HySPA: Hybrid Span Generation for Scalable Text-to-Graph Extraction

Liliang Ren, Chenkai Sun, Heng Ji, Julia Hockenmaier
University of Illinois, Urbana Champaign
Department of Computer Science
{liliang3,chenkai5,hengji,juliahmr}@illinois.edu

Abstract

Text-to-Graph extraction aims to automatically extract information graphs consisting of mentions and types from natural language texts. Existing approaches, such as table filling and pairwise scoring, have shown impressive performance on various information extraction tasks, but they are difficult to scale to datasets with longer input texts because of their second-order space/time complexities with respect to the input length. In this work, we propose a **Hy**brid **SP**an Gener**A**tor (**HySPA**) that invertibly maps the information graph to an alternating sequence of nodes and edge types, and directly generates such sequences via a hybrid span decoder which can decode both the spans and the types recurrently in linear time and space complexities. Extensive experiments on the ACE05 dataset show that our approach also significantly outperforms state-of-the-art on the joint entity and relation extraction task.[1]

1 Introduction

Information Extraction (IE) can be viewed as a Text-to-Graph extraction task that aims to extract an information graph (Li et al., 2014; Shi et al., 2017) consisting of mentions and types from unstructured texts, where the nodes of the graph are mentions or entity types and the edges are relation types that indicate the relations between the nodes. A typical approach towards graph extraction is to break the extraction process into sub-tasks, such as Named Entity Recognition (NER) (Florian et al., 2006, 2010) and Relation Extraction (RE) (Sun et al., 2011; Jiang and Zhai, 2007), and either perform them separately (Chan and Roth, 2011) or jointly (Li and Ji, 2014; Eberts and Ulges, 2019).

Recent joint IE models (Wadden et al., 2019; Wang and Lu, 2020; Lin et al., 2020) have shown impressive performance on various IE tasks, since they can mitigate error propagation and leverage inter-dependencies between the tasks. Previous work often uses pairwise scoring techniques to identify relation types between entities. However, this approach is computationally inefficient because it needs to enumerate all possible entity pairs in a document, and the relation type is a *null* value for most of the cases due to the sparsity of relations between entities. Also, pairwise scoring techniques evaluate each relation type independently and thus fail to capture interrelations between relation types for different pairs of mentions.

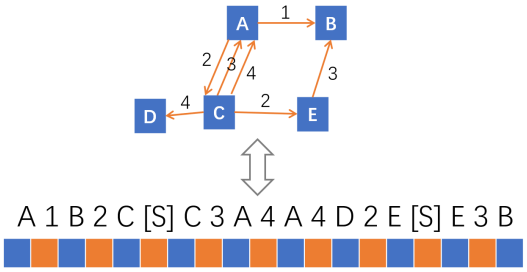

Figure 1: We represent directed multigraphs as *alternating* sequences of nodes (blue) and edges (orange). Here, the graph is traversed by Breadth First Search (BFS) with an ascending ordering of nodes and edge types. "[s]" or [SEP] is a virtual edge type, representing the end of each BFS level.

Another approach is to treat the joint information extraction task as a table filling problem (Zhang et al., 2017; Wang and Lu, 2020), and generate two-dimensional tables with a Multi-Dimensional Recurrent Neural Network (Graves et al., 2007). This can capture interrelations among entities and relations, but the space complexity grows quadratically with respect to the length of the input text, making this approach impractical for long sequences.

Some attempts, such as Seq2RDF (Liu et al., 2018) and IMoJIE (Kolluru et al., 2020), leverage the power of Seq2seq models (Cho et al., 2014)

[1]Our code is publicly available at https://github.com/renll/HySPA

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 4066–4078

to capture the interrelations among mentions and types with first-order complexity, but they all use a pre-defined vocabulary for mention prediction, which largely depends on the distribution of the target words and will not be able to handle unseen out-of-vocabulary words.

