proScript: Partially Ordered Scripts Generation

Keisuke Sakaguchi,[1] **Chandra Bhagavatula**,[1] **Ronan Le Bras**,[1]
Niket Tandon,[1] **Peter Clark**,[1] **Yejin Choi**[1,2]
[1]Allen Institute for Artificial Intelligence
[2]Paul G. Allen School of Computer Science & Engineering, University of Washington

Abstract

Scripts – prototypical event sequences describing everyday activities – have been shown to help understand narratives by providing expectations, resolving ambiguity, and filling in unstated information. However, to date they have proved hard to author or extract from text. In this work, we demonstrate for the first time that pre-trained neural language models can be finetuned to *generate* high-quality scripts, at varying levels of granularity, for a wide range of everyday scenarios (e.g., bake a cake). To do this, we collect a large (6.4k) crowdsourced partially ordered scripts (named proScript), that is substantially larger than prior datasets, and develop models that generate scripts by combining language generation and graph structure prediction. We define two complementary tasks: (i) edge prediction: given a scenario and unordered events, organize the events into a valid (possibly partial-order) script, and (ii) script generation: given only a scenario, generate events and organize them into a (possibly partial-order) script. Our experiments show that our models perform well (e.g., F1=75.7 on task (i)), illustrating a new approach to overcoming previous barriers to script collection. We also show that there is still significant room for improvement toward human level performance. Together, our tasks, dataset, and models offer a new research direction for learning script knowledge.

1 Introduction

Scripts (Schank and Abelson, 1975) represent structured commonsense knowledge about prototypical events in everyday situations/scenarios such as *bake a cake* (Figure 1). However, while scripts have been shown to help understand narratives by providing expectations, resolving ambiguity, and filling in unstated information (Chambers and Jurafsky, 2008; Modi et al., 2017, inter alia), they have proved hard to author or extract from text, with only small script databases available (Regneri et al., 2010; Chambers, 2017; Ostermann, 2020).

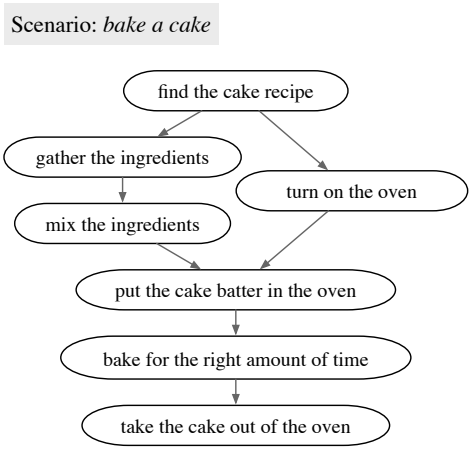

Figure 1: We collected 6.4k of partially ordered scripts (proScript) and developed models that take a scenario (e.g., bake a cake) as the input and *generate* a (possibly partial-order) script. In proScript, an event (node) requires that all the precedent events and paths are happened/executed in advance.

In this work, we show for the first time that pre-trained neural language models (LMs) can be adapted to *generate* high-quality scripts, including appropriately partial ordering events where a specific temporal ordering is required only when it is necessary. LMs have previously been shown to successfully generate stories (Rashkin et al., 2020), summaries (Lewis et al., 2020), and commonsense facts (Bosselut et al., 2019; Hwang et al., 2020). Here we investigate their application to *script generation*. First, we collect large amount (6.4k) of partially ordered script from crowdsourcing with a similar but simplified collection method (Ciosici et al., 2021). We call the dataset as proScript (PaRtial Order SCRIPT), and this is substantially larger and more diverse than prior (crowdsourced) dataset such as DeScript (Regneri et al., 2010) that has 40 scripts. In proScript, all the events/paths need to be happened/executed (cf. AND arcs in AND/OR graphs), whereas prior work on scripts

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 2138–2149
November 7–11, 2021.

do not distinct core and optional/alternative events explicitly. Additionally, temporal duration of each event is annotated (e.g., *take the cake out of the oven* typically takes one minute in the *bake a cake* script), which will potentially link script knowledge with temporal reasoning in future work.[1]

Second, with the collected data, we introduce two complementary tasks: **script edge prediction** and **entire script generation**. In the edge prediction task, given a scenario and unordered intermediate events, models must organize the events as a valid partial-order script. On the other hand, the script generation task is to generate intermediate events and the partial-order of those events for a given scenario. This task requires both natural language generation (for nodes) and graph structure prediction (for edges).

Finally, based on our proposed dataset, we develop models for both edge prediction and entire script generation tasks. As Chambers (2017) has revealed that models trained and evaluated on missing events prediction (i.e., *narrative cloze*) are insufficient to assess script knowledge, our evaluation scheme evaluate the entire script. We compare the models against baselines, and show that our models outperform the baselines for both the edge prediction and the script generation tasks. Nonetheless, there is a significant room for improvement toward human-level performance – e.g., for edge prediction, the best model achieves 75.71 of F1 score while human achieves 89.28, and for script generation, the best model obtains a graph edit distance of 4.97 (i.e., number of human edits), while human-created scripts achieve 2.98 on average.
Our contributions are thus:

- A new dataset (`proScript`) of crowdsourced scripts that is substantially larger than prior (manually crafted) datasets
- Two complementary task definitions against `proScript`
- Two new models for these task, providing the first demonstration that generative models can be successfully applied, although it is still significantly below human levels.

2 Related Work

Script as narrative chain Mooney and DeJong (1985) and Chambers and Jurafsky (2008, inter alia) have investigated automatically inducing scripts from (unstructured) corpus. In particular, Chambers and Jurafsky (2008) introduced scripts as *narrative chain*, where verbs with the participants information (e.g., (*claimed, subj*), and (*accused, obj*)) named *narrative events* are partially ordered according to causal and temporal relations. They also introduced *narrative cloze* task, where a model is expected to predict one removed narrative event, given all the other narrative events, while our proposed task requires to *generate* scripts as a partial-order graph for a given scenario. The "script as narrative chain" approach has been actively studied (Jans et al., 2012; Modi and Titov, 2014; Pichotta and Mooney, 2014; Rudinger et al., 2015; Granroth-Wilding and Clark, 2016; Weber et al., 2018; Belyy and Van Durme, 2020), but it has its drawbacks. First, the source corpora is mainly from a news domain rather than everyday scenarios, and induced narrative chains contain a number of non-script events such as reporting verbs (Mostafazadeh et al., 2016; Chambers, 2017). Second, events are highly abstracted as tuples of verb and the dependency (*subj* or *obj*) (Ostermann, 2020). Third, the evaluation scheme for the narrative cloze task is insufficient to evaluate script knowledge (Chambers, 2017).

