SubeventWriter: Iterative Sub-event Sequence Generation with Coherence Controller

Zhaowei Wang[1], **Hongming Zhang**[2], **Tianqing Fang**[1], **Yangqiu Song**[1],
Ginny Y. Wong[3], **& Simon See**[3]
[1]Department of Computer Science and Engineering, HKUST
[2]Tencent AI Lab, Bellevue, USA
[3]NVIDIA AI Technology Center (NVAITC), NVIDIA, Santa Clara, USA
`{zwanggy, tfangaa, yqsong}@cse.ust.hk`
`hongmzhang@global.tencent.com`, `{gwong, ssee}@nvidia.com`

Abstract

In this paper, we propose a new task of sub-event generation for an unseen process to evaluate the understanding of the coherence of sub-event actions and objects. To solve the problem, we design *SubeventWriter*, a sub-event sequence generation framework with a coherence controller. Given an unseen process, the framework can iteratively construct the sub-event sequence by generating one sub-event at each iteration. We also design a very effective coherence controller to decode more coherent sub-events. As our extensive experiments and analysis indicate, *SubeventWriter*[1] can generate more reliable and meaningful sub-event sequences for unseen processes.

1 Introduction

Natural language understanding involves deep understanding of events. In the NLP community, there have been many event understanding tasks. Most of them focus on parsing events into involved entities, time, and locations as semantic roles (Kingsbury and Palmer, 2002; Li et al., 2013; Lv et al., 2020; Lin et al., 2020; Du and Cardie, 2020; Zhang et al., 2021; Lyu et al., 2021a), or identifying their binary relations such as temporal or causal relations (Berant et al., 2014; Smith et al., 2018; Sap et al., 2019; Wang et al., 2020, 2021). However, our natural language can be used to describe relations more than binary ones. For example, processes (Craig et al., 1998), also known as scripts (Schank and Abelson, 1977) or activities (Mourelatos, 1978), are complex events constituted by a sequence of sub-events. Understanding processes can be more challenging than individual or pair of events.

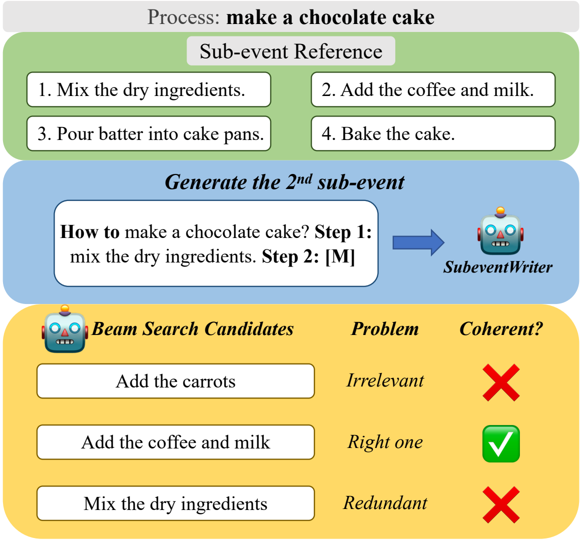

Figure 1: A motivating example of *SubeventWriter*, which generates one sub-event at a time iteratively. We show the process "make a chocolate cake" and the second iteration of the generation. By considering coherence, we can re-rank candidates and reach the right sub-event. [M] is a mask token.

As shown in Figure 1, to complete the process of making a chocolate cake, we need to consider a sequence of actions, "mix," "add," "pour," and "bake," which involves different objects, e.g., dry ingredients, coffee, milk, etc. Those actions should follow a logically coherent procedure while the objects should be all related to the target, chocolate cake. Thus, building such a coherent sequence should take the whole sub-events into consideration.

There have been two categories of related studies to processes, namely process induction and narrative cloze tasks. Zhang et al. (2020a) proposed a task to learn the hierarchical structure called process induction, where a model needs to generate a sub-event sequence to finish a given process. Their framework aggregates existing events so that it can conceptualize and instantiate similar processes. However, the aggregation procedure does not consider the coherence of actions and their objects. In addition, to build the dataset, they extracted events using a dependency parser

---

[1] Code is available at `https://github.com/HKUST-KnowComp/SubeventWriter`.

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 1590 - 1604
December 7-11, 2022 ©2022 Association for Computational Linguistics

with pre-defined verb-argument templates (Zhang et al., 2020b, 2022). Such structured events might harm coherence as only head words are retained after extraction. Consider the first sub-event in Figure 1. After parsing, we lost the indispensable modifier "dry" and the sub-event becomes (*mix*, *ingredients*)[2], which includes the wet ingredients (e.g., "milk") in the second sub-event. Thus, the logical relation between the two adjacent sub-events (i.e., coherence (Van Dijk, 1980)) is defective.

On the other hand, narrative cloze tasks (Chambers and Jurafsky, 2008; Granroth-Wilding and Clark, 2016; Chambers, 2017; Mostafazadeh et al., 2016) evaluate whether a model can predict the missing (usually the last) event in a narrative. These tasks essentially evaluate the semantic similarity and relatedness between the target event and the context. However, they did not emphasize how all events in the contexts are unified as a whole process in an ordered and coherent way.

To evaluate complex process understanding, we propose a new generation-based task to directly generate sub-event sequences in the free-text form, as shown in Figure 1. In the task, better generation of a process means better understanding of the coherence among action verbs as well as their operational objects. In fact, we find that generating free-text events is a non-trivial task, even with existing strong pre-trained models like T5 (Raffel et al., 2020) and BART (Lewis et al., 2020). First, generating an overlong piece of text containing several temporally ordered sub-events at once is challenging to current pre-trained models (Zhou et al., 2022; Lin et al., 2021; Brown et al., 2020). Next, sub-events are generated without considering the coherence of actions and their objects, which might give rise to irrelevant or redundant results.

To solve the task, we propose *SubeventWriter* to generate sub-events iteratively in the temporal order. *SubeventWriter* only generates the next sub-event in each generation iteration, given the process and prior generated sub-events. It eases the generation difficulty by decomposing the sub-event sequence. Moreover, sub-events should be coherently organized to complete a process. To consider coherence in each iteration, we can get a few sub-event candidates from the beam search and select the most coherent one, as shown in Figure 1. In *SubeventWriter*, we introduce a coherence controller to score whether a candidate is coherent with the process and prior generated sub-events. As a result, *SubeventWriter* can construct more reliable and meaningful sub-event sequences.

To evaluate our framework, we extract a large-scale general-domain process dataset from WikiHow [3], containing over 80k examples. We conduct extensive experiments with multiple pre-trained models, and automatic and human evaluations show that *SubeventWriter* can produce more meaningful sub-event sequences compared to existing models by a large margin. Moreover, we conduct few-shot experiments to demonstrate that our framework has a strong ability to handle few-shot cases. Last but not least, we evaluate the generalization ability of *SubeventWriter* on two out-of-domain datasets: SMILE (Regneri et al., 2010) and DeScript (Wanzare et al., 2016). The results manifest our framework can generalize well.

