Don’t Parse, Generate! A Sequence to Sequence Architecture for Task-Oriented Semantic Parsing

Subendhu Rongali
University of Massachusetts Amherst
Amherst, MA, USA
srongali@cs.umass.edu

Luca Soldaini
Amazon Alexa Search
Manhattan Beach, CA, USA
lssoldai@amazon.com

Emilio Monti
Amazon Alexa
Cambridge, UK
monti@amazon.co.uk

Wael Hamza
Amazon Alexa AI
New York, NY, USA
waelhamz@amazon.com

ABSTRACT
Virtual assistants such as Amazon Alexa, Apple Siri, and Google Assistant often rely on a semantic parsing component to understand which action(s) to execute for an utterance spoken by its users. Traditionally, rule-based or statistical slot-filling systems have been used to parse “simple” queries; that is, queries that contain a single action and can be decomposed into a set of non-overlapping entities. More recently, shift-reduce parsers have been proposed to process more complex utterances. These methods, while powerful, impose specific limitations on the type of queries that can be parsed; namely, they require a query to be representable as a parse tree.

In this work, we propose a unified architecture based on Sequence to Sequence models and Pointer Generator Network to handle both simple and complex queries. Unlike other works, our approach does not impose any restriction on the semantic parse schema. Furthermore, experiments show that it achieves state of the art performance on three publicly available datasets (ATIS, SNIPS, Facebook TOP), relatively improving between 3.3% and 7.7% in exact match accuracy over previous systems. Finally, we show the effectiveness of our approach on two internal datasets.

KEYWORDS
Natural Language Understanding, Semantic Parsing, Voice Assistants, Sequence to Sequence models

ACM Reference Format:

∗Work done when the author was at Amazon Alexa AI for a summer internship.
We propose a unified architecture to solve the task of semantic parsing for both simple and complex queries. This architecture can also be adapted to handle queries containing slots with overlapping spans. It consists of a Sequence to Sequence model and a Pointer Generator Network. We choose a pretrained BERT [2] model as our encoder. Our decoder is modeled after the transformer decoder described in Vaswani et al. [33] and is augmented with a Pointer Generator Network [9, 34] which allows us to learn to generate pointers to the source sequence in our target sequence. Figure 3 shows this architecture parsing an example query. We train the model using a cross-entropy loss function with label smoothing.

In this section, we first describe how we formulate queries and their semantic parses as sequences with pointers for our architecture. We then describe our encoder and decoder components.

### 2.1 Query Formulation

A Sequence to Sequence architecture is trained on samples with a source sequence and a target sequence. When some words in the target sequence are contained in the source sequence, they can be replaced with a separate pointer token that points to that word in the source to be able to apply the Pointer Generator Network.

Take the example query from Figure 1. In our architecture, we use the query as our source sequence. The target sequence is constructed by combining the intent with all the slots, in order, with each slot also containing its source words. The source and target sequences now look as follows.

**Source:** play the song don't stop believin by journey
**Target:** PlaySongIntent SongName( @ptr1, @ptr4, @ptr5 )SongName ArtistName( @ptr2 )ArtistName

Here, each token @ptri is a pointer to the ith word in the source sequence. So @ptr3, @ptr4 and @ptr5 point to the song words don’t stop believin, and @ptr7 points to the artist word journey. The slots have open and close tags since they are enclosing a consecutive span of source tokens. The intent is just represented as a single tag at the beginning of the target sequence. We can do this for simple queries since they consist of just one intent. The target vocabulary hence consists of all the available intents, two times the number of different slots, and the pointers.

Complex queries with multiple intents and nested slots can also be transformed easily into this formulation. Figure 2 shows an example from the Facebook TOP dataset along with its parse tree. This query How far is the coffee shop can be converted into our formulation as follows.

**Source:** How far is the coffee shop
**Target:** [IN:GET_DISTANCE @ptr6, @ptr7 ]SL:DESTINATION [IN:GET_RESTAURANT_LOCATION @ptr8, @ptr9 ]SL:TYPE_FOOD @ptr4, SL:TYPE_FOOD @ptr5, IN:GET_RESTAURANT_LOCATION SL:DESTINATION IN:GET_DISTANCE

We made a minor modification to the reference parses from the TOP dataset for our formulation. We replaced the end-brackets with custom end-brackets corresponding to the intent or slot they close.