Script as paraphrase sets *Script as paraphrase sets* (Regneri et al., 2010; Modi et al., 2016; Wanzare et al., 2016) is more recent approach to gather script knowledge, where crowd workers are asked to write down a sequence of events for a given everyday scenario (e.g., bake a cake) and the collected sequences (called event sequence description) are aligned with paraphrased events being clustered. The collected (partially ordered) scripts cover wide variety of everyday situations compared to narrative chains (news domain), but one shortcoming of this approach is the scalability; it is not easy to scale because of the cost for manual data collection (Chambers, 2017; Ostermann, 2020). In fact, Modi et al. (2016) crowdsourced 1000 stories that cover only 10 scripts, and similarly Regneri et al. (2010) end up with collecting 40 scripts. The limited amount of data hinders learning script knowledge by models. Furthermore, they provide no evaluation metric on the dataset for assessing model's script knowledge.

Story generation and tracking state changes Neural models have been demonstrated to successfully generate stories (Kiddon et al., 2016; Peng et al., 2018; Zhai et al., 2019; Rashkin et al., 2020)

[1]The dataset and code are available at `https://proscript.allenai.org/`

as well as tracking state changes in procedural texts (Henaff et al., 2017; Bosselut et al., 2018; Dalvi et al., 2018; Tandon et al., 2020). Our work is related in terms of generating higher-level agenda (or plot) of a story and understanding latent preconditions and effects between events. However, a main difference between these studies and *scripts* is that story generation and state change tracking explicitly generate and/or predict character's mental states and entity's physical attributes (e.g., temperature), whereas *scripts* focuses on essential *core events* (Chambers, 2017) in partial order.

3 Definitions

`proScript` We define `proScript` as a directed acyclic graph (DAG), $G(V, E)$ with a given scenario (s), where V is a set of essential events $\{v_1, ...v_i, ...v_{|V|}\}$ and E is a set of temporal ordering constraints between events $\{e_{ij}\}$ which means that the events v_i must precede the event v_j ($v_i \prec v_j$).[2] DAGs effectively encode the partial-ordering of core events–crucial for representing events which can be performed in any order. For example, in a *bake a cake* scenario, one can "gather the ingredients" and "turn on the oven" in any order (Figure 1). We emphasize that scripts should not include non-core events such as discourse related events (e.g., reporting verbs) as Chambers (2017) proposed. In `proScript`, we also exclude alternative events in a `proScript` DAG. For example, in a *bake a cake* scenario, "get ingredients" and "buy ingredients" are alternative events with each other because either one is only necessary in the scenario. By excluding alternative events, we can resolve ambiguity of the edges in partial order structure as temporal relations or alternative paths. Regneri et al. (2010) and Modi et al. (2016) do not discriminate this ambiguity.[3]

With the definition, we introduce `proScript` task in two complementary settings: script edge prediction and entire script generation.

Edge Prediction The script edge prediction task is to predict a set of partial-ordered edges (E) of the script $G(V, E)$, given a scenario and a set of unordered intermediate events $v \in V$.

[2]Technically, `proScript` is a transitive reduction of a DAG. In short, transitive reduction of G does not have any short cut edges between nodes. In `proScript`, we add a single *root* node (v_r) and scenario (s) as a unique leaf node.

[3]We focus on events and the partial-ordering for the *protagonist* (Chambers and Jurafsky, 2008), and leave the identification of other participants for future work.

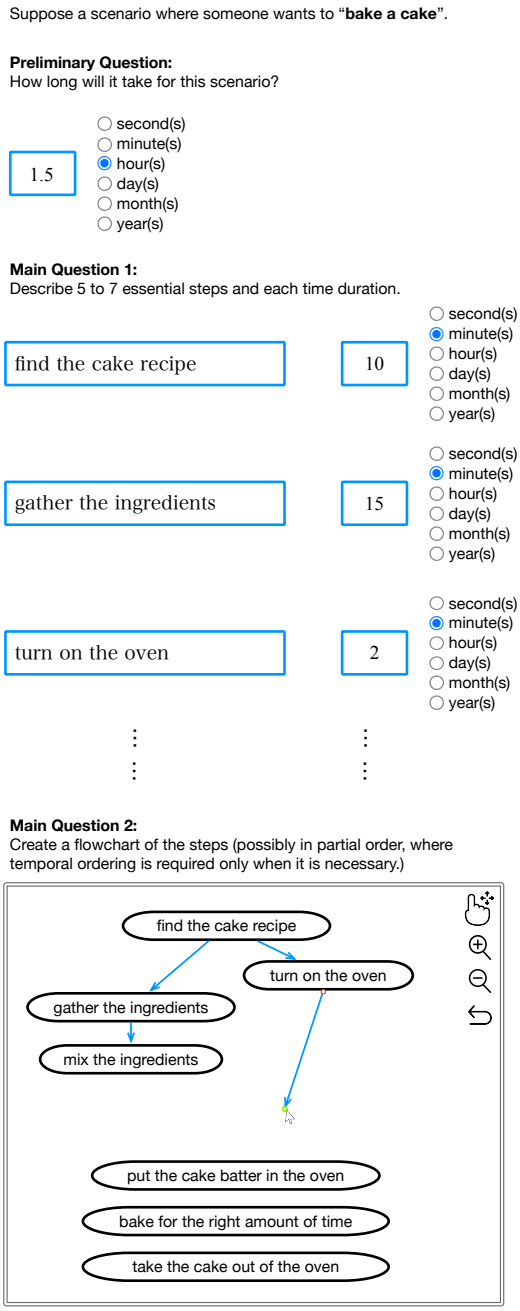

Figure 2: Annotation procedure for `proScript` data collection.

