noy, and L. Zelnik-Manor. An Image is Worth 16x16 Words, What is a Video Worth? *arXiv:2103.13915*, 2021.

Y. Shen, L. Wang, and E. Elhamifar. Learning to Segment Actions from Visual and Language Instructions via Differentiable Weak Sequence Alignment. In *CVPR*, 2021.

S. Sohn, J. Oh, and H. Lee. Hierarchical Reinforcement Learning for Zero-shot Generalization with Subtask Dependencies. In *NeurIPS*, 2018.

S. Sohn, H. Woo, J. Choi, and H. Lee. Meta Reinforcement Learning with Autonomous Inference of Subtask Dependencies. In *ICLR*, 2020.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. In *NeurIPS*, 2017.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv:1910.03771*, 2019.

D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese. Neural Task Programming: Learning to Generalize Across Hierarchical Tasks. In *ICRA*, 2018.

D. Zhukov, J.-B. Alayrac, R. G. Cinbis, D. Fouhey, I. Laptev, and J. Sivic. Cross-Task Weakly Supervised Learning from Instructional Videos. In *CVPR*, 2019.

A SUBTASK STATE PREDICTION

A.A TRAINING DETAILS

Preprocessing. We employ ProceL dataset (Elhamifar and Naing, 2019), which includes twelve task with 54.5 videos and 13.1 subtasks per task on average with subtask label and timing annotations (start and end times). We first convert all videos in ProceL (Elhamifar and Naing, 2019) to 30 fps. Then, we extract the verb phrases of *verb+(prt)+dobj +(prep+pobj)*[1] from Automated Speech Recognition (ASR) output, using the implementation in SOPL (Shen et al., 2021). For both vision and text data, we extract the frame and verb phrases features from the 'ViT-B/32' variant of CLIP (Radford et al., 2021). We extract the features with each subtask's start and end position and load the CLIP features, instead of video frames, for faster data reading. We choose the first label of each subtask in a video and convert the temporal segment labels to *not started*, *in progress*, and *completed* for the existing subtasks.

Training. We set the hidden dimension of each feedforward dimension in 16-head modality fusion attention to be the same as the CLIP representation embedding size of 512 and use a single fully-connected layer when projecting to the status prediction. During training, we randomly drop up to 25% of subtasks from each task sequence and then subsample at least three frames from each subtask region, but no more than 190 frames in total. On the other hand, for testing, we did not drop any subtask and sample 3 frames per subtask in an equidistance manner (no randomness). For all the language features, because the length of the ASR varies from video to video, we use three consecutive sentences per frame while setting the center of the sentence closest to the selected frame, inspired by Miech et al. (2020). We train all models with a learning rate of 3e-4 with the Adam (Kingma and Ba, 2015) optimizer with cosine scheduling, following BERT (Devlin et al., 2019). We set the batch size as 32 and trained each model for 600 epochs, with 100 steps of warm-up. Each model is trained on an 18.04 LTS Ubuntu machine with a single NVIDIA A100 GPU on CUDA 11.3, cuDNN 8.2, and PyTorch 1.11.

A.B ABLATION STUDY

Since a subtask sequence $\mathbf{S}$ in video stores the start and end frame numbers, we can directly compare the completion prediction result of all subtasks by checking $\mathbb{I}[\hat{s}_t[n] \geq 1]$. However, because the label for $\mathbf{S}$ only covers the labeled part of the sequence, we first split 15% of the data as the validation set and hand-annotated the subtask state of all subtask labels for the videos in the validation set. For both the ground-truth subtask graph and the subtask state labels, we asked three people to manually annotate after watching all videos in the task. We choose the majority answer among three as the label for the subtask state labels, and we iterate multiple rounds of ground-truth subtask graph labeling until the label converges among three people. We performed an ablation study with the VisionOnly (train without $\mathbf{X}^e$), our model without having first binary cross entropy loss in Equation (1) (denoted as w/o *in progress* state), as well as our model with the skip connection (adding $+\mathbf{F}^e$ after multihead attention before feeding to the transformer model, following Transformer (Vaswani et al., 2017). We denote this as 'Vision+ASR') by measuring the binary completion prediction accuracy per task with the hand-annotated labels. In addition to this, we performed additional experiments with the pretrained ViT (Dosovitskiy et al., 2021) and VisualBERT (Li et al., 2019) models from Huggingface (Wolf et al., 2019), replacing the T5 (Raffel et al., 2020) model. Specifically, we use 'google/vit-large-patch16-224', 'uclanlp/visualbert-vqa-coco-pre', and 'google/t5-v1_1-large' pretrained weights for this ablation. We also tried a variation of VisualBERT (indicated as VisualBERT†) where we directly feed the frames $\mathbf{F}^e$ and sentences $\mathbf{X}^e$, instead of our multihead attention layer in Section 4.2, as input. We set b in Equation (1) as 1 for all of the models.