Finally, we show how we can express queries from datasets that don’t conform to either the slot-filling or Shift-reduce systems. Take the following example from the healthcare domain, where the task is to extract a patient diagnosis and related information from a clinician’s notes.

**Source:** The pt. was diagnosed with GI upper bleed today.
**Annotations:** Bleeding_Event (GI bleed), Anatomical_Site (upper)
Figure 3: Our architecture - Sequence to Sequence model with Pointer Generator Network (Seq2Seq-Ptr). The model is currently decoding the symbol after MediaType by looking at the scores over the tagging vocabulary and the attentions over the source pointers. It generates $@$ptr2 since it has the highest overall score.

A traditional slot filling system wouldn’t know which non-consecutive slots to combine, while a shift-reduce parser cannot split the middle word into a separate tag. In our architecture, we simply formulate the target sequence as follows.

Target: Bleeding_Event($@$ptr5 $@$ptr7$)Bleeding_Event
Anatomical_Site($@$ptr6$)Anatomical_Site

2.2 BERT Encoder

Language model pretraining has been shown to improve the downstream performance on many NLP tasks [2, 24, 27]. The idea is to train a language model on a large amount of text using a next word prediction objective to learn good representations for each of the words. These representations can then be fine-tuned on a given NLP task to improve the performance of an existing model. Pretrained models improve the performance of task models since they already contain a lot of useful semantic information learned through the pretraining phase. This has even more significance when the task-specific dataset is fairly small. Some examples of pretrained models in literature include word embeddings such as Word2Vec [19] and Glove [23], and contextualized representations such as ELMo [24], OpenAI-GPT [27], and BERT [2].

We choose BERT to encode the source sequence in our architecture. BERT (Bidirectional Encoder Representations from Transformers) is a language representation model architecture based on Transformers [33]. The original publicly available model was pretrained on a millions of lines of text from BooksCorpus and English Wikipedia. Unlike other language models (ELMo, OpenAI-GPT), which are trained to predict the next token given the previous sequence of words, BERT uses a composite objective that combines masked word prediction and next sentence prediction.

BERT’s architecture is based on a multi-layer bidirectional Transformer, originally implemented in Vaswani et al. [33]. The detailed implementation of this architecture can be found in Devlin et al. [2]. For our experiments, we use three different variants of BERT.

For the three public datasets, we used the checkpoint released by Devlin et al. [2]. Experiments on the two internal datasets were carried out using a model we pretrained over a large sample of queries from the live traffic of Amazon Alexa. We also experimented with a publicly-available variant of BERT called RoBERTa [17]. RoBERTa (A Robustly Optimized BERT Pretraining Approach) uses the same architecture as BERT but changes the pretraining process. The next sentence prediction objective is removed and a dynamic masking scheme is used instead of a static one like the original BERT implementation. RoBERTa was also trained with longer sequences, higher batch-sizes, and for a longer time, and was reported to match or exceed the performance of BERT in several NLP benchmarks. The detailed implementation can be found in Liu et al. [17]. Finally as an ablation study, we experimented with an encoder with no pretrained weights.

2.3 Decoder with Pointer Generator Network

We use the transformer decoder proposed in Vaswani et al. [33] in our architecture. The self-attention mechanism in the decoder learns to attend to target words before the current step, as well as all the source words in the encoder.

We set up the decoder with different numbers of units, layers, and attention-heads for different tasks based on the size and complexity of the queries. These details are provided in the experiments section.

In a traditional Sequence to Sequence model, the target words are generated from the decoder hidden states through a feed-forward layer that obtains unnormalized scores over a target vocabulary
distribution. In our architecture, we use a Pointer Generator Network to generate two different kinds of target words: words from the target vocabulary consisting of parse symbols (the intent and slot delimiters), and words that are simply pointers to the source sequence. Our Pointer Generator Network is based on the models in Vinyals et al. [34] and See et al. [29], and is closest in implementation to Jia and Liang [9].

