THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TASK GRAPH BASED MISTAKE DETECTION IN INSTRUCTIONAL VIDEOS

PARKER SELL
SPRING 2024

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Huijuan Xu
Professor of Computer Science and Engineering
Thesis Supervisor

John Hannan
Professor of Computer Science and Engineering
Honors Advisor

*Signatures are on file in the Schreyer Honors College.

Abstract

The primary focus of this thesis is to examine the impact of integrating a structural task graph into a visual recognition network to accurately identify and segment errors in the assembly of toy cars. We have introduced enhancements to two baseline networks that specifically encode the structural and sequential intricacies of assembly processes. These enhancements have led to state-of-the-art performance in visual-only mistake recognition task, marking a 3.7% increase in the F1-score over existing benchmarks within the Assembly101 dataset [1]. Moreover, our work pioneers in addressing the temporal mistake segmentation task which does not rely on ground truth action segments during test time. The advancements presented have yielded substantial improvements over baseline models, with a 5% increase in F1 @ 10, 3.8% at F1 @ 25, and 1.8% at F1 @ 50. Our results indicate the significant role that graph construction and attention-based mechanisms play in enhancing mistake recognition and temporal mistake segmentation tasks, setting a new precedent for future research in mistake detection.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Professor Huijuan Xu for taking me on as one of her first Master's students here at Penn State. She spent countless hours guiding me throughout every semester which has significantly shaped my research skills today. It was through Professor Xu's connections that I met my amazing lab partners, Wenchao Ma, Xiaohan Zou, Shu Zhao and Yichao Chen, who have supported me greatly through this project. Without all of them, I would not have been able to get this far.

I would also like to thank Professor Rui Zhang and Ranran Haoran Zhang who kindly invited me to work alongside them in the Alexa Prize Taskbot challenge. Through their help, I learned more than I could have hoped about modern natural language processing techniques. Working with them in this challenge significantly improved my engineering and research skills and brought many joyful moments.

Chapter 1

Introduction

In today's modern world, it is commonplace to begin a complex task such as cooking a new recipe, making a repair to a bike, or assembling furniture, by referring to a YouTube video for guidance. Instructional videos have speedup the process of learning do-it-yourself (DIY) tasks significantly. However, without a feedback system to aid the process, individuals can often stray from the original procedure and make critical mistakes that cause a faulty end result.

Recently, research and industry has focused on creating the many different components that would allow an AI assistant to aid users in correctly completing a complex procedure. A recent example is the Alexa Prize TaskBot challenge hosted by Amazon Science which focused on augmenting the conversational AI, Alexa, to guide users through complex real-world tasks [2]. In order for an AI assistant to be effectively introduced into real-world scenarios, it would need to understand a user's current progress within the procedure, relate it to the original procedure or final intended result, and communicate the necessary interventions to correct any mistakes that occur. While procedural understanding models have become adept at learning from instructional videos and text, accurately identifying errors in these videos remains a significant challenge. This work specifically focuses on creating models that improve upon the mistake detection tasks in the toy car assembly environment.

Common procedural understanding research has focused on the action recognition, anticipation, and segmentation tasks in a various number of settings including zero-shot, self-supervised, semi-supervised, and supervised. There are several SoTA techniques that generally perform well on instructional video dataset benchmarks like YouCookII [3], CrossTask [4], TastyVideosV2 [5], EPIC-Kitchens [6] and COIN [7]. A new study suggests that world knowledge stored within Large Language Models (LLMs) can be used for tasks that involve understanding procedures [8]. This has led to the exploration of techniques that combine large vision and language models. This technique takes advantage of natural language processing (NLP) approaches to use information from unstructured documents such as recipes and WikiHow articles [9]. The knowledge graph is another

source of external information that is commonly created in this setting. Compared to the LLM, a knowledge graph provides a structured, easily understandable source of information [10]. On the model architecture side, techniques that utilize attention and transformer networks have continued to perform well [11, 12, 13, 14].

A similar setting to mistake recognition is the video anomaly detection task [15, 16]. This task is particularly applicable to public surveillance settings, where the model processes untrimmed video to identify anomalous events. Typically the anomaly detection setting is framed as a one-class classification problem in which training is only conducted on non-anomalous samples. This setting requires the model to learn the distribution of the training data and identify any anomalous behavior during test time by comparing the difference between the predicted result and the label. While some mistake recognition datasets are framed under the one-class classification setting [17, 18], others like Assembly101 [1] are not since the mistake recognition setting requires a precise understanding of both the overall procedure and the fine-grained step implementations.

As mentioned in the results of [1], detecting mistakes in the assembly environment is a challenging task. One of the biggest challenges is that there are many different types of errors including procedural, extraneous, and technique errors that require a model to have a detailed step-level spatial and temporal understanding. To add to this difficulty, a correct procedure has many different viable execution orders, which requires knowledge of the necessary preconditions between each step. Another inherent property of mistake recognition datasets is the imbalance between the number of correct actions and mistake actions within a sequence. Inherently, it makes sense that, in a single assembly sequence, it only requires one mistake in a series of several other correct actions to skew the end result. Additionally, one nuanced difference, when compared to the cooking domain, is that the assembly domain more readily allows for the individual to make a correction of a previous mistake. In Assembly101 [1], this means that the procedure state can have even more variations.

Despite the rising prevalence of instruction video datasets with error steps, there are relatively few models that address the mistake recognition task specifically. For the Assembly101 dataset [1], the only follow-up model [19] is an algorithmic method that assumes ground truth text action annotations. We find this to be an unreasonable assumption since the collection process is manual and time consuming. Furthermore, even more realistic than the initial benchmark set by [1], we propose a more realistic setting called temporal mistake segmentation that removes the assumption of the ground truth action segmentations during test time. As shown in Figure 1.1, this involves classifying mistakes on a frame-level. In instructional video datasets, this is called the temporal action segmentation setting.

To address these two tasks we create a method that maps out the temporal interconnections among different procedural stages via the formulation of a task graph. This task graph is not merely a static representation; it is enhanced through the integration of each node into a graph encoder network, thereby extracting structural information. The resultant node embeddings are synthesized, providing a refined input that enriches traditional classification and segmentation networks, such as TempAgg [12] or LTContext [20]. The contributions of this thesis are as follows:

- We propose a more realistic setting of mistake recognition called temporal mistake segmentation, which drops the assumption that ground truth action segmentations are known during test time. This setting is more applicable as it can be applied to real-world data without the need for expensive frame-wise annotations.

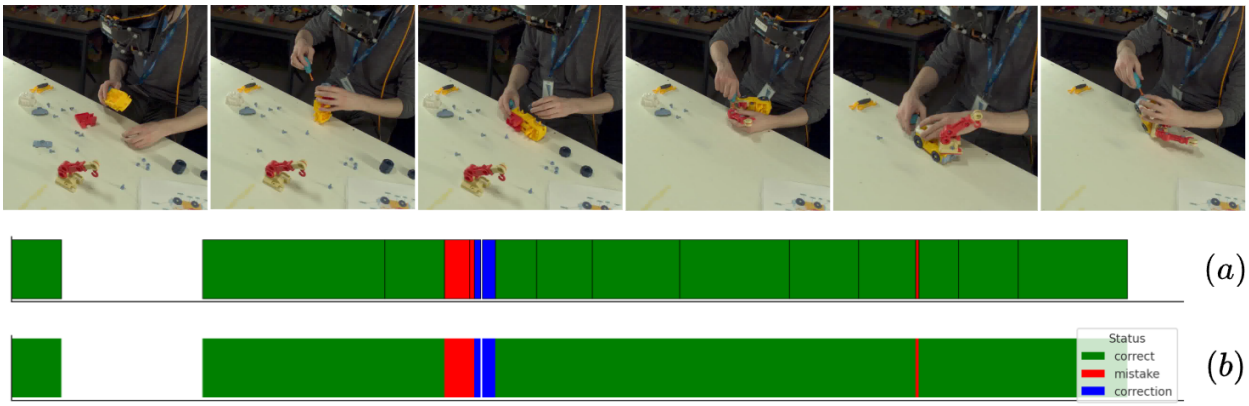


Figure 1.1: Mistake detection in the Assembly101 [1] dataset. Time series (a) demonstrates the classification of step segments into correct, mistake, or correction using ground-truth action segmentation. Time series (b) showcases temporal mistake segmentation, evaluated on the frame-level without ground-truth segmentation.

