Zero-Shot Script Parsing

Fangzhou Zhai, Vera Demberg and **Alexander Koller**
Dept. of Language Science and Technology
Saarland Informatics Campus
Saarland University
{fzhai, vera, koller}@coli.uni-saarland.de

Abstract

Script knowledge (Schank and Abelson, 1977) is useful for a variety of NLP tasks. However, existing resources only cover a small number of activities, limiting its practical usefulness. In this work, we propose a zero-shot learning approach to **script parsing**, the task of tagging texts with scenario-specific event and participant types, which enables us to acquire script knowledge without domain-specific annotations. We (1) learn representations of potential event and participant mentions by promoting class consistency according to the annotated data; (2) perform clustering on the event / participant candidates from unannotated texts that belongs to an unseen scenario. The model achieves 68.1/74.4 average F1 for event / participant parsing, respectively, outperforming a previous CRF model that, in contrast, has access to scenario-specific supervision. We also evaluate the model by testing on a different corpus, where it achieved 55.5/54.0 average F1 for event / participant parsing.

1 Introduction

Script knowledge is a type of commonsense knowledge that captures how people conduct everyday activities (Schank and Abelson, 1977). It expresses that in a certain **scenario**, **participants** tend to act out **events** in a certain order; an example from the scenario FIXING A FLAT TIRE is shown in Fig. 1. Humans use script knowledge to fill in events that are not explicitly mentioned in a text, and script knowledge is useful for many downstream NLP applications, including referent prediction (Ahrendt and Demberg, 2016; Modi et al., 2017), discourse classification (Lee et al., 2020), and story generation (Zhai et al., 2019, 2020).

A key challenge in dealing with script knowledge is coverage: it is costly and time-consuming to spell out the prototypical events and participants of a scenario and how they can be expressed in language. Existing resources are mostly crowd-sourced (Regneri et al., 2010; Modi et al., 2016); they annotate stories from a limited number of scenarios with script events and participants (cf. figure 1). **Script parsers**, which predict these event and participant labels given a text, can achieve high accuracies on scenarios that were seen in training (Ostermann et al., 2017; Zhai et al., 2021). Nonetheless, these parsers only operate on known scenarios: they could only predict event types on the same scenario as they were trained. The limited coverage restricts the practical usefulness of script parsers and thus the practical usefulness of script knowledge for downstream tasks in general.

In this paper, we acquire script knowledge by tackling the task of **zero-shot script parsing**: we present the first system which accurately performs

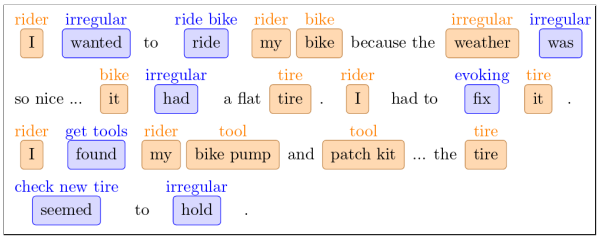

Figure 1: A story about FIXING A FLAT TIRE from InScript. Script parsing identifies events and participants from texts. The picture is taken from Zhai et al. (2021).

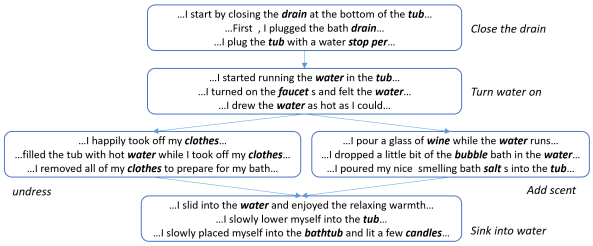

Figure 2: A part of the temporal script graph for TAKING A BATH inferred from our parsing result. Each node illustrates 3 random verbalizations from the cluster. The gold classes are shown on the side. Further edges that could be inferred by transitivity are omitted. We see one could either *undress* first or *add scent* (to the bath tub) first before *sink into water*.

Proceedings of the 29th International Conference on Computational Linguistics, pages 4049–4060
October 12–17, 2022.

script parsing on scenarios that were not seen at training time. For instance, given training data that talks about TAKING A BATH and GOING TO A RESTAURANT, the parser labels events and participants in the FIXING A FLAT TIRE story of Fig. 1, allowing us to arrange script knowledge into a graphical form as Fig. 2. This offers a way of overcoming the coverage limitations of script knowledge, by generalizing from the training scenarios to arbitrary other ones.

Our method learns to extract script-specific representations from general-purpose pretrained word embeddings, and then uses agglomerative clustering at inference time to group together natural-language phrases that refer to the same event or participant of the unseen script. Finally, we evaluate our model on MCScript, a different corpus than the training data.

Our model achieves a micro-F1 score on zero-shot event labeling of up to 68.1 and on participant labeling of up to 74.4, on par with the supervised model of Ostermann et al. (2017) that assumes training data for the same scenario. We find that our method yields script graphs with reasonable event clusters that are temporally ordered in a reasonable way; the majority of errors on event labeling are due to issues with the granularity of events. We also find in probing tasks that our model learns to amplify information about sentence ordering from the pretrained embeddings, while suppressing low-level information about morphology and syntax, which are less relevant for the script parsing task. In order to investigate its potential for practical use, we also evaluated the model on a different, unannotated corpus, where the model achieved $55.5/54.0$ average F1 for event / participant parsing.