Script Generation The script generation task is to predict a partial order script $G(V, E)$, but only the scenario is given. Models are additionally expected to *generate* events (V) in natural language.

4 Datasets

Source of Scenarios We collected scenarios from DeScript (Wanzare et al., 2016), VirtualHome (Puig et al., 2018), and ROCStories (Mostafazadeh et al., 2016). DeScript consists of 40 daily scenarios (e.g., making coffee) and we

use all of them. VirtualHome is constructed to learn activities interactively in a household in a 3D simulated world. It has 233 indoor tasks (e.g., turn on light) and we include them as scenarios. Since these two datasets have only small amount of scenarios, we additionally extracted phrases for scenarios from ROCStoreis (Mostafazadeh et al., 2016), by manually curating patterns with *want(ed) to ...* (e.g., go to Hawaii), *need(ed) to ...* (e.g, get a haircut) and *look(ing) to* (e.g, buy a television). The scenarios we collected from ROCStories include both high-level long-term ones (e.g., open a small business) and fine-grained short-term ones (e.g., sign into an email account).

Crowdsourcing `proScript` For the collected scenarios, we crowdsource the corresponding `proScript` on the Amazon Mechanical Turk. Our crowdsourcing procedure (Figure 2) is similar but simplified method to (Ciosici et al., 2021). First, given a scenario (e.g., bake a cake), each crowdworker is required to describe five to seven *core events* that they are essential for the scenario (Chambers, 2017) with the estimated time it takes to complete each event.[4] In the second question, crowdworkers confirm the set of steps and they are asked to create a flowchart (DAG) by connecting the steps possibly in partial order. When crowdworkers make a submission, validation function is executed to check if the created flowchart is a valid (transitive reduction of) DAG that does not contain a cycle/loop and any short cut edge.

Due to the complex nature of this crowdsourcing procedure, it is crucial to maintain the quality. To filter out noisy scripts and resolve conflicts, two different workers are asked to sort the same set of events in partial order (i.e., the same as the second question described above),[5] and we filter out scripts (DAGs) that have low agreement.[6] To collect `proScript` with both micro and macroscopic scenarios, we iteratively picked events in the DAGs and use them as an additional source of finer-grained scenarios. For example, *turn on the oven* is a new fine-grained scenario derived from *bake a cake*.

[4]We set the number around 5-7 to balance the cognitive load on the crowdworkers and to stay within budget. We found this number to be a good balance given the spectrum of granularity in our dataset.

[5]In our crowdsourcing tasks, we maintained a pay rate of 12$/hr or higher. For example, crowd workers were paid $0.8 for the script creation and $0.4 for the validation.

[6]Technically, we compute F1 scores between the DAGs (§5.3) and set a threshold to filter out. Our cutoff F1 is 65, which we arrived at based on manual analysis. We keep the script with the highest F1 (break ties by a random coin toss).

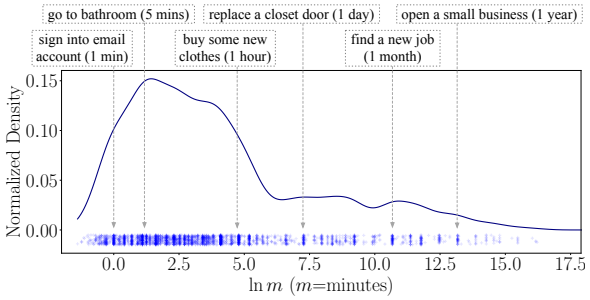

Figure 3: Normalized histogram of time duration in `proScript` dataset. We see the dataset contains scripts with various time granularity.

Dataset Statistics In total, we collected 6,414 valid scripts that include 311,502 pairs of events, and we split the `proScript` into training (3,252 scenarios), development (1,085), and test set (2,077). The training and development sets consist of scenarios collected from ROCStories, and the test set consists of those from ROCStories, DeScript, and VirtualHome. This helps us evaluate in- and out-of-domain performance.

	train	dev	test (in)	test (out)
source	ROC	ROC	ROC	DeScript VirtualHome
scenarios	3,252	1,085	1,106	971

The average number of events in `proScript` scenarios is 5.45 and the maximum degrees of DAGs in the training set are distributed as follows: 2,198 scripts (67.6%) for degree 1, 915 scripts (28.1%) for degree 2, 108 scripts (3.3%) for degree 3, 31 scripts (0.9%) for degree 4 and above.

Figure 3 shows the normalized histogram of the typical time to take for each script in `proScript` dataset. Most of the scripts take between a minute and an hour (e.g., "go to bathroom", "buy some new clothes"), while there are a reasonable amount of high-level long-term scripts (e.g., "find a new job", "open a small business").

5 `proScript` Edge Prediction

5.1 Models

For the `proScript` edge prediction task (§3), we implement a two-step approach baseline (*pairwise model*) and compare it with our proposed end-to-end neural method ($\texttt{proScript}_{\text{edge-pred}}$).

Pairwise Model We implement a two-step baseline where we train a binary classifier to predict the

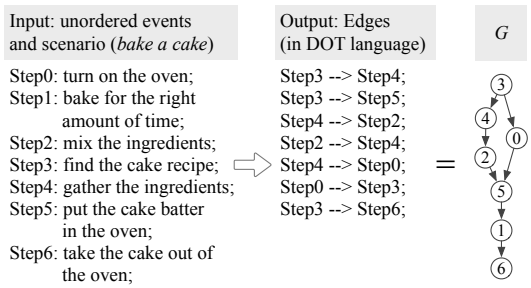

Figure 4: Example of input and output for the $\texttt{proScript}_{\text{edge-pred}}$ model. The input is a flattened sequence of events, and the output is a flattened sequence of edges of the (predicted) partial-order script in DOT language (Gansner et al., 1993).

precedence between pairs of events, followed by building a partial order script G by aggregating the predicted relations across all pairs of events.