- We present an encoding network, made specifically to improve mistake recognition abilities of downstream tasks. At the heart of our network's innovation is the creation of a task graph tailored to a particular toy, which is then seamlessly merged with a temporal visual network through the use of attention. By doing so, it captures the assembly process's structural and sequential rules, thereby boosting the efficacy of the temporal model.
- Our methodology has set new benchmarks for the mistake recognition and temporal mistake segmentation tasks, surpassing existing baselines. The demonstrated performance attests the efficiency and effectiveness of our model in the precise identification of errors within assembly processes.

Chapter 2

Literature Review

Prior to selecting Assembly101 [1] as our dataset of choice, we conducted evaluations of various instructional video datasets containing error steps. Our approach, which emphasizes task graphs, identified Assembly101 as offering the greatest potential for enhancement, given its wide variety of specific tasks. Additionally, we explored techniques for building and conditioning our graph to enhance mistake recognition and temporal mistake segmentation performance. Given that mistake recognition is a relatively nascent area within action recognition, we further elaborate on tasks related to action understanding.

2.1 Instructional Video Datasets with Error Steps

Recent research has introduced many datasets that are focused on the instructional videos with error steps domain [17, 21, 18, 22, 1, 23, 24, 19]. Since this task is similar to the established video anomaly detection task, the ATA [17] dataset and the BRIO-TA [18] dataset focus on a split of the dataset that only contains anomalies during test time. However, these datasets contain simple errors at the video level rather than being localized to a specific timestamp. The complexity of the mistake recognition task is specifically more challenging and applicable when it includes errors such as procedural errors, execution errors, and backtracking to correct previously introduced errors. This is because recognizing these errors requires a much finer level of understanding of the current state of the object.

In our research, we found that there are generally three main types of errors, as shown in Table 2.1: procedural, extraneous, and technique. Procedural errors are those that occur due to an incorrect temporal alignment of steps. The most common example would be an ordering mistake in which a step is enacted before another step was completed. In mistake datasets that are synthetically created from other datasets [25, 26, 27], this also can happen in a form a swap of two video clips. The second type of error is an extraneous error where the action that occurs is inefficient or

Datasets	Procedural	Extraneous	Technique	Genuine	Corrections
BrioTA [18]	Swapping, Omission	Timing	✗	✗	✗
ATA [17]	Omission	Timing, Extra part	Balancing	✗	✗
CaptianCook4D [22]	Order, Omission	Timing	Preparation, Measurement	Both	✗
IndustReal [21]	Order, Omission	✓*	Incorrect Part Type	Both	✓*
HA-VID [23]	✓*	✓*	✓*	✓	✓*
HoloAssist [24]	✓*	✓*	✓*	✓	✓*
Assembly101 [1]	Order, Accumulation	Removal	Orientation	✓	✓

Table 2.1: Overview of relevant instructional video datasets with error steps. Examples of specific error types are shown in the table. The genuine column describes whether each dataset contains real unintentional errors made by the participants (✓) or whether the participants followed a script to force the error to occur artificially(✗). *Note: The asterisk indicates that the errors exist; however, they are not explicitly annotated

unnecessary. In practice, this is when a part is unnecessarily removed [19], an extra part is added to the object [17], or an extraneous amount of time is wasted idle [17, 18, 22]. Out of all three error types, the technique error often requires the most fine-grained understanding. For instance, in the case of [22], this can be as subtle as incorrectly stirring some corn and having a few kernels fall out of the bowl. In assembly datasets, a technique error is more commonly related to the orientation, type, and balance of a part.

Another noteworthy characteristic of the aforementioned error datasets is whether they contain realistic unintentional errors or if the participants were instructed to feign an error. We describe these as genuine errors as shown in Table 2.1. The genuine setting is more useful in training a real-world model since the model needs to recognize more subtle mistakes. There are a few datasets [25, 26, 27] which have synthetically created errors by modifying existing error-free datasets. We found this to be orthogonal to our direction of research, because the end setting isn't realistic. However, following the pre-training paradigm and transferring the learned knowledge from synthetic datasets to realistic benchmarks, as done in [21], could be a potential direction for future research.

As shown in Table 2.1, only half of the error datasets contain corrections to past errors. The practice of incorporating corrections is largely relevant to assembly environments. In cooking scenarios, rectifying a mistake typically necessitates starting over, as it is often impractical to correct the error without doing so. However, this setting requires more advanced state understanding since the current state must be compared to past actions more closely.

Our research focuses on the more realistic task of visual-only mistake recognition initially proposed in the Assembly101 dataset [1]. This dataset contains annotated correction steps, contains only genuine mistakes, has the highest number of different objects, and has annotated fine grained actions. IndustReal [21] operates in the object detection space by proposing a new task called procedure step recognition and focusing on understanding the spatial constraints of a single object. HoloAssist [24] focuses on the collaboration between an instructor which interactively guides the participant through any mistakes and questions. CaptainCook4D [22] is the only error-focused dataset to exist in the cooking domain which innately contains many complex fine errors. In the end, the Assembly101 dataset is more applicable to our task graph-focused approach, as the diverse number of objects has the highest potential to benefit from enhanced structural information.

Despite recent interest in mistake detection, most of these works only provide baselines like [12] without more sophisticated techniques. The exception is [19] which proposes a rule-based algorithm to construct a spatial and temporal graph from textual annotations of the ground truth state.

However, this setting is unrealistic due to the assumptions of a ground truth text oracle during test time. Additionally, the work is unable to adequately detect different technique errors such as part misorientation since it doesn't utilize the rich visual modality. Our work focuses on improving baseline models like [12, 20] to perform well in the supervised visual-only setting.

2.2 Graph Construction Methods

There are several approaches to creating structured task graphs from instructional videos, including [28, 29, 11]. The motivation is that by distilling instructional knowledge into a more succinct structured format, a network has increased performance on step detection or next subtask prediction.

One of the advantages of instructional videos, especially with the rise of LLMs, is that they are often a narrated demonstration of the task. This narration can act as a self-supervised way to create a link between the video and external text sources such as Wikihow [9] for better procedural understanding as in [30, 11]. In [11], they create a procedural knowledge graph, but it is heavily dependent on the linked external text sources. However, error datasets are more frequently user executions, where users simply enact the procedure (with or without instructions). Since the user is not demonstrating the procedure, there is no audio that can be used to find a link between actions and an external procedural recipe. Recently, [31] has explored using visual instruction manuals and aligning them with video demonstrations through the IKEA-in-the-Wild (IAW) dataset. However, in the case of Assembly101 [1], there are no visual instruction manuals requiring all knowledge to be derived solely from the video and action annotations.

Currently, graph generation works are divided among those that use weighted [32, 33] and unweighted edge weights [19, 28, 34, 29, 35, 36] as a way to model the temporal relationship between different action subtasks. Weighted edges often show the probability of one task having a relation between another and a highest probability path can be computed to determine the most likely candidate path as demonstrated in [32]. Distinctly, in an unweighted graph, the edge defines a necessary precondition between two actions. Since detecting an ordering mistake in assembly requires knowledge of whether an ordering constraint has been violated, the unweighted graph is the most intuitive representation.

Some previous approaches [35, 36] focus on the supervised graph generation problem, in which a manually curated graph is provided as annotation and a large language model (LLM) is utilized for generation. The more ideal setting is the unsupervised case as this type of annotation is not available for most instructional videos. As demonstrated in [28, 29], by using the temporal action annotations present in most instructional videos as input, a task graph can be constructed through an algorithm called Inductive Logical Programming (ILP). This algorithm is a specific type of learning that specializes in using a limited amount of training data to output a set of logical rules that generalize the training data [37]. In our case, the goal is to acquire a temporal ordering between all the correct actions in each assembled toy. Given the temporal action annotations for each toy, the ILP algorithm outputs a set of logical preconditions between steps that can be used to construct the task graph.

In contrast to a simulated environment [34], which has complete information about which actions are eligible to be completed next, in the real world scenario the temporal action annotations are noisy and sparse. Therefore, to obtain the logical preconditions, [28] boils this algorithm down

to greedy edge selection based on the typical ordering of actions. As these works operate on data that does not contain errors, they can only assume the positive eligibility of a task. In mistake recognition, with the inclusion of ordering mistakes, we modify the algorithm proposed by [28] to more closely fit this domain difference as detailed in Sec 3.1.2.

