Flow graph to Video Grounding for Weakly-supervised Multi-Step Localization

Nikita Dvornik, Isma Hadji, Hai Pham, Dhaivat Bhatt, Brais Martinez, Afsaneh Fazly, and Allan D. Jepson

Samsung AI Center

Abstract. In this work, we consider the problem of weakly-supervised multi-step localization in instructional videos. An established approach to this problem is to rely on a given list of steps. However, in reality, there is often more than one way to execute a procedure successfully, by following the set of steps in slightly varying orders. Thus, for successful localization in a given video, recent works require the actual order of procedure steps in the video, to be provided by human annotators at both training and test times. Instead, here, we only rely on generic procedural text that is not tied to a specific video. We represent the various ways to complete the procedure by transforming the list of instructions into a procedure flow graph which captures the partial order of steps. Using the flow graphs reduces both training and test time annotation requirements. To this end, we introduce the new problem of flow graph to video grounding. In this setup, we seek the optimal step ordering consistent with the procedure flow graph and a given video. To solve this problem, we propose a new algorithm - Graph2Vid - that infers the actual ordering of steps in the video and simultaneously localizes them. To show the advantage of our proposed formulation, we extend the CrossTask dataset with procedure flow graph information. Our experiments show that Graph2Vid is both more efficient than the baselines and yields strong step localization results, without the need for step order annotation.

Keywords: procedures, flow graphs, instructional videos, localization

1 Introduction

Understanding video content from procedural activities has recently seen a surge in interest with various applications including future anticipation [33, 13], procedure planning [5, 1], question answering [39] and multi-step localization [41, 36, 23, 22, 10]. In this work, we tackle multi-step localization, i.e., inferring the temporal location of procedure steps present in the video. Since fully-supervised approaches [12, 21, 36] entail expensive labeling efforts, several recent works perform step localization with weak supervision. The alignment-based approaches [29, 4, 10] are of particular interest here as for each video they only require the knowledge of step order to yield framewise step localization.

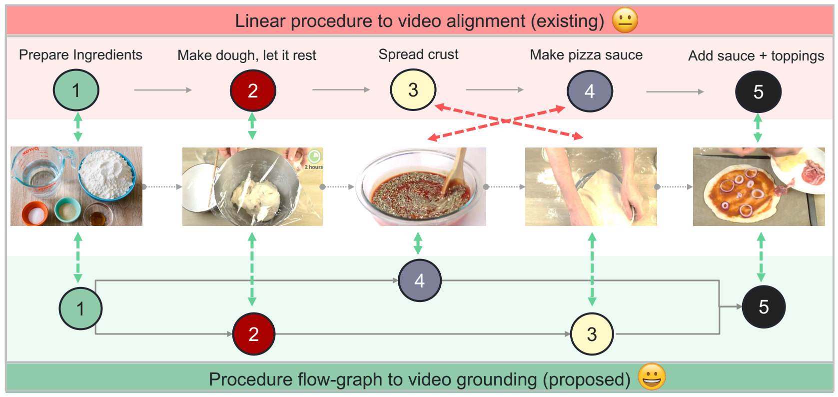

Fig. 1: Graph-to-Sequence Grounding. (top) instructional videos do not always strictly follow a prototypical procedure order (e.g., recipe). (bottom) Therefore, we propose a new setup where procedural text is parsed into a flow graph that is consequently grounded to the video to temporally localize all steps using our novel algorithm.

However, all such alignment-based approaches share a common issue. They all assume that a given procedure follows a strict order, which is often not the case. For example, in the task of making a pizza, one can either start with steps related to making dough, then steps involved in making the sauce, or vice-versa, before finally putting the two preparations together. Since the general procedure (e.g., recipe) does not define a unique order of steps, the alignment-based approaches rely on human annotations to provide the exact steps order for each video. In other words, step localization via alignment requires using per-video step order annotations during inference, which limits the practical value of this setup.

To this end, we propose a new approach for step localization that does not rely on per-video step order annotation. Instead, it uses the general procedure description, common to all the videos of the same category (e.g., the recipe of making pizza independent of the video sequence), to localize procedure steps present in any video. Fig. 1 illustrates the proposed problem setup. We propose to represent a procedure using a flow graph [32, 18], i.e., a graph-based procedure representation that encodes the partial order of instruction steps and captures all the feasible ways to execute a procedure. This leads us to the novel problem of multi-step localization from instructional videos under the graph-based setting, which we call flow graph to video grounding. To support the evaluation of our work we extend the widely used CrossTask dataset [41] with recipes and corresponding flow graphs. Importantly, in this work, the flow graphs are obtained by parsing procedural text (e.g., a recipe) freely available online using an off-the-shelf parser, which makes the annotation step automatic and reduces the amount of human annotation even further.

To achieve our goal of step localization from flow graphs, we introduce a novel solution for graph-to-sequence grounding - Graph2Vid. Graph2Vid is an algorithm that, given a video and a procedure flow graph, infers the temporal

location of every instruction step such that the resulting step sequence is consistent with the procedure flow graph. Our proposed solutions grounds each step in the video by: **(i)** expanding the original flow graph into a meta-graph, that concisely captures all topological sorts [37] of the original graph and **(ii)** applying a novel graph-to-sequence alignment algorithm to find the best alignment path in the metagraph with the given video. Importantly, our alignment algorithm has the ability to "drop" video frames from the alignment, in case there is not a good match among the graph nodes, which effectively models the no-action behavior. Moreover, our Graph2Vid algorithm naturally admits a differentiable approximation and can be used as a loss function for training video representation using flow graph supervision. As we show in Section 3, this can further improve step localization performance with flow graphs.
Contributions. In summary the main contributions of this work are fourfold.