The results are shown in Table A. First of all, we can see that our model with the skip connection could lead the model to be overfitted to the train set, which is the reason behind our decision not to add a skip connection to the MSG^2 model. Also, we found inferior performance when we train without ASR or *in progress* state information, so we perform subtask state prediction with both

[1] parenthesis denotes optional component, prt: particle, dobj: direct object, prep: preposition, pobj: preposition object.

Table A: **Ablation Result on Subtask State Prediction in ProceL (Elhamifar and Naing, 2019).** We denote video-input only case as "VisionOnly", video with the narration text data, but with skip connection as "Vision + ASR", and our subtask state prediction model as "**Ours**" and measure the performance of completion prediction. We denote the pretrained transformer model as "ViT" and "VisualBERT", following the pretrained model names (Dosovitskiy et al., 2021; Li et al., 2019). Task label indexes are (a) Assemble Clarinet (b) Change Tire (c) Perform CPR (d) Setup Chromecast (e) Change Toilet Seat (f) Make Peanut Butter and Jelly Sandwich (g) Jump Car (h) Tie Tie (i) Change iPhone Battery (j) Make Coffee, (k) Repot Plant and (l) Make Salmon Sandwich, respectively.

Module	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)	Avg
VisionOnly	72.41	74.91	80.21	93.68	72.94	91.11	85.58	93.65	89.52	88.93	79.45	70.90	82.77
Vision + ASR	71.83	74.76	83.12	89.42	69.05	91.07	83.80	90.08	86.36	87.78	78.16	72.64	81.51
Ours (from scratch)	63.56	68.93	74.73	90.36	66.61	79.02	71.54	87.76	69.54	80.47	72.42	65.97	74.24
Ours (w/o *in progress* state)	70.67	74.42	82.72	93.28	72.41	89.95	85.56	93.94	88.01	88.85	79.59	74.63	82.84
Ours	71.25	74.93	83.51	94.56	73.42	89.76	85.94	94.71	91.14	89.54	79.44	75.58	**83.65**
VisualBERT (Li et al., 2019)	69.09	73.84	79.63	69.59	58.45	91.68	84.01	90.40	83.18	83.83	62.07	67.73	76.13
VisualBERT†	59.01	59.99	72.98	71.07	64.66	78.93	77.69	72.33	75.52	75.33	63.43	68.10	69.92
ViT (Dosovitskiy et al., 2021)	58.69	59.92	74.11	68.84	59.69	79.09	78.29	70.39	74.90	78.17	70.03	67.45	69.96

vision and language modality with *in progress* for the rest of the paper. We additionally found that the model predicts *completed* for the unlabeled tasks more clearly one subtask after the original prediction timing. We conjecture that the end-time of a subtask, which is annotated by a human annotator, is often noisy and annotated to a slightly earlier time step where subtask is still *in progress*. Thus, we grab the states from the next subtask timing and use *completed* if $\hat{s}[n] \geq 0$ instead in graph generation. We also trained our model from scratch instead of finetuning a pretrained T5 encoder-based model. We believe the performance gap between 'Ours' and 'Ours (from scratch)' shows the effectiveness of finetuning from the pretrained weights. In addition to this, we tested with other pretrained Transformers and found that the pretrained T5 encoder-based model performs best among all the transformer models. Interestingly, ViT and VisualBERT were worse than T5, which seems to indicate that language priors are more useful for modeling subtask progression.