2 Related work

Scripts were introduced as an approach to capturing commonsense knowledge in AI by Schank and Abelson (1977); see also Barr and Feigenbaum (1981). Much research in NLP has simplified the acquisition of script knowledge to identifying "event chains" in narrative text (Chambers and Jurafsky, 2008, 2009). Event chains represent typical sequences of events, each represented by one predicate, and can be learned from large corpora. Other work has followed in this tradition (Jans et al., 2012; Modi and Titov, 2014; Pichotta and Mooney, 2014; Rudinger et al., 2015).

In this paper, we instead build upon work by Regneri et al. (2010, 2011), who explicitly capture script knowledge about a given scenario in a *temporal script graph* (see Fig. 2). A TSG specifies the abstract events and participants that make up a script with their temporal ordering; each of these events and participants can be expressed in language in many different ways. A TSG is also more expressive than mere event sequences, because it encodes different manners how a scenario play out in real life. Regneri et al. learned script graphs by crowdsourcing. We instead rely on manually script-annotated corpora (Modi et al., 2016; Wanzare et al., 2016).

Trained with scenario-specific supervision, script parsing can be performed accurately. Ostermann et al. (2017) developed a linear CRF model to perform script parsing as a sequence labelling task. Zhai et al. (2021) developed a hierarchical model for supervised script parsing, making use of pre-trained contextualized word embeddings. The model learns patterns at the word level, as well as the narrative level with respective sequence models. These existing approaches are limited to scenarios for which training data is available, whereas our work focuses on unseen scenarios.

Zero-shot learning is a family of methods that establish a classifier for **unseen** classes, based on labelled data from **seen** classes. One common approach is to learn a latent representation space that all instances embed into, thus the knowledge of the source domain, encoded in the labelled training instances, could be transferred to the target domain. It tackles data scarcity in various situations, such as machine translation for low-resource languages (e.g. Pham et al., 2019; Zhang et al., 2020; Johnson et al., 2017), generation (Duan et al., 2019; Philip et al., 2020), text classification (see, e.g. Yin et al., 2019) and question answering (e.g. Banerjee and Baral, 2020).

3 Data

We work with **InScript** (Modi et al., 2016) and **MCScript** (Ostermann et al., 2018, 2019).

3.1 InScript

InScript is a crowdsourced corpus of around 100 stories about each of 10 scenarios (see Fig. 1 for an example). The authors were asked to write a story about a given scenario (such as GOING GROCERY SHOPPING) "as if to a child", step by step. InScript was then hand-annotated with event and

participant classes; it also contains coreference and dependency annotations.

3.2 Preprocessing

Following Ostermann et al. (2017), we distinguish between (1) events that are 'related to the scenario', or commonly seen in a typical instantiation of the scenario, which we call **regular events**, and (2) the ones that take place in the course of a specific story, but are not directly related to the scenario, which we call **irregular events**. For example, in Figure 1, *I found my bike pump* describes the regular event 'get tools', whereas *the weather **was** nice* is irregular. We collapse all the subclasses of irregular events[1] into a single irregular event class for each scenario. 12,902 (33.5%) event instances in InScript are regular. We also distinguish **regular participants** from **irregular participants** in a similar manner: participants like 'rain' in Fig. 1 are considered irregular to the FIXING A FLAT TIRE scenario, as they are not directly relevant to the scenario per se. Irregular participant instances take a smaller proportion of 19.6%.

3.3 MCScript

MCScript is a question answering dataset that focuses on script knowledge. Here we use its background text (not the questions), which comprises around 20 stories on each of 200 scenarios. These stories are not annotated with script events and participants; they are stylistically similar to those in InScript, as they also consist of relatively simple language and focus on explaining the scenario in detail.

We use MCScript to evaluate our model trained on InScript. To this end, we annotate 20 random scenarios from MCScript to specify script events and participants: 10 as validation set and 10 as test set. The set of labels we used is consistent with that of InScript, in that we adopt the same set of special labels (for example label **Unrel** for events and participants not relevant to the scenario). The annotation was performed by two experts. Similar preprocessing is performed to MCScript.

[1]These labels are UNREL (unrelated to the scenario), RELNSCR (related to the scenario but not a script event / participant), OTHER and UNKNOWN.

4 Zero-shot Script Parsing

4.1 Task Description

Our parser is tasked to predict event and participant annotations for a scenario that was not seen in training. Thus, our model must learn to group verbs and noun phrases from an unseen scenario into abstract events and participants, without knowing what the gold label set of events and participants are.

The basic idea of our zero-shot script parser is as follows. We will learn a transformation φ which maps pretrained general-purpose word embeddings into a representation space that is suitable for script parsing. Identifying verb tokens as *candidates* for event descriptions and noun and pronoun tokens as candidates for participant descriptions, we will train φ such that candidates that describe the same event or participant are close together in the representation space, whereas candidates for different events and participants are distant. To parse a text from an unseen scenario, we will apply φ to the word embeddings of all candidates and perform clustering to group them into events and participants.

We train and evaluate the model under a few different settings. **(1) InScript.** We split InScript into eight training, one validation, and one test scenario. During inference, the model takes the unannotated stories of the test scenario as input and labels them with events and participants that are consistent with the gold annotations. We rotate the validation / test scenario to perform a ten-fold cross-validation. **(2) MCScript.** We use all 10 scenarios in InScript as the training set, whereas validation and test are performed on the newly annotated MCScript validation set and test set.