1. We introduce flow graph to video grounding - a new task of multi-step localization in instructional videos given generic procedure flow graphs.
2. We extend the CrossTask dataset by associating procedure text with each category and parsing the instructions into a procedure flow graph.
3. We propose a new graph-to-sequence grounding algorithm (i.e., Graph2Vid) and show that Graph2Vid outperforms baseline approaches in step localization performance and efficiency.
4. We show Graph2Vid can be used as a loss function to supervise video representations with flow graphs.

The code will be made available at github.com/SamsungLabs/Graph2Vid.

2 Related work

Sequence-to-sequence alignment. Sequence alignment has recently seen growing interest across various tasks [34, 7, 4, 11, 14, 3, 6, 2], in particular, the methods seeking global alignment between sequences by relying on Dynamic Time Warping (DTW) [30, 7, 4, 14, 6]. Some of these methods propose differentiable approximations of the discrete operations (i.e., the min operator) to enable training with DTW [7, 14]. Others, allow DTW to handle outliers in the sequences [24, 3, 31, 27, 35, 25]. Of particular note, the recently proposed Drop-DTW algorithm [10] combines the benefits of all those methods as it allows dropping outliers occurring anywhere in the sequence, while still maintaining the ability of DTW to do one-to-many matching and enabling differentiable variations. However, as most other sequence alignment algorithms, Drop-DTW matches sequence elements with each other in a linear order, not consider possible element permutations within each sequence. In this work, we propose to extend Drop-DTW to work with partially ordered sequences. This is achieved by representing one of the sequences as a directed cyclic graph, thereby relaxing the strict order requirement.
Graph-to-sequence alignment. Aligning graphs to sequences is an important topic in computer science. One of the pioneering works in this area proposed a

Dynamic Programming (DP) based solution for pattern matching where the target text is represented as a graph [26]. Many follow up works extend this original idea via enhancing the alignment procedure. Examples include, admitting additional dimensions in the DP tables for each alternative path [19], improving the efficiency of the alignment algorithm [28, 16] or explicitly allowing gaps in the alignment, thereby achieving sub-sequence to graph matching [17]. A common limitation among all these methods is the assumption that only *one* of the paths in the graph aligns to the query sequence, while alternative paths and their corresponding nodes do not appear in the query sequence. Therefore, the goal in graph-to-sequence *alignment* is to find the specific path that best aligns with the query sequence. In contrast, we consider the novel problem of graph-to-sequence *grounding*. In particular, we consider the task where all nodes in the graph have a match in the query sequence and therefore our task is to ground each node in the sequence, while finding the optimal traversal in the graph that best aligns with the sequence. This problem is strictly harder than graph-to-sequence alignment and can not be readily tackled by the existing algorithms.

Video multi-step localization The task of video multi-step localization has gained a lot of attention in the recent years [22, 20, 10] particularly thanks to instructional videos dataset availability that support this research area [41, 36, 40]. The task consists of determining the start and end times of all steps present in the video, based on a description of the procedure depicted in the video. Some methods rely on full supervision using fine-grained labels indicating start and end times of each step (e.g., [12, 21, 36]). However, these methods require extensive labeling efforts. Instead, other methods propose weakly supervised approaches where only steps order information is needed to yield framewise step localization [15, 8, 29, 4, 41, 10]. However, these methods lack flexibility as they require *exact* order information to solve the step localization task. Here, we propose a more flexible approach where only partial order information, as given by a procedure flow graph, is required to localize each step. In particular, given a procedure flow graph, describing all possible step permutations that result in successful procedure execution, our method localizes the steps in a given video, by automatically grounding the steps of the graph in the video.

3 Our approach

In this section, we describe our approach for flow graph to video grounding. We start with a motivation and formal definition of our proposed flow graph to sequence grounding problem. Next, we describe in detail our proposed solution to tackle the task of video multi-step localization using flow graphs.

3.1 Background

Ordered steps to video alignment. If the true order of steps in a video (i.e., as they happen in a video) is given, the task of step grounding reduces to a well-defined problem of steps-to-video alignment, which can be solved with some existing sequence alignment method. In particular, the recent Drop-DTW [10] algorithm suits the task particularly well thanks to a unique set of desired properties: **(i)** it operates on sequences of continuous vectors (such as video and step embeddings) **(ii)** it permits one-to-many matching, allowing multiple video frames to match to a single step, and **(iii)** it allows for dropping elements from the sequence, which in turn allows for ignoring video frames that are unrelated to the sequence of steps. In Drop-DTW, the alignment is formulated as minimization of the total match cost between the video clips and instruction steps. It is solved using dynamic programming and can be made differentiable (see Alg. 1 in [10]). That is, given a video, $\mathbf{x}$, and a sequence of steps, $\mathbf{v}$, Drop-DTW returns the alignment cost, c^*, and alignment matrix, M^*, indicating the correspondences between steps and video segments.

Procedure flow graphs. In more realistic settings, procedure steps for many processes, such as cooking recipes, are often given as a set of steps in a partial order. Specifically, the partial ordering dictates that certain steps need to be completed before other steps are started, but that other subsets of steps can be done in any order. For example, when thinking of making a salad, one can cut tomatoes and cucumbers in one order or the other, however we are certain that both ingredients must be cut before mixing them into the salad. This is an example of a procedure with partially ordered steps; i.e., there are multiple valid ways to complete the procedure, all of which can be conveniently represented with a flow graph.