A.C Visualization of Subtask Completion

We present predicted subtask completion in Figure A. Our model predicts *completed* states from missing subtasks (labeled in red). Such predicted subtask states help generate better graphs.

B Graph Generation

B.A Background: Subtask Graph Inference using Layer-wise Inductive Logic Programming

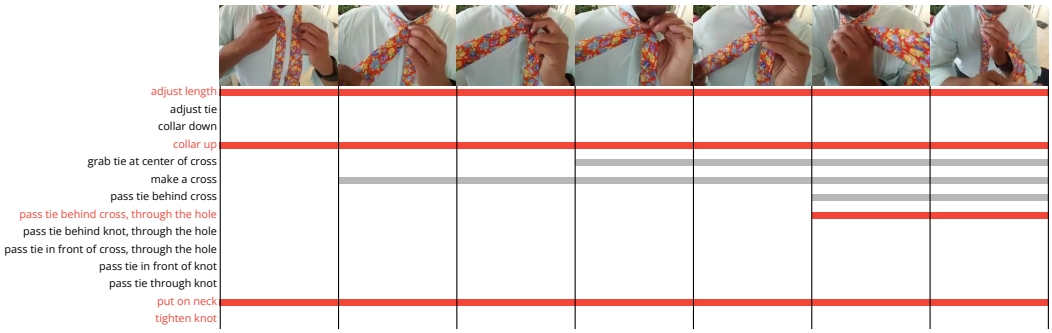
For a task τ that consists of N_τ subtasks, we define the *completion* vector $\mathbf{c} \in \{0,1\}^{N_\tau}$ and *eligibility* vector $\mathbf{e} \in \{0,1\}^{N_\tau}$ where c_n indicates if the n-th subtask has completed and e_n indicates if the n-th subtask is eligible (*i.e.*, its precondition is satisfied). Given $\mathcal{D} = \{(\mathbf{c}^j, \mathbf{e}^j)\}_{j=1}^{|\mathcal{D}|}$ as training data, Sohn et al. (2020) proposed an Inductive Logic Programming (ILP) algorithm which finds the subtask graph G that maximizes the binary classification accuracy (Equation (A)):

$$\hat{G} = \underset{G}{\operatorname{argmax}}\, P(f_G(\mathbf{c}) = \mathbf{e}) \tag{A}$$

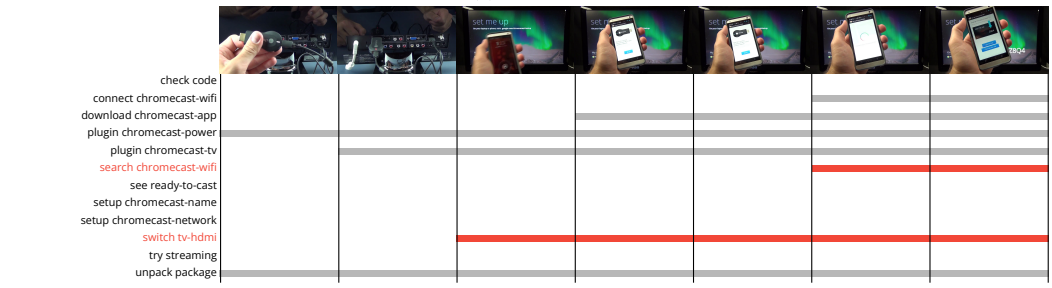
$$= \underset{G}{\operatorname{argmax}} \sum_{j=1}^{|\mathcal{D}|} \mathbb{I}[\mathbf{e}^j = f_G(\mathbf{c}^j)] \tag{B}$$

$$= \left(\underset{G_1}{\operatorname{argmax}} \sum_{j=1}^{|\mathcal{D}|} \mathbb{I}\left[e_1^j = f_1(\mathbf{c}^j)\right], \ldots, \underset{G_{N_\tau}}{\operatorname{argmax}} \sum_{j=1}^{|\mathcal{D}|} \mathbb{I}\left[e_{N_\tau}^j = f_{N_\tau}(\mathbf{c}^j)\right] \right), \tag{C}$$

where $\mathbb{I}[\cdot]$ is the element-wise indicator function, $f_G : \mathbf{c} \mapsto \mathbf{e}$ is the precondition function defined by the subtask graph G, which predicts whether subtasks are eligible (*i.e.*, the precondition is satisfied)