2 Textual Sub-event Sequence Generation

We formally define the sub-event sequence generation task as follows. Given a process S, we ask the model to generate sub-event sequences E, which are steps to solve the process. This task is essentially a *conditional language modeling* problem. Specifically, given a process S consisting of n tokens: $x_1, x_2, \ldots, x_n$ and a sequence E consists of m sub-events $e_1, e_2, \ldots, e_m$ (each sub-event refers to a sentence containing t_i tokens: $y_{i,1}, y_{i,2}, \ldots, y_{i,t_i}$), models aim to learn the conditional probability distribution by maximizing the following conditional probabilities in Eq. (1):

$$\begin{aligned} P_\theta(E|S) &= \prod_{i=1}^{m} P_\theta(e_i|e_{<i}, S) \\ P_\theta(e_i|e_{<i}, S) &= \prod_{j=1}^{t_i} P_\theta(y_{i,j}|y_{i,<j}, e_{<i}, S). \end{aligned} \tag{1}$$

3 The *SubeventWriter* Framework

Figure 2 illustrates the details of the proposed *SubeventWriter* framework. For a given process, the framework decomposes the generation into multiple iterations. The sequence-to-sequence (seq2seq) language model generates a few candidates for the next sub-event in each iteration. We then leverage a coherence controller to re-rank the generated candidates by considering whether they are coherent with the process and prior generated

---

[2] The matched pre-defined template is (*verb*, *object*).

[3] wikihow.com

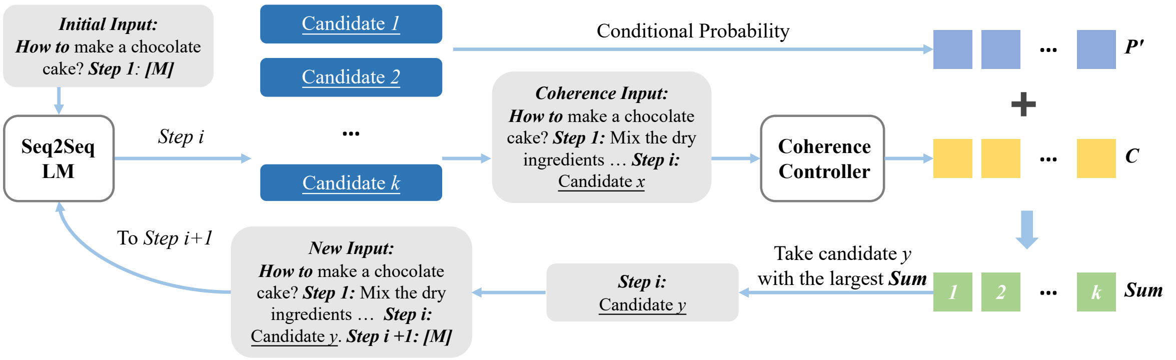

Figure 2: The overview of our *SubeventWriter*. In each iteration, the Seq2Seq language model takes the process and prior generated sub-events as input and generates a few candidates for the next sub-event. Then the coherence controller is used to select the most coherent candidate as the next sub-event.

sub-events. The coherence controller is a discriminative model that can assign a coherence score to a sub-event sequence. It is fine-tuned independently on our synthetic data generated according to our manually designed coherence rules. Finally, the framework appends the generated sub-event to the end of the input to serve as new context and start the next iteration. The detailed description of *SubeventWriter* components is as follows:

3.1 Iterative Event-level Decoding

The iterative event-level decoding scheme is built on top of seq2seq language models, including T5 (Raffel et al., 2020) and BART (Lewis et al., 2020). We describe training and inference details as follows.

Training: The seq2seq language models are fine-tuned to decode one sub-event each time in chronological order. For each process with its sub-event sequences in the training data, we create an augmented set of training examples with each sub-event in the sequence as the output in turns. For example, if the valid sequence of a process S consists of temporally ordered sub-events e_1, e_2, and e_3, we then create four training examples: $S \rightarrow e_1$, $S \cup \{e_1\} \rightarrow e_2$, $S \cup \{e_1, e_2\} \rightarrow e_3$, and $S \cup \{e_1, e_2, e_3\} \rightarrow$ *none*, where "*none*" is a special token to end sequences. The order of adding sub-events e_i follows the temporal order, which ensures that the model only needs to predict what will happen next without a longer-term forecast.

To minimize the gap between pre-training and fine-tuning, we design a textual prompt template to construct input in human language. If we want to generate the $i + 1$ *th* sub-event given a process S and sub-events $e_1, e_2, ..., e_i$, the template takes the form of "How to S? Step 1: e_1 ... Step i: e_i. Step i+1: [M]" as the example shown in Figure 2. [M] is the mask token of the model. More examples of input/output are shown in Appendix A.1.

Inference: During the inference, we apply the seq2seq language models iteratively to generating the sub-event sequence of a process. The aforementioned prompt template is also used. For instance, the model first generates sub-event e_1 for a process S. It then takes S and e_1 as input and generates the second sub-event e_2. The model repeats this process until the special token "*none*" is generated, which means no more sub-events are required. Then, generated sub-events are concatenated into a sequence as the final output.

3.2 Coherence Controller

As a sub-event sequence should be coherent to complete a process, we propose a coherence controller to control the iterative event-level decoding. At each iteration, the coherence controller considers whether each sub-event candidate is coherent with the given process and sub-events generated in previous iterations. Considering that sub-events (one or more sentences) are diverse and complicated, here we employ a coherence model (Jwalapuram et al., 2021) based on BERT (Devlin et al., 2019) as the coherence controller to score sub-event candidates.

We train the coherence controller as a binary classification task to discriminate coherent sub-event sequences from incoherent ones. Following previous works (Mesgar and Strube, 2018; Moon et al., 2019), we regard a human-written sub-event sequence as coherent, and we synthetically build two types of incoherent sub-event sequences by corrupting the local or global coherence of the human-written one. For ***local coherence***, we randomly copy a sub-event in the current process and place this ***duplicate sub-event*** at a random location. In this way, the relation between two sub-events adjacent to the duplicate sub-event is corrupted, entitled local coherence in linguistics (Van Dijk, 1980). For ***global coherence***, we randomly choose a sub-event from other processes with a different theme and insert this ***irrelevant sub-event*** at a random location. In this way, the theme among all sub-events is corrupted, called global coherence (Van Dijk, 1980). We show positive and two types of negative examples in Appendix A.2.

We use the cross-entropy loss shown in Eq. 2 to optimize the coherence controller, where y and $\hat{y}$ are label and coherence scores, respectively. Since y equals 1 for positive examples, our model will give higher scores for more coherent input. For each positive example, we sample N negative examples by corrupting local coherence and the same number by corrupting global coherence ($2N$ in total). Thus, we balance the loss function by dividing negative loss by $2N$:

$$\mathcal{L}_{coh} = -(y \log \hat{y} + \frac{1}{2N}(1-y)\log(1-\hat{y})). \tag{2}$$

At the inference stage (Figure 2), we concatenate the process and the current generated sequence into the input to the coherence controller. For example, in the i th iteration, the Seq2Seq language model with beam search returns top-k possible sub-event candidates: $\hat{e}_{i1}, \hat{e}_{i2}, \ldots, \hat{e}_{ik}$. We construct the input $S; \hat{e}_1, \hat{e}_2, \ldots, \hat{e}_{i-1}, \hat{e}_{ij}$ for every candidate $\hat{e}_{ij}$, given process S and prior sub-events $\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_{i-1}$. With such input, the coherence controller computes coherence scores $C(\hat{e}_{ij})$. As the sequence-to-sequence model can return the logarithm of conditional probability $P'(\hat{e}_{ij}) = \log P_\theta(\hat{e}_{ij}|\hat{e}_{<i}, S)$ (Eq. 1) for each candidate, we re-rank candidates and return the best one according to the sum of the two scores:

$$\hat{e}_i = \arg\max_{\hat{e}_{ij}} \{P'(\hat{e}_{ij}) + \lambda C(\hat{e}_{ij})\}, \tag{3}$$

where λ is a hyper-parameter to weight coherence scores. Appendix A.3 gives a concrete example of the inference stage of the coherence controller.