To solve these problems, we propose a first-order approach that invertibly maps the target graph to an alternating sequence of nodes and edges, and applies a hybrid span generator that directly learns to generate such alternating sequences. Our main contributions are three-fold:

- We propose a general technique to invertibly map between an information graph and an alternating sequence (assuming a given graph traversal algorithm). Generating an alternating sequence is equivalent to generating the original information graph.
- We propose a novel neural decoder that is enforced to only generate alternating sequences by decoding spans and types in a hybrid manner. For each decoding step, our decoder only has linear space and time complexity with respect to the length of the input sequence, and it can capture inter-dependencies among mentions and types due to its nature as a sequential decision process.
- We conduct extensive experiments on the Automatic Content Extraction (ACE) dataset which show that our model achieves state-of-the-art performance on the joint entity and relation extraction task which aims to extract a knowledge graph from a piece of unstructured text.

2 Modeling Information Graphs as Alternating Sequences

An **information graph** can be viewed as a heterogeneous multigraph (Li et al., 2014; Shi et al., 2017) $G = (V, E)$, where V is a set of nodes (typically representing spans (t_s, t_e) in the input document) and E is a multiset of edges with a node type mapping function $\phi : V \rightarrow Q$ and an edge type mapping function $\psi : E \rightarrow R$. Node and edge types are assumed to be drawn from a finite vocabulary. Node types can be used e.g. to represent entity types (PER, ORG, etc.), while edge types may represent relations (PHYS, ORG-AFF, etc.) between the nodes. In this work, we represent node types as separate nodes that are connected to their node v by a special edge type, [TYPE]. [2]

Representing information graphs as sequences Instead of directly modeling the space of heterogeneous multigraphs, $\mathcal{G}$, we build a mapping $s^\pi = f_s(G, \pi)$ from $\mathcal{G}$, to a sequence space S^π. f_s depends on a (given) ordering π of nodes and their edges in G, constructed by a graph traversal algorithm like Breadth First Search (BFS) or Depth First Search (DFS), and an internal ordering of nodes and edge types. We assume that the elements s_i^π of the resultant sequences s^π are drawn from finite sets of node representations V (defined below), node types Q, edge types R (incl. [TYPE]), and "virtual" edge types U: $\forall\ s_i^\pi \in s^\pi,\ s_i^\pi \in V \cup Q \cup R \cup U$. Virtual edge types $U = \{[\text{SOS}], [\text{EOS}], [\text{SEP}]\}$ do not represent edges in G, but serve to control the generation of the sequence, indicating the start/end of sequences and the separation of levels in the graph.

We furthermore assume that $s^\pi = s_0^\pi, ..., s_n^\pi$ that represent graphs have an **alternating** structure, where $s_0^\pi, s_2^\pi, s_4^\pi, ...$ represent nodes V, and $s_1^\pi, s_3^\pi, ...$ represent actual or virtual edges. In the case of BFS, we exploit the fact that it visits nodes level by level, *i.e.*, in the order $\mathrm{p}_i, \mathrm{c}_{i1}, ..., \mathrm{c}_{ik}, \mathrm{p}_j$ (where c_{ik} is the k-th child of parent p_i, connected by edge e_{ik}, and p_j may or may not be equal to one of the children of p_i), which we turn into a sequence,

$$s^\pi = \mathrm{p}_i, \psi(\mathrm{e}_{i1}), \mathrm{c}_{i1}, ..., \psi(\mathrm{e}_{ik}), \mathrm{c}_{ik}, [\text{SEP}], \mathrm{p}_j, ...$$

where we use the special edge type [SEP] to delineate the levels in the graph. This representation allows us to unambiguously recover the original graph, if we know which type of graph traversal is assumed (BFS or DFS).[3] Algorithm 1 (which we use to translate graphs in the training data to sequences) shows how an alternating sequence for a given graph can be constructed with BFS traversal. Figure 1 shows the alternating sequence for an information multigraph. The length $|s^\pi|$ is bounded linearly by the size of the graph $O(|s^\pi|) = O(|V| + |E|)$ (which is also the complexity of typical graph traversal algorithms like BFS/DFS).