We now describe our decoding process. For each input source sequence \( \{x_1 \ldots x_n\} \), we use the BERT encoder to encode it into a sequence of encoder hidden states \( \{e_1 \ldots e_n\} \). Having generated the first \( t-1 \) output tokens, the transformer decoder generates the token at step \( t \) as follows.

First, the decoder produces the decoder hidden state at time \( t, d_t \) by building multi-layer multi-head self-attention on the encoded output as well as the embeddings of the previously generated output sequence as described in Vaswani et al. [33]. We feed \( d_t \) through a dense layer to produce scores \( \{s_1 \ldots s_{|V|}\} \) for each word in the vocabulary \( V \). \( V \) contains all symbols necessary for the parse (ints, slots) but not regular words appearing on the source side.

We also use \( d_t \) as a query and compute unnormalized attention scores \( \{a_1 \ldots a_n\} \) with the encoded sequence. Concatenating the unnormalized attention scores (size \( n \)) and the output of the dense layer (size \( |V| \)), we obtain an unnormalized distribution over \( |V| + n \) tokens, the first \( |V| \) of which are the output parsing vocabulary and the last \( n \) of which are the \( @ptr_1 (0 < i < n) \) words pointing to the source tokens. We then feed this through a softmax layer to obtain the final probability distribution. This probability is used in the loss function during training and will be used to choose the next token to generate during inference. Since the transformer decoder uses embeddings of previously generated tokens, we use a set of special embeddings to represent \( @ptr_1 \) tokens.

In the example in Figure 3, we are trying to predict a target word after the token ‘MediaType’, at step 5. As shown in the figure, we compute the scores \( \{a_1 \ldots a_4\} \) (blue, left) over each of the source tokens, and the scores \( \{s_1 \ldots s_{|V|}\} \) (green, right) over the parsing vocabulary. We expect the model to produce the highest score for \( a_3 \), which corresponds to \( @ptr_2 \), representing the word ‘hits’.

3 DATASETS

We test our approach on five different datasets (three publicly available, two internal), which we describe in this section.

3.1 Facebook TOP

The Task Oriented Parsing (TOP) [6] dataset contains complex hierarchical and nested queries that make the task of semantic parsing more challenging. It contains around 45k annotations with 25 intents and 36 slots, randomly split into 31k training, 5k validation and 9k test utterances. The dataset mainly consists of user queries about navigation and various public events. An example from this dataset can be seen in Figure 2. The IN: prefix stands for intent while SL: is for slot. We can see how there are multiple intents and nested slots in the semantic interpretation. This makes the query much harder to interpret and parse using a simple slot tagging model that tags each word with a single slot.

3.2 SNIPS

The SNIPS dataset [1] is a public dataset that is used for training and testing semantic parsing models for voice assistants. It consists of utterances that belong to seven different intents: SearchCreativeWork, GetWeather, BookRestaurant, PlayMusic, AddToPlaylist, RateBook, and SearchScreeningEvent. Each intent contains around 2000 examples to train and 100 to test.

This dataset contains only simple queries with single intents and flat slots. An example is ‘Will there be fog in Tahquamenon Falls State Park’; where the intent is GetWeather and the slots are condition_description for fog and geographic_poi for Tahquamenon Falls State Park.

The dataset was originally used to evaluate models in the Snips Voice Platform. It has since been a widely used dataset to benchmark the performance of various task-oriented parsing models.

3.3 ATIS

The Airline Travel Information System (ATIS) [26] corpus is a widely used dataset in spoken language understanding. It was built by collecting and transcribing audio recordings of people making flight reservations in the early 90s. It consists of simple queries.

There are seventeen different goals or intents such as Flight or Aircraft capacity. This distribution is however skewed, with the Flight intent covering about 70% of the total queries. An example from this dataset consists of the query ‘How much is the cheapest flight from Boston to New York tomorrow morning?’ The intent is Airfare, while the slots tag important information like the departure and arrival cities, and the departure times.

The ATIS corpus has supported research in the field of spoken language understanding for more than twenty years. Some researchers have performed extensive error analysis on the state of the art discriminative models for this dataset and reported that despite really low error rates, there exist many unseen categories and sequences in the dataset that can benefit from incorporating linguistically motivated features [31]. This supports the continued utility of ATIS as a research corpus.