(a) Tie Tie



(b) Setup Chromecast

Figure A: **Completion Prediction Examples.** We plot predicted subtask completions from (a) Tie Tie and (b) Setup Chromecast. The missing labels in the original dataset are colored red, and each colored row represents the completion of a subtask at the matched frame on top. The presence of a horizontal bar at a particular time-step indicates the subtask is in *completed* state at the time-step. Bars in red show subtask completion states that were inferred by our model but were missing in the dataset.

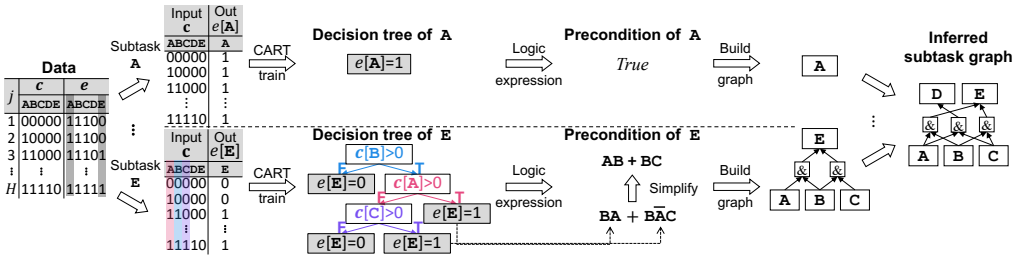


Figure B: The inductive logic programming (ILP) module takes all the completion and eligibility vectors $(\{(\mathbf{c}^j, \mathbf{e}^j)\}_{j=1}^{|\mathcal{D}|})$ as input, and builds a binary decision tree to infer the precondition G. For example, in case of subtask $\texttt{E}$ (the bottom row in the Figure), it takes $(\{(\mathbf{c}^j, e^j[\texttt{E}])\}_{j=1}^{|\mathcal{D}|})$ as (input, output) pairs of training dataset, and constructs a decision tree by choosing a variable (*i.e.*, component of completion vector $\mathbf{c}$, which corresponds to each subtask) at each step that best splits the true (*i.e.*, $e[\texttt{E}] = 1$) and false (*i.e.*, $e[\texttt{E}] = 0$)-labeled data samples. Then, the decision tree is represented as a logic expression (*i.e.*, precondition), simplified, transformed to a subtask graph form, and merged together to form a subtask graph. The figure was adopted from Sohn et al. (2020) and modified to match our notation style.

from the subtask completion vector $\mathbf{c}$, and $f_n : \mathbf{c} \mapsto e_n$ is the precondition function of n-th subtask defined by the precondition G_n.

Figure B illustrates the detailed process of subtask graph inference in Sohn et al. (2020) from the completion and eligibility data. The precondition function f_n is modeled as a binary decision tree where each branching node chooses the best subtask (*e.g.*, subtask B in the first branching in the bottom row of Figure B) to predict whether the n-th subtask is eligible or not based on Gini impurity Breiman (1984). Each binary decision tree constructed in this manner is then converted into a logical expression (*e.g.*, $BA + B\bar{A}C$ in the bottom row of Figure B) that represents precondition. Finally, we build the subtask graph by consolidating the preconditions of all subtasks.

B.B SUBTASK GRAPH INFERENCE FROM REAL-WORLD DATA

Layer-wise Precondition Inference. One major problem of inferring the precondition independently for each subtask is the possibility of forming a *cycle* in the resulting subtask graph, which leads to a causality paradox (*i.e.*, subtask A is a precondition of subtask B and subtask B is a precondition of subtask A). To avoid this problem, we perform precondition inference in a *layer-wise* fashion similar to Sohn et al. (2020).