A procedure flow graph is a Directed Acyclic Graph (DAG) $\mathcal{G} = (V, E)$, where V is a set of nodes and E is a set of directed edges. Each node $v_i \in V$ represents a procedure step and an edge $e_j \in E$ connecting v_k and v_l declares that the procedure step v_k must be completed before v_l begins in any instruction execution. If a node v_k has multiple ancestors, all the corresponding steps must be completed before beginning instruction step v_k. In this work, we assume that $\mathcal{G}$ has a single root and sink nodes. For convenience, we automatically add them to the graph if they are not already present. From the definition of the flow graph, it follows that every topological sort [37] of the nodes in $\mathcal{G}$ (see Fig. 2, step 2) is a valid way to complete the procedure. This is an important property that forms the foundation of our Graph2Vid approach, described next.

Flow graphs to video grounding. We define the task of grounding a flow graph $\mathcal{G}$ in a video $\mathbf{x} = [x_i]_{i=1}^N$, where N is the total number of frames, as the task of finding a disjoint set of corresponding video segments, $s_l = [x_i]_{i=start_l}^{end_l}$ for each node $v_l \in \mathcal{G}$ of the flow graph, such that the resulting segmentation conforms to the flow graph. Specifically, in a pair of resulting video segments, $(s_i,\ s_j)$, segment s_i can only occur before s_j in the video if the corresponding node n_i is a predecessor of n_j in the flow graph $\mathcal{G}$. In this work, we assume that every procedure step v_l appears in the video exactly once.

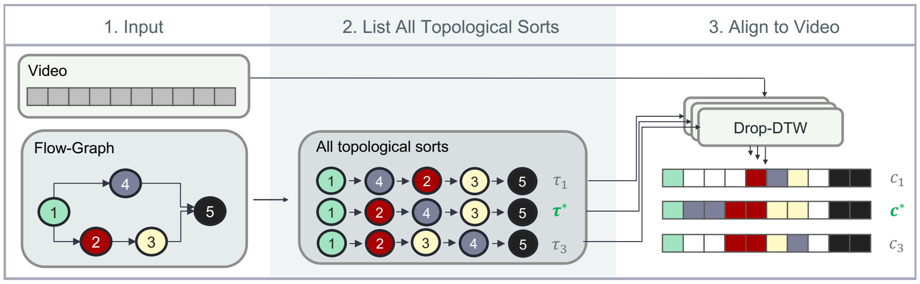

Fig. 2: Brute-force approach for flow-graph to video grounding. Given the the flow-graph, the brute-force approach explicitly considers every topological sort and aligns it to the video with Drop-DTW independently, making the inference inefficient.

3.2 Graph to video grounding - a brute-force approach

Before introducing our method, we first discuss a naive solution to the problem. As previously mentioned, we know that the procedure execution in the video follows some topological sort of the flow graph, $\mathcal{G}$. Thus, one can derive an algorithm for flow graph to video grounding by explicitly considering all the topological sorts of $\mathcal{G}$ as depicted in Fig. 2. More specifically, one can generate all topological sorts of the flow graph, try aligning each of them to the video, and select the best alignment as step grounding. This essentially reduces the problem of flow graph grounding to sequence-to-sequence alignment which can be solved using Drop-DTW. The procedure would iterate over all topological sorts $\tau \in \mathcal{T}$ of the flow graph $\mathcal{G}$ and align a sequence of nodes (re-ordered according to the topological sort) $\mathbf{v}^\tau = [v_{\tau_i}]_{i=1}^{|V|}$ to the video sequence, and pick the alignment M^* with the topological sort τ^* that has the minimum alignment cost c^*:

$$c^* = \texttt{Drop-DTW}(\mathbf{v}^{\tau^*}, \mathbf{x}); \quad \tau^* = \arg\min_{\tau \in \mathcal{T}} \texttt{Drop-DTW}(\mathbf{v}^\tau, \mathbf{x}) \tag{1}$$

While simple, this approach has one crucial downside - its efficiency. As we show in Sec. 3.5, this algorithm has exponential complexity and becomes infeasible even for small flow graphs. Thus, we need a more scalable solution to the problem.

3.3 Graph2Vid - our approach

In this section we present a new, more efficient approach for flow graph to video grounding, that we term Graph2Vid (see the overview in Fig. 3). Graph2Vid operates in two stages: **(i)** given the flow graph, $\mathcal{G}$, we pack all the topological sorts of $\mathcal{G}$ into a novel compact graph-based representation $\mathcal{S}$ (that we call the tSort graph); **(ii)** we align the obtained tSort graph, $\mathcal{S}$, to the video, $\mathbf{x}$, using a proposed graph-to-video alignment algorithm, which we dub Graph-Drop-DTW. To compute the alignment we embed the procedure text in each node of the graph, v_τ, and the clips of the video sequence, x_i, using a model pre-trained on the large HowTo100M dataset [22]. The key to superior efficiency of Graph2Vid is the interplay between the tSort graph structure and the graph-to-video alignment algorithm, which allows for polynomial complexity of flow graph to video grounding[1]. Moreover, Graph2Vid allows for a differentiable approximation that can be used for training neural networks. In the following, we explain how to construct the tSort graph, $\mathcal{S}$, from the original flow graph, $\mathcal{G}$, how to extend Drop-DTW to perform graph-to-sequence alignment, and finally, how to use Graph2Vid as a differentiable loss function.