3.4 Internal Datasets

Our internal datasets consist of millions of user utterances that are used to train and test Amazon Alexa. For our experiments, we sampled two datasets of utterances, one from the music domain and the other from video domain. Utterances in these domains naturally included a large amount of entities (e.g. artists and albums names, movie and video titles), and thus represent a good benchmark for the ability of any neural model to generalize over a diverse set of queries. The example in Figure 3 is from the music domain.

The sampled music domain dataset contains 6.2M training and 200K test utterances, with 23 intents and 100 slots. The video domain dataset contains 1M training and just 5K test utterances; parses in this dataset are comprised of 24 distinct intents and 59 slots.

4 BASELINE MODELS

We benchmark our performance on the internal datasets by comparing it to a well tuned RNN based model. The model learns to perform joint intent and slot tagging using a bidirectional LSTM and a Conditional Random Field (CRF) [8]. We further enhanced
We leveraged gradual unfreezing to effectively tune the language with attention. There is also a model that works completely with just attention, the slot gated full attention model. The final baseline, Don’t Parse, Generate! A Sequence to Sequence Architecture for Task-Oriented Semantic Parsing

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>exact match</th>
<th>intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR with ELMo embeddings [4]</td>
<td>83.93</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR ensemble + ELMo + SVMRank [4]</td>
<td>87.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ2SEQ-Ptr (no pretraining)</td>
<td>79.25</td>
<td>97.43</td>
<td></td>
</tr>
<tr>
<td>SEQ2SEQ-Ptr (BERT encoder)</td>
<td>83.13</td>
<td>97.91</td>
<td></td>
</tr>
<tr>
<td>SEQ2SEQ-Ptr (RoBERTa encoder)</td>
<td>86.67</td>
<td>98.13</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Results on TOP [6]. Our system (SEQ2SEQ-Ptr) outperforms the best single method (SR + ELMo) by 3.3%. It is close to the best ensemble approach (SR + ELMo + SVRank).

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>exact match</th>
<th>intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint BiRNN [7]</td>
<td>73.20</td>
<td>96.90</td>
<td></td>
</tr>
<tr>
<td>Attention BiRNN [16]</td>
<td>74.10</td>
<td>96.70</td>
<td></td>
</tr>
<tr>
<td>Slot Gated Full Attention [5]</td>
<td>75.50</td>
<td>97.00</td>
<td></td>
</tr>
<tr>
<td>CAPSULENLU [36]</td>
<td>80.90</td>
<td>97.30</td>
<td></td>
</tr>
<tr>
<td>SEQ2SEQ-Ptr (no pretraining)</td>
<td>85.43</td>
<td>97.00</td>
<td></td>
</tr>
<tr>
<td>SEQ2SEQ-Ptr (BERT encoder)</td>
<td>86.29</td>
<td>98.29</td>
<td></td>
</tr>
<tr>
<td>SEQ2SEQ-Ptr (RoBERTa encoder)</td>
<td>87.14</td>
<td>98.00</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Results on SNIPS [1]. Our system (SEQ2SEQ-Ptr) outperforms the previous state of the art by 7.7%.

This baseline by replacing its embedding and encoder layers with a language model pretrained on a subset of Alexa’s live traffic. These components were fine-tuned on the two datasets described in Section 3.4.

For the ATIS and SNIPS datasets, we use the top four performing methods reported by Zhang et al. [36] as baselines. All these models perform joint intent and slot tagging. There are two variants that use RNNs: a simple RNN based model, and an RNN model augmented with attention. There is also a model that works completely with just attention, the slot gated full attention model. The final baseline, CapsuleNLU, uses Capsule Networks [28].

For the TOP dataset, we pick a model based on Recurrent Neural Network Grammars (RNNG) [3], the Shift Reduce Parser. We provide a brief overview of this model as described in Gupta et al. [6] - the parse tree is constructed using a sequence of transitions, or actions. The transitions are defined as a set of SHIFT, REDUCE, and the generation of intent and slot labels. SHIFT action consumes an input token (that is, adds the token as a child of the right most open sub-tree node) and REDUCE closes a sub-tree. The third set of actions is generating non-terminals: the slot and intent labels. The model learns to perform one of these actions at each step in time.