2.3 Graph Conditioning Methods

The methods for conditioning a model on a graph can be divided into two categories: those derived from machine learning techniques and those from programming logic. The machine learning based method focuses on finding a representation of the graph in the form of an embedding. Works like [38, 39, 28] use a graph nueral network (GNN) as an encoder. In [38], they use the graph for increased performance in the closed question and answering domain. They further update the GNN output using cross-attention pooling between each candidate graph and the context provided in the question. We follow a similar approach to attend the graph representation to the frames of the current video.

Recently, in the neurosymbolic community, they have found that logical rules [40] can directly benefit data-driven models that focus on analyzing images and graph data. Recently, models have extended this to the action understanding temporal domain [41, 42]. These works create a context-free grammar by iterating through the training set text annotations and extracting relevant rules. Then these rules are used to guide the output of the learning based model. Specifically, DLT [41] employs a neurosymbolic evaluator to introduce a secondary loss into a temporal action segmentation model, compelling the model's output to adhere to the extracted rules.

Another example is [32] which replaces model output that has a low confidence score with more confident output from the task graph. For instance, in the keystep recognition task, if the model is unsure about the predictions for steps 3 and 4, they use Dijkstra's algorithm to find the most likely path from steps 2 and 5. In this way, they can condition on the structured task graph to edit the predicted output.

In this paper, we focus on the learning based method, but future work could consider utilizing programming logic as it explicitly represents the rules extracted from the textual annotations.

2.4 Action Understanding Tasks

The mistake recognition task is a specific subset of the action recognition or action classification task. We propose a new setting where ground truth segment boundaries are not provided which makes our work similar to the temporal action segmentation task or the procedure segmentation task. Therefore, we explore some of the models in these tasks as the model space of the error detection task is sparse.

Video action classification or action recognition that is an established task [43, 44, 45] with several established pre-trained and end-to-end models [43, 46, 47, 48, 49]. The primary difference between the action recognition task and mistake recognition is that there are usually more classes. For example, the action recognition task in Assembly101 [1] has up to 1,380 different possible classes. However, in mistake recognition, there are typically only two or three classes consisting of mistake, correct, and correction (correction is not always explicitly included). In some datasets,

there is a fine mistake recognition task where the number of specific errors can range from 6 to around 20. In most fine action recognition settings, the snippets consist of only a few seconds of the video compared to the coarse action recognition settings which range from a few seconds to several minutes. This is different from the aforementioned fine mistake recognition task as the clip length in this setting is not shortened.

A similar setting to our temporal mistake segmentation task is the supervised temporal action segmentation task [50]. In both of these tasks, the ground truth segment boundaries are not used during test time. In this setting, there a few notable models based on representation networks [12, 51], temporal convolutional networks [52, 53, 20], refining proposals [54] and transformers [55, 56, 57]. There is also a similar task, where the procedural action boundaries are more important than the action labels called the video procedure segmentation proposed in [58]. In our model, we enhance a relevant fully supervised temporal action recognition model called LTContext [20] with our graph embedding. This model offers an efficient way to apply attention over the entire video through windowed attention and long-context attention strategies.

Chapter 3

Model

Our method starts by creating a task graph that captures the temporal dependencies between various steps in the execution process. We then gather structural information about each graph by incorporating each node into a graph encoder network. Once we obtain updated node embeddings, we aggregate the node embeddings of each graph and incorporate them into a baseline classification network such as TempAgg [12] or LTContext [20]. We describe any changes made to these networks to better utilize our graph embeddings in the mistake recognition and temporal mistake segmentation task.

3.1 Task Graph Generation

Inspired by the use of the ILP algorithm to generate a task graph based on the temporal action annotations of an instructional video [28, 34], we have similarly adapted this algorithm to fit our domain. There are a few primary differences in the task sequences present in the assembly domain compared to the ProceL dataset used in [28]. Specifically, the sequences are untrimmed, contain mistakes, contain backtracking to fix mistakes(corrections), contain cycles due to in-progress tasks, and has fewer global temporal ordering constraints. As a result, to utilize the advantages of ILP we conduct some pre-processing and improvements to the algorithm.

3.1.1 Pre-processing

Our temporal knowledge graph focuses on representing the correct ordering constraints in a directed acyclic graph (DAG). In the graph, every node represents an action label, which, within the context of Assembly101, is defined as a triplet consisting of *(verb, this, that)*. Additionally, each directed edge (u, v) indicates that node u must precede node v as a necessary precondition. Therefore, we first remove cycles and all corrections in the training transcripts by taking only the

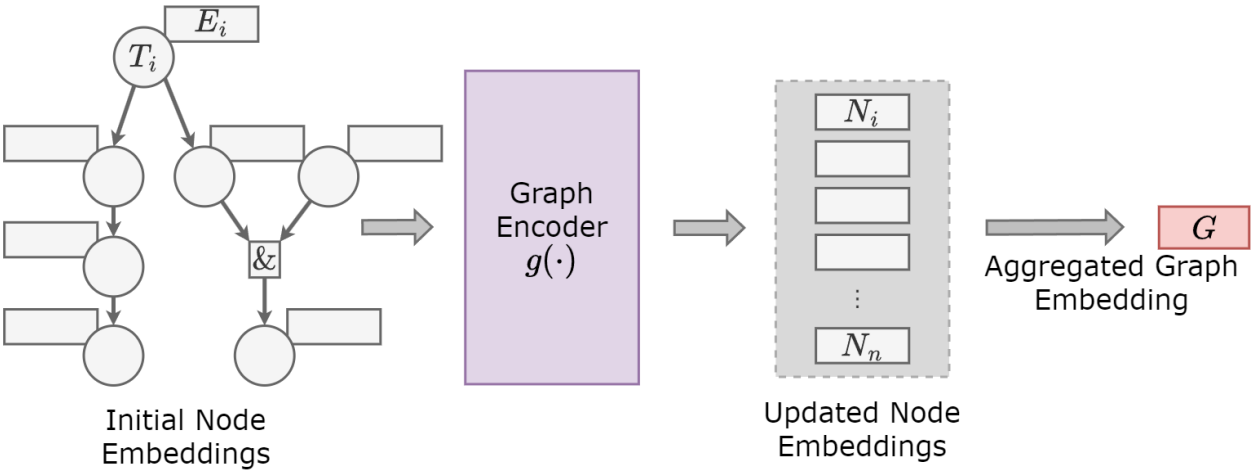


Figure 3.1: Task graph embedding process. Converts textual labels T_i of action steps into a final graph embedding G. Each text label T_i is initialized to embeddings E_i, followed by node embedding updates via graph encoder $g(\cdot)$ to N_i. Aggregation of N_i embeddings yields final graph embedding G.

last occurrence of each action. This is a reasonable assumption, especially for cycles that occur due to in-progress actions (i.e. adding wheels to a chassis), because the last occurrence symbolizes the final completion of that state. Although our training examples incorporate mistake actions, we deliberately omit these actions from potential node selection because our graph does not represent incorrect temporal relationships.

3.1.2 ILP Algorithm Improvement

After pre-processing, we follow the similar two stage approach of [28] in which the candidate steps are first sorted into layers to prevent cycles and then edge functions f_n are iteratively selected based on a greedy objective J_{prec}. Since our real-world observations are similarly sparse (only one data point per action per video), we optimize precision using the following equation:

$$J_{\text{prec}} = \frac{\mathbb{E}_{(c,e_n)}[e_n \cdot f_n(c)]}{\mathbb{E}_c[f_n(c)]}$$

In this equation, c represents the binary completion vector, where a value of 1 indicates the subtask with that id was completed prior to the current step in the sequence. The eligibility vector e_n is a one-hot encoded vector that represents the current state in the sequence. Since we have mistakes in our training examples, we represent the eligibility value e_n for these mistake actions as a negative weight of -1. Our modification just switches the sign to decrease the numerator of J_{prec} when the edge function includes f_n. We qualitatively find that procedural ordering errors are the most relevant in temporal understanding, and so we only include these error types.

Finally, compared to the cooking domain, assembly tasks exhibit fewer global ordering constraints. In other words, there are multiple potential starting points when beginning to assemble a toy. Despite the possibility for actions to be carried out in parallel, when ordering constraints do arise, they tend to be more subtle and usually occur between parts that are closely connected.