4.2 Regular candidate identification

Throughout the paper, we will focus on *regular* candidates, because irregular candidates are not our primary goal, as the target of script acquisition is identifying candidates that describe the scenario, i.e. the **regular** candidates; furthermore, irregular candidates make a diverse group of instances without much semantic similarity to each other. Thus they would not cluster easily in the representation space. Therefore, we ignore irregular candidates in training. During inference, we evaluate against the original gold standard.

We train a classifier to distinguish regular and irregular candidates so the latter could be excluded from training. The classifier is also used in the test

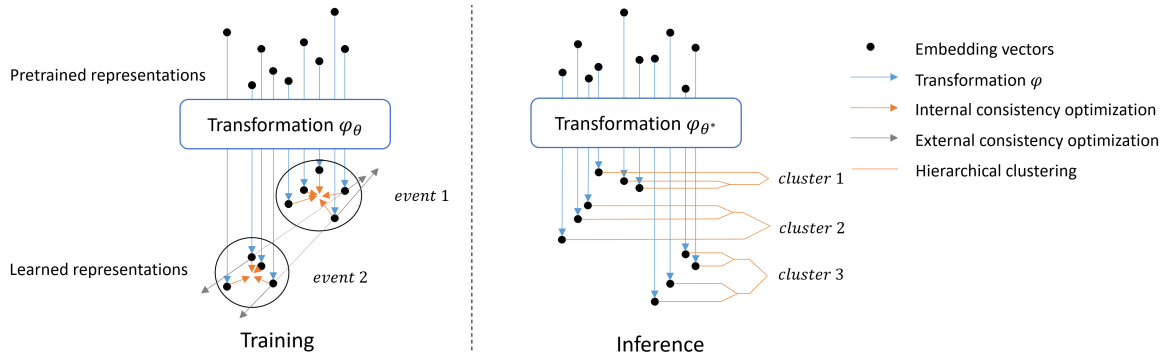

Figure 3: The overall framework. We learn a representation from annotated corpus and apply it to unannotated texts. The coreference and dependency terms are not visualized.

phase to exclude irregular candidates from clustering. We use the supervised script parser proposed in Zhai et al. (2021), but train it only to distinguish regular candidates from irregular candidates. We obtain training data for this task by grouping the original labels into one of REGULAR_EVENT, IRREGULAR_EVENT, REGULAR_PARTICIPANT and IRREGULAR_PARTICIPANT. The train / validation set of this model is constructed from the respective training set specified for each settings in §4.1. These classifiers achieve on average 85 points F1-score.

4.3 Training

We now describe how to learn φ. For any given text that we want to parse, we run XLNet (Yang et al., 2019) to obtain contextualized word embeddings $f(c)$ for each event and participant candidate c. We also considered using BERT (Kenton and Toutanova, 2019) / ROBERTA (Liu et al., 2019), but our input length exceeds the 512 word-piece limit hardwired in the pre-trained versions of these models.

Afterwards, we train φ to minimize distances within the same event and participant class and maximize them between different ones (§4.3.1); the general framework is illustrated in Fig. 3. We will then describe several extensions to the loss function (§4.3.2–§4.3.3) and then discuss replacing XLNet embeddings with more specialized word embeddings (§4.3.4)[2].

4.3.1 Learning script-neutral representations

Let $\mathcal{C}$ be the set of all event candidates or the set of all participant candidates in a text, and let $\pi(\mathcal{C})$ be a partition of $\mathcal{C}$ which clusters candidates into equivalence classes; at training time, each class contains the candidates that are labeled with the same event or participants. We define $\pi(c)$ as the element of the partition to which the candidate c belongs. Given a pre-trained embedding function f and the transformation φ_θ that we want to learn, we consider the average distance between instances belonging to different clusters:

$$d_{ext}(\pi(\mathcal{C});\theta) = \underset{\substack{c,c' \in \mathcal{C}: \\ \pi(c) \neq \pi(c')}}{mean} d(\varphi_\theta(f(c)), \varphi_\theta(f(c')))$$

The transformation φ is implemented by fine-tuning the last layer of the encoder.

We would like to push the embeddings of two candidates from different classes apart if they are too close to each other. We do so by maximizing the *external consistency* of the partition π:

$$\gamma_{ext}(\pi(\mathcal{C});\theta) = \underset{\substack{c,c': \pi(c) \neq \pi(c'), \\ d(\varphi_\theta(f(c)), \varphi_\theta(f(c'))) \\ < \sigma_1 d_{ext}(\pi(\mathcal{C});\theta)}}{mean} d(\varphi_\theta(f(c)), \varphi_\theta(f(c')))$$

$\sigma_1 \in (0, 1)$ is a threshold that quantifies being 'too close'. This definition captures the intuition that φ should map candidates from different classes to dissimilar vectors.

Likewise, consider the average distances between embeddings of candidates from same classes:

$$d_{int}(\pi(\mathcal{C});\theta) = \underset{c,c': \pi(c)=\pi(c')}{mean} d(\varphi_\theta(f(c)), \varphi_\theta(f(c')))$$

We would like to pull the embeddings of two candidates from the same class towards each other if they are too far away. In a similar spirit, we maximize the *internal consistency* of π:

[2]Our code and data are available at https://github.com/coli-saar/A3_USSP_coling22

(a) There is a **bus** stop down the street from my house . If you take **it** going south , **it** leads to the city...