Formally, the classifier takes a pair of events (v_i, v_j) and predicts the precedence e_{ij} – i.e. the event v_i precedes ($\prec$) v_j .

$$e_{ij} = p(v_i \prec v_j | v_i, v_j) \quad (1)$$

Scores by the classifier are used as weights to create an adjacency matrix of G which is then automatically converted into a partial-order script with heuristics – when G contains a cycle, we iteratively remove edges by choosing the one with minimum weight until we get a valid DAG.

$\texttt{proScript}_{\text{edge-pred}}$ We propose an end-to-end neural model, which takes all the (unordered) events (v) and the scenario (s) as the input (x) and predicts the edges (E) in a partial-order script (G) at one time. To represent E in a linear format (y), we use DOT, a graph description language (Gansner et al., 1993) as shown in Figure 4.[7] By flattening the nodes and edges of G, we apply neural encoder-decoder models. Formally, flattened unordered events and scenario as x are embedded as continuous representation (emb(x)) by the encoder, then the decoder will generate tokens (y) as follows:

$$p(y_1, \ldots, y_N | x_1, \ldots, x_M) = \quad (2)$$
$$\prod_{n=1}^{N} p(y_n | \text{emb}(x_1, \ldots, x_M), y_1, \ldots, y_{n-1}).$$

Compared to the pairwise model, the $\texttt{proScript}_{\text{edge-pred}}$ model uses information from all the events jointly to build partial-order script with a broader context.

[7] Madaan and Yang (2020) have previously shown that finetuned LMs can generate valid DOT language.

5.2 Evaluation Metrics

Given $\hat{G}(V, \hat{E})$ as a predicted (partial order) script and $G(V, E)$ as the correct (oracle) script, the F1 score is defined as follows:

$$\text{Precision} = \frac{|E \& \hat{E}|}{|E|}, \text{ Recall} = \frac{|E \& \hat{E}|}{|\hat{E}|}$$
$$\text{F1 score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}.$$

For evaluating human performance, we show randomly shuffled steps to crowdworkers and ask them to create a partial-order script. We compute the F1-score of the script with the reference script.

5.3 Experiments

Setup For the binary classifier (pairwise model), we use two variants of the Transformer (Vaswani et al., 2017): RoBERTa-large (Liu et al., 2019) and T5-11B (Raffel et al., 2020).

When training (i.e., fine-tuning) RoBERTa,[8] we use a grid-search for choosing the best hyper-parameters from the best performed model on the development set: epochs {1, 2, 3}, learning rate {1e-5, 1e-6, 1e-7}, batch size {16, 24, 32}. For training the T5 model as the pairwise model, we followed a default set of hyper-parameters that are recommended in Raffel et al. (2020).[9]

For the $\texttt{proScript}_{\text{edge-pred}}$ model, we use the T5 with different model sizes (Large and 11B) and training sizes (100, 1k, and all 3.2k) to see how these factors affect the performance.[10] We followed a default set of hyper-parameters for the T5 models.

Results The results are shown in Table 1. We find that the pairwise and $\texttt{proScript}_{\text{edge-pred}}$ models significantly outperform the random baseline where the edges are randomly assigned. The $\texttt{proScript}_{\text{edge-pred}}$ T5-11B model outperforms the pairwise T5-11B model. This indicates that the $\texttt{proScript}_{\text{edge-pred}}$ model benefits from a larger context from the input to predict edges more accurately, although there is still a significant room for improvement toward human-level performance.[11] Regarding the difference between

[8] We used the implementation from Huggingface Transformers (Wolf et al., 2019).

[9] https://github.com/google-research/text-to-text-transfer-transformer

[10] We also used BART (Lewis et al., 2020) and GPT2 (Radford et al., 2019) as the baselines, but we found that both failed to generate canonical DOT language.

[11] We find that 99% of the outputs from $\texttt{proScript}_{\text{edge-pred}}$ are valid DOT language.

	dev			test (all)			test (in domain)			test (out domain)		
Models	F1	P	R	F1	P	R	F1	P	R	F1	P	R
random	21.30	21.08	21.72	21.03	21.00	21.26	20.57	20.52	20.84	21.32	21.27	21.58
Pairwise (RoB)	65.75	67.05	64.71	61.29	62.85	60.06	63.25	64.97	61.89	59.06	60.44	57.98
Pairwise (T5)	70.96	71.93	69.76	67.64	69.44	66.18	69.50	71.41	67.96	65.51	67.20	64.16
proScr(11B-100)	56.05	56.58	55.75	52.26	52.91	51.89	54.98	55.67	54.59	49.16	49.76	48.83
proScr(11B-1k)	65.98	66.49	65.71	60.55	61.24	60.15	64.64	65.40	64.20	55.89	56.51	55.54
proScr(L-all)	66.25	66.89	65.83	63.64	64.22	63.27	65.76	66.38	65.35	61.23	61.76	60.91
proScr(11B-all)	**78.20**	78.48	78.14	**75.71**	75.93	75.72	**77.75**	78.03	77.71	**73.37**	73.54	73.46
Human	89.32	89.60	89.21	89.28	89.91	88.86	90.04	90.54	89.74	88.71	89.44	88.18

Table 1: Results for proScript edge prediction task. In this table, proScript refers to $\text{proScript}_{\text{edge-pred}}$.

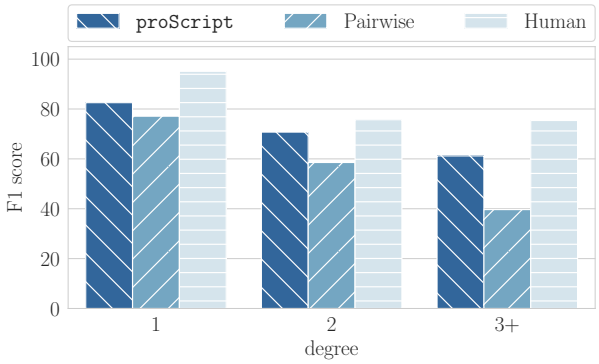

Figure 5: Model performance by pairwise (T5-11B), $\text{proScript}_{\text{edge-pred}}$ (T5-11B) and human according to the maximum degree of the script (DAG).