4 Experiments

We conduct extensive experiments and compare *SubeventWriter* with a wide selection of baselines.

4.1 Dataset

We collect processes and corresponding event sequences from the WikiHow website[4] (Koupaee and Wang, 2018), where each process is associated with a sequence of temporally ordered human-annotated sub-events. We randomly split them into the training, validation, and testing sets. As a result, we got 73,847 examples for the training set and 5,000 examples for both validation and testing sets, whose average sub-event sequence length is 4.25.

4.2 Evaluation Metric

For each pair of a predicted sequence and a ground truth sequence, we compute BLEU-1 (Papineni et al., 2002), BLEU-2, ROUGE-L (Lin, 2004), and BERTScore (Zhang et al., 2019) between them and take the average of each metric over all data. For inference cases with multiple references, we take the best performance among all references.

4.3 Baseline Methods

We compare our framework to three methods:

All-at-once Seq2Seq: An intuitive solution to the textual sub-event sequence generation task would be modeling it as an *end-to-end sequence-to-sequence (Seq2Seq) problem*, where Seq2Seq language models are fine-tuned to predict all sub-events at once, given a process as input. Here we test multiple Seq2Seq language models: T5-base/large/3b and BART-base/large. We refer to this baseline as "All-at-once" for short in following sections.

Top-one Similar Sequence: Following previous work (Zhang et al., 2020a), another naive yet potentially strong baseline is Top-one Similar Sequence. For each unseen process in the validation or testing set, the baseline finds the most similar process in the training data. The sub-event sequence of the most similar process is then regarded as the prediction. If more than one sub-event sequence exists for the most similar process, we randomly pick one from them. Here, we consider two methods to measure similarities: cosine similarity of Glove (Pennington et al., 2014) and Sentence-BERT (SBERT) (Reimers and Gurevych, 2019) embeddings.

Zero-shot Large LM: Large language models (LMs) have shown stronger performance on extensive NLP tasks (Raffel et al., 2020). The third baseline we introduce is prompting large language

---

[4] wikihow.com

Models	B-1	B-2	R-L	BERT	$\Delta_{B\text{-}1}$	$\Delta_{B\text{-}2}$
Zero-shot Large LM (GPT-J 6b)	13.88	0.33	16.47	45.43	-	-
Zero-shot Large LM (T5-11b)	20.14	0.76	14.11	54.55	-	-
Top-1 Similar Sequence (Glove)	16.31	0.99	11.63	57.24	-	-
Top-1 Similar Sequence (SBERT)	18.39	2.21	13.46	59.94	-	-
All-at-once Seq2Seq (BART-base)	21.01	4.52	18.83	58.79	-	-
All-at-once Seq2Seq (BART-large)	21.84	4.73	18.94	59.45	-	-
All-at-once Seq2Seq (T5-base)	20.33	5.63	20.22	52.15	-	-
All-at-once Seq2Seq (T5-large)	24.27	7.11	21.76	57.58	-	-
All-at-once Seq2Seq (T5-3b)	27.99	8.72	23.36	62.03	-	-
SubeventWriter (BART-base)	29.62	8.35	21.59	60.42	↑ 8.61	↑ 3.83
SubeventWriter (BART-large)	31.31	9.41	22.52	61.83	↑ 9.47	↑ **4.68**
SubeventWriter (T5-base)	30.74	8.89	22.44	61.81	↑ **10.41**	↑ 3.26
SubeventWriter (T5-large)	33.01	10.39	23.07	64.19	↑ 8.74	↑ 3.28
SubeventWriter (T5-3b)	**34.75**	**11.30**	**24.17**	**65.67**	↑ 6.76	↑ 2.58

Table 1: Performance of all frameworks on the testing set of the WikiHow dataset. *SubeventWriter* is our model. We abbreviate BLEU-1, BLEU-2, ROUGE-L, and BERTScore to B-1, B-2, R-L, and BERT, respectively. Compared to All-at-once Seq2Seq, improvements of our frameworks are shown under $\Delta_{B\text{-}1}$ and $\Delta_{B\text{-}2}$ for each size of T5 and BART. We also include the performance of all models on the validation set in Appendix C.

models in the zero-shot setting. We consider GPT-J (Wang and Komatsuzaki, 2021) and T5-11b, which contain ~6 billion and ~11 billion parameters, respectively. We choose the prompt template "How to S? Generate the events to solve it." for every process S.

4.4 Implementation Details

We fine-tune *SubeventWriter* and All-at-once Seq2Seq based on T5-base/large/3b and BART-base/large for four epochs. The best checkpoint is selected according to the sum of all metrics on the validation set. The grid search explored learning rates of 1e-5, 5e-5, 1e-4, 5e-4, batch size of 32 and 64, and weight λ of coherence scores (Eq. 3) of 0.5, 1, 2, 5. We test multiple LMs for the coherence controller and choose BERT-base due to its efficiency. We show more details in Appendix B.

5 Main Evaluation

We show the results on the testing set of the WikiHow dataset in Table 1. In general, *SubeventWriter* can generate relevant sub-event sequences, outperforming all baseline frameworks by a great margin. For example, 11.30% of bi-grams generated by *SubeventWriter* (T5-3b) are covered by the references, increasing by 2.58% absolutely and 29.6% relatively compared to All-at-once Seq2Seq (T5-3b). Even though GPT-J and T5-11b are much larger, the smallest fine-tuned *SubeventWriter* (BART-base) can still surpass them.

Besides, our framework improves more significantly on smaller-sized language models and is parameter efficient. With T5 going down from "3b" to "base", we observe improvements increase (e.g., from 6.76% to 10.41% for BLEU-1). Also, *SubeventWriter* (T5-base) achieves comparable performance compared to All-at-once Seq2Seq (T5-3b) with only about 12% parameters[5] because generating one event each time is not hard to T5-base with the help of the coherence controller.

Another interesting observation is that when comparing between *SubeventWriter* based on T5 and BART, T5 always performs slightly better than BART in both "base" and "large" sizes. Such advances are consistent with intuition since T5 is about 1.5x - 2x larger than BART.

In the rest of this section, we conduct more analysis to demonstrate the reason behind the success of *SubeventWriter*.

5.1 Ablation Study

To measure the contribution of each module to the final results, we conduct an ablation study on *SubeventWriter* in Table 2.

The first ablation experiment drops the coherence controller in *SubeventWriter* (◇ w/o CoCo), which verifies the effectiveness of controlling coherence. Then, the second ablation experiment further drops iterative event-level decoding (◇ w/o CoCo & ITER.) and substantiates the iter-

---

[5] We include the parameters of both BERT-base in the coherence controller and T5-base.