(b) ...$I_{passenger}$ **fed** my *coins*$_{money}$ into the slot where you put your money...
...$I_{passenger}$ boarded the bus and **paid** for my ride with my *change*$_{money}$...

(c) ...the bus *arrived*$_{bus\_stops}$ at the **bus stop** closest to the beach...
...I would need the bus to *stop*$_{bus\_stops}$ next to **the hospital**...

Figure 4: Examples.(a) coreference chain. (b) events sharing similar participants. (c) participants that have similar event dependents.

$$\gamma_{int}(\pi(\mathcal{C});\theta) = 1 - \underset{\substack{c,c':\pi(c)=\pi(c'),\\ d(\varphi_\theta(f(c)),\varphi_\theta(f(c')))\\ >\sigma_2 d_{int}(\pi;\theta)}}{mean} d(\varphi_\theta(f(c)),\varphi_\theta(f(c'))) \quad (1)$$

We write $d(\cdot,\cdot)$ for the distance function in the representation space. Empirically, the following variant of cosine distance worked well:

$$\sqrt{1 - cos(\angle(v,w))}$$

Here $\angle(v,w)$ denotes the angle between v and w.

We obtain an overall consistency measure γ, which we maximize in training; λ_i is a hyperparameter that balances the terms. [3]

$$\gamma(\pi(\mathcal{C});\theta) = \gamma_{ext}(\pi(\mathcal{C});\theta) + \lambda_i \gamma_{int}(\pi(\mathcal{C});\theta)$$

4.3.2 Coreference

We can now further refine this baseline consistency model with script-specific knowledge. First, within a text, noun phrases that refer to the same entity form a **coreference chain**: for example, all mentions of the bus in the scenario TAKING A BUS (Fig. 4a). Therefore, these noun phrases should belong to the same participant cluster and have similar representations.

We capture this intuition as follows. Let $\eta(\mathcal{C}^c)$ be the set of all coreference chains that consists of participant candidates $\mathcal{C}^c$. Like π above, $\eta(\mathcal{C}^c)$ also specifies an equivalence relation, in that two candidates are in the same class iff they are in the same coreference chain. We can thus formulate a coreference-based consistency measure as

$$\beta(\theta) := \gamma_{int}(\eta(\mathcal{C}^c);\theta)$$

Note that minimizing this quantity imposes only a soft constraint; coreferent entities are rewarded for being in the same class, not forced into them. This increases robustness against noise in the coreference annotations.

4.3.3 Event-participant dependencies

Second, events and participants in a script are tightly linked: if two verbs have arguments from the same participant class, they tend to describe the same event (Fig. 4b); and if two noun phrases are arguments of the same event, they tend to describe the same participant (Fig. 4c).

Let c_p be the set of event candidates that have participant p as an argument; we encourage φ to map the elements of c_p to similar representations. Let $\xi(\mathcal{C}_e^d)$ be the set of all c_p, namely a partition of event candidates based on dependencies. Analogously, let $\xi(\mathcal{C}_p^d)$ be the set of participant candidate sets that depend on the same events. We can formulate a dependency-based consistency measure as

$$\alpha(\theta) = \gamma_{int}(\xi(\mathcal{C}_e^d);\theta) + \gamma_{int}(\xi(\mathcal{C}_p^d);\theta)$$

The final training objective, with hyperparameters λ_c, λ_d and cluster assignment $\pi^*(\mathcal{C}^e)$ of event candidates and $\pi^*(\mathcal{C}^p)$ of participant candidates specified by the annotations in InScript, is [4]

$$\theta^* = \underset{\theta}{argmax}[\gamma(\pi^*(\mathcal{C}_e^d);\theta) + \lambda_p \gamma(\pi^*(\mathcal{C}_p^d);\theta) + \lambda_c \beta(\theta) + \lambda_d \alpha(\theta)] \quad (2)$$

4.3.4 Specialized word embeddings

We further investigated whether our zero-shot approach can benefit by using more specialized word embeddings as input instead of the general-purpose XLNet embeddings. We thus replaced f with the representations from the pre-final layer of the supervised script parser of Zhai et al. (2021) trained and validated on our training data. These representations are also based on XLNet, but then trained to

[3] This function could be seen as a variant of triplet loss introduced by (Dong and Shen, 2018), but relaxed with the thresholds and uses a different distance metric. Empirically, these measures improve the stability of training and result in a moderate performance improvement.

[4] As was correctly pointed out by one of our reviewers, untyped dependencies can be noisy for our purpose. We do not use typed dependencies as typing the links amplifies the data sparsity, making it difficult to generalize. Empirically, using untyped dependencies still granted a moderate performance improvement.

predict InScript events and participants on known scenarios.

4.4 Inference

4.4.1 Clustering

At inference time, we first determine the event and participant candidates by taking the nouns, pronouns and verbs, and classify them for regularity. We then acquire a representation $\varphi_{\theta^*}(f(c))$ for each candidate c and group them into classes by clustering (cf. Fig. 3).