We report scores of three experimental setups with the shift reduce parser from Einolghozati et al. [4]: a simple shift reduce parser, a shift reduce parser augmented with ELMo embeddings, and an ensemble of these models augmented with ELMo and an SVM language model reranker.

5 EXPERIMENTAL SETUP

All our models were trained on a machine with 8 NVIDIA Tesla V100 GPUs, each with 16GB of memory. When using pretrained encoders, we leveraged gradual unfreezing to effectively tune the language model layers on our datasets. We used the “Base” variant of BERT and RoBERTa encoders, which uses 768-dimensional embeddings, 12 layers, 12 heads, and 3072 hidden units. When training from scratch, we used a smaller encoder consisting of 512-dimensional embeddings, 6 layers, 8 heads, and 1024 hidden units.

Depending on the dataset, we used either a 128 units, 4 layers, 3 heads, and 512 hidden units decoder (Facebook TOP, ATIS, SNIPS) or a larger 512 units, 6 layers, 8 heads, and 1024 hidden units decoder (internal Music and Video datasets). We used bi-linear product attention to score the source words in the Pointer Network.

While training, the cross entropy loss function was modified with label smoothing with $\epsilon = 0.1$. We used the Adam [12] optimizer with noam learning rate schedule [33], each adjusted differently for different datasets. At inference time, we used beam search decoding with a beam size of 4.

6 RESULTS AND DISCUSSION

We use exact match (EM) accuracy as the main metric to measure the performance of our models across all datasets. Under this metric, the entire semantic parse for a query has to match the reference parse to be counted as correct. Because EM is generally more challenging than slot-level precision and recall or semantic error rate [30], it is better suited to compare high performing systems like the ones studied in this work. For completeness, we also report the intent classification accuracy for our models.

The results from our experiments are documented in Tables 1-5. Our models match or beat the baselines across all datasets on both exact match and intent classification accuracies. We see significant improvements on both simple and complex datasets.

6.1 Complex Queries

We achieve an improvement of 2.7 (+3.3%) EM accuracy points on the TOP dataset over the state-of-the-art single model on this dataset (Table 1). Our Seq2Seq-Ptr model with RoBERTa encoder is only surpassed by the ensemble model reported in Einolghozati et al. [4] (+0.6% EM accuracy points.)

In addition, we find that even without specifying any hard requirements for the grammar of the parse trees in the complex queries, 98% of the generated parses are well formatted. For the simple query datasets, it was greater than 99% but difference is expected since the grammar is easier to learn there.

During error analysis, we found an interesting example in the TOP dataset where we believe our model generates a valid, more
We report results of our sequence to sequence model (Seq2Seq-Ptr) on four datasets (SNIPS, ATIS, internal music, internal video) that contain simple queries in Tables 2, 3, 4, and 5.

6.2 Simple Queries

We report results of our sequence to sequence model (Seq2Seq-Ptr) on four datasets (SNIPS, ATIS, internal music, internal video) that contain simple queries in Tables 2, 3, 4, and 5.

On the SNIPS and ATIS datasets, we note that the best version of our method (Seq2Seq-Ptr with RoBERTa encoder) achieves a significant improvement in EM accuracy over existing baselines (+7.7% and +4.5% respectively.) Using a BERT encoder causes a slight decrease in performance, but still achieves a meaningful improvement over the previous state of the art [36]; this is consistent with what has been observed on other NLP tasks [17]. If no pretraining is used, performance is further reduced but it is notable that this variant still beats all the baselines on the SNIPS dataset.

For our internal Alexa datasets, we note that the proposed Seq2Seq-Ptr method obtains comparable results to a BiLSTM-CRF tagger on the music domain, and slightly better EM accuracy (+1.9%) on the video domain. We believe our model wasn’t able to outperform the baseline on the music domain because the entities in this domain are very diverse, especially song or album names. Sequence tagging methods therefore benefit from having to solve a simpler task of having to tag each word in the sequence, as opposed to our un constrained model. We would however like to note that our from-scratch variants beat the from-scratch baselines on both domains. Also curiously, the performance of Seq2Seq-Ptr with a BERT encoder fell behind that of a sequence to sequence model trained from scratch. Since the scratch model uses a smaller transformer encoder (6 layers with 8 heads per layer instead of 12/12), we believe it was able to converge more effectively than the BERT encoder.