[2] Q includes a [NULL] node type for the case when the input text does not have an information graph.

[3] In the case of DFS, [SEP] tokens appear after leaf nodes. Parents appear once for each child.

Algorithm 1 Alternating sequence construction algorithm with BFS

Input : Ordered adjacency dictionary of an information graph G, positions of nodes in the input text p_q, frequency of edge types in the training set p_r

Output : An alternating sequence y^π

Sort the nodes in G according to p_q
For each node v in G, sort the neighbors and the edges of v according to p_q and p_r respectively
Instantiate y^π as an empty list
for u *in* G **do**
  if u *is not visited* **then**
    Initialize an empty queue q
    Mark u as visited and enqueue u to q
    while q *is not empty* **do**
      Dequeue the a node w from q
      if w *in* G **then**
        Append w and all the neighbors of w with their edge types to y^π
        Append the separation edge type, [SEP], to y^π
        Mark all unvisited neighbors of w as visited and enqueue them to q
      end
    end
  end
end
Return y^π

Node and Edge Representations Our node and edge representations (explained below) rely on the observation that there are only two kinds of objects in an information graph: spans (as addresses to pieces of input texts) and types (as representations of abstract concepts). Since we can view types as special spans of length 1 grounded on the vocabulary of all types, $Q \cup R \cup U$, we only need $O(nm + |Q \cup R \cup U|)$ number of indices to unambiguously represent the spans grounded on a concatenated representation of the type vocabulary and the input text, where n is the maximum input length, m is the maximum span length, and $m \ll n$. We denote these indices as *hybrid spans* because they consist of both the spans of texts and the length-1 spans of types. These indices can be invertibly mapped back to types or text spans depending on their magnitudes (details of this mapping are explained in Section 3.2). With this joint indexing of spans and types, the task of generating an information graph is thus converted to generating an alternating sequence of *hybrid spans*.

Generating sequences We model the distribution $p(s^\pi)$ by a sequence generator h with parameters θ ($|s^\pi|$ is the length of the s^π):

$$p(s_i^\pi | s_0^\pi, ..., s_{i-1}^\pi) = h(s_0^\pi, ..., s_{i-1}^\pi, \theta),$$
$$p(s^\pi) = \prod_{i=1}^{|s^\pi|} p(s_i^\pi | s_0^\pi, ..., s_{i-1}^\pi),$$

We will address in the following sections how to enforce the sequence generator, h, to only generate sequences in the space S^π, since we do not want h to assign non-zero probabilities to arbitrary sequences that do not have a corresponding graph.

3 HySPA: Hybrid Span Generation for Alternating Sequences

In order to directly generate a target sequence that alternates between nodes that represent spans in the input and a set of node/edge types that depend on our extraction task, we first build a hybrid representation H that is a concatenation of the hidden representations from edge types, node types and the input text. This representation functions as both the context space and the output space for our decoder. Then we invertibly map both the spans of input text and the indices of the types to the *hybrid spans* grounded on the representation H. Finally, hybrid spans are generated auto-regressively through a hybrid span decoder to form the alternating sequence $y^\pi \in S^\pi$. By translating the graph extraction task to a sequence generation task, we can easily use beam-search decoding to reduce possible exposure bias (Wiseman and Rush, 2016) of the sequential decision process and thus find globally better graph representation.

High-level overview of HySPA: The HySPA model takes a piece of text (e.g. a sentence or passage), and the pre-defined node and edge types as input, and outputs an alternating sequence representation of an information graph. We enforce the generation of this sequence to be alternated by applying an alternating mask to the output probabilities. The detailed architecture is described in the following subsections.