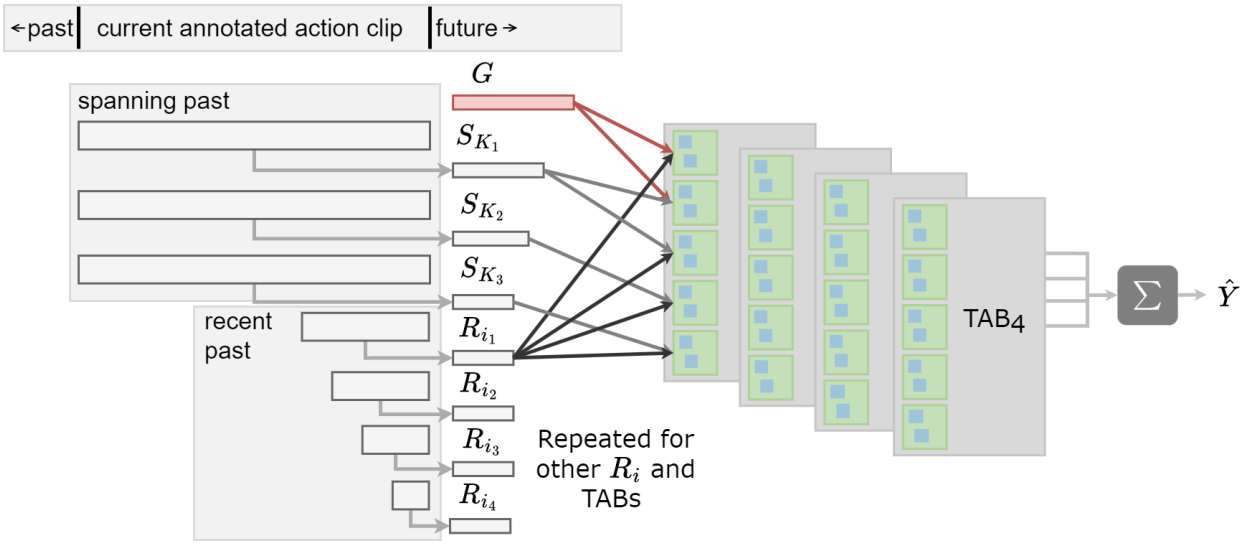


Figure 3.2: Improved mistake recognition model with task graph embedding G. Incorporates two additional coupling blocks in each TAB, facilitating comparison of graph embedding G with spanning and recent vectors, adapted from TempAgg [12]

Therefore, by using the part-to-part annotations provided by [19], we are able to construct a spatial graph and remove the temporal ordering constraints that occur between parts not connected to each other spatially.

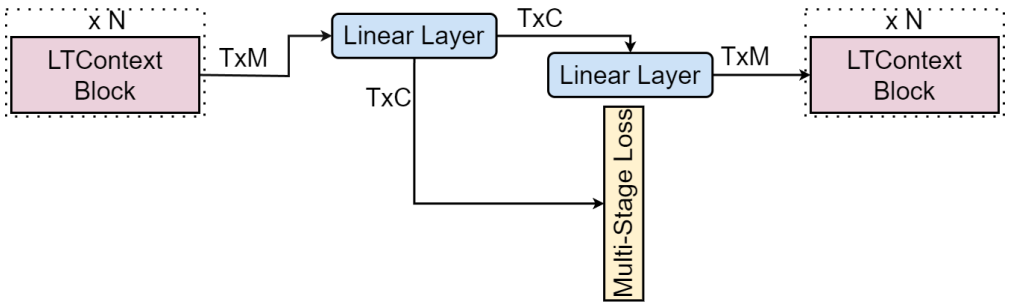
3.2 Task Graph Embedding

As shown in Figure 3.1, once we have used ILP to obtain a graph for each toy, we need to extract structural knowledge about each graph in the form of an embedding G. We start by initializing each node's embedding to a vector E_i based on the textual label of each node. Then we utilize a graph encoder network $g(\cdot)$ to obtain updated node embeddings. Finally, we explore different methods of aggregating the node embeddings N_i for each graph into one final graph embedding G.

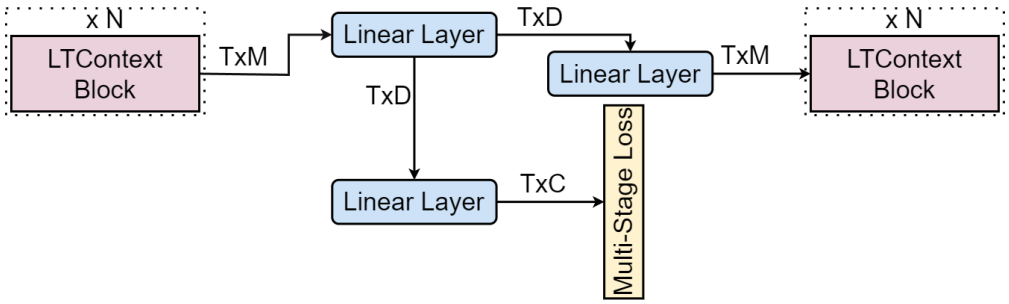
We experiment with the well-known embedding networks BERT [59] and CLIP [60] to initialize the node embeddings E_i for each node n_i in the graph. Similar to [38], we utilize the GAT [61] network, which relies on attention to further update connected node embeddings.

3.3 Mistake Recognition Model

We utilize TempAgg [12] which is an attention based aggregation network that relies on non-local blocks [47]. We add two coupling blocks to the originally configured temporal aggregation block (TAB) so that each graph embedding can be combined with the spanning and recent visual features. This is similar to the cross-attention pooling approach of [38] since each coupling block contains non-local blocks to compare the graph embedding to the visual features. We make ablations described in Sec 4.3 to show that Figure 3.2 is the best method of incorporating the graph embedding within the TempAgg model.



(a) Original implementation from LTContext [20]



(b) Our modification to the connection between LTContext blocks

Figure 3.3: Dimensionality preservation in temporal mistake segmentation model. Unlike the original (a), our modification (b) prevents dimension reduction to C between LTContext blocks. For temporal mistake segmentation with D=26 and C=3, this ensures minimal information loss in later stages.

3.4 Temporal Mistake Segmentation Model

We chose LTContext as our baseline model since it is the state-of-the-art model in the supervised temporal action segmentation task for Assembly101. Additionally, this model is less computationally expensive than transformer alternatives like [55], since it does not attend over every frame. Instead it utilizes windowed and long-context attention. Since there is no baseline for this task, the baseline is the original LTContext model where the number of classes C is set to 3 for each class in Assembly101: mistake, correct, and correction. LTContext was originally tested on the temporal action segmentation tasks within the Assembly101 and Breakfast datasets, which contain 202 and 48 distinct classes, respectively. During our experiments, we found that this difference between the number of classes created a bottleneck. Therefore, we modified the original connection between the LTContext blocks to better fit the reduced number of classes as shown in Figure 3.3. Empirically, we find this new dimension D performs highest when set to 26.

To incorporate our graph embedding, we follow a similar strategy to the graph neural prompting method in [38]. However, since we are using a transformer, we cannot insert this as a special token. Instead, as shown in Figure 3.4, we simply duplicate the single graph embedding to be the same length as the input frame sequence and concatenate the two vectors together. This ensures that each frame embedding has information about the graph.

We empirically find that cross-attention pooling is necessary in this task to better fuse the graph embedding vector and the visual frame features. Therefore, we further update the node

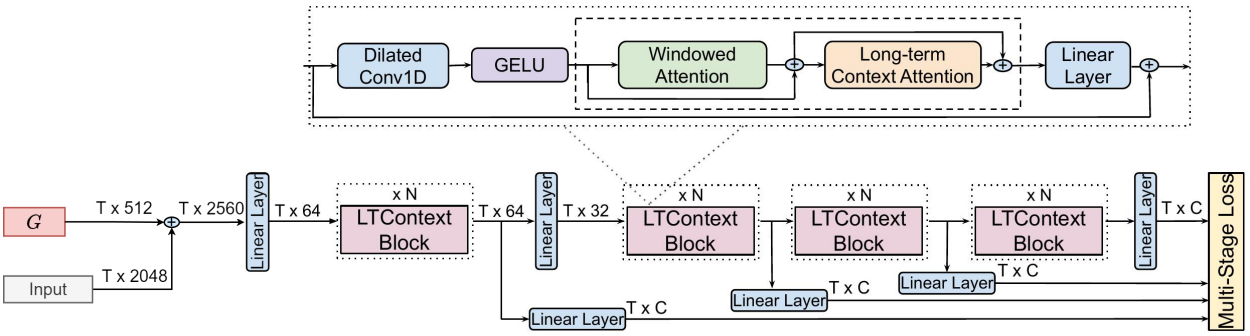


Figure 3.4: Improved temporal mistake segmentation model with task graph embedding G. Adapted from LTContext [20], the graph embedding is duplicated for each frame and concatenated.