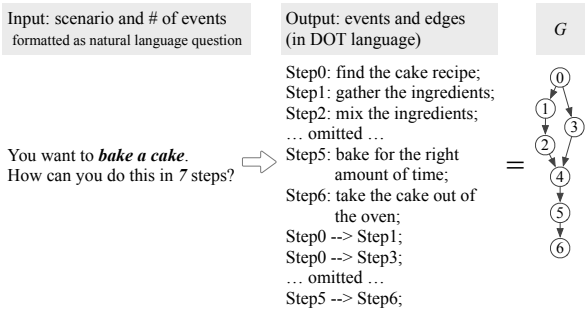

Figure 6: Example of input and output for $\text{proScript}_{\text{gen}}$. The input is a scenario and number of events to generate in natural text format, and the output is a sequence of events and edges of the script.

in and out of domain, we find that the in-domain performance is higher than the out-of-domain performance, whereas human performance is robust regardless of the domain difference. We also see that the training set (100, 1k, all) and model sizes (Large, 11B) significantly affect the performance of $\text{proScript}_{\text{edge-pred}}$.

Figure 5 shows the performance of the pairwise (T5-11B) model, $\text{proScript}_{\text{edge-pred}}$ (T5-11B) and human according to the (maximum) degree of the script DAGs. We find that scripts with higher degree are more difficult to predict for both $\text{proScript}_{\text{edge-pred}}$ and pairwise models, whereas human shows smaller decrease for predicting higher-degree scripts.

6 proScript Generation

6.1 Models

$\text{proScript}_{\text{gen}}$ The proScript generation task combines natural language generation (i.e. generating events in natural language) with graph structure prediction over the generated events (i.e. organizing the events into a DAG). Our approach ($\text{proScript}_{\text{gen}}$) is to formulate it as an end-to-end problem, similar to the $\text{proScript}_{\text{edge-pred}}$ for the proScript edge prediction task (§5.1).

Given a scenario (s) and the number of events to generate in the script, $\text{proScript}_{\text{gen}}$ generates events and edges for the partial-order script (G) in DOT language (Figure 6). Formally, we use the same encoder-decoder framework (eq.2) except that a scenario and number of steps to generate are described in natural text as x and the decoder is expected to *generate* both events and the edges (as y) in the script jointly.

Transfer learning from WikiHow data Transfer learning often helps improve the performance when it is (pre-)trained on a similar task (Peters et al., 2018; Devlin et al., 2019). As additional resource for pre-training $\text{proScript}_{\text{gen}}$, we use procedural texts extracted from WikiHow,[12] which contains 130k instances of a sequence of essential steps for a given topic in various categories (e.g., health, finance, hobbies, etc.). It is important to note that all the procedures in WikiHow are formatted as *sequences* rather than a partial-order, and therefore the model is biased towards generating sequences. We refer to this approach as $\text{proScript}_{\text{gen-transfer}}$.

Pipeline approach An alternative approach is to use $\text{proScript}_{\text{gen}}$ followed by the

[12] https://www.wikihow.com/

proScript$_{\text{edge-pred}}$ model. The approach relies on proScript$_{\text{gen}}$ to generate a set of events but allows to fix the predicted edges via the proScript$_{\text{edge-pred}}$ model. We refer to this approach as proScript$_{\text{gen-pipe}}$, and study whether it can improve the performance over proScript$_{\text{gen}}$.

6.2 Evaluation Metrics

Chambers (2017) emphasizes the importance of human annotation for evaluating script knowledge. However, human evaluation for the proScript generation task is challenging because it involves natural text generation and graph structure prediction. As in the text generation tasks such as machine translation and text summarization, there are several possible correct answers. Therefore, we use two complementary evaluation metrics for the proScript generation task: (i) graph edit distance, and (ii) pairwise comparison. These are the absolute and relative measures of performance, respectively. Graph edit distance (Abu-Aisheh et al., 2015) computes the distance between two graphs. Formally, given two graphs G_1 and G_2,

$$\text{GED}(G_1, G_2) = \min_{G_1 \xrightarrow{d_1,\ldots,d_k} G_2} \sum_{i=1}^{k} \text{cost}(d_i) \quad (3)$$

where $d_1, \ldots, d_k$ is a list of graph edit operations from G_1 to G_2. The operations include deletion, insertion, and replacement for vertex and edge. Each operation has its cost and we set the cost to be 1 for all the operations in our evaluation for simplicity. We use an averaged graph edit distance between a model-generated script and the revised scripts by two human annotators. For evaluating human performance, a crowdworker writes a partial-order script, given a scenario. Then, similarly to the model evaluation, two human annotators are asked to revise the partial-order script, and we take the average of the two graph edit distances.

The graph edit distance is indicative of the quality of the generated scripts; higher-quality scripts must have smaller graph edit distances to the gold-standard (i.e. they require a smaller number of human revisions).

For the relative measure, we employ pairwise human judgments where we ask annotators to *compare* a pair of scripts generated by proScript$_{\text{gen}}$ with those from the other approaches.

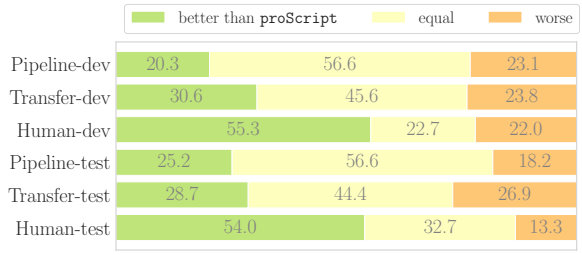

Figure 7: Pairwise judgments (%) between proScript$_{\text{gen}}$ and the other approaches.

6.3 Experiments

Setup For our proScript$_{\text{gen}}$, we use T5-11B. Similarly to the proScript$_{\text{edge-pred}}$, we follow the default set of hyper-parameters recommended in (Raffel et al., 2020). For proScript$_{\text{gen-transfer}}$, we pre-train the proScript$_{\text{gen}}$ with the 130k procedures, and finetune it on the proScript dataset. For the proScript$_{\text{gen-pipe}}$, we first obtain the actions generated by proScript$_{\text{gen}}$ (ignoring the edges), and use the set of events as input for proScript$_{\text{edge-pred}}$, which is trained (see §5.3) to predict the edges.