3.1 Text and Types Encoder

Figure 2 shows the encoder architecture of our proposed model. For the set of node types, Q, and the set of edge types, R, and the virtual edge types, U,

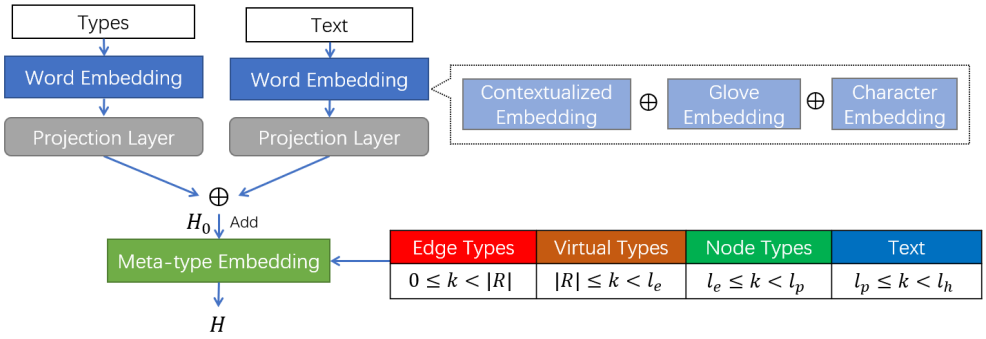

Figure 2: The encoder architecture of our model, where the $\oplus$ symbol is the concatenation operator, k is the index of the word vectors in H_0, and $l_e = |R| + |U|$. The colored table on the right indicates the assignment of the meta-types for different blocks of the concatenated word vectors from H_0.

we arrange the type list, $\mathbf{v}$ as a concatenation of the label names of the edge types, virtual edge types and node types, *i.e.*,

$$\mathbf{v} = \hat{R} \oplus \hat{U} \oplus \hat{Q}$$
$$\hat{R} = [R_1, ..., R_{|R|}]$$
$$\hat{U} = [U_1, ..., U_{|U|}]$$
$$\hat{Q} = [Q_1, ..., Q_{|Q|}]$$

where $\oplus$ means the concatenation operator between two lists, and $\hat{R}, \hat{U}, \hat{Q}$ are the lists of the type names in the sets R, U, Q, respectively (e.g. $\hat{Q}$ = ["Geopolitics", "Person", ...]). Note that the concatenation order between the lists of type names can be arbitrary as long as it is kept consistent throughout the whole model. Then, as in the embedding part of the table-sequence encoder (Wang and Lu, 2020), for each type, v_i, we embed the label tokens of the types with the contextualized word embedding from a pre-trained language model, the GloVe embedding (Pennington et al., 2014) and the character embedding,

$$E_1 = \text{ContextualizedEmbed}(\mathbf{v}), \in R^{l_p \times d_c}$$
$$E_2 = \text{GloveEmbed}(\mathbf{v}), \in R^{l_p \times d_g}$$
$$E_3 = \text{CharacterEmbed}(\mathbf{v}), \in R^{l_p \times d_k}$$
$$E_4 = E_1 \oplus E_2 \oplus E_3 \in R^{l_p \times d_e},$$
$$E_v = E_4 W_0^T \in R^{l_p \times d_m},$$

where $l_p = |R| + |U| + |Q|$ is the number of all kinds of types, $W_0 \in R^{d_e \times d_m}$ is the weight matrix of the linear projection layer, $d_e = d_c + d_g + d_k$ is the total embedding dimension and d_m is the hidden size of our model. After we obtain the contextualized embedding of the tokens of each type $v_i \in \mathbf{v}$, we take the average of these token vectors as the representation of v_i and freeze its update during training. More details of the embedding pipeline can be found in Appendix A.

This embedding pipeline is also used to embed the words in the input text, $\mathbf{x}$. Unlike the pipeline for the type embedding, we represent the word as the contextualized embedding of its first sub-token from the pre-trained Language Model (LM, e.g. BERT (Devlin et al., 2018)), and finetune the LM in an end-to-end fashion.