Creating the tSort graph. As we have shown previously, grounding a flow-graph, $\mathcal{G}$, to a video requires considering all the topological sorts of $\mathcal{G}$, yet their explicit consideration is infeasible due to the exponential number of topological sorts. We note that the cause of such inefficiency is the redundancy and high overlaps between the topological sorts. This motivates us to encode all topological sorts into a tSort graph, $\mathcal{S}$, that effectively shares the common parts of different topological sorts and provides a more compact representation. Fig. 3 illustration how for a simple flow graph input we construct the tSort graph encoding all the topological sorts (Fig. 2). Each path from root to sink in the tSort graph, $\mathcal{S}$, spells out a topological sort of $\mathcal{G}$. Thus, listing all root to sink paths in $\mathcal{S}$ is equivalent to listing all topological sorts of the original flow graph, $\mathcal{G}$.
Algorithm 1 gives a procedure for constructing the tSort graph $\mathcal{S}$. The first step to constructing the tSort graph is to connect all the nodes on separate threads in the original flow-graph $\mathcal{G}$ (i.e., like nodes $\{4\}$ and $\{2, 3\}$ in Fig 3) with undirected edges. Since the instruction steps on separate threads may follow one after another in an actual instruction execution, thus connection between them must exist. These connections yield an augmented graph, $\mathcal{G}_{aug}$. Then, in this augmented graph $\mathcal{G}_{aug}$, we run Breat First Search (BFS) traversal to find all the paths that lead from the root to the sink node of $\mathcal{G}_{aug}$ and conform to the original flow-graph $\mathcal{G}$. During the BFS traversals, the paths that have visited the same set of nodes so far are merged together and mapped into a single node in the tSort graph $\mathcal{S}$. Merging the traversals with the same set of visited nodes enables the tSort graph to represent all topological sorts efficiently. For more details on the tSort graph construction and a more efficient implementation description, we refer the reader to supplemental material.
Graph to sequence alignment using Graph-Drop-DTW. Having access to the tSort graph, $\mathcal{S}$, (whose every path from root to sink is a valid topological sort of the flow graph, $\mathcal{G}$), we can cast flow graph to video grounding as graph-to-video alignment problem [26]. In this case, the graph-to-video alignment finds a traversal of the graph $\mathcal{S}$ that best aligns with the given video $\mathbf{x}$. Importantly, directly aligning the tSort graph to a video discovers the optimal path in the graph and the best alignment of this path to the video *simultaneously* (see Fig 3, step 3). This is in contrast to the naive solution in Sec. 3.2, which uses sequence alignment as a subroutine to find the optimal topological sort.

[1] The tSort graph is polynomial for an assumed subset of flow graphs with a fixed maximum number of threads.

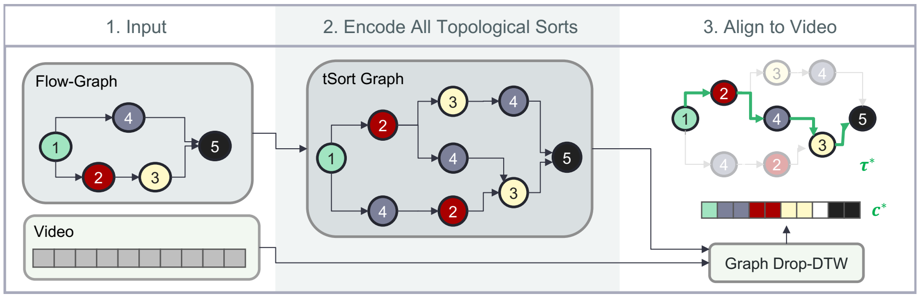

Fig. 3: Flow-graph to video grounding with Graph2Vid. Given the the procedure flow-graph and a video as input our method packs all topological sorts into a tSort Graph and then uses Graph-Drop-DTW to align it to the given video, producing video segmentation and the optimal topological sort τ^*.

Algorithm 1 tSort-graph Construction

Inputs: $\mathcal{G}$- flow graph, s - root node in $\mathcal{G}$
$\mathcal{G}_{aug} = \text{aug}(\mathcal{G})$ ▷ *connect the nodes on parallel threads*
$E_{tSort} = []$ ▷ *init edge set of the tSort-graph*
$q = \text{queue}((\text{s}, \oslash))$ ▷ *init BFS queue*
while q **do**
  $v, P = q.\text{pop}()$ ▷ *active node v, set of visited nodes P*
  for v_d in get_descendants(v, $\mathcal{G}_{aug}$) **do**
    $P_d = P.\text{add}(v_d)$ ▷ *extend the visited nodes set*
    if get_predecessors(v_d, $\mathcal{G}$) in P **then** ▷ *the path P_d conforms to $\mathcal{G}$*
      $q.\text{append}((v_d, P_d))$
      $E_{tSort}.\text{add}(((v, P), (v_d, P_d)))$ ▷ *add edge to tSort graph*
$\mathcal{S}$ = build_graph_from_edges(E_{tSort}) ▷ *build the tSort graph*
Output: $\mathcal{S}$

In order to solve graph to video grounding, the graph-to-sequence alignment algorithm must have the following properties: **(i)** operate on continuous vectors, **(ii)** permit one-to-many matching and **(iii)** allow for unmatched sequence elements. To the best of our knowledge, a graph-to-sequence alignment algorithm with such properties does not exist. However, Drop-DTW [10] for sequence alignment satisfies all 3 criteria (see Sec. 3.1) Thus, we propose a new algorithm - Graph-Drop-DTW - which is an extension of Drop-DTW for graph-to-sequence alignment. We base Graph-Drop-DTW on Alg.1 in [10] and modify the dynamic programming recursion to take into account the graph structure as follows:

$$
\begin{aligned}
D^+_{i,j} &= C_{i,j} + \min\{\min_{k \in A(i)} \{D_{k,j-1}\}, D_{i,j-1}\} \qquad (2)\\
D^-_{i,j} &= d^x_j + D_{i,j-1}\\
D_{i,j} &= \min\{D^+_{i,j}, D^-_{i,j}\},
\end{aligned}
$$

where $C_{i,j}$ is the cost of matching the i-th node of the graph to the j-th video clip, and d_j^x is the cost of dropping the j-th video clip and not matching it to any node in the graph. We follow [10] and define $C_{i,j}$ as negative log-likelihood of the video clip j belonging to step i, and compute the drop cost d_j^x as a 30-th bottom percentile of $C_{i,j}$. Different from Drop-DTW, when computing $D^+_{i,j}$, Graph-Drop-DTW takes into consideration all the predecessors of the node i, (i.e., $k \in A(i)$) and selects the one minimizing the alignment cost. Intuitively, the minimum operation over the predecessors, $\min_{k \in A(i)}$, in Eq. (2), dynamically finds the traversal of the graph that aligns with the video best. Similar to Drop-DTW, Graph-Drop-DTW outputs the alignment cost c^* and the alignment path M^*, representing the optimal matching between the nodes of the input graph $\mathcal{S}$ and the video clips of $x_i \in \mathbf{x}$. It is important to note that Graph-Drop-DTW can only drop elements from the sequence (as a direct extension of Alg.1 in [10]) and does not support dropping nodes from the graph.