Models	**B-1**	**B-2**	**R-L**	**BERT**
Ours (BART-large)	**31.31**	**9.41**	**22.52**	**61.83**
◇ w/o CoCo	26.78	7.79	22.04	59.53
◇ w/o CoCo & ITER.	21.84	4.73	18.94	59.45
Ours (T5-large)	**33.01**	**10.39**	**23.07**	**64.19**
◇ w/o CoCo	30.41	9.14	22.75	62.19
◇ w/o CoCo & ITER.	24.27	7.11	21.76	57.58

Table 2: Ablation study on *SubeventWriter*. "w/o CoCo" refers to ablation of coherence controller. Further ablation of iterative event-level decoding is shown in "w/o CoCo & ITER." See Appendix D.1 for results on other sizes of BART and T5.

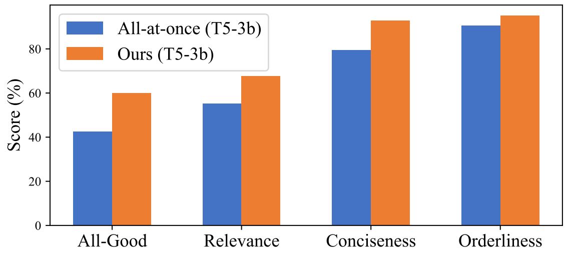

Figure 3: Human evaluation scores of *SubeventWriter* ("Ours") and All-at-once Seq2Seq ("All-at-once") based on T5-3b. "All-Good" means sub-events that satisfy all three aspects.

ative event-level decoding can boost the performance. The coherence controller depends on iterative event-level decoding as it controls sub-events one by one in chronological order. Thus, we cannot only drop iterative event-level decoding while the coherence controller is kept (no ◇ w/o ITER.).

From the results in Table 2, we observe that both the coherence controller and the iterative event-level decoding play essential roles in generating high-quality sub-event sequences. Dropping each of them will cause drastic decreases in all metrics. Taking *SubeventWriter* (BART-large) as an example, the BLEU-1 decreases by 4.53% without the coherence controller. When the iterative event-level decoding is further removed, the BLEU-1 declines to 21.84%, which is only 70% of the original BLEU-1 score.

5.2 Human Evaluation

We perform human evaluation to complement automatic evaluation. As sub-event sequences are complicated and diverse, we decompose them and score every sub-event in the following 3 aspects:
Relevance: whether a sub-event is relevant to solving the given process, measuring how well sub-events focus on the same theme (global coherence).
Conciseness: whether a sub-event is not redundant to others in the same sequence. We introduce this aspect since generating duplicates is a common failure of language models (Brown et al., 2020) and destroys local coherence.
Orderliness: whether a sub-event is placed in proper order, considering its prior sub-events. As the order of sub-events irrelevant to the given process is not defined clearly, we only consider the order of sub-events that satisfy the first aspect (Relevance).

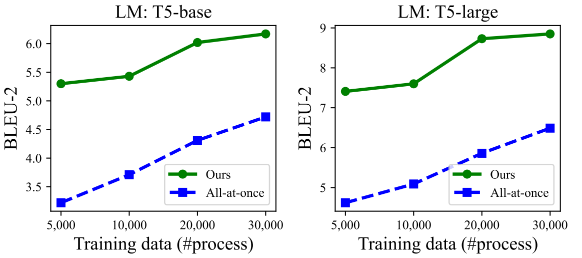

Figure 4: Few-shot learning performance of *SubeventWriter* ("Ours") and All-at-once Seq2Seq ("All-at-once") based on T5 is shown. We also include the results of other metrics in Appendix D.2.

We choose to evaluate the generation of *SubeventWriter* (T5-3b) and All-at-once (T5-3b) as they have the best quantitative performance. We randomly select 50 processes from the testing set, containing about 200 sub-events. Three experts are asked to evaluate every sub-event, yielding 1,800 total ratings for each model (200 sub-events × 3 aspects × 3 experts). We take the majority vote among three votes as the final result for each sub-events. The IAA score is 83.78% calculated using pairwise agreement proportion, and the Fleiss's κ (Fleiss, 1971) is 0.57.

We show the average scores in Figure 3. We can observe that both models achieve acceptable scores in orderliness. *SubeventWriter* generates more relevant and less redundant sub-events in comparison to All-at-once Seq2Seq as the global and local coherence in the coherence controller reflects the relevance and conciseness, respectively. Our model also produces more "All-Good" sub-events, which satisfy all three aspects.

5.3 Few-shot Learning Ability

We conduct main evaluation on the WikiHow dataset, which contains a large training set. To better understand the generalization ability of *Subeven-*

Models	SMILE				DeScript			
	B-1	**B-2**	**R-L**	**BERT**	**B-1**	**B-2**	**R-L**	**BERT**
Top-1 Similar Sequence (Glove)	24.42	0.91	12.54	52.66	39.26	5.50	16.30	56.80
Top-1 Similar Sequence (SBERT)	25.98	1.85	11.62	53.28	45.89	8.57	17.02	58.73
All-at-once (BART-base)	42.22	10.14	20.51	54.49	66.90	28.61	28.35	60.99
All-at-once (BART-large)	46.25	11.45	21.80	55.79	71.04	33.27	29.29	62.15
All-at-once (T5-base)	29.07	5.56	22.57	45.38	59.07	20.43	32.53	54.69
All-at-once (T5-large)	37.31	11.07	25.54	55.69	62.97	27.57	33.69	59.91
All-at-once (T5-3b)	40.11	10.23	26.41	60.29	74.60	44.15	36.14	68.50
Ours (BART-base)	44.67	10.83	24.07	54.19	74.02	36.93	31.03	61.69
Ours (BART-large)	48.78	16.22	27.17	56.53	80.95	43.93	33.70	64.39
Ours (T5-base)	43.71	10.44	24.09	56.90	76.41	39.74	32.73	63.84
Ours (T5-large)	45.13	12.68	26.51	60.29	82.54	47.34	35.29	68.32
Ours (T5-3b)	**53.41**	**15.21**	**28.69**	**60.68**	**86.27**	**51.24**	**37.10**	**70.17**

Table 3: Performance of zero-shot transfer learning on SMILE and DeScript. *SubeventWriter* ("Ours") outperforms the All-at-once Seq2Seq baseline ("All-at-once") by a large margin.

Dataset	#Process	#Seq	Avg-Ref	Avg-Len
SMILE	22	386	17.55	9.06
DeScript	42	3845	91.55	8.32

Table 4: Statistics of SMILE and DeScript. **#Process**, **#Seq**, **Avg-Ref**, and **Avg-Len** are the number of processes, sub-event sequences, average sequences per process, and average sub-events per sequence, respectively.

tWriter, we conduct few-shot experiments to confirm its ability to generalize with fewer data.

Referring to the size of the validation set (5,000 examples), we conduct experiments with training data of 5,000, 10,000, 20,000, and 30,000 shots (1x, 2x, 4x, 6x as large as the validation set)[6]. As shown in Figure 4, *SubeventWriter* achieves better performance compared to All-at-once Seq2Seq, demonstrating that *SubeventWriter* owns the ability to generalize with fewer data.