To this end, we first infer the layer of each subtask from the (noise reduced) subtask state labels $\{\mathbf{S}^i\}_{i=1}^{|\mathbf{V}_\tau|}$ for the task τ. Sohn et al. (2020) infer the layer of each subtask by finding the minimal set of subtask completions to perfectly discriminate the eligibility of a subtask. However, this approach is not applicable to our setting due to a lack of data points with ineligible subtasks; *i.e.*, we can perfectly discriminate the eligibility by predicting a subtask to be *always* eligible. Instead, we propose to extract the parent-child relationship from the subtask state labels $\mathbf{S}$. Intuitively speaking, we consider subtask n to be the ancestor of subtask m if subtask n (almost) always precedes subtask m in the videos, and assign at least one greater depth to subtask m than subtask n. Specifically, we count the (long-term) transitions between all pairs of subtasks and compute the *transition purity* as follows:

$$\text{purity}_{n\to m} = \frac{\text{\# occurences subtask } n \text{ preceds subtask } m \text{ in the video}}{\text{\# occurences subtask } n \text{ and subtask } m \text{ appears together in the video}} \tag{D}$$

We consider subtask n to be the ancestor of subtask m if the transition purity is larger than a threshold δ (*i.e.*, $\text{purity}_{n\to m} > \delta$). Intuitively, when the data has higher noise, we should use lower δ. We used $\delta = 0.96$ for ProceL and $\delta = 0.55$ for CrossTask in the experiment. Note that this is only a *necessary* condition, and it cannot guarantee to extract *all* of the parent-child relationships, especially when the precondition involves OR ($|$) relationship. Thus, we use this information only for deciding the layer of each subtask and do not use it for inferring the precondition.

After each subtask is assigned to its depth, we perform precondition inference at each depth in order of increasing depth. By definition, the subtasks at depth$= 0$ do not have any precondition. When inferring the precondition of a subtask in depth l, we use the completion of subtasks in depth $1, \dots, (l-1)$. This ensures that the edge in the subtask graph is formed from the lower depth to the higher depth, which prevents the cycle.

Recency Weighting. To further improve the graph generation, we propose to take the temporal information into account. In fact, the conventional ILP does not need to take the time step into account since it assumes eligibility data to be available at every time step. However, in our case, we are only given a single data point with positive eligibility per video clip per subtask, where incorporating the temporal information can be very helpful. Motivated by this, we propose to assign the weight to each data sample according to the *recency*; *i.e.*, we assign higher weight if a subtask has become eligible more recently. We first rewrite Equation (3) as follows:

$$\hat{f}_n = \underset{f_n}{\text{argmax}} \left\{ P\left(e_n = 1 | f_n(\mathbf{c}) = 1\right) - \alpha C\left(f_n\right) \right\} \tag{E}$$

$$\simeq \underset{f_n}{\text{argmax}} \frac{\frac{1}{|\mathcal{D}_n|}\sum_{j=1}^{|\mathcal{D}_n|} \mathbb{I}\left(f_n(\mathbf{c}^j) = 1, e_n^j = 1\right)}{\frac{1}{2^N}\sum_{\mathbf{c}} \mathbb{I}\left(f_n(\mathbf{c}) = 1\right)} - \alpha C\left(f_n\right) \tag{F}$$

$$= \underset{f_n}{\text{argmax}} \frac{\frac{1}{|\mathbf{X}_\tau|}\sum_{i=1}^{|\mathbf{X}_\tau|} \frac{1}{T}\sum_{t=1}^{T} \mathbb{I}\left(f_n(\mathbf{c}_t^i) = 1, e_{t,n}^i = 1\right)}{\frac{1}{2^N}\sum_{\mathbf{c}} \mathbb{I}\left(f_n(\mathbf{c}) = 1\right)} - \alpha C\left(f_n\right) \tag{G}$$