Graph2Vid for flow graph grounding Finally, Graph2Vid can be defined as the complete pipeline that chains tSort graph creation and its alignment to the video. Precisely, given the procedure flow graph, $\mathcal{G}$, and the video sequence, $\mathbf{x}$, Graph2Vid first transforms $\mathcal{G}$ into the tSort graph, $\mathcal{S}$, then aligns the graph, $\mathcal{S}$, to the video, $\mathbf{x}$, using Graph-Drop-DTW. This effectively provides the desired correspondence between every node, $v_i \in \mathcal{G}$, and a video segment in $\mathbf{x}$, that conforms to the flow graph, $\mathcal{G}$.

3.4 Graph2Vid for representation learning

We now describe a differentiable approximation of Graph2Vid to learn video representations using flow graphs as the source of supervision. To use Graph2Vid as a loss function, we must be able to backpropagate gradients with respect to the video input. That is, Graph-Drop-DTW must be differentiable. A differentiable version of Graph-Drop-DTW can be obtained by simply using a soft approximation of the min operator (e.g., [7, 14]) in Eq. (2). Here, we substitute the min in Eq 2 with the smooth-min operator [14] defined as

$$\text{smoothMin}(x; \gamma) = x \cdot \text{softmax}(x/\gamma) = \frac{x \cdot e^{x/\gamma}}{\sum_j e^{x_j/\gamma}}, \tag{3}$$

where $\gamma > 0$ is a hyper-parameter controlling the trade-off between smoothness and the error of the approximation.

With this differentiable version of Graph2Vid, we can use the matching cost between a video and it's corresponding flow graph as a training signal. Intuitively, the lower the cost of matching of a video to its corresponding procedure flow graph, the better are the learned representations. Specifically, we define the Graph-Drop-DTW loss as the cost of grounding the flow graph, $\mathcal{G}$, to video, $\mathbf{x}$:

$$\mathcal{L}_{\text{G}}(Z, \mathbf{x}) = c^* = \text{Graph-Drop-DTW}(\mathcal{G}, \mathbf{x}). \tag{4}$$

As we show in Sec. 4, using $\mathcal{L}_{\text{G}}$ for weakly-supervised learning (with flow-graph supervision) improves step localization performance.

3.5 On the algorithm's complexity

To develop some intuition on both the size of the generated tSort-graphs, and on the speed-up over the naive approach, we consider simple model problems where the flow graph, $\mathcal{G}$, consists of T separate, linearly-ordered threads, with $n_1, n_2, \ldots, n_T \geq 1$ nodes in each thread, for a total of $\sum_{t=1}^{T} n_t = n$ steps. As we show in supplemental, in this case, the number of topological sorts of $\mathcal{G}$, $N_{ts}(\mathrm{G})$ grows exponentially with n and has the complexity $O(T^n)$. On the other hand, the number of nodes in the tSort graph $\mathcal{S}$, $|V_S|$, is polynomial in the number of nodes n, i.e., $|V_S| = O(Tn^T)$ which is better than exponential growth in $N_{ts}(\mathrm{G})$, provided that T is not too large. As shown in the supplemental, the asymptotic speedup of Graph2Vid over the brute-force approach can be roughly described by the ratio $N_{ts}(\mathrm{G})/|V_s|$ which is still exponential in n, giving a large advantage to Graph2Vid. This is further confirmed in Sec. 4.5, where Graph2Vid is orders of magnitude faster than the brute-force approach on real procedure flow-graphs.

3.6 Creating flow graph from procedural text

To construct flow graphs from procedural text,we considered two automated alternatives; namely, a rule-based and a learning-based approach. In addition, we considered a manual approach, where we explicitly define nodes and edges.

Rule-based graph parsing. Starting from regular procedural instructions, we first extract relevant text entities from each step description, including action verbs, direct and prepositional objects as suggested in [32]. Next, we define a set of rich semantic rules for the graph constructor to connect the text entities. Once edges are defined using those entities, flow graphs are collapsed into coarse sentence-level graphs to be used in our experiments.

Learning-based graph parsing. Using the same procedural instructions, we consider a learning based approach, which relies on two steps. First, we identify 10 named entities using the the tagger of [38]. Second, we used the parser proposed recently in [9] to automatically find edges between the named entities defined in the first step. Once again, we finally collapse the fine-grained flow graphs into coarse sentence-level graphs as done with the rule-based parser.
A detailed description of both approaches is in the supplemental material.

4 Experiments

To demonstrate the strengths of the proposed Graph2Vid algorithm, we first present our dataset construction in Sec. 4.1. Then we describe the metrics used to evaluate the task of multi-step localization from graphs as well as the adopted baselines in Sec. 4.2. Finally, we summarize our results in Secs. 4.3 through 4.5.