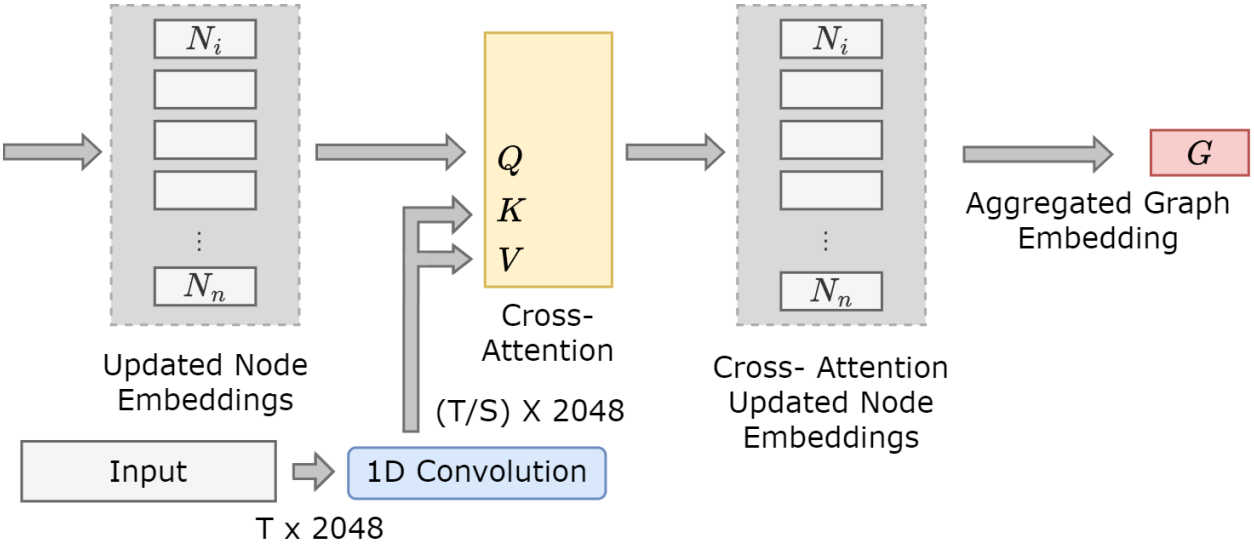


Figure 3.5: Integration of visual frame data into graph embedding G through cross-attention. Enhances node embeddings, initially created by the graph encoder $g(\cdot)$. To manage the disparity where $T >> n$, a 1D convolutional layer downsamples frames from T to (T/S).

embeddings outputted from the graph encoder as shown in Figure 3.5. The ablations performed in Sec 4.4 shows that this is a necessary step in the model. Additionally, since the average number of nodes in each graph is much smaller than the number of input frames, we use an one dimensional convolution layer with stride S set to 250 to reduce the dimensionality of the input frames.

Chapter 4

Experiments and Results

We report our experimental assumptions, our metrics, and our intermediary model results used to reach our final model on the main tasks of mistake recognition and temporal mistake segmentation. For mistake recognition, we see a 3.7% improvement in F1-score through our final model. In temporal mistake segmentation, our enhancements considerably improve the baseline for each F1-score threshold. Finally, we show some examples of our generated task graphs and present our qualitative analysis of the intuition behind this construction.

4.1 Experimental Setup

In the graph construction process, we strictly utilize videos from the training set to assemble the graphs, thereby maintaining the separation between the graph information and the testing split. We leave the settings for the parameters in the greedy objective in the ILP algorithm similar to those reported in [28]. For the graph encoder, GAT, we ablate the number of layers, number of heads per layer, and number of features per layer. Given the highly imbalanced nature of our data, for both settings we use Balanced Softmax (BS) [62] to predict more accurately across all classes.

In the ground truth setting, we follow the hyperparameters from [1] to recreate the baseline TempAgg model on our data split. We cannot compare with the results shown in this paper [1] since this split is not publicly released. However, the follow-up paper [19] releases fine error annotations, part-to-part annotations, and uses an average of five random results as their split. To ensure accurate generalization on the toy level, we follow the split [19] where one action sequence from each toy is randomly sampled to be in the test set. Aside from our modifications to integrate the graph embeddings into the TempAgg network, we maintain the same recent and spanning dimensions and sampling parameters as used in [1]. One small architecture change that is not depicted in Figure 3.2 is that a linear layer is used to match the length of the inputs to the coupling blocks.

	Mistake		Correction		Correct		Accuracy	F1-score
	R	P	R	P	R	P		
TempAgg [12]	40.5	42.9	46.7	27.2	87.5	90.7	76.6	55.0
TempAgg w only BERT	38.4	35.9	42.7	31.4	88.1	88.0	75.2	56.3
TempAgg Couple w all S_K	30.8	2.0	48.1	12.6	75.3	99.0*	74.1	36.5
TempAgg CA w first S_K	39.4	37.6	46.6	33.0	87.0	91.2	76.4	55.4
TempAgg init w CLIP	41.0	25.3	44.2	35.2	86.5	**93.3**	75.4	55.8
TempAgg Couple w first S_K	**41.4**	**47.7**	**51.4**	**36.9**	**89.0**	88.8	**76.9**	**58.7**

Table 4.1: Main results for mistake recognition task with ground truth action segments for coarse-grained mistake recognition on Assembly101. TG denotes task-graph embeddings from the graph encoder mentioned in Section 3.2. R stands for the recall metric and P stands for the precision metric. *Note: This is due to naive solution of predicting correct most of the time.

In the segmentation setting, we again maintain similar hyperparameters to the model described in [20]. For consistency, we use the same split from the mistake recognition setting.

4.2 Evaluation Metrics

For the mistake recognition task, we focus on the typical recall and precision metrics. Additionally we report the overall accuracy independent of the class. However, this metric does not adequately describe performance when the data is imbalanced. Instead the macro F1-score is a more important metric, because accuracy can be too focused on the dominant class correct.

On the segmentation task where ground truth segments are not used, we utilize the standard metrics for temporal action segmentation [50]: edit score, mean over frames (MoF), and F1-score thresholded over intersection levels of 10%, 25%, and 50%. These metrics focus on evaluating the predictions on a frame-level. Since we are looking for increased model performance over all three classes, we use the macro F1-score which uses an average of each classes individual precision and recall scores to calculate the overall F1-score. Since the dataset is imbalanced, prevents supporting a model that primarily predicts one class. Comparatively, as shown in Table 4.5 we see that the micro F1-score highlights a model whose F1-score is highest for the most dominant class. Similarly to the mistake recognition task, MoF or accuracy is also heavily weighted towards the dominant class. Therefore, even though we include MoF, we find that this metric rewards a naive prediction where correct is predicted for all frames.

4.3 Mistake Recognition Results

As shown in Table 4.1, we report a 3.7% improvement in F1-score when we add our proposed task graph model to the TempAgg model in the mistake recognition task. Importantly, we see that over each class's recall and precision, the task graph shows a noticeable improvement. The one exception where our metric is below the TempAgg baseline is the precision metric for the correct

Number of layers	Mistake R	Mistake P	Correction R	Correction P	Correct R	Correct P	Accuracy	F1-score
1	38.7	6.1	46.7	40.7	80.6	**98.1**	76.7	47.5
2	37.3	44.7	49.0	**45.6**	89.4	85.9	75.0	58.5
4	36.1	46.7	48.4	42.7	**90.1**	85.1	74.5	57.9
3	**41.4**	**47.7**	**51.4**	36.9	89.0	88.8	**76.9**	**58.7**

Table 4.2: Ablation results on the number of layers in the GAT GNN for the TempAgg Couple w first S_K model in the mistake recognition task. R stands for the recall metric and P stands for the precision metric. We selected 3 layers since it is the most balanced model with the highest F1-score.

class. However, we find this difference to be insignificant, since the F1-score for the baseline is only 0.17% higher than our final model as shown in Figure 3.2.