5.4 Zero-shot Transfer Learning

To further verify the generalization ability of *SubeventWriter*, we test it on two small-scale and domain-specific datasets: SMILE (Regneri et al., 2010) and DeScript (Wanzare et al., 2016). Both contain hundreds of human-curated sub-event sequences pertaining to human activities. Statistics for each dataset are in Table 4.

We directly use *SubeventWriter* fine-tuned on the WikiHow dataset to test its zero-shot transferring ability because it performs well on the WikiHow dataset. Since we do not tune hyper-parameters on these datasets, we treat each entire dataset as a testing set, and there is no validation set.

We report the zero-shot transferring results on *SubeventWriter* on SMILE and DeScript in Table 3. Among all baseline methods introduced in Section 4.3, we choose Top-one Similar Sequence and All-at-once Seq2Seq as baselines. The method Zero-shot Large LM does not fit WikiHow data, so it is not suitable to test the zero-shot transferring ability. From Table 3, we can find the performance on SMILE and DeScript is higher than the WikiHow dataset for all models since more references are provided. We find *SubeventWriter* surpasses Top-one Similar Sequence and All-at-once Seq2Seq on both datasets and all model sizes. Such improvements indicate that our framework is able to learn non-trivial knowledge about sub-event sequences and has a strong generalization ability.

5.5 Cutting Down the Model Parameters

Most of our experiments use T5-base (~220M parameters) and T5-large (~770M parameters), or the counterpart of BART, but in practice, we might prefer to use smaller models due to computational limitations. Here, we investigate the impact of model size by using T5-small (~60M parameters) models. Table 5 presents the results for fine-tuning All-at-once Seq2Seq (T5-small) and *SubeventWriter* (T5-small). Since the coherence controller is based on BERT-base (~110M parameters), we remove it from *SubeventWriter* to keep the number of parameters consistent for a fair comparison.

There are two meaningful observations. First, *SubeventWriter* (T5-small) can still provide superior sub-event sequences compared to All-at-once

---

[6] The full training set is 15x as large as the validation set

Models	**B-1**	**B-2**	$\delta_{B\text{-}1}$	$\delta_{B\text{-}2}$
All-at-once	17.18	4.25	↓ 3.15	↓ 1.38
SubeventWriter	**23.53**	**5.67**	↓ 5.03	↓ 2.60

Table 5: Performance of using T5-small. $\delta_{B\text{-}1}$ and $\delta_{B\text{-}2}$ indicate performance drops when replacing T5-base with T5-small. See Appendix D.3 for full results.

Models	**Valid**		**Test**	
	MAE	RMSE	MAE	RMSE
All-at-once (BART-large)	2.25	2.60	2.27	2.64
All-at-once (T5-large)	1.37	1.88	1.39	1.89
Ours (BART-large)	1.40	1.86	1.45	1.90
Ours (T5-large)	**1.34**	**1.80**	**1.38**	**1.84**

Table 6: Regression errors of sub-event sequence length of *SubeventWriter* and All-at-once Seq2Seq, which verify our framework can predict precise lengths.

Process1: How to make a felt heart card?
Reference: "Measure heart shapes." -> "Cut out heart shapes." -> "Glue into place on a card."

SubeventWriter w/o CoCo:
"Cut out a heart shape." -> "Glue the heart to a card base." -> "Add a sentiment."
SubeventWriter:
"Cut out a heart shape." -> "Glue the heart to a card base." -> "Add embellishments."

Process2: How to make Brownie batter dip?
Reference: "Whip the butter." -> "Add two cups of sugar." -> "Add other ingredients." ->"Serve the dip"

SubeventWriter w/o CoCo:
"Melt the butter." -> "Add the sugar and flour." -> "Add the chocolate chips." -> "Add the chocolate chips." -> "Serve."
SubeventWriter:
"Melt the butter." -> "Add the sugar and flour." -> "Add the remaining ingredients." -> "Serve"

Figure 5: Case Study. We show the generation without coherence controller (w/o CoCo) to illustrate how *SubeventWriter* works. We mark digressive (in Process1) and redundant (in Process2) sub-events with red. Their corrections are marked with green.

Seq2Seq (T5-small) by a large margin (e.g., 6.35% in BLEU-1). Second, both *SubeventWriter* and All-at-once Seq2Seq perform worse when we replace T5-base with T5-small since the model size reduces to 27% of the original one.

5.6 Comparison of Sub-event Sequence Length

While prior experiments and analysis mainly focus on the generated content, we also compare lengths of generated sub-event sequences with the ground truth to better assess *SubeventWriter*. We consider the lengths as a regression problem and decide to use two metrics: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

From Table 6, we can observe that *SubeventWriter* achieves less mean absolute and root mean squared error than All-at-once Seq2Seq, which indicates that our framework can generate sequences with more precise numbers of sub-events.

5.7 Case Study

We show two sub-event sequences produced by *SubeventWriter* in Figure 5. We also generate sub-events without the coherence controller to analyze how it works. From the first example, we can observe that *SubeventWriter* without the coherence controller produces a digressive sub-event "Add a sentiment." The coherence controller can correct the generation and keep a consistent theme across the process and all sub-events (global coherence). From the second example, two redundant sub-events, "Add the chocolate chips," are generated with flawed discourse relation between them (local coherence). We can see the coherence controller rectifies the redundancy by considering coherence in decoding.

6 Related Work

Understanding events has been a challenging task in NLP for a long time (Chen et al., 2021), to which the community has dedicated many works. Chambers and Jurafsky (2008) first introduced the narrative cloze task, where models are asked to predict the next event from given ones. After them, a few works are devoted to better modeling the event representations (Pichotta and Mooney, 2014, 2016; Granroth-Wilding and Clark, 2016; Li et al., 2018; Ding et al., 2019; Bai et al., 2021). Mostafazadeh et al. (2016) studied the story cloze test, where a system needs to choose the correct ending for a short story. Nonetheless, those tasks emphasize the semantic similarity and relatedness among events, ignoring how events are organized coherently.

A similar work to ours is process induction (Zhang et al., 2020a), where they proposed a statistical framework to generate a sub-event sequence of a given process. The framework aggregates existing events with conceptualization and instantiation. The difference between our work and theirs is that we consider the coherence of both actions and their objects in generation. Tasks about processes in different forms are also studied, including sub-event sequence typing (Chen et al., 2020; Pepe et al., 2022), sub-event selection (Zhang et al., 2020c), chronological ordering (Jin et al., 2022), script construction with specified length (Lyu et al., 2021b), and multi-relation prediction (Lee and Goldwasser, 2019). Compared to their settings, our work directly tackles the most challenging one, where models are asked to generate whole sub-event sequences.

7 Conclusion

In this paper, we try to construct coherent sub-event sequences by considering coherence in event-level decoding. Our *SubeventWriter* generates sub-events iteratively. A coherence controller is introduced to re-rank candidates in each iteration. The extensive experiments demonstrate the effectiveness of *SubeventWriter*.

8 Acknowledgement

The authors of this paper were supported by the NSFC Fund (U20B2053) from the NSFC of China, the RIF (R6020-19 and R6021-20) and the GRF (16211520) from RGC of Hong Kong, the MHKJFS (MHP/001/19) from ITC of Hong Kong and the National Key R&D Program of China (2019YFE0198200) with special thanks to HKMAAC and CUSBLT, and the Jiangsu Province Science and Technology Collaboration Fund (BZ2021065). We also thank the support from NVIDIA AI Technology Center (NVAITC) and the UGC Research Matching Grants (RMGS20EG01-D, RMGS20CR11, RMGS20CR12, RMGS20EG19, RMGS20EG21).