Then, we add the recency weight $w_{t,n}$ and modify Equation (G) as follows:

$$\hat{f}_n \simeq \underset{f_n}{\text{argmax}} \frac{\frac{1}{|\mathbf{X}_\tau|}\sum_{i=1}^{|\mathbf{X}_\tau|} \frac{1}{T}\sum_{t=1}^{T} w_{t,n}\, \mathbb{I}\left(f_n(\mathbf{c}_t^i) = 1, e_{t,n}^i = 1\right)}{\frac{1}{2^N}\sum_{\mathbf{c}} \mathbb{I}\left(f_n(\mathbf{c}) = 1\right)} - \alpha C\left(f_n\right), \tag{H}$$

where

$$w_{t,n} = \max(0.1, \lambda^{t_n - t}), \tag{I}$$

$0 < \lambda < 1$ is the discount factor, t_n is the time step when the precondition for subtask n became satisfied.

Hyperparameters. We used $\alpha = 0.2$ and $\lambda = 0.7$ in our experiments.

B.C TASK-LEVEL GRAPH GENERATION RESULTS

We present graph generation metrics for each task separately in Table B.

Table B: **Task-level Graph Generation Result in ProceL (Elhamifar and Naing, 2019).** Task label indexes (a-l) are identical to Table A.

Metric	Method	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)	Avg
Accuracy	proScript	54.69	55.56	62.50	63.54	53.57	62.50	58.93	57.14	57.14	54.17	55.00	55.21	57.50
	MSGI	50.00	55.56	53.12	55.21	48.81	52.50	58.93	55.36	51.79	58.33	50.00	61.11	54.23
	MSGI+	57.03	66.67	53.12	60.42	57.74	68.75	61.61	67.86	64.29	63.54	58.44	64.58	62.00
	MSG^2-noSSP	73.44	80.56	81.25	84.38	88.69	90.00	76.79	98.21	85.71	81.25	72.50	66.67	81.62
	MSG^2 (Ours)	87.50	79.17	90.62	84.38	83.33	85.00	67.86	100.00	87.50	83.33	82.50	66.67	**83.16**
SPOC	proScript	72.08	65.03	57.14	60.61	64.52	70.00	65.38	62.09	58.24	47.73	60.00	65.28	62.34
	MSGI	83.33	66.34	69.64	58.33	60.00	61.11	57.69	50.55	59.89	67.42	74.44	66.67	64.62
	MSGI+	77.92	70.59	69.64	62.12	64.76	82.22	69.23	84.07	72.53	65.91	73.33	76.39	72.39
	MSG^2-noSSP	87.92	92.48	91.07	93.18	86.90	90.00	82.97	93.41	89.56	90.15	77.78	84.72	88.35
	MSG^2 (Ours)	95.83	88.89	92.86	93.18	90.00	92.22	89.01	100.00	90.11	87.12	83.33	76.39	**89.91**

B.D GENERATED GRAPHS

To evaluate the performance of our graph generation approach, we hand-annotate subtask graphs for each task in the ProceL dataset, and we plot the subtask graphs for two tasks in Figure C. In addition to the hand-annotated graphs, we also plot the subtask graphs generated from MSG^2 and MSG^2 without Subtask State Prediction. When we compare the ground-truth with MSG^2, our predictions closely resemble the ground-truth graphs, compared with MSG^2 without Subtask State Prediction. These results show that our subtask state prediction improves subtask labels in the data, which leads to more accurate generated graphs.

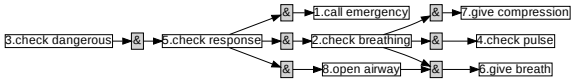
C NEXT STEP PREDICTION WITH SUBTASK GRAPH