4.1 Dataset construction

To evaluate our new formulation of graph-to-video grounding using the proposed Graph2Vid approach, we need a dataset with procedure steps captured in flow graphs and corresponding Ground Truth (GT) start and end times for each node in these graphs. To this end, we extend the widely used CrossTask datast [41] following three main steps: **(i)** For each procedure class (e.g., *Build floating shelves* or *Making pancakes*), we grab the procedure text from the web [2]. **(ii)** We extract flow graphs from the procedural text following the methods described in Section 3.6, such that each node in the flow graph corresponds to a step from the procedure text. **(iii)** Finally, we manually find correspondence between nodes in the graph and step instructions provided with the original datasets. These correspondences are used to associate the original GT temporal annotations of each step with nodes in our flow graph. These GT temporal annotations, now associated with our graph nodes, are used to evaluate our model on the task of multi-step localization. Importantly, these annotations are only necessary for evaluation, and not for training or inference.

4.2 Metrics and baselines

We evaluate the performance of our Graph2Vid approach on multi-step localization using two different metrics: **(i) Framewise accuracy (Acc.)** [36], which is defined as the ratio between the number of frames assigned the correct step label (not including background) and the total number of frames, and **(ii) Intersection over Union (IoU)** [40], which is defined as the sum of the intersections between the predicted and ground truth time intervals for each step label divided by the sum of their unions.

As we are the first to tackle multi-step localization under this new paradigm of flow graph to video grounding, we consider three increasingly strong baselines for comparisons. **(i) Bag of steps.** In this baseline, we consider every steps as being a node in a separate thread in a graph (i.e., no graph structure or order is imposed). **(ii) Linear Procedure.** Here, we read the instructions extracted from the procedure text linearly and assume this order as the default ordering of steps. **(iii) GT Step Sequence.** This is the *upper bound* on our Graph2Vid; it uses the ground truth step order (i.e., as they happen in each video) provided with the dataset. For baseline (i) we use the proposed Graph-Drop-DTW to obtain a segmentation as we treat the bag of steps as a disconnected graph. On the other hand, in baselines (ii) and (iii), we use the Drop-DTW algorithm [10] to obtain the segmentations, which we chose for two main reasons. First, this algorithm directly relates to the proposed Graph-Drop-DTW. Second, Drop-DTW is currently state-of-the-art on CrossTask for step localization [10].

4.3 Graph2Vid for step localization

We begin by evaluating the proposed Graph2Vid technique as an inference time method for step localization in instructional videos. For this purpose, we follow

[2] We find the procedure text of CrossTask in www.wikihow.com

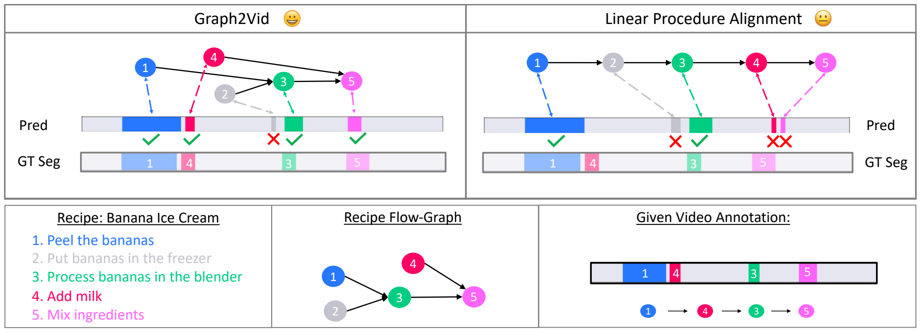

Fig. 4: Graph2Vid vs Linear Procedure Alignment for step localization on CrossTask. Given a video of making Banana Ice Cream (bottom), we compare video segmentation (into steps) produced by both methods. Graph2Vid (top-left) localizes all the present recipe steps by grounding the flow-graph into the video, while aligning linear recipe sequence to the video (top-right) fails in 2 out of 4 steps. This is because the order of instruction in the video (bottom-right) is different from the linear recipe, however it conforms to the flow-graph and thus can be grounded by Graph2Vid. Both methods incorrectly predict the step "2", that is actually not present in the video.

previous work [22, 10] and rely on video and text features extracted from a model pre-trained on the HowTo100M dataset [23]. First, the results in Tab 1 show better performance of the "Linear Procedure" baseline compared to the "Bag of Steps". This indicates that treating the recipe as a linear sequence of steps (i.e., having some prior on the order of steps) allows for better step localization than treating the recipe as unordered set of steps. However, treating the recipe as a flow graph (automatically extracted using the learning-based parser of Sec. 3.6) and grounding it in the video with Graph2Vid yields superior localization performance. This confirms the advantage of using the more flexible graph structure of the recipe for step localization. See Fig. 4 for an illustration of how Graph2Vid takes advantage of the nonlinear flow-graph structure in an example from the dataset. Unfortunately, there is still a large gap in performance between Graph2Vid and our upper bound, which relies on ground-truth ordered steps provided by human annotators. Closer examination of these results revealed that this gap is largely attributed to the fact that the GT of CrossTask does not conform to the assumed recipe flow graph structure, which we use in our Graph2Vid formulation. Specifically, while the flow graphs assume that each step happens once across the video, the GT of CrossTask allows for repeated steps (e.g., [`cut tomato, cut cucumber, cut tomato, mix ingredients`]), where these repetitions are often a consequence of video post-production.

4.4 Graph2Vid for representation learning

Here, we show the benefits of using the differentiable approximation of Graph2Vid for weakly-supervised representation learning. We start from the same video and text features used in Sec 4.3 and train a two-layer multi-layer perceptron (i.e.,

Table 1: Graph2Vid as an inference time procedure.

Inference Method	Acc.↑	IoU ↑
Graph2Vid (ours)	**24.8**	**16.8**
Linear Procedure [10]	22.3	15.1
Bag of Steps	20.5	13.7
GT Step Sequence [10]	32.4	21.2

Table 2: Graph2Vid as a training loss.