In Table 4.1, we also show our intermediate models to validate our choices for our final model (*TempAgg Couple w first* S_K). *TempAgg w only BERT* demonstrates the results from removing the graph encoder $g(\cdot)$. The worst result in the table, *TempAgg Couple w all* S_K, demonstrates how adding coupling blocks to attend to each spanning vector negatively impacts performance. *TempAgg CA w first* S_K shows how adding a cross-attention network like 3.5 into TempAgg is unnecessary since our integration inherently includes attention between the visual frames and the graph encoding. To test if the final model *TempAgg Couple w first* S_K can be improved by changing the BERT [59] initial embedding to CLIP [60], we include the model *TempAgg init w CLIP*. Our final results show that *TempAgg Couple w first* S_K demonstrate the highest results. This model integrates the graph embedding G, initially set with BERT embeddings, with the first spanning vector and every recent vector. It does not employ cross-attention (CA) for updating the graph embedding, which is a step shown to be essential for the temporal mistake segmentation task.

In Table 4.2, we demonstrate validation for our choice of three layers using the mistake recognition task. For each layer, we report the best results we found from ablating the models number of heads and features per layer. We found similar results to be true for the temporal mistake segmentation task.

4.4 Temporal Mistake Segmentation Results

In the temporal mistake recognition task, we report our overall average results in Table 4.3 and our per class results for micro F1-score in Table 4.4. Our proposed enhancements improve from the baseline LTContext by 5% in F1 @ 10, by 3.8% in F1 @ 25, and 1.8% in F1 @ 50. We provide our intermediate models to show how we constructed our final model. The *LTContext w BS* is just the original baseline with the Balanced Softmax [62] applied for the loss function. By itself, we can see it show improvement over the baseline in the per-class F1-scores. In *LTContext w only BERT*, we remove encoder $g(\cdot)$ or GAT [61], to show the improvement we have over simply max-pooling the initial BERT [59] embeddings to get G. Next, we add GAT back in, but don't include the cross attention modification shown in Figure 3.5. We see that for the LTContext model

	Edit score	F1@10	F1@25	F1@50	MoF
LTContext [20]	56.9	28.3	27.4	21.9	**90.6**
LTContext w BS	55.8	30.2	27.4	21.3	87.7
LTContext w only BERT	57.1	30.4	28.3	22.3	88.4
LTContext w TG wo CA	50.2	28.6	26.4	21.7	86.1
LTContext w TG D=3	52.2	30.4	27.8	21.3	86.2
LTContext w TG	**57.8**	**33.3**	**31.2**	**23.7**	85.9

Table 4.3: Main results for the temporal mistake segmentation task. TG denotes task-graph embeddings from the graph encoder mentioned in Section 3.2. BS stands for Balanced Softmax [62]. CA stands for cross-attention module. D is the intermediary dimension between blocks. MoF stands for Mean over Frames.

	Mistake F1			Correction F1			Correct F1		
	10	25	50	10	25	50	10	25	50
LTContext [20]	9.6	8.4	3.2	1.3	0.9	0.4	73.7	72.6	62.1
LTContext w BS	11.2	8.0	3.2	6.0	3.5	**2.7**	72.5	70.3	57.8
LTContext w only BERT	10.6	7.0	4.1	5.2	3.3	1.7	74.9	74.1	60.8
LTContext w TG wo CA	5.5	3.5	3.0	4.5	2.1	0.9	75.3	73.3	61.1
LTContext w TG D=3	6.8	5.0	3.0	**8.5**	3.6	0.2	74.6	73.9	60.7
LTContext w TG	**14.5**	**12.1**	**6.8**	7.7	**5.4**	1.4	**76.6**	**75.4**	**62.7**

Table 4.4: Per class F1-score results for the temporal mistake segmentation task. TG denotes task-graph embeddings from the graph encoder mentioned in Section 3.2. CA stands for cross-attention module. D is the intermediary dimension between blocks. MoF stands for Mean over Frames.

it is necessary to include this module. Finally, we add in cross attention, but leave D at 3 as was originally designed in the baseline LTContext. We find that the final model *LTContext w TG* shows the highest performance overall. The exceptions occur among MoF which doesn't account for our class imbalance and F1 Correction scores in which the low sample size causes higher variance.

4.5 Qualitative Analysis of Task Graph Generation

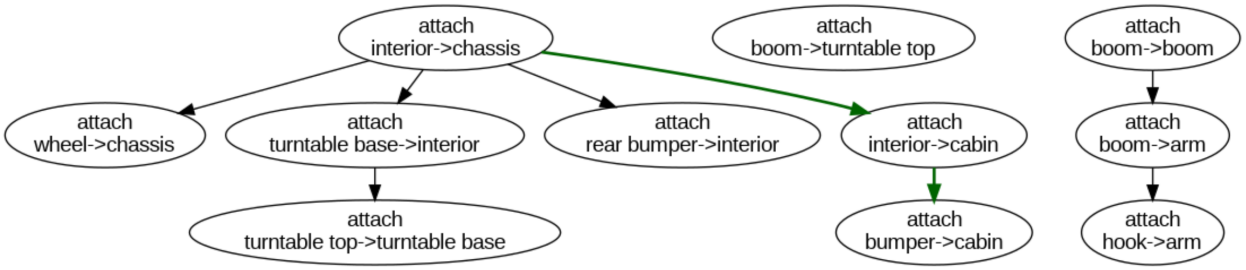
Figures 4.1 and 4.2 show examples of task graphs generated from the Assembly101 dataset. In each of the displayed graphs, there are a few edges that are highlighted green. These show the edges that the ILP algorithm correctly defines as necessary preconditions. We determined this by manually evaluating all video executions for that toy and looking at the mistakes made. The spatial graphs are constructed using part-to-part annotations by [19] and are only shown for illustration purposes. Only the task graphs constructed using ILP are embedded in our network. By examining the mistake annotations and spatial graph, one can understand different necessary preconditions.

	Mistake F1			Correction F1			Correct F1			Micro F1			Macro F1		
	10	25	50	10	25	50	10	25	50	10	25	50	10	25	50
[20]	9.6	8.4	3.2	1.3	0.9	0.4	73.7	72.6	62.1	**57.5**	**56.4**	**48.9**	30.2	27.4	21.3
Ours	**14.5**	**12.1**	**6.8**	**7.7**	**5.4**	**1.4**	**76.6**	**75.4**	**62.7**	57.1	55.1	46.1	**33.3**	**31.2**	**23.7**

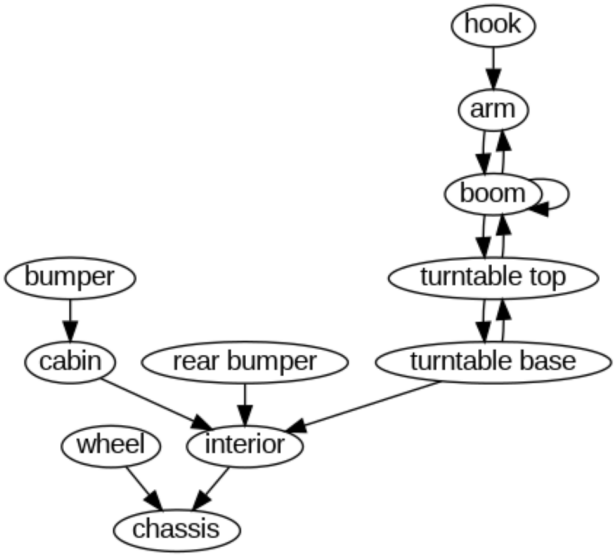
Table 4.5: Empirical evidence for the choice of macro F1-score over micro F1-score. Micro F1-score is highest for the baseline model despite the baseline performing worse on all per class F1-scores. Macro F1-score is highest for the *LTContext w TG* model which performs highest on all per class F1-scores.

For example, in Figure 4.1 the action "attach bumper to cabin" was annotated as a mistake when completed before "attach interior to cabin". From this mistake, we can determine that "attach interior to cabin" must occur before "attach bumper to cabin".