As defined in §3, we use graph edit distance and pairwise judgments to evaluate the quality of the generated scripts. For computing graph edit distances, we select 500 scripts (250 for dev and test sets) and ask crowdworkers to revise the generated scripts as necessary (e.g., add/delete/replace the events and the edges). We use the revised scripts as gold-standard. Each script is revised by two annotators, and we compute the average of the graph edit distances.

In pairwise judgments, we compare the scripts generated by proScript$_{\text{gen}}$ with those from the other approaches. We randomly select 150 pairs, and ask three crowdworkers to judge whether the script generated by proScript$_{\text{gen}}$ is *better*, *worse*, or *equal to* the other (i.e. transfer, pipeline, or human). We use majority vote to decide the final pairwise human judgment between the two scripts.

Results The pairwise judgment result is shown in Figure 7. We see that the pipeline and transfer models show slight preference over the proScript$_{\text{gen}}$ (except pipeline-dev), although the difference is not large. We also see that the transfer model constantly have more preference over the proScript$_{\text{gen}}$ than the pipeline model in both dev and test sets. Regarding the pairwise comparison with human-created plans, proScript$_{\text{gen}}$ still has a significant room for im-

Split	Models	Graph Edit Dist	V-Del	V-Ins	V-Rep	E-Del	E-Ins	E-Rep
dev	proScript$_{gen}$	**4.73**	0.426	0.192	0.581	1.558	1.308	0.671
	proScript$_{gen\text{-}transfer}$	4.79	0.337	0.195	0.679	1.491	1.281	0.775
	proScript$_{gen\text{-}pipe}$	4.88	0.397	0.159	0.560	1.705	1.407	0.661
	Human	2.78	0.155	0.161	0.144	1.123	1.011	0.199
test	proScript$_{gen}$	**4.97**	0.581	0.142	0.656	1.668	1.184	0.709
	proScript$_{gen\text{-}transfer}$	5.38	0.438	0.213	0.775	1.713	1.402	0.835
	proScript$_{gen\text{-}pipe}$	5.41	0.594	0.143	0.671	1.880	1.292	0.787
	Human	2.98	0.168	0.149	0.130	1.276	1.074	0.189
test (in domain)	proScript$_{gen}$	**4.57**	0.513	0.158	0.633	1.471	1.108	0.687
	proScript$_{gen\text{-}transfer}$	5.03	0.339	0.299	0.649	1.575	1.496	0.677
	proScript$_{gen\text{-}pipe}$	5.10	0.561	0.147	0.630	1.765	1.217	0.744
	Human	3.03	0.168	0.211	0.154	1.223	1.091	0.206
test (out domain)	proScript$_{gen}$	**5.43**	0.659	0.124	0.681	1.894	1.270	0.735
	proScript$_{gen\text{-}transfer}$	5.76	0.549	0.115	0.916	1.867	1.296	1.013
	proScript$_{gen\text{-}pipe}$	5.81	0.659	0.116	0.795	1.961	1.267	0.941
	Human	2.91	0.170	0.074	0.102	1.340	1.054	0.170

Table 2: Results for proScript generation task (dev, test, in-domain test and out-of-domain test set). We measure the average graph edit distance between generated script and the two human revisions (lower the better). We also show the average number of each graph edit operation ({Delete, Insert, Replace} $\times$ {Vertex, Edge}). Random (edge) baseline shows 11.06 edit distance for the dev set and 10.95 for the test set.

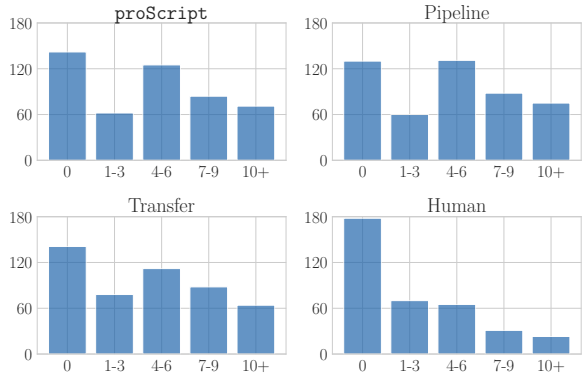

Figure 8: Histograms of graph edit distance (in dev set). The number of scripts (y-axis) according to the (binned) graph edit distance (x-axis).

provement toward human level.

Table 2 shows the average graph edit distance between the generated script and the human revisions. We find that neither transfer nor pipeline help to improve the graph edit distance over proScript$_{gen}$, indicating that proScript$_{gen}$ is already a strong baseline (see examples in Appendix). The reason of no improvement by the transfer approach may be because WikiHow consists of sequences rather than partially ordered steps. No improvement by the pipeline approach indicates that the proScript$_{gen}$ can directly generate valid script in both events and edges. Further studies for improvements are needed for future work.

Graph edit distance (Table 2) is absolute evaluation, and it objectively treats each graph edit equally. Pairwise comparison (Figure 7) is relative evaluation comparing generated scripts with the script quality/goodness being considered by human annotators. These two metrics produce slightly different results but this is not strictly a contradiction, as they are measuring different things.

In terms of the edit types, many of the edits are edge-related, suggesting that proScript$_{gen}$ and the variants are all good at generating events but struggles with ordering them. Regarding in- and out-of domains in the test sets, we observe that proScript$_{gen}$ and the variants have slightly better performance for in-domain scripts than out-of domain, while human created scripts are not affected by domains. These findings are consistent with the result in the edge prediction task (§5.3).

Figure 8 shows a histogram of the graph edit distance. It is evident that human created scripts are corrected less often than scripts generate by proScript$_{gen}$, whereas the scripts from proScript$_{gen}$ and the variants often have a large number of edits (e.g., 4 or more). It is interesting to see that fewer number of scripts have 1 to 3 edits (except scripts created by human). The reason is because one simple revision tends to yield multiple graph edits (e.g., one node insertion yields multiple edge insertions).