Training Method	Acc. ↑	IoU ↑
Graph2Vid (ours)	**26.3**	**19.1**
Linear Procedure + Drop-DTW	25.0	16.6
Bag of Steps + Soft Clustering	25.4	17.3
Pre-trained Features [22]	25.3	17.1
GT Step Seq. + Drop-DTW	35.7	25.3

MLP) on top of the video representation. During training, we assume no access to ground-truth ordered step sequences, but only to the task label (e.g., making pizza) and its corresponding flow graph. The flow graph is obtained automatically from the procedure text description of the task using the learning-based graph parser descibed in Sec. 3.6. Here, we compare Graph2Vid for representation learning with two other methods: **(i)** procedure to video alignment with Drop-DTW [10] (i.e., "Linear Procedure + Drop-DTW"), and **(ii)** aligning video to a set of instruction steps using soft clustering (i.e., "Bag of Steps + Soft Clustering"). We elaborate on these baselines in the supplementary material.

Once again, the results in Tab. 2 speak in favor of the proposed Graph2Vid approach, which better benefits from training compared to the considered baselines. Interestingly, training with "Linear Procdure" does slightly worse than no training at all, indicating that aligning the video to a potentially out-of-order sequences, which is often the case for a fixed instruction list, results in a poor training signal. In contrast, allowing Graph2Vid to infer the optimal topological sort of the flow-graph for aligning with the video results in a better training signal and improves video representations. Finally, using the GT ordered steps for training - a much richer source of supervision - yields best overall performance, however at the price of extra labeling effort.

4.5 Ablation Study

Here we evaluate the role of the flow graph parser and study the inference speed of our proposed Graph2Vid.

Role of the flow graph parser. To better understand the connection between the flow graph construction method and Graph2Vid localization performance, we compare the two different parsers described in Sec. 3.6 as well as graphs obtained from manual annotation. As expected, the results summarized in Tab. 3, show that graphs from the rule-based parser yield performance slightly inferior to the manually generated graphs. Surprisingly, the flow-graphs from the learning-based parser do better than the manually annotated flow-graphs on CrossTask. After visually comparing the flow graphs produced by learning-based parser to the manual annotations, we realize that the former sometimes "misses" the edges present in the manual graph. This essentially allows for more flexible step ordering when aligning to a video, which benefits step localization on the CrossTask dataset.

This is because many CrossTask videos depict procedures that do not conform to the manually labeled flow graph, due to the post-editing of the videos. We provide more detailed analysis of this matter in Supplemental.

Table 3: Graph2Vid using different flow graph parsers.

Method	Acc.↑	IoU ↑
Manual Annotation	24.8	16.8
Rule-based Parser	24.3	16.3
Learning-based Parser	**25.3**	**17.1**

Evaluation of execution time. Here, we evaluate the speed of Graph2Vid for inference on CrossTask and compare it to running the brute force solution for flow graph grounding that considers all topological sorts explicitly (see Sec. 3.2). The flow graphs in CrossTask (obtained using the learning-based parser) have an average of 8.6 nodes and 2.7 separate threads. Such graphs, on average, produce $1,700$ topological sorts and generate tSort graphs, $\mathcal{S}$, with about 60 nodes. With such compact tSort graphs, Graph2Vid takes $\approx$ 57ms to ground a flow graph to a video. In contrast, the brute-force procedure requires $\approx 3.2 * 10^4$ ms, which is almost 3 order of magnitudes slower than Graph2Vid. This result speaks decisively in favor of our Graph2Vid approach for flow-graph to video grounding and confirms our theoretical derivations in Sec 3.5.

5 Conclusion

In summary, we introduced a new formulation for step localization in instructional videos using procedure flow graphs. In particular, we proposed the novel task of flow graph to video grounding, which relies on task level procedure description to yield a step-wise segmentation of instructional videos. To this end, we rely on automatically generated *task-level* procedure flow graphs for step localization instead of relying on manually annotated, *per-video* step sequences. This effectively makes the proposed solution more scalable and practical. To solve the task of flow graph to video alignment, we developed a new graph-alignment-based algorithm - Graph2Vid - that demonstrates superior localization performance and efficiency compared to baselines. In addition, we could improve video representations by training with flow graphs as supervisory signal and using Graph2Vid as a loss function.

We believe that flow graphs are a more natural and informative representation of procedural activities, compared to linear instructions. Moreover, procedure flow graphs are only needed at the task level, rather than on a per video bases. As such, we believe the proposed formulation to hold a great promise in minimizing labeling efforts and defines new avenues for further research.

Acknowledgement

We thank Ran Zhang for the help with flow graph creation and processing.