We use agglomerative clustering, a bottom-up hierarchical clustering algorithm that iteratively merges the most similar pair of clusters. It terminates when either the number of clusters decreases to a pre-defined quantity or the minimum dis-similarity between the current clusters goes beyond a predefined threshold. As the number of event and participant classes vary across scenarios, we do not fix the number of cluster, but instead define a dissimilarity threshold estimated from the training scenarios.

4.4.2 Protagonists

We give special treatment to the **protagonist** of each scenario – for example, the passenger in TAKING A TRAIN or the customer in GROCERY SHOPPING. The protagonist is the most frequent participant in all scenarios and always makes the largest class of participant candidates. We thus identify it by following the longest coreference chain, instead of feeding these referents to the neural pipeline. Therefore, the identification of protagonists is excluded from training and performed symbolically in the reference phase. This simple heuristic yields an F-score of 98 at inference time for the protagonist class.

4.4.3 Coreference Chains at Inference Time

During the training phase we encourage the embedding vectors associated with candidates from the same coreference chain to be more similar to each other. But during the inference phase, we could also directly perform coreference resolution on the text. We take into account this information by refining the distance in §4.3 to

$$\sqrt{1 - cos(\angle(v, w)) - \lambda I_c(v, w)}$$

here $I_c(v, w) = 1 \Leftrightarrow$ the candidates associated with vectors v, w are in the same coreference chain. λ is a hyper-parameter.

5 Evaluation

On InScript, we evaluate our method with 10-fold cross-validation where each fold is the data associated with a scenario. Note that the texts in the validation and test data are always from scenarios that were unseen in training. On MCScript, the model is evaluated on the MCScript test set.

5.1 Metric

Given a cluster assignment, what we are interested in is how well the predicted clusters align with gold classes. However, the gold classes are unknown to us; the outcome of clustering includes a number of indexed clusters, like $cluster_1, cluster_2$, and we do not know which gold cluster should these clusters be compared to. To tackle this issue, we need a 'best' assignment of the clusters to the gold classes, with which we can evaluate the 'accuracy' of the clustering results as if it were a classification task. One approach is to find the assignment that maximizes this accuracy. This is a *linear assignment* problem, which is solved in cubic time by the Hungarian algorithm (see, e.g. Kuhn, 1955), thus tractable given the scale of our problem. We call the F1 score evaluated according to this optimal assignment **Hungarian F1**, and use it as our main evaluation metric. This metric allows us to compare the results of the clustering-based parsers to that of the classification-based parsers.

5.2 Baselines

We compare the results of our zero-shot parser to a number of baselines. First, we compare against the supervised script parsers of Zhai et al. (2021) and Ostermann et al. (2017). For the former, we take the performance report from the original paper; for the latter, we retrain the model to evaluate on the train-test split of Zhai et al. This data split defines a supervised task, thus the performance of these parsers are not directly comparable to ours.

Second, we compare against a baseline where we cluster event and participant candidates at inference time based on the bare XLNet embeddings, rather than the ones that were transformed by our learned φ_{θ^*}. Finally, in addition to our *full* model, as specified by equation 2 with the specialized embeddings of §4.3.4, we also present results for ablated versions without the extensions regarding event-participant dependencies (*dep*), coreference (*coref*), and specialized embeddings (*specialized*).

model	gold regularity	task	events		participants	
			macro F1	micro F1	macro F1	micro F1
Ostermann et al. (2017)	✓	supervised	58.1	66.0	n/a	n/a
Zhai et al. (2021)	X	supervised	75.1	85.7	80.3	90.3
Bare XLNet	X	zero-shot	40.2	53.2	39.3	60.5
w/o dep, coref, specialized	X	zero-shot	$46.0_{\pm 2.8}$	$58.4_{\pm 2.7}$	$47.5_{\pm 2.6}$	$\mathbf{75.7}_{\pm 1.8}$
w/o dep, coref	X	zero-shot	$48.6_{\pm 5.2}$	$62.7_{\pm 3.7}$	$44.5_{\pm 3.3}$	$71.8_{\pm 2.1}$
w/o dep	X	zero-shot	$51.0_{\pm 3.7}$	$66.8_{\pm 4.3}$	$\mathbf{52.0}_{\pm 3.7}$	$74.8_{\pm 2.9}$
Full model	X	zero-shot	$\mathbf{53.4}_{\pm 1.8}$	$\mathbf{68.1}_{\pm 2.3}$	$51.7_{\pm 1.6}$	$74.4_{\pm 1.4}$
Bare XLNet	✓	zero-shot	43.1	51.6	43.9	61.0
w/o dep, coref, specialized	✓	zero-shot	$46.1_{\pm 1.9}$	$55.4_{\pm 2.2}$	$51.1_{\pm 2.7}$	$\mathbf{75.3}_{\pm 1.3}$
w/o dep, coref	✓	zero-shot	$55.3_{\pm 2.8}$	$65.8_{\pm 2.8}$	$52.5_{\pm 3.1}$	$73.6_{\pm 2.1}$
w/o dep	✓	zero-shot	$56.7_{\pm 3.3}$	$67.4_{\pm 3.7}$	$\mathbf{53.6}_{\pm 2.9}$	$74.2_{\pm 2.0}$
Full model	✓	zero-shot	$\mathbf{57.6}_{\pm 1.3}$	$\mathbf{68.1}_{\pm 1.3}$	$52.8_{\pm 1.4}$	$73.7_{\pm 1.4}$