C.A DETAILS ABOUT NEXT STEP PREDICTION WITH GRAPHS

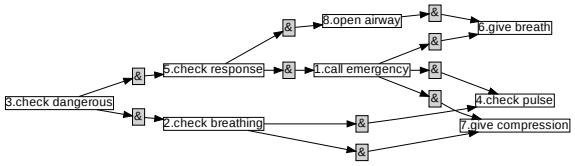
From the subtask label $\mathbf{s}_t$ and predicted subtask state $\hat{\mathbf{s}}_t$, we first obtain the subtask completion $\mathbf{c}_t$ by checking whether each subtask state is *completed*. Then, we compute the subtask eligibility $\mathbf{e}_t$ from the completion $\mathbf{c}_t$ and subtask graph G as $\mathbf{e}_t = f_G(\mathbf{c}_t)$. When predicting the next step subtask, we exploit the fact that a subtask can be completed only if 1) it is eligible and 2) it is incomplete. Thus, we compute the subtask prediction mask $\mathbf{m}_t$ as $\mathbf{m}_t = \mathbf{e}_t \odot (1 - \mathbf{c}_t)$, where $\odot$ denotes element-wise multiplication. Lastly, among the eligible and incomplete subtasks, we assign a higher probability if a subtask has been eligible more recently: $p_{t+1}[n] \propto m_t[n] \cdot \rho^{\Delta t_n^{\text{elig}}}$, where $p_t[n]$ is the probability that n-th subtask is (or will be) completed at time t, Δt_n^{elig} is the time steps elapsed since the n-th subtask has been eligible, and $0 < \rho < 1$ is the discount factor. We used $\rho = 0.9$ in the experiment.

C.B TRAINING DETAILS

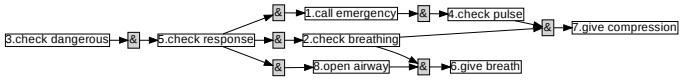
We apply our subtask graph generation method to the next subtask prediction task in Section 4.2. For both ProceL (Elhamifar and Naing, 2019) and CrossTask (Zhukov et al., 2019), we first convert all videos to 30 fps, obtain the verb phrases, and extract CLIP features, following Appendix A.a. We split 15% of the data as the test set. During training, we randomly select a subtask and feed the data up to the selected subtask with subsampling at least three frames from each subtask region but no more than 190 frames in total for all models. For evaluation, we provide all previous subtasks in the dataset and sample 3 frames per subtask in an equidistant manner (without any random sampling). All the other settings are the same as subtask state prediction.



1. Human-annotated graph

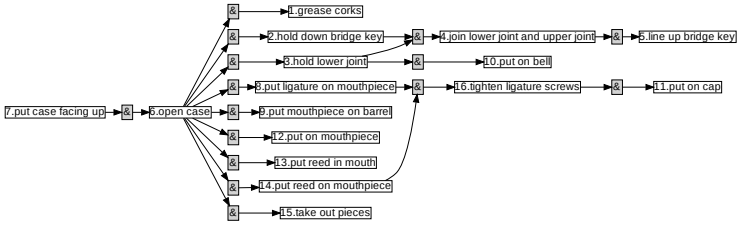


2. MSG^2 without Subtask State Prediction

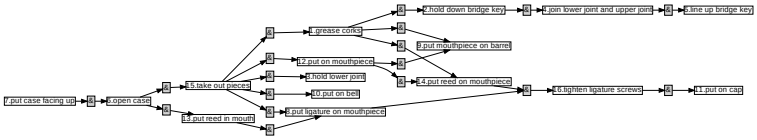


3. MSG^2

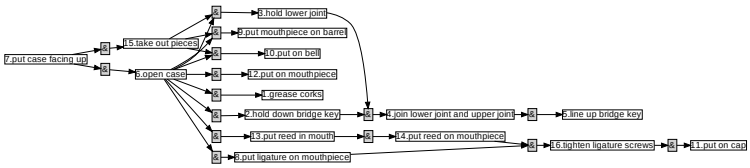
(a) Perform CPR



1. Human-annotated graph



2. MSG^2 without Subtask State Prediction



3. MSG^2

(b) Assemble Clarinet

Figure C: **Ground Truth and Generated Graphs.** We plot the subtask graphs for (a) Perform CPR and (b) Assemble Clarinet. In each subfigure, the three graphs correspond to 1. Human-annotated graph, 2. MSG^2 without Subtask State Prediction, and 3. MSG^2 from the ProceL dataset, respectively.