References

1. Bi, J., Luo, J., Xu, C.: Procedure planning in instructional videos via contextual modeling and model-based policy learning. In: Proceedings of the International Conference on Computer Vision (ICCV) (2021)
2. Cai, X., Xu, T., Yi, J., Huang, J., Rajasekaran, S.: DTWNet: A dynamic time warping network. In: Advances in Neural Information Processing Systems (NeurIPS) (2019)
3. Cao, K., Ji, J., Cao, Z., Chang, C., Niebles, J.C.: Few-shot video classification via temporal alignment. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
4. Chang, C., Huang, D., Sui, Y., Fei-Fei, L., Niebles, J.C.: D3TW: Discriminative differentiable dynamic time warping for weakly supervised action alignment and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
5. Chang, C.Y., Huang, D.A., Xu, D., Adeli, E., Fei-Fei, L., Niebles, J.C.: Procedure planning in instructional videos. In: Proceedings of the European Conference on Computer Vision (ECCV) (2020)
6. Chang, X., Tung, F., Mori, G.: Learning discriminative prototypes with dynamic time warping. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
7. Cuturi, M., Blondel, M.: Soft-DTW: A differentiable loss function for time-series. In: International Conference on Machine Learning (ICML) (2017)
8. Ding, L., Xu, C.: Weakly-supervised action segmentation with iterative soft boundary assignment. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
9. Donatelli, L., Schmidt, T., Biswas, D., Köhn, A., Zhai, F., Koller, A.: Aligning actions across recipe graphs. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (2021)
10. Dvornik, N., Hadji, I., Derpanis, K.G., Garg, A., Jepson, A.: Drop-DTW: Aligning common signal between sequences while dropping outliers. In: Advances in Neural Information Processing Systems (NeurIPS) (2021)
11. Dwibedi, D., Aytar, Y., Tompson, J., Sermanet, P., Zisserman, A.: Temporal cycle-consistency learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
12. Fabian Caba Heilbron, Victor Escorcia, B.G., Niebles, J.C.: ActivityNet: A large-scale video benchmark for human activity understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
13. Girdhar, R., Grauman, K.: Anticipative Video Transformer. In: Proceedings of the International Conference on Computer Vision (ICCV) (2021)
14. Hadji, I., Derpanis, K.G., Jepson, A.D.: Representation learning via global temporal alignment and cycle-consistency. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
15. Huang, D., Fei-Fei, L., Niebles, J.C.: Connectionist temporal modeling for weakly supervised action labeling. In: Proceedings of the European Conference on Computer Vision (ECCV) (2016)
16. Jain, C., Zhang, H., Gao, Y., Aluru, S.: On the complexity of sequence to graph alignment. Journal of Computational Biology **27** (2020)
17. Kavya, V.N.S., Tayal, K., Srinivasan, R., Sivadasan, N.: Sequence alignment on directed graphs. Journal of computational biology : a journal of computational molecular cell biology **261**, 53–67 (2019)

18. Kiddon, C., Ponnuraj, G.T., Zettlemoyer, L., Choi, Y.: Mise en place: Unsupervised interpretation of instructional recipes. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP) (2015)
19. Lee, C., Grasso, C., Sharlow, M.F.: Multiple sequence alignment using partial order graphs . Bioinformatics **18**(3), 452–464 (03 2002)
20. Luo, H., Ji, L., Shi, B., Huang, H., Duan, N., Li, T., Li, J., Bharti, T., Zhou, M.: UniVL: A unified video and language pre-training model for multimodal understanding and generation. arXiv preprint arXiv:2002.06353 (2020)
21. Ma, M., Fan, H., Kitani, K.M.: Going deeper into first-person activity recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
22. Miech, A., Alayrac, J.B., Smaira, L., Laptev, I., Sivic, J., Zisserman, A.: End-to-End Learning of Visual Representations from Uncurated Instructional Videos. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
23. Miech, A., Zhukov, D., Alayrac, J.B., Tapaswi, M., Laptev, I., Sivic, J.: HowTo100M: Learning a Text-Video Embedding by Watching Hundred Million Narrated Video Clips. In: Proceedings of the International Conference on Computer Vision (ICCV) (2019)
24. Müller, M.: Information Retrieval for Music and Motion. Springer-Verlag, Berlin, Heidelberg (2007)
25. Nakatsu, N., Kambayashi, Y., Yajima, S.: A longest common subsequence algorithm suitable for similar text strings. Acta Inf. (1982)
26. Navarro, G.: Improved approximate pattern matching on hypertext. Theoretical Computer Science **237**(1), 455–463 (2000)
27. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology (1970)
28. Rautiainen, M., Mäkinen, V., Marschall, T.: Bit-parallel sequence-to-graph alignment. Bioinformatics **35**(19), 3599–3607 (2019)
29. Richard, A., Kuehne, H., Iqbal, A., Gall, J.: NeuralNetwork-Viterbi: A framework for weakly supervised video learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
30. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken processing recognition. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) **26** (1978)
31. Sakurai, Y., Faloutsos, C., Yamamuro, M.: Stream monitoring under the time warping distance. In: International Conference on Data Engineering (ICDE) (2007)
32. Schumacher, P., Minor, M., Walter, K., Bergmann, R.: Extraction of procedural knowledge from the web: A comparison of two workflow extraction approaches. In: Proceedings of the 21st International Conference on World Wide Web (2012)
33. Senner, F., Yao, A.: Zero-shot anticipation for instructional activities (2019)
34. Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., Levine, S.: Time-contrastive networks: Self-supervised learning from video. In: IEEE International Conference on Robotics and Automation (ICRA) (2018)
35. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology (1981)
36. Tang, Y., Ding, D., Rao, Y., Zheng, Y., Zhang, D., Zhao, L., Lu, J., Zhou, J.: COIN: A large-scale dataset for comprehensive instructional video analysis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

37. Wikipedia: Topological sorting — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Topological%20sortingoldid=1062117596 (2022), [Online; accessed 07-March-2022]
38. Yamakata, Y., Mori, S., Carroll, J.: English recipe flow graph corpus. In: Proceedings of the 12th Language Resources and Evaluation Conference (2020)
39. Yang, A., Miech, A., Sivic, J., Laptev, I., Schmid, C.: Just Ask: Learning to Answer Questions from Millions of Narrated Videos. In: Proceedings of the International Conference on Computer Vision (ICCV) (2021)
40. Zhou, L., Xu, C., Corso, J.J.: Towards automatic learning of procedures from web instructional videos. In: AAAI Conference on Artificial Intelligence (2018)
41. Zhukov, D., Alayrac, J.B., Cinbis, R.G., Fouhey, D., Laptev, I., Sivic, J.: Cross-task weakly supervised learning from instructional videos. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)