Table 1: Results on InScript. We show the average over ten-fold cross validation and five training runs when feasible. These quantities are the Hungarian versions of F1 defined in §5.1. Some models train and inference according to the regularity annotations in InScript, instead of the predictions of our regular candidate identifier. Ostermann et al. and Zhai et al. use a data split where the models see the test scenario during training; the other variants use the zero-shot split described in §3.

model	gold regularity	events		participants	
		macro F1	micro F1	macro F1	micro F1
Bare XLNet	X	24.1	25.0	17.8	19.5
w/o dep, coref, specialized	X	$44.2_{\pm 0.57}$	$51.2_{\pm 0.83}$	$37.0_{\pm 1.4}$	$39.0_{\pm 0.72}$
w/o dep, coref	X	$45.7_{\pm 0.63}$	$50.2_{\pm 0.63}$	$37.9_{\pm 1.1}$	$41.9_{\pm 1.0}$
w/o dep	X	$48.2_{\pm 1.4}$	$53.3_{\pm 1.3}$	$40.8_{\pm 1.8}$	$45.4_{\pm 1.7}$
Full model	X	$\mathbf{49.6}_{\pm 1.3}$	$\mathbf{55.5}_{\pm 1.6}$	$\mathbf{42.6}_{\pm 1.6}$	$\mathbf{54.0}_{\pm 2.0}$

Table 2: Results on MCScript averaged from five parallel training runs. All models train and inference based on the regularity predictions of our regular candidate identifier.

For each of the clustering-based methods, we report two results: one where we assume gold information about whether an event or participant candidate is regular, and one where this is predicted by the classifier from §4.2. All variants use the same number of trials for hyperparameter tuning. Afterwards, we do 5 parallel training sessions to test the models' robustness against random initializations, and report mean and std.

6 Results

6.1 Evaluation on InScript

The results on InScript are shown in Table 1. All variants of our model outperform clustering based on raw XLNet embeddings by a considerable margin. Our model also performs on par with Ostermann's, although we do not have access to scenario-specific supervision whereas Ostermann's does, and our model additionally performs participant parsing. In general, we obtain a higher micro-F1 for participants than for events. This is due to the more skewed distribution of the sizes of the participant class sizes than those of the event classes.

The model extensions boost parsing accuracy significantly. Access to coreference information improves participant parsing performance. Dependency information grants a performance boost in event parsing. A closer inspection shows that with dependency information, the parser is better at grouping together event candidates with different verbs but same arguments. For example, event *sink into water* in TAKING A BATH could be evoked by *slide into water*, *sink into water*, *slip into the tub*, *lower into the tub*, etc. The verbs in these event candidates all share arguments *I* and *water* or *tub*, which our parser correctly clusters together. Without dependency information, the parser mostly groups together candidates whose predicate is 'sink', the most frequent verbalization of the event.

The performance of our script parser differs from fold to fold: we get 70.1 micro-F1 for participant parsing on TAKING A BATH, but only 43.8 on BORROWING A BOOK FROM LIBRARY. These differ-

ences result from two factors. (1) Similarities between the training scenarios and the test scenario. (2) Differences in the qualities of the original annotation among different scenarios.

6.2 Evaluation on MCScript

The results on MCScript is given in table 2. Note that the test scenarios only have around 20 stories each, as opposed to the 100 stories available for each InScript scenario. Shifting to a different domain and having fewer stories per scenario available incurred a considerable performance drop (on average over 10 points F1 score). The performance drop is especially noticeable for participant parsing, which was unexpected. A closer look indicates that participants in MCScript scenarios follow much more skewed distributions than in InScript: instances from the less frequent half of the participant classes in MCScript take a proportion of only 8% whereas this quantity is 18% for InScript. As a result, there are more classes that only have a handful of instances than InScript, worsening the already low-resource setting.

All our model variants still outperform the XLNet baseline by a large margin, with the full model achieving the best performance, at 49.6/55.5 macro/micro F1 for events, and 42.6/54.0 macro/micro F1 for participants. Each model component contributes to this performance according to the ablation study.

7 Further analysis

7.1 Temporal script graphs

Given the clustering results, we can induce temporal script graphs for unseen scenarios. We establish temporal order as follows: we say event e_1 **precedes** event e_2 if, and only if, in stories where they both occur, the proportion where e_1 occurs before e_2 is beyond a threshold ζ. If neither e_1 precedes e_2 nor e_2 precedes e_1, we decide they could follow arbitrary order.

If we view the construction of temporal script graphs as a task of retrieving temporally ordered event pairs, and evaluate our results against that inferred from annotations in InScript, our clustering results yields 75 points F1 score. Observe that the model has learned that each event can be expressed in many different ways that are semantically similar only in the context of the scenario (Fig. 2).

7.2 Probing Script-specific Embeddings

We conjectured that the transformation φ was needed to distill the relevant information for script parsing out of the pretrained XLNet embeddings. We investigate whether this is true by freezing the embeddings $\varphi(f(c))$ (zero-shot) and the pretrained embeddings $f(c)$ (XLNet) and training models for a variety of NLP tasks that take these embeddings as input.