Limitations

The main limitation is that our *SubeventWriter* framework lacks knowledge while it needs to understand multiple entities in processes and how they interact. As shown in Figure 6, we ask the framework "How to make strawberry cupcakes?" However, *SubeventWriter* ignores "strawberry" in the question, which shows *SubeventWriter* does not have knowledge about general cupcakes and strawberry cupcakes. Thus, it cannot infer the way to make strawberry cupcakes from making cupcakes. Future work can investigate effective ways to integrate more knowledge and give models stronger reasoning ability. For example, Zhang et al. (2020a) utilized the hierarchical structure among events to conceptualize and instantiate similar processes.

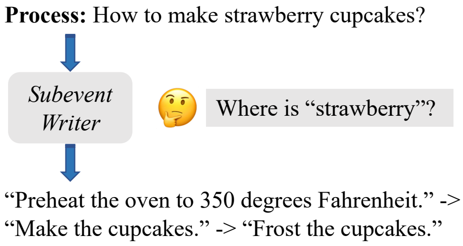

Figure 6: An error analysis. The *SubeventWriter* ignores "strawberry" in the question and answers how to make cupcakes.

Models	**B-1**	**B-2**	**R-L**	**BERT**
SubeventWriter (T5-3b)	34.75	11.30	24.17	65.67
Human Performance	54.47	28.33	38.28	73.18

Table 7: Human performance on the WikiHow dataset. We add *SubeventWriter* (T5-3b), which achieves the best machine performance.

We also test human performance to show the limitations of *SubeventWriter* and the large room for improvements. Notice that a process usually owns multiple ground truth references, which are annotated by humans. For every process, we randomly select a sub-event sequence from ground truth as a human prediction. The randomly selected one will be excluded from references.

From the results in Table 7, we can observe that there is still a notable gap between machine performance and human performance. For example, the BLEU-2 of human performance is more than twice of *SubeventWriter* (T5-3b) (28.33% vs. 11.30%).

References

Long Bai, Saiping Guan, Jiafeng Guo, Zixuan Li, Xiaolong Jin, and Xueqi Cheng. 2021. Integrating deep event-level and script-level information for script event prediction. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9869–9878.

Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D. Manning. 2014. Modeling biological processes for reading comprehension. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1499–1510, Doha, Qatar. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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.

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

Muhao Chen, Hongming Zhang, Qiang Ning, Manling Li, Heng Ji, Kathleen McKeown, and Dan Roth. 2021. Event-centric natural language processing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Tutorial Abstracts*, pages 6–14.

Muhao Chen, Hongming Zhang, Haoyu Wang, and Dan Roth. 2020. What are you trying to do? semantic typing of event processes. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 531–542.

Edward Craig et al. 1998. *Routledge encyclopedia of philosophy: Index*, volume 8. Taylor & Francis.

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

Xiao Ding, Kuo Liao, Ting Liu, Zhongyang Li, and Junwen Duan. 2019. Event representation learning enhanced with external commonsense knowledge. 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 4894–4903.

Xinya Du and Claire Cardie. 2020. Event extraction by answering (almost) natural questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 671–683.

Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378.

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.

Zijian Jin, Xingyu Zhang, Mo Yu, and Lifu Huang. 2022. Probing script knowledge from pre-trained models. *arXiv preprint arXiv:2204.10176*.

Prathyusha Jwalapuram, Shafiq Joty, and Xiang Lin. 2021. Rethinking self-supervision objectives for generalizable coherence modeling. *arXiv preprint arXiv:2110.07198*.

Paul Kingsbury and Martha Palmer. 2002. From treebank to propbank. In *Proceedings of the Third International Conference on Language Resources and Evaluation, Las Palmas, Spain*, pages 1989–1993.

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

I-Ta Lee and Dan Goldwasser. 2019. Multi-relational script learning for discourse relations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4214–4226.

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.

Qi Li, Heng Ji, and Liang Huang. 2013. Joint event extraction via structured prediction with global features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, Sofia, Bulgaria*, pages 73–82.

Zhongyang Li, Xiao Ding, and Ting Liu. 2018. Constructing narrative event evolutionary graph for script event prediction. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4201–4207.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Shih-Ting Lin, Nathanael Chambers, and Greg Durrett. 2021. Conditional generation of temporally-ordered event sequences. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference* *on Natural Language Processing (Volume 1: Long Papers)*, pages 7142–7157.

Ying Lin, Heng Ji, Fei Huang, and Lingfei Wu. 2020. A joint neural model for information extraction with global features. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Virtual Event*, pages 7999–8009.

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*.

Shangwen Lv, Fuqing Zhu, and Songlin Hu. 2020. Integrating external event knowledge for script learning. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 306–315.

Qing Lyu, Hongming Zhang, Elior Sulem, and Dan Roth. 2021a. Zero-shot event extraction via transfer learning: Challenges and insights. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 322–332.

Qing Lyu, Li Zhang, and Chris Callison-Burch. 2021b. Goal-oriented script construction. In *Proceedings of the 14th International Conference on Natural Language Generation*, pages 184–200.

Mohsen Mesgar and Michael Strube. 2018. A neural local coherence model for text quality assessment. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 4328–4339.

Han Cheol Moon, Tasnim Mohiuddin, Shafiq Joty, and Xu Chi. 2019. A unified neural coherence model. *arXiv preprint arXiv:1909.00349*.

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.

Alexander PD Mourelatos. 1978. Events, processes, and states. *Linguistics and philosophy*, 2(3):415–434.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Sveva Pepe, Edoardo Barba, Rexhina Blloshmi, and Roberto Navigli. 2022. Steps: Semantic typing of event processes with a sequence-to-sequence approach.

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.

Karl Pichotta and Raymond Mooney. 2016. Learning statistical scripts with lstm recurrent neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

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*, 21:1–67.

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.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. 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 3982–3992.

Maarten Sap, Ronan LeBras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. 2019. ATOMIC: An atlas of machine commonsense for if-then reasoning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, Honolulu, USA*, pages 3027–3035.

Roger C Schank and Robert P Abelson. 1977. Scripts, plans, goals, and understanding: an inquiry into human knowledge structures.

Noah A. Smith, Yejin Choi, Maarten Sap, Hannah Rashkin, and Emily Allaway. 2018. Event2Mind: Commonsense inference on events, intents, and reactions. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Melbourne, Australia*, pages 463–473.

Teun A Van Dijk. 1980. The semantics and pragmatics of functional coherence in discourse. *Speech act theory: Ten years later*, pages 49–65.

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

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Haoyu Wang, Muhao Chen, Hongming Zhang, and Dan Roth. 2020. Joint constrained learning for event-event relation extraction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 696–706.

Haoyu Wang, Hongming Zhang, Muhao Chen, and Dan Roth. 2021. Learning constraints and descriptive segmentation for subevent detection. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5216–5226.

Lilian DA Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. 2016. A crowdsourced database of event sequence descriptions for the acquisition of high-quality script knowledge.