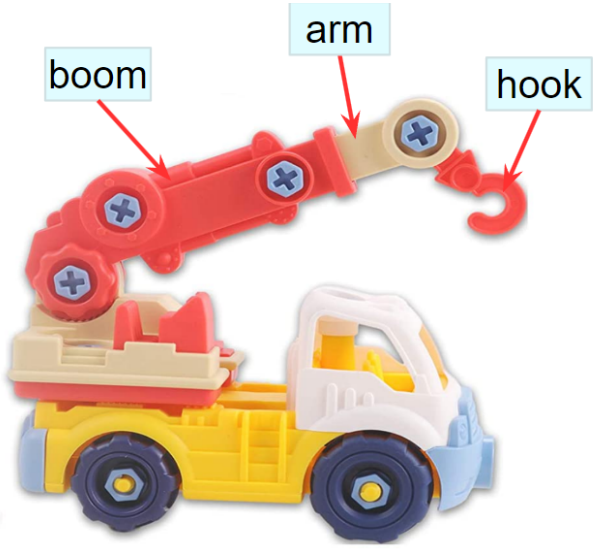
As previously discussed in Sec 3.1.2, the assembly process often allows for parallel actions to occur. By observing Figure 4.2, we can see that there are three actions that can be started. This property was our motivation to remove edges between nodes that were not spatially connected. This helps prevent the ILP algorithm from unnecessarily creating edges between parts which are not necessarily preconditions and instead are just commonly assembled in that order.



(a) Task graph

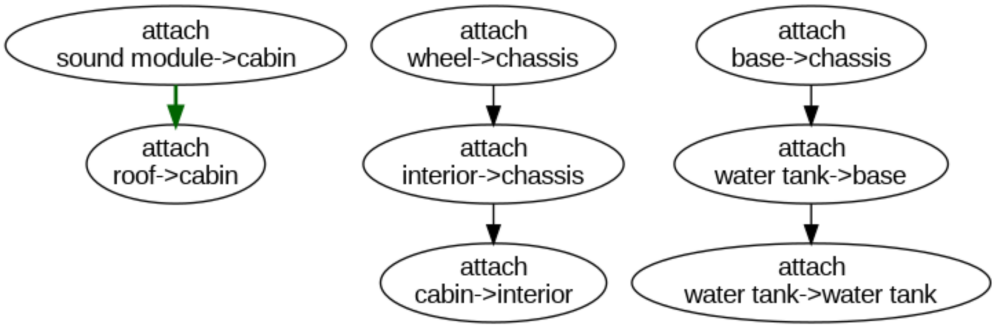


(b) Spatial graph

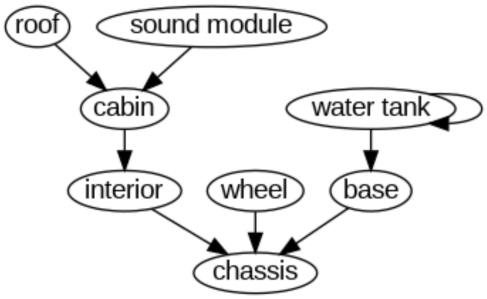


(c) Picture of the toy crane (b05b)

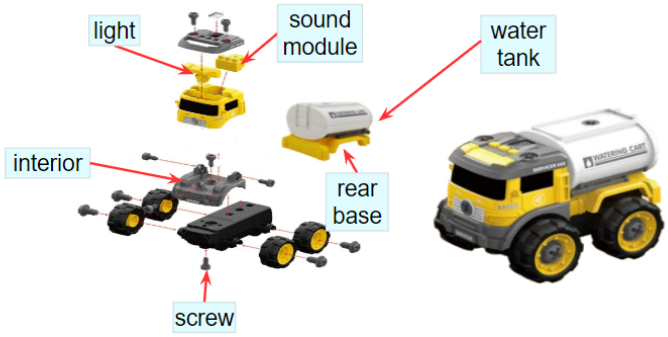
Figure 4.1: Task graph visualization for Assembly101 dataset generated by ILP algorithm (object b05b). Green edges in the task graph highlight correctly identified preconditions that were validated via counterexamples in mistake sequences. Spatial graphs, built from part-to-part annotations by [19], serve for illustrative purposes only.



(a) Task graph



(b) Spatial graph



(c) Picture of the toy water tanker (c06d)

Figure 4.2: Task graph visualization for Assembly101 dataset generated by ILP algorithm (object c06d). Green edges in the task graph highlight correctly identified preconditions that were validated via counterexamples in mistake sequences. Spatial graphs, built from part-to-part annotations by [19], serve for illustrative purposes only.

Chapter 5

Conclusion

The main objective of this work was to determine how well a structural task graph contributes to the mistake recognition task. Our proposed modifications to the TempAgg network effectively encode the structural and sequential rules of the assembly process and achieve state-of-the-art performance in the visual-only mistake recognition task with a 3.7% increase in F1-score. We modify the existing baselines to fully integrate our generated graph into each baseline. Additionally, we find our model is the first to predict on the more realistic and challenging temporal mistake segmentation setting of Assembly101 where ground truth action segments are not provided during test time. Our proposed enhancements improve from the baseline by 5% in F1 @ 10, by 3.8% in F1 @ 25, and 1.8% in F1 @ 50. These improvements on the mistake recognition and temporal mistake segmentation tasks demonstrate the effectiveness of our graph construction and embedding methods.

Future research aimed at employing structural task graphs for mistake detection presents a wide array of possibilities for ongoing exploration and advancements. The utilization of LLMs to inject world knowledge into task graphs presents a promising avenue for expanding their generalization capabilities. Similarly, the investigation of rule-based neurosymbolic evaluators as an alternative form of graph conditioning opens new pathways for these tasks. The experimentation of more advanced graph encoders techniques involving transformers [63] and more complex positional encodings could offer further improvement. Furthermore, the exploration of diverse datasets such as IndustReal [21], HoloAssist [24], and CaptainCook4D [22] offers different arenas in which to apply the structural task graph. These research directions can possibly create richer, more adaptable task graph methodologies that can further enhance mistake detection and generalization across various domains.

Bibliography

[1] F. Sener, D. Chatterjee, D. Shelepov, K. He, D. Singhania, R. Wang, and A. Yao. Assembly101: A large-scale multi-view video dataset for understanding procedural activities. *CVPR 2022*, 2022.

[2] Eugene Agichtein, Michael Johnston, Anna Gottardi, Lavina Vaz, Cris Flagg, Yao Lu, Shaohua Liu, Sattvik Sahai, Giuseppe Castellucci, Jason Ingyu Choi, Prasoon Goyal, Di Jin, Saar Kuzi, Nikhita Vedula, Lucy Hu, Samyuth Sagi, Luke Dai, Hangjie Shi, Zhejia Yang, Desheng Zhang, Chao Zhang, Daniel Pressel, Heather Rocker, Leslie Ball, Osman Ipek, Kate Bland, James Jeun, Oleg Rokhlenko, Akshaya Iyengar, Arindam Mandal, Yoelle Maarek, and Reza Ghanadan. Advancing conversational task assistance: the second alexa prize taskbot challenge. In *Alexa Prize TaskBot Challenge 2 Proceedings*, 2023.

[3] Luowei Zhou, Chenliang Xu, and Jason Corso. Towards automatic learning of procedures from web instructional videos. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[4] Dimitri Zhukov, Jean-Baptiste Alayrac, Ramazan Gokberk Cinbis, David Fouhey, Ivan Laptev, and Josef Sivic. Cross-task weakly supervised learning from instructional videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3537–3545, 2019.

[5] Fadime Sener, Rishabh Saraf, and Angela Yao. Transferring knowledge from text to video: Zero-shot anticipation for procedural actions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[6] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, et al. Scaling egocentric vision: The epic-kitchens dataset. In *Proceedings of the European conference on computer vision (ECCV)*, pages 720–736, 2018.

[7] Yansong Tang, Dajun Ding, Yongming Rao, Yu Zheng, Danyang Zhang, Lili Zhao, Jiwen Lu, and Jie Zhou. Coin: A large-scale dataset for comprehensive instructional video analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1207–1216, 2019.

[8] Qi Zhao, Ce Zhang, Shijie Wang, Changcheng Fu, Nakul Agarwal, Kwonjoon Lee, and Chen Sun. Antgpt: Can large language models help long-term action anticipation from videos? *arXiv preprint arXiv:2307.16368*, 2023.

[9] Mahnaz Koupaee and William Yang Wang. Wikihow: A large scale text summarization dataset. *arXiv preprint arXiv:1810.09305*, 2018.

[10] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[11] Honglu Zhou, Roberto Martín-Martín, Mubbasir Kapadia, Silvio Savarese, and Juan Carlos Niebles. Procedure-aware pretraining for instructional video understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10727–10738, June 2023.