We probe with the following tasks. (1) POS tagging and (2) named entity recognition; these mostly depend on the token itself and its local context. (3) Noun phrase chunking, which is determined by sentence-level syntax. (4) Sentence ordering, where we randomly shuffle the order of the sentences in a story and train a binary classifier to detect whether the story is shuffled. The task would need information across the entire story to conduct. (1)-(3) are formulated as sequence labelling tasks; (4) is a binary classification. The experiments are conducted on InScript, with the same data split as is used to train our representation. InScript includes POS annotations; for NER and chunking, the labels are generated with Spacy (Honnibal and Montani, 2017, model *en_core_web_trf*).

In each of these probing tasks, both representations use the same amount of GPU budget. See Fig. 5 for the results. The transformed representation $\varphi(f(c))$ incurs performance drops on most tasks, compared to the general-purpose embeddings $f(c)$. However, the performance on sentence ordering sees a noticable improvement. This supports our hypothesis that φ amplifies higher level features, which are more important to script parsing than to generic language modelling. In comparison, lower-level information about morphology and syntax are deemphasized.

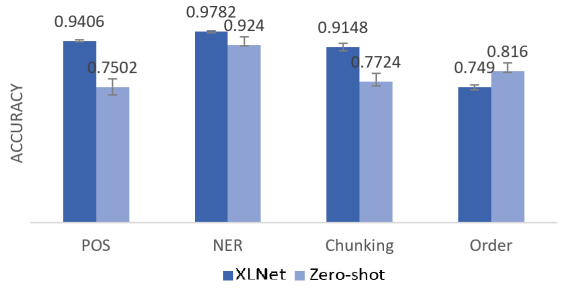

Figure 5: Performance on probing tasks. Our representation clearly favours the sentence ordering task. The error bars show one standard deviation. All differences between these pairs are significant at $\alpha = 0.05$ according to independent T-test.

7.3 Error analysis

We now have a look at the errors made by our parser. The errors fall into the following categories.

Granularity

Many events could be divided into multiple subevents, forming a hierarchy of events. For example, in the TAKING A BATH scenario, we have *prepare for bath*, *undress* and *grab a towel*. A similar phenomenon is observed for the participants. This fact manifested itself into various types of errors. For example, the set of event labels in InScript often consists of events of different granularities, frequently rendering multiple cluster assignments feasible (e.g. ... *I **took** a clean towel with me* ... in either *prepare for bath* or *grab a towel*). As a result, the parser sometimes confuses one event cluster with another that includes it, or group together different events that actually fit together (*turn on water* and *fill tub with water*). Granularity accounts for two thirds the event errors and one sixth participant errors.

Shared predicate or argument

Some wrongly clustered events share the verb or some arguments with another class, especially when light verbs are involved, which makes the distinction harder. For example, in TAKING A TRAIN, a few *get ticket* events (e.g. "I took the ticket from him") are predicted as *conductor checks ticket* (e.g. "I gave the ticket to him").

8 Conclusion

We have presented the first approach to script parsing without scenario-specific knowledge. We do this by clustering specialized word representations which are trained by optimizing cluster consistency; the model is further improved by the use of coreference and event-participant dependency information. The model greatly outperforms a baseline with general-purpose word embeddings, and performs on par with an earlier supervised model.

Our model thus makes it possible, for the first time, to label large quantities of unannotated data with script information. In future work, we plan to experiment also with corpora that contain a larger variety of naturally occurring texts than MCScript, in which the sentences are relatively simple, pertinent to the scenario and are often in the correct temporal order.

Acknowledgement

This research was funded by the German Research Foundation (DFG) as part of SFB 1102 (Project-ID 232722074) "Information Density and Linguistic Encoding".

References

Simon Ahrendt and Vera Demberg. 2016. Improving event prediction by representing script participants. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 546–551.

Pratyay Banerjee and Chitta Baral. 2020. Self-supervised knowledge triplet learning for zero-shot question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 151–162, Online. Association for Computational Linguistics.

Avron Barr and Edward Feigenbaum. 1981. *The Handbook of Artificial Intelligence: Volume 2*. William Kaufman Inc, Los Altos, CA.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Nathanael Chambers and Dan Jurafsky. 2008. Unsupervised learning of narrative event chains. *Proceedings of ACL-08: HLT*, pages 789–797.

Nathanael Chambers and Dan Jurafsky. 2009. Unsupervised learning of narrative schemas and their participants. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 602–610. Association for Computational Linguistics.

Xingping Dong and Jianbing Shen. 2018. Triplet loss in siamese network for object tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 459–474.

Xiangyu Duan, Mingming Yin, Min Zhang, Boxing Chen, and Weihua Luo. 2019. Zero-shot cross-lingual abstractive sentence summarization through teaching generation and attention. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3162–3172, Florence, Italy. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

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.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR (Poster)*.

Harold W Kuhn. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

I-Ta Lee, Maria Leonor Pacheco, and Dan Goldwasser. 2020. Weakly-supervised modeling of contextualized event embedding for discourse relations. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4962–4972, 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.

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.

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.

Simon Ostermann, Ashutosh Modi, Michael Roth, Stefan Thater, and Manfred Pinkal. 2018. Mcscript: A novel dataset for assessing machine comprehension using script knowledge. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Simon Ostermann, Michael Roth, and Manfred Pinkal. 2019. Mcscript2. 0: A machine comprehension corpus focused on script events and participants. In *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (* SEM 2019)*, pages 103–117.

Simon Ostermann, Michael Roth, Stefan Thater, and Manfred Pinkal. 2017. Aligning script events with narrative texts. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*, pages 128–134.