7 RELATED WORK

The task of semantic parsing for intent and slot detection is well established in literature. Traditionally, this was done with slot filling systems that classify the query and then label each word in the query. There were a few approaches that followed this system, using Recurrent Neural Networks [16, 18]. Researchers have also experimented with Convolutional Neural Networks and showed good results [11] and more recently, Capsule Networks [28, 36].

Prior to the advent of deep learning models, the task of sequence labeling was tackled with the use of Conditional Random Fields (CRF) [10, 14, 22]. CRFs learn pairwise potentials on labeling subsequent words which allow models to find more probable label sequences for a given query.

Most of this work is valid for semantic parsing for simple queries which boils down to a sequence labeling task. To handle more complex cases with hierarchical slots such as the example in Figure 2, researchers have experimented with Sequence to Sequence models and models based on Recurrent Neural Network Grammars (RNN) [3]. RNNs were shown to perform better on complex queries than RNN or Transformer-based Sequence to Sequence models [6]. Researchers have also explored models involving logical forms and discourse for language representation [15, 32, 35].

The Pointer Generator Network in our architecture was introduced in Vinyals et al. [34]. It was used in NLP applications where some words from the source sequence reappeared in the target sequence such as text summarization and style transfer [21, 25, 29]. They were also used to copy out of vocabulary words from the source to target in machine translation [13]. Our implementation of the Pointer Network is closest to the architecture in Jia and Liang [9]. By using pointers to represent the source tokens and imposing no particular logical form over our target sequence, we can handle any kind of queries for parsing. This makes our architecture as expressive as logical forms, while also being able to learn as easily as simple slot tagging systems.

8 CONCLUSION

We propose a unified architecture for the task of semantic parsing for different kinds of queries. We show that our architecture matches or outperforms existing approaches across multiple datasets: internal music and video datasets, SNIPS, ATIS, and Facebook TOP. We significantly outperform the current state of the art models on the public datasets TOP (3.3%), SNIPS (7.7%), and ATIS (4.5%).

We describe how to apply this architecture to both simple queries and complex queries with hierarchical and nested slots. We also describe how to formulate any set of queries with non-conforming grammars to work with our architecture, making this model applicable to many different types of semantic parsing. We leave the non-conforming grammar task to future work.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Exact match</th>
<th>Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiLSTM-CRF (no pretraining)</td>
<td>baseline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiLSTM-CRF (pretrained LM)</td>
<td>+3.0%</td>
<td>+0.1%</td>
<td></td>
</tr>
<tr>
<td>Seq2Seq-Ptr (no pretraining)</td>
<td>-0.5%</td>
<td>-0.7%</td>
<td></td>
</tr>
<tr>
<td>Seq2Seq-Ptr (BERT encoder)</td>
<td>-2.2%</td>
<td>-0.8%</td>
<td></td>
</tr>
<tr>
<td>Seq2Seq-Ptr (RoBERTa encoder)</td>
<td>-3.5%</td>
<td>-0.7%</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Results on an internal dataset (music). The best configuration of our method (Seq2Seq-Ptr) is comparable to a BiLSTM-CRF pretrained on a large conversational dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Exact match</th>
<th>Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiLSTM-CRF (no pretraining)</td>
<td>baseline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiLSTM-CRF (pretrained LM)</td>
<td>+3.0%</td>
<td>+0.1%</td>
<td></td>
</tr>
<tr>
<td>Seq2Seq-Ptr (no pretraining)</td>
<td>+2.9%</td>
<td>-0.1%</td>
<td></td>
</tr>
<tr>
<td>Seq2Seq-Ptr (BERT encoder)</td>
<td>+0.1%</td>
<td>-0.2%</td>
<td></td>
</tr>
<tr>
<td>Seq2Seq-Ptr (RoBERTa encoder)</td>
<td>+4.9%</td>
<td>-0.2%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Results on an internal dataset (video). The best configuration of our method (Seq2Seq-Ptr) outperforms a pre trained BiLSTM-CRF network by 1.9%.