[12] Fadime Sener, Dipika Singhania, and Angela Yao. Temporal aggregate representations for long-range video understanding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 154–171. Springer, 2020.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[15] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 589–597, 2016.

[16] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6479–6488, 2018.

[17] Reza Ghoddoosian, Isht Dwivedi, Nakul Agarwal, and Behzad Dariush. Weakly-supervised action segmentation and unseen error detection in anomalous instructional videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10128–10138, October 2023.

[18] Kosuke Moriwaki, Gaku Nakano, and Tetsuo Inoshita. The brio-ta dataset: Understanding anomalous assembly process in manufacturing. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 1991–1995. IEEE, 2022.

[19] Guodong Ding, Fadime Sener, Shugao Ma, and Angela Yao. Every mistake counts in assembly, 2023.

[20] Emad Bahrami, Gianpiero Francesca, and Juergen Gall. How much temporal long-term context is needed for action segmentation? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10351–10361, 2023.

[21] Tim J Schoonbeek, Tim Houben, Hans Onvlee, Fons van der Sommen, et al. Industreal: A dataset for procedure step recognition handling execution errors in egocentric videos in an industrial-like setting. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4365–4374, 2024.

[22] Rohith Peddi, Shivvrat Arya, Bharath Challa, Likhitha Pallapothula, Akshay Vyas, Jikai Wang, Qifan Zhang, Vasundhara Komaragiri, Eric Ragan, Nicholas Ruozzi, et al. Captaincook4d: A dataset for understanding errors in procedural activities. *arXiv preprint arXiv:2312.14556*, 2023.

[23] Hao Zheng, Regina Lee, and Yuqian Lu. Ha-vid: A human assembly video dataset for comprehensive assembly knowledge understanding, 2023.

[24] Xin Wang, Taein Kwon, Mahdi Rad, Bowen Pan, Ishani Chakraborty, Sean Andrist, Dan Bohus, Ashley Feniello, Bugra Tekin, Felipe Vieira Frujeri, et al. Holoassist: an egocentric human interaction dataset for interactive ai assistants in the real world. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 20270–20281, 2023.

[25] Rishi Hazra, Brian Chen, Akshara Rai, Nitin Kamra, and Ruta Desai. Egotv: Egocentric task verification from natural language task descriptions. *arXiv preprint arXiv:2303.16975*, 2023.

[26] Yicheng Qian, Weixin Luo, Dongze Lian, Xu Tang, Peilin Zhao, and Shenghua Gao. Svip: Sequence verification for procedures in videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19890–19902, June 2022.

[27] Medhini Narasimhan, Licheng Yu, Sean Bell, Ning Zhang, and Trevor Darrell. Learning and verification of task structure in instructional videos. *arXiv preprint arXiv:2303.13519*, 2023.

[28] Yunseok Jang, Sungryull Sohn, Lajanugen Logeswaran, Tiange Luo, Moontae Lee, and Honglak Lee. Multimodal subtask graph generation from instructional videos, 2023.

[29] Lajanugen Logeswaran, Sungryull Sohn, Yunseok Jang, Moontae Lee, and Honglak Lee. Unsupervised task graph generation from instructional video transcripts. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 3392–3406, Toronto, Canada, July 2023. Association for Computational Linguistics.

[30] Xudong Lin, Fabio Petroni, Gedas Bertasius, Marcus Rohrbach, Shih-Fu Chang, and Lorenzo Torresani. Learning to recognize procedural activities with distant supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13853–13863, 2022.

[31] Jiahao Zhang, Anoop Cherian, Yanbin Liu, Yizhak Ben-Shabat, Cristian Rodriguez, and Stephen Gould. Aligning step-by-step instructional diagrams to video demonstrations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2483–2492, 2023.

[32] Kumar Ashutosh, Santhosh Kumar Ramakrishnan, Triantafyllos Afouras, and Kristen Grauman. Video-mined task graphs for keystep recognition in instructional videos, 2023.

[33] Yu Zhou, Sha Li, Manling Li, Xudong Lin, Shih-Fu Chang, Mohit Bansal, and Heng Ji. Non-sequential graph script induction via multimedia grounding. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5529–5545, Toronto, Canada, July 2023. Association for Computational Linguistics.

[34] Sungryull Sohn, Hyunjae Woo, Jongwook Choi, and Honglak Lee. Meta reinforcement learning with autonomous inference of subtask dependencies. In *International Conference on Learning Representations (ICLR)*, 2020.

[35] Keisuke Sakaguchi, Chandra Bhagavatula, Ronan Le Bras, Niket Tandon, Peter Clark, and Yejin Choi. proScript: Partially ordered scripts generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2138–2149, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[36] Niket Tandon, Aman Madaan, Peter Clark, Keisuke Sakaguchi, and Yiming Yang. Interscript: A dataset for interactive learning of scripts through error feedback. *arXiv preprint arXiv:2112.07867*, 2021.

[37] Andrew Cropper and Sebastijan Dumančić. Inductive logic programming at 30: a new introduction. *Journal of Artificial Intelligence Research*, 74:765–850, 2022.

[38] Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V. Chawla, and Panpan Xu. Graph neural prompting with large language models. *CoRR*, abs/2309.15427, 2023.

[39] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. *Advances in neural information processing systems*, 31, 2018.

[40] Sungyong Seo, Sercan Arik, Jinsung Yoon, Xiang Zhang, Kihyuk Sohn, and Tomas Pfister. Controlling neural networks with rule representations. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 11196–11207. Curran Associates, Inc., 2021.

[41] Ziwei Xu, Yogesh Rawat, Yongkang Wong, Mohan S Kankanhalli, and Mubarak Shah. Don't pour cereal into coffee: Differentiable temporal logic for temporal action segmentation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 14890–14903. Curran Associates, Inc., 2022.

[42] Yaqi Xie, Fan Zhou, and Harold Soh. Embedding symbolic temporal knowledge into deep sequential models. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4267–4273, 2021.

[43] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[44] Haroon Idrees, Amir R Zamir, Yu-Gang Jiang, Alex Gorban, Ivan Laptev, Rahul Sukthankar, and Mubarak Shah. The thumos challenge on action recognition for videos "in the wild". *Computer Vision and Image Understanding*, 155:1–23, 2017.

[45] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 961–970, 2015.

[46] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, 27, 2014.

[47] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[48] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016.

[49] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6202–6211, 2019.

[50] Guodong Ding, Fadime Sener, and Angela Yao. Temporal action segmentation: An analysis of modern techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[51] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[52] Yazan Abu Farha and Jurgen Gall. Ms-tcn: Multi-stage temporal convolutional network for action segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3575–3584, 2019.

[53] Dipika Singhania, Rahul Rahaman, and Angela Yao. C2f-tcn: A framework for semi-and fully-supervised temporal action segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[54] Yuchi Ishikawa, Seito Kasai, Yoshimitsu Aoki, and Hirokatsu Kataoka. Alleviating oversegmentation errors by detecting action boundaries. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2322–2331, 2021.

[55] Fangqiu Yi, Hongyu Wen, and Tingting Jiang. Asformer: Transformer for action segmentation. *arXiv preprint arXiv:2110.08568*, 2021.

[56] Nadine Behrmann, S Alireza Golestaneh, Zico Kolter, Juergen Gall, and Mehdi Noroozi. Unified fully and timestamp supervised temporal action segmentation via sequence to sequence translation. In *European Conference on Computer Vision*, pages 52–68. Springer, 2022.

[57] Nicolas Aziere and Sinisa Todorovic. Multistage temporal convolution transformer for action segmentation. *Image and Vision Computing*, 128:104567, 2022.

[58] Lei Ji, Chenfei Wu, Daisy Zhou, Kun Yan, Edward Cui, Xilin Chen, and Nan Duan. Learning temporal video procedure segmentation from an automatically collected large dataset. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1506–1515, 2022.

[59] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[60] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[61] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[62] Jiawei Ren, Cunjun Yu, Xiao Ma, Haiyu Zhao, Shuai Yi, et al. Balanced meta-softmax for long-tailed visual recognition. *Advances in neural information processing systems*, 33:4175–4186, 2020.

[63] Yuankai Luo. Dagformer: Directed acyclic graph transformer. *arXiv preprint arXiv:2210.13148*, 2022.