C.C TASK-LEVEL NEXT STEP PREDICTION RESULTS

We share task-level next step prediction results in Table C. All the method labels correspond to Table 2.

Table C: **Task-level Next Step Prediction Results.** We perform next subtask prediction on ProceL (Elhamifar and Naing, 2019) and CrossTask (Zhukov et al., 2019) datasets and measure the accuracy(%). Task label indexes (a-l) in ProceL task are the same as Table A. Task labels (i-x) in CrossTask task are (i) Add Oil to Your Car, (ii) Build Simple Floating Shelves, (iii) Change a Tire, (iv) Grill Steak, (v) Jack Up a Car, (vi) Make a Latte, (vii) Make Banana Ice Cream, (viii) Make Bread and Butter Pickles, (ix) Make French Strawberry Cake, (x) Make French Toast, (xi) Make Irish Coffee, (xii) Make Jello Shots, (xiii) Make Kerala Fish Curry, (xiv) Make Kimchi Fried Rice, (xv) Make Lemonade, (xvi) Make Meringue, (xvii) Make Pancakes and (xviii) Make Taco Salad.

	Model	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)	Avg
ProceL	STAM	23.61	23.81	21.43	46.88	25.00	63.04	57.14	34.29	4.88	24.53	27.50	6.25	29.86
	ViViT	26.39	20.24	19.05	50.00	25.00	28.26	55.36	40.00	4.88	32.08	22.50	0.00	26.98
	proScript	15.57	10.34	37.50	36.17	10.62	32.76	21.05	28.06	13.19	8.33	12.70	0.00	18.86
	MSGI	10.66	13.10	32.81	25.53	7.96	25.86	14.74	24.46	13.19	13.54	7.94	19.23	17.42
	MSGI+	22.13	36.55	32.81	26.60	25.66	37.93	20.00	34.53	16.48	27.08	7.94	30.77	26.54
	MSG^2-noSSP	31.97	56.55	43.75	73.40	51.33	51.72	46.32	69.06	50.55	48.96	30.16	26.92	48.39
	MSG^2 (Ours)	40.16	51.72	56.25	73.40	62.83	68.97	44.21	69.06	59.34	48.96	39.68	50.00	**55.38**

	Model	(i) (x)	(ii) (xi)	(iii) (xii)	(iv) (xiii)	(v) (xiv)	(vi) (xv)	(vii) (xvi)	(viii) (xvii)	(ix) (xviii)	Avg
CrossTask	STAM	30.77	58.82	48.21	38.82	27.27	26.98	34.00	21.31	15.69	40.17
		47.91	78.57	48.94	34.83	38.60	36.90	54.93	42.86	37.63	
	ViViT	34.62	43.14	49.11	34.12	63.64	34.92	60.00	13.11	11.76	41.96
		48.37	82.14	50.00	32.58	38.60	35.71	46.48	43.57	33.33	
	proScript	21.37	50.65	37.93	20.07	90.62	33.33	48.15	22.52	34.12	36.78
		33.44	45.54	33.78	26.36	33.72	38.89	42.57	24.49	24.40	
	MSGI	30.77	32.47	23.45	14.60	71.88	33.33	39.51	18.02	18.82	32.31
		22.51	33.04	36.49	33.33	45.35	32.54	34.65	34.69	26.19	
	MSGI+	30.77	32.47	22.76	14.60	71.88	33.33	39.51	27.93	18.82	32.72
		20.58	33.04	36.49	33.33	45.35	32.54	34.65	34.69	26.19	
	MSG^2-noSSP	67.52	72.73	64.83	45.99	90.62	44.74	48.15	32.43	37.65	53.39
		63.99	50.89	52.03	47.29	48.84	36.51	58.42	63.78	34.52	
	MSG^2 (Ours)	67.52	72.73	68.28	41.24	90.62	46.49	48.15	32.43	37.65	**54.42**
		63.99	55.36	57.43	48.06	48.84	46.03	58.42	64.80	31.55	