Hongming Zhang, Muhao Chen, Haoyu Wang, Yangqiu Song, and Dan Roth. 2020a. Analogous process structure induction for sub-event sequence prediction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1541–1550.

Hongming Zhang, Xin Liu, Haojie Pan, Haowen Ke, Jiefu Ou, Tianqing Fang, and Yangqiu Song. 2022. Aser: Towards large-scale commonsense knowledge acquisition via higher-order selectional preference over eventualities. *Artificial Intelligence*, page 103740.

Hongming Zhang, Xin Liu, Haojie Pan, Yangqiu Song, and Cane Wing-Ki Leung. 2020b. Aser: A large-scale eventuality knowledge graph. In *Proceedings of the web conference 2020*, pages 201–211.

Hongming Zhang, Haoyu Wang, and Dan Roth. 2021. Zero-shot label-aware event trigger and argument classification. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1331–1340.

Li Zhang, Qing Lyu, and Chris Callison-Burch. 2020c. Reasoning about goals, steps, and temporal ordering with wikihow. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4630–4639.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Yucheng Zhou, Tao Shen, Xiubo Geng, Guodong Long, and Daxin Jiang. 2022. Claret: Pre-training a correlation-aware context-to-event transformer for event-centric generation and classification. *Proceedings of the 60th Annual Meeting of the ACL.*

	Process: cook eggs **Reference:** Place eggs in a pot of water. → Bring the water to a boil. → Turn off the heat and place the eggs in cold water.
1	**Input:** How to cook eggs? Step 1: [M] **Output:** Place eggs in a pot of water.
2	**Input:** How to cook eggs? Step 1: Place eggs in a pot of water. Step 2: [M] **Output:** Bring the water to a boil.
3	**Input:** How to cook eggs? Step 1: Place eggs in a pot of water. Step 2: Bring the water to a boil. Step 3: [M] **Output:** Turn off the heat and place the eggs in cold water.
4	**Input:** How to cook eggs? Step 1: Place eggs in a pot of water. Step 2: Bring the water to a boil. Step 3: Turn off the heat and place the eggs in cold water. Step 4: [M] **Output:** none

Table 8: An example of the process "cook eggs" with the prompt template. Each sub-event in the sequence is regarded as the output in turns. [M] is a masked token used in pre-train models, like <extra_id_0> of T5.

A Input and Output Examples

We list examples of input and output in this section,

A.1 Examples of the Prompt Template

In Table 8 and Table 9, we give two examples to show how we train *SubeventWriter* with the prompt template. We train *SubeventWriter* to generate one sub-event each time in chronological order and append it back to the input. If all sub-events are generated, *SubeventWriter* generates "none".

A.2 Examples of Training the Coherence Controller

In Table 10, we give positive and negative examples to show how we train the coherence controller. For negative examples, we provide examples for using a duplicate sub-event to corrupt the ***local coherence*** and using an irrelevant sub-event to corrupt the ***global coherence***.

A.3 Examples of the Inference Stage of the Coherence Controller

In Table 11, we give two candidates for the third sub-event in the process "make a felt hear card". We also show the input to the coherence controller, which is concatenated from the process, sub-events generated in prior iterations, and current candidates. The coherence controller can assign coherent input higher scores and penalize incoherent input.

	Process: buy a house **Reference:** Getting your financials in order. → Shopping for a home. → Making an offer and finalizing the deal.
1	**Input:** How to buy a house? Step 1: [M] **Output:** Getting your financials in order.
2	**Input:** How to buy a house? Step 1: Getting your financials in order. Step 2: [M] **Output:** Shopping for a home.
3	**Input:** How to buy a house? Step 1: Getting your financials in order. Step 2: Shopping for a home. Step 3: [M] **Output:** Making an offer and finalizing the deal.
4	**Input:** How to buy a house? Step 1: Getting your financials in order. Step 2: Shopping for a home. Step 3: Making an offer and finalizing the deal. Step 4: [M] **Output:** none

Table 9: An example of the process "buy a house" with the prompt template. Each sub-event in the sequence is regarded as the output in turns. [M] is a masked token used in pre-train models, like <extra_id_0> of T5.

	Process: make a felt heart card **Sub-events generated in prior iterations:** Cut out a heart shape. → Glue the heart to a card base.
1	**Candidate:** Add a sentiment. **Input:** How to make a felt heart card? Step 1: Cut out a heart shape. Step 2: Glue the heart to a card base. Step 3: Add a sentiment. **Coherence Score:** 0.23 (low score)
1	**Candidate:** Add embellishments. **Input:** How to make a felt heart card? Step 1: Cut out a heart shape. Step 2: Glue the heart to a card base. Step 3: Add embellishments. **Coherence Score:** 0.82 (high score)

Table 11: An example of the process "make a felt heart card" for the inference stage of the coherence controller. We compare two candidates for the third sub-events. We can see that the coherence controller can score the coherent candidate higher.

B Implementation Details

We conduct all experiments on 8 NVIDIA A100 GPUs.

B.1 Coherence Controller

The coherence controller is fine-tuned on BERT-base due to efficiency. We also tested three other variants of the Transformer (Vaswani et al., 2017): BERT-large, RoBERTa-large (Liu et al., 2019), and RoBERTa-base. We fine-tune them with sub-event sequences from WikiHow to keep the domain consistent inside *SubeventWriter*. Two negative examples are sampled using ***duplicate sub-event*** and the same number using ***irrelevant sub-event*** ($2N = 4$ in total).

Process: have a relaxing evening **Reference:** Turn the lights down. → Put on some music or some relaxing nature sounds. → Make sure the temperature is comfortable. → Turn your phone off.
Example: How to have a relaxing evening? Step 1: Turn the lights down. Step 2: Put on some music or some relaxing nature sounds. Step 3: Make sure the temperature is comfortable. Step 4: Turn your phone off. **Label:** Positive
Example: How to have a relaxing evening? Step 1: Turn the lights down. Step 2: Put on some music or some relaxing nature sounds. Step3: Turn the lights down. Step 4: Make sure the temperature is comfortable. Step 6: Turn your phone off. **Label:** Negative (with a duplicate sub-event)
Example: How to have a relaxing evening? Step 1: Turn the lights down. Step 2: Put on some music or some relaxing nature sounds. Step 3: Make sure the temperature is comfortable. Step 4: Place eggs in a pot of water. Step 5: Turn your phone off. **Label:** Negative (with an irrelevant sub-event)

Table 10: Positive and negative examples of the process "have a relaxing evening" for the training stage of the coherence controller. We mark the sub-event used to build negative examples with blue color.

Models	Local	Global	All
BERT-base	95.76	90.52	93.14
BERT-large	95.57	91.62	93.59
RoBERTa-base	97.11	92.16	94.63
RoBERTa-large	96.63	94.17	95.40

Table 12: Accuracy of coherence controllers. "Local" and "Global" refer to testing sets with corrupted local and global coherence, respectively. "All" contains all testing data of both sets.

We build two testing sets with positive and negative samples of 1:1. Negative examples in the first testing set are examples with corrupted local coherence, while those in the second set are examples with corrupted global coherence. Accuracy is shown in Table 12 on both testing sets. We also show accuracy on all testing data of both sets ("All"). We can observe that BERT-base already achieves satisfying accuracy (93.14%). Using larger models does not improve too much and increases computation cost.