Ngoc-Quan Pham, Jan Niehues, Thanh-Le Ha, and Alexander Waibel. 2019. Improving zero-shot translation with language-independent constraints. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 13–23, Florence, Italy. Association for Computational Linguistics.

Jerin Philip, Alexandre Berard, Matthias Gallé, and Laurent Besacier. 2020. Monolingual adapters for zero-shot neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4465–4470, Online. 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.

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.

Michaela Regneri, Alexander Koller, Josef Ruppenhofer, and Manfred Pinkal. 2011. Learning script participants from unlabeled data. In *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pages 463–470.

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. 1977. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press.

Lilian DA Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. 2016. Descript: 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.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763.

Wenpeng Yin, Jamaal Hay, and Dan Roth. 2019. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3914–3923, Hong Kong, China. Association for Computational Linguistics.

Fangzhou Zhai, Vera Demberg, and Alexander Koller. 2020. Story generation with rich details. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2346–2351, Barcelona, Spain (Online). International Committee on Computational Linguistics.

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.

Fangzhou Zhai, Iza Škrjanec, and Alexander Koller. 2021. Script parsing with hierarchical sequence modelling. In *Proceedings of* SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, pages 195–201.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. 2020. Improving massively multilingual neural machine translation and zero-shot translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639, Online. Association for Computational Linguistics.

A Implementation and Optimization

The model was implemented with AllenNLP 1.2 Gardner et al. (2017). The pre-trained XLNet model we used was *xlnet-base-cased* (`https://github.com/zihangdai/xlnet/`). The training is further regularized with weight decay. The optimization is performed with adam (Kingma and Ba, 2015) in conjunction with early-stopping which monitors validation loss; the hyper-parameter tuning is performed with random hyper-parameter search (Bergstra and Bengio, 2012). Optimization takes on average 5 hours on a singe Tesla v100.

The implementations of agglomerative clustering and Hungarian algorithm are from the *scipy* library. Table 3 shows the hyper-parameters for the best performing single run on each fold (full model, predicted regularity).

B Examples

Table 4 illustrates a couple of sample clusters. The candidates vary in their surface forms.

fold	lr	weight decay	λ_{ei}	λ_{pi}	λ_p	σ_1	σ_2	λ_c	λ_d
0	1.10E-05	0.000155	1.03	0.0147	0.343	0.887	0.263	0.00372	0.00795
1	2.06E-05	0.00677	0.457	0.882	0.125	0.392	0.391	0.0101	0.0485
2	0.000126	0.000896	0.0105	0.0148	0.0144	0.903	0.728	0.00533	0.0118
3	2.91E-05	0.000166	0.0237	0.0649	0.256	0.373	0.544	0.0619	0.00867
4	7.08E-05	0.00051	0.15	0.0711	0.745	0.681	0.562	0.0121	0.0216
5	0.000353	0.00121	1.51	0.0371	0.00211	0.171	0.692	0.0872	0.016
6	0.000423	1.07E-05	0.204	0.018	0.177	0.113	0.632	0.242	0.00517
7	7.88E-05	0.00303	0.734	0.0136	0.517	0.189	0.948	0.0047	0.111
8	2.72E-05	7.32E-05	0.568	0.00151	0.0589	0.676	0.566	0.00424	0.0475
9	2.09E-05	5.19E-05	0.0327	0.555	0.243	0.788	0.95	0.0866	0.0119

Table 3: Hyper-parameters. Note that these are the hyper-parameter combinations that yielded the best performance in a round of random hyper-parameter search. Thus the quantities in this table do not represent the best choices of each single hyper-parameter.

ground truth	text
turn water on bath	... I might drain the tub and **put** in more water ...
sink into water	... I turn off the faucet and **sink** into bliss ...
sink into water	... Then I **slid** into the water and enjoyed the relaxing warmth for twenty or more minutes...
sink into water	... I gingerly **lowered** myself into the nice warm water and immediately began to relax...
sink into water	... I eased my way into the tub and let myself **sink** into the water ...
sink into water	... I slowly **sunk** the rest of my body , and closed my eyes...
sink into water	... the tub was full and ready . I **slipped** into the tub and soaked in the bliss...
washing tools	...then I lather up with either **soap** or shower gel ...
water	...After I scrub really good and finish singing , I pour **water** continuously on my body...
washing tools	...I pour water continuously on my body until all the **soap** was he s off...
washing tools	...I cleaned my hair with some **shampoo** and washed my body with a wash cloth and rinsed...
washing tools	...shampooed my hair and applied some **conditioner** then washed my body...
washing tools	...applied some condition er then washed my body using some **liquid body wash**...
washing tools	...After I have washed everything , I rinse the **soap** from my body with the water in the tub...
washing tools	...on the corner of the bathtub . I lather ed **it** up and washed my arms , my legs...
washing tools	...take a wash cloth and **soap** or body wash to give yourself a good scrub down...
washing tools	...You can put the **shampoo** in your hair...
washing tools	...place your head under the faucet to rinse out the **soap** . Enjoy your bath !
washing tools	...washed myself with a **wash cloth** and soap . Then I leaned my head against...
washing tools	...stepped into the bath tub . I used **soap** and a wash cloth to clean myself...

Table 4: Example output clusters. Top: event; bottom: participant. The table presents a random selection of instances from these clusters as the original output could contain hundreds of instances.