Error Analysis We performed manual error analysis for the scripts generated by each model. We selected 40 random scripts that have non-zero graph edit distance and classified the human revisions into

Revision types	generated script (subgraph)	revised script (subgraph)
missing event	wait for the plane → exit the plane	wait for the plane → get on the plane → exit the plane
incorrect order	get off the car → drive to the zoo	drive to the zoo → get off the car
irrelevant or redundant event	put clothes in dryer → place clothes into dryer → dry clothes	put clothes in dryer → dry clothes
order ambiguity by context	get a visa → ... → get off the plane → trip to a foreign country	get off the plane → get a visa (on arrival) → trip to a foreign country
granularity of events	get out of the bed → go to the kitchen	get out of the bed → open the bedroom door → go to the kitchen
paraphrased	move into new apartment	move to a new apartment

Table 3: Examples for each revision type.

Revision types		proScript	Transfer	Pipeline	Human
crucial errors	(edge) incorrect order	15.79	21.62	24.32	10.00
	(node) missing event	5.26	2.70	2.70	0.00
	(node) irrelevant/redundant event	10.53	13.51	2.70	0.00
minor revisions	(edge) order ambiguity by context	31.58	32.43	40.54	33.33
	(node) granularity of events	31.58	24.32	21.62	26.67
	(node) paraphrased event	0.00	0.00	5.41	6.67
wrong revisions		5.26	5.41	2.70	23.33

Table 4: Revision type distribution (%) by each model.

7 types: (1) incorrect order of events, (2) missing event, (3) irrelevant/redundant event, (4) order ambiguity by context, (5) granularity of (core) events, (6) paraphrased event, and (7) wrong human revision/correction (examples are shown in Table 3). Approximately, the first three error types indicate that the script has crucial errors, the next three types are trivial/minor revisions where both generated and revised scripts are plausible. The last type of revision is the one where the revised script is wrong (or worse).

Table 4 shows the statistics of each error type. We see that edge-related revisions are more frequent than node-related revisions. The generated nodes are of a high quality (among all revisions by human, 10.53% of them are related to irrelevant or redundant nodes), and the majority of revisions are minor modifications. This is consistent with the results in graph edit distance. Overall, we find that minor revisions are more frequent than crucial errors, indicating that $\texttt{proScript}_{\text{gen}}$ and the variants generates reasonably good scripts. In contrast, crucial errors are quite rare in human created scripts, indicating a significant room for future innovation.

7 Conclusions

We show for the first time that pre-trained neural language models can be adapted to *generate* partial order scripts. We collect 6,400 partially ordered script from crowdsourcing (proScript), which is substantially larger than prior manually crafted datasets. With the proScript dataset, we introduced two complementary task and models, which combine language generation and graph structure prediction, providing the first demonstration that generative models can be successfully applied to script generation, although it is still below human performance. We believe that proScript dataset and models would advance future work on various NLP tasks such as story generation, machine comprehension, temporal reasoning, and high-level planning.

References

Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. 2015. An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems. In *4th International Conference on Pattern Recognition Applications and Methods 2015*, Lisbon, Portugal.

Anton Belyy and Benjamin Van Durme. 2020. Script induction as association rule mining. In *Proceedings of the First Joint Workshop on Narrative Understanding, Storylines, and Events*, pages 55–62, Online. Association for Computational Linguistics.

Antoine Bosselut, Corin Ennis, Omer Levy, Ari Holtzman, Dieter Fox, and Yejin Choi. 2018. Simulating action dynamics with neural process networks. In *International Conference on Learning Representations*.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Çelikyilmaz, and Yejin Choi. 2019. Comet: Commonsense transformers for automatic knowledge graph construction. In *Proceed-*

ings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL).

Nathanael Chambers. 2017. Behind the scenes of an evolving event cloze test. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pages 41–45, Valencia, Spain. Association for Computational Linguistics.

Nathanael Chambers and Dan Jurafsky. 2008. Unsupervised learning of narrative event chains. In *Proceedings of ACL-08: HLT*, pages 789–797, Columbus, Ohio. Association for Computational Linguistics.

Manuel R. Ciosici, Joseph Cummings, Mitchell DeHaven, Alex Hedges, Yash Kankanampati, Dong-Ho Lee, R. Weischedel, and Marjorie Freedman. 2021. Machine-assisted script curation. *ArXiv*, abs/2101.05400.

Bhavana Dalvi, Lifu Huang, Niket Tandon, Wen-tau Yih, and Peter Clark. 2018. Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1595–1604, New Orleans, Louisiana. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *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. Association for Computational Linguistics.

Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem phong Vo. 1993. A technique for drawing directed graphs. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 19(3):214–230.

Mark Granroth-Wilding and Stephen Clark. 2016. What happens next? event prediction using a compositional neural network model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Mikael Henaff, J. Weston, Arthur D. Szlam, Antoine Bordes, and Y. LeCun. 2017. Tracking the world state with recurrent entity networks. *ArXiv*, abs/1612.03969.

Jena D Hwang, Chandra Bhagavatula, Ronan Le Bras, Jeff Da, Keisuke Sakaguchi, Antoine Bosselut, and Yejin Choi. 2020. Comet-atomic 2020: On symbolic and neural commonsense knowledge graphs. *arXiv preprint arXiv:2010.05953*.

Bram Jans, Steven Bethard, Ivan Vulić, and Marie Francine Moens. 2012. Skip n-grams and ranking functions for predicting script events. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 336–344, Avignon, France. Association for Computational Linguistics.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339, Austin, Texas. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Aman Madaan and Yi-Ming Yang. 2020. Neural language modeling for contextualized temporal graph generation. *ArXiv*, abs/2010.10077.

Ashutosh Modi, Tatjana Anikina, Simon Ostermann, and Manfred Pinkal. 2016. InScript: Narrative texts annotated with script information. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3485–3493, Portorož, Slovenia. European Language Resources Association (ELRA).

Ashutosh Modi and Ivan Titov. 2014. Inducing neural models of script knowledge. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 49–57, Ann Arbor, Michigan. Association for Computational Linguistics.

Ashutosh Modi, Ivan Titov, Vera Demberg, Asad Sayeed, and Manfred Pinkal. 2017. Modeling semantic expectation: Using script knowledge for referent prediction. *Transactions of the Association for Computational Linguistics*, 5:31–44.