B.2 Best Hyper-parameters

We collect the best hyper-parameters of *SubeventWriter* and All-at-once Seq2Seq in Table 13, including learning rate, batch size, and the weight λ

Models	Ours			All-at-once	
	LR	BS	λ	LR	BS
BART-base	5e-5	32	2	5e-5	32
BART-large	5e-5	32	1	1e-5	32
T5-base	5e-4	64	5	1e-3	32
T5-large	5e-5	32	0.5	5e-4	32
T5-3b	5e-5	64	0.5	1e-4	32

Table 13: The best hyper-parameters for *SubeventWriter* ("Ours") and All-at-once Seq2Seq ("All-at-once"). "LR", "BS" and "λ" refer to learning rate, batch size and the weight λ, respectively.

of coherence scores (Eq. 3).

C Results on WikiHow Validation Dataset

We collect the performance on the validation set of the WikiHow dataset in Table 14. *SubeventWriter* also works well on validation data.

D Main Evaluation and Analysis

We provide complementary results of main evaluation and analysis as follows.

D.1 Full Results of Ablation Study

Here we present the ablation study results of *SubeventWriter* based on all BART and T5 models in Table 15 and Table 16, respectively.

Models	B-1	B-2	R-L	BERT
Ours (BART-base)	**29.62**	**8.35**	**21.59**	**60.42**
⋄ w/o CoCo	25.82	7.18	21.24	56.60
⋄ w/o CoCo & ITER.	21.01	4.52	18.83	58.79
Ours (BART-large)	**31.31**	**9.41**	**22.52**	**61.83**
⋄ w/o CoCo	26.78	7.79	22.04	59.53
⋄ w/o CoCo & ITER.	21.84	4.73	18.94	59.45

Table 15: Ablation study results on BART-base and BART-large.

D.2 Full Results of Few-shot Learning

We offer full results of few-shot learning on the testing set of WikiHow dataset in Table 17 for All-at-once Seq2Seq and Table 18 for *SubeventWriter*.

Models	B-1	B-2	R-L	BERT
Ours (T5-base)	**30.74**	**8.89**	**22.44**	**61.81**
⋄ w/o CoCo	28.56	8.27	22.01	58.89
⋄ w/o CoCo & ITER.	20.33	5.63	20.22	52.15
Ours (T5-large)	**33.01**	**10.39**	**23.07**	**64.19**
⋄ w/o CoCo	30.41	9.14	22.75	62.19
⋄ w/o CoCo & ITER.	24.27	7.11	21.76	57.58
Ours (T5-3b)	**34.75**	**11.30**	**24.17**	**65.67**
⋄ w/o CoCo	33.63	10.90	24.15	65.57
⋄ w/o CoCo & ITER.	27.99	8.72	23.36	62.03

Table 16: Ablation study results on T5-base, T5-large and T5-3b.

All-at-once	#SHOT	B-1	B-2	R-L	BERT
T5-base	n = 5K	16.37	3.22	16.78	47.65
	n = 10K	16.60	3.71	17.44	48.50
	n = 20K	18.43	4.31	18.30	49.90
	n = 30K	18.65	4.72	18.54	50.22
T5-large	n = 5K	20.15	4.62	19.07	54.27
	n = 10K	20.59	5.09	19.69	55.21
	n = 20K	22.35	5.86	20.45	56.93
	n = 30K	23.00	6.49	21.06	56.52

Table 17: Few-shot learning results of All-at-once Seq2Seq based on T5-base and T5-large on the testing set.

D.3 Full Results of T5-small

We show the full results of *SubeventWriter* (T5-small) and All-at-once Seq2Seq (T5-small) in Table 19. The performance drops compared to *SubeventWriter* (T5-base) and All-at-once Seq2Seq (T5-base), respectively, as shown in Table 20. Notice that we test the *SubeventWriter* without the coherence controller. Thus, to calculate performance changes, please refer to "⋄ w/o CoCo" of *SubeventWriter* (T5-base) in the ablation study (Table 15 and Table 16).

Ours	#SHOT	B-1	B-2	R-L	BERT
T5-base	n = 5K	23.68	5.30	19.71	54.77
	n = 10K	24.00	5.43	19.77	55.29
	n = 20K	25.47	6.02	20.43	56.45
	n = 30K	25.49	6.17	20.57	56.98
T5-large	n = 5K	29.07	7.41	21.48	62.24
	n = 10K	29.14	7.60	21.99	62.43
	n = 20K	30.23	8.73	22.28	63.42
	n = 30K	30.67	8.85	22.73	63.42

Table 18: Few-shot learning results of *SubeventWriter* based on T5-base and T5-large on the testing set.

Models	**B-1**	**B-2**	**R-L**	**BERT**	$\Delta_{B\text{-}1}$	$\Delta_{B\text{-}2}$
Zero-shot Large LM (GPT-J)	14.02	0.42	16.53	45.33	-	-
Zero-shot Large LM (T5-11b)	20.31	0.97	14.13	54.72	-	-
Top-1 Similar Sequence (Glove)	16.74	1.08	11.85	57.55	-	-
Top-1 Similar Sequence (SBERT)	18.12	1.86	13.06	59.95	-	-
All-at-once Seq2Seq (BART-base)	21.09	4.41	18.70	58.84	-	-
All-at-once Seq2Seq (BART-large)	22.39	4.77	19.09	59.50	-	-
All-at-once Seq2Seq (T5-base)	20.51	5.52	19.76	51.83	-	-
All-at-once Seq2Seq (T5-large)	24.39	7.30	21.64	57.23	-	-
All-at-once Seq2Seq (T5-3b)	28.22	8.60	22.98	62.08	-	-
SubeventWriter (BART-base)	29.70	8.26	21.33	60.26	↑ 8.61	↑ 3.85
SubeventWriter (BART-large)	31.57	9.44	22.17	61.95	↑ 9.18	↑ **4.67**
SubeventWriter (T5-base)	32.09	9.31	22.51	61.99	↑ **11.58**	↑ 3.79
SubeventWriter (T5-large)	33.97	10.65	23.33	64.40	↑ 9.58	↑ 3.35
SubeventWriter (T5-3b)	**35.64**	**12.07**	**24.08**	**65.79**	↑ 7.42	↑ 3.47

Table 14: Performance of all frameworks on the validation data of the WikiHow dataset.

Split	**Models**	**B-1**	**B-2**	**R-L**	**BERT**
valid	All-at-once	17.80	4.28	18.27	47.96
	SubeventWriter	**22.98**	**5.47**	**19.58**	**51.38**
test	All-at-once	17.18	4.25	17.90	48.26
	SubeventWriter	**23.53**	**5.67**	**19.69**	**51.26**

Table 19: Performance of using T5-small on validation and testing sets. “valid” and “test” are shortened forms of validation and testing.

Split	**Models**	$\delta_{B\text{-}1}$	$\delta_{B\text{-}2}$	$\delta_{R\text{-}L}$	δ_{BERT}
valid	All-at-once	2.71	1.24	1.49	3.87
	SubeventWriter	6.76	3.12	2.64	7.48
test	All-at-once	3.15	1.38	2.32	3.89
	SubeventWriter	5.03	2.60	2.32	7.63

Table 20: Performance drops when we replace T5-base with T5-small.