After obtaining the type embedding E_v, and the text embedding E_x respectively, we concatenate them along the sequence length dimension to form the hybrid representation H_0. Since H_0 is a concatenation of word vectors from four different types of tokens, *i.e.*, edge types, virtual edge types, node types and text, a meta-type embedding is applied to indicate this type difference between the blocks of vectors from the representation H_0, as shown in Figure 2. The final context representation H is obtained by element-wise addition of the meta-type embedding and H_0,

$$H_0 = E_v \oplus E_x \in R^{l_h \times d_m},$$
$$H_s = \text{MetaTypeEmbed}(H_0) \in R^{l_h \times d_m},$$
$$H = H_0 + H_s \in R^{l_h \times d_m},$$

where $l_h = l_p + |x|$ is the height of our hybrid representation matrix H.

3.2 Invertible Mapping between Spans & Types and Hybrid Spans

Given a span in the text, $t = (t_s, t_e) \in \mathbb{N}^2, t_s < t_e$, we convert the span t to an index k, $k \ge l_p$, in the representation H via the mapping g_k,

$$k = g_k(t_s, t_e) = t_s m + t_e - t_s - 1 + l_p \in \mathbb{N},$$

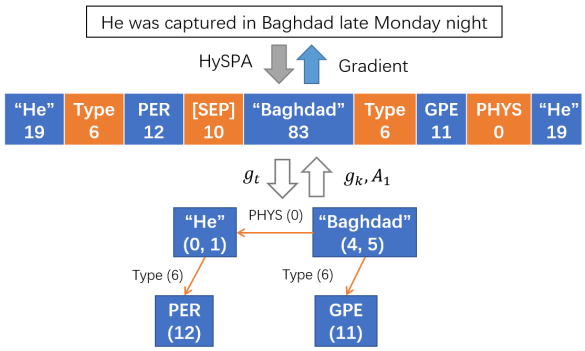

Figure 3: An example of the alternating sequence representation (in the middle) of a knowledge graph (at bottom) from the ACE05 training set, where A_1 means the Algorithm 1. We take $m = 16$ and $l_p = 19$ for this example. "19" in the alternating sequence is the index for the span (0,1) of "He", "83" is the index for the span (4,5) of "Baghdad", and "10" is the index of the virtual edge type, [SEP]. The input text (on top) for this graph is "He was captured in Baghdad late Monday night".

where m is the maximum length of spans, and $l_p = |R| + |U| + |Q|$. We keep the type indices in the graph unchanged because they are smaller than l_p and $k \geq l_p$. Since, for an information graph, the maximum span length, m, of a mention is often far smaller than the length of the text, *i.e.*, $m \ll n$, we can then reduce the bound of the maximum magnitude of k from $O(n^2)$ to $O(nm)$ by only considering spans of length smaller than m, and thus maintain linear space complexity for our decoder with respect to the length of the input text, n. Figure 3 shows a concrete example of our alternating sequence for a knowledge graph in the ACE05 dataset.

Since t_s, t_e, k are all natural numbers, we can construct an inverse mapping g_t that converts the index k in H back to $t = (t_s, t_e)$,

$$t_s = g_{t_s}(k) = -\max(0, -k + l_p) + \lfloor \max(0, k - l_p)/m \rfloor + l_p,$$
$$t_e = g_{t_e}(k) = g_{t_s}(k) + \max(0, k - l_p) \bmod m,$$

where $\lfloor \cdot \rfloor$ is the integer floor function and $\bmod$ is the modulus operator. Note that $g_t(k)$ can be directly applied to the indices from the types segment of H and remain their values unchanged, *i.e.*,

$$g_t(k) = (k, k), \forall k < l_p, k \in \mathbb{N}.$$

With this property, we can easily incorporate the mapping g_t into our decoder to map the alternating sequence y^π back to the spans in the hybrid representation H.

3.3 Hybrid Span Decoder

Figure 4 shows the general model architecture of our hybrid span decoder. Our decoder takes the context representation H as input, and recurrently decodes the alternating sequence y^π given a start-of-sequence token.

Hybrid Span Encoding via Attention Given the alternating sequence y^π, and the mapping g_t (section