Raymond J Mooney and Gerald DeJong. 1985. Learning schemata for natural language processing. In *IJCAI*, pages 681–687.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 839–849, San Diego, California. Association for Computational Linguistics.

Simon Ostermann. 2020. *Script Knowledge for Natural Language Understanding*. Ph.D. thesis, Saarland University, Germany.

Nanyun Peng, Marjan Ghazvininejad, Jonathan May, and Kevin Knight. 2018. Towards controllable story generation. In *Proceedings of the First Workshop on Storytelling*, pages 43–49, New Orleans, Louisiana. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Karl Pichotta and Raymond Mooney. 2014. Statistical script learning with multi-argument events. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 220–229, Gothenburg, Sweden. Association for Computational Linguistics.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Hannah Rashkin, Asli Celikyilmaz, Yejin Choi, and Jianfeng Gao. 2020. PlotMachines: Outline-conditioned generation with dynamic plot state tracking. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4274–4295, Online. Association for Computational Linguistics.

Michaela Regneri, Alexander Koller, and Manfred Pinkal. 2010. Learning script knowledge with web experiments. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 979–988, Uppsala, Sweden. Association for Computational Linguistics.

Rachel Rudinger, Pushpendre Rastogi, Francis Ferraro, and Benjamin Van Durme. 2015. Script induction as language modeling. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1681–1686, Lisbon, Portugal. Association for Computational Linguistics.

Roger C. Schank and Robert P. Abelson. 1975. Scripts, plans and knowledge. In *IJCAI*, pages 151–157.

Niket Tandon, Keisuke Sakaguchi, Bhavana Dalvi, Dheeraj Rajagopal, Peter Clark, Michal Guerquin, Kyle Richardson, and Eduard Hovy. 2020. A dataset for tracking entities in open domain procedural text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6408–6417, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Lilian D. A. Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. 2016. A crowdsourced database of event sequence descriptions for the acquisition of high-quality script knowledge. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3494–3501, Portorož, Slovenia. European Language Resources Association (ELRA).

Noah Weber, Leena Shekhar, Niranjan Balasubramanian, and Nathanael Chambers. 2018. Hierarchical quantized representations for script generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3783–3792, Brussels, Belgium. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Fangzhou Zhai, Vera Demberg, Pavel Shkadzko, Wei Shi, and Asad Sayeed. 2019. A hybrid model for globally coherent story generation. In *Proceedings of the Second Workshop on Storytelling*, pages 34–45, Florence, Italy. Association for Computational Linguistics.

A Appendix

A.1 Reproducibility

For training RoBERTa-large as a pairwise model, we use Quadro RTX 8000 (48GB memory), which takes around 4.5 hours to train a model. RoBERTa-large consists of 355M parameters with 24 layers, 1,024 of hidden embedding size, and 16 of the attention heads. T5-large model has 770M parameters with 24-layers, 1024-hidden-state, 4096 feed-forward hidden-state, and 16 attention heads. T5-11B models has 11B parameters with 24-layers, 1024-hidden-state, 65,536 feed-forward hidden-state, 128 attention heads. We use TPU (v3-8) on google cloud platform. It takes 3 hours in average to train a edge prediction model, and 5 hours for plan generation models.

A.2 Plans generated by $\texttt{proScript}_{\textbf{gen}}$

We show some example scripts generated by $\texttt{proScript}_{\text{gen}}$ in Figure 9. In each example, $\texttt{proScript}_{\text{gen}}$ which takes scenario and the number of steps as the input (e.g., *play the organ*, in 5 steps) and generates a script DAG.

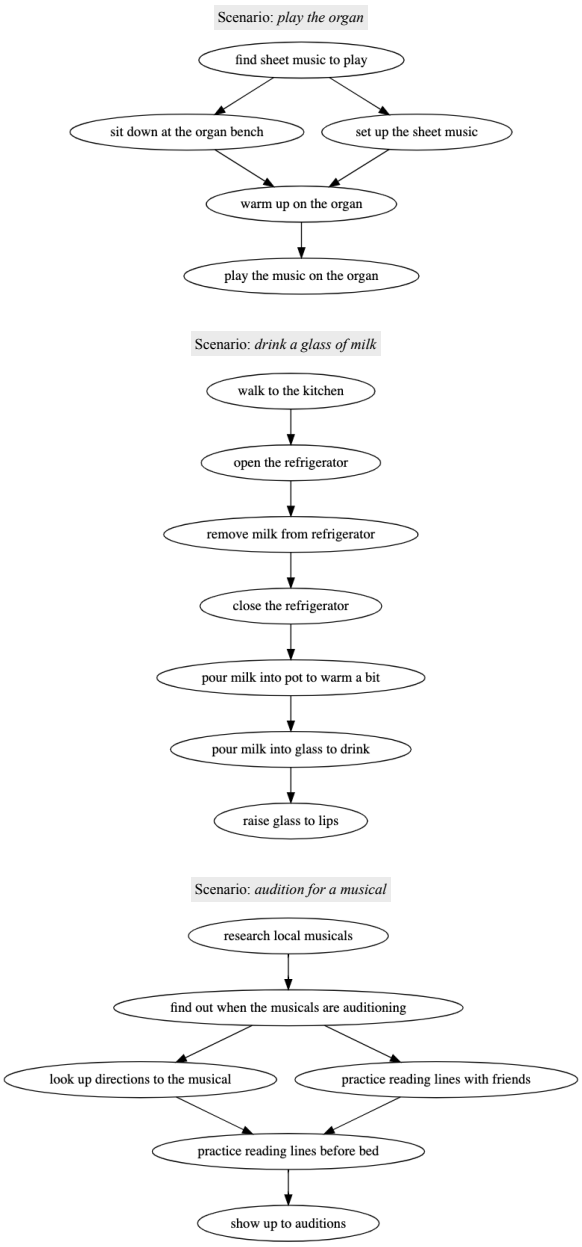

Figure 9: Example scripts generated $\texttt{proScript}_{\text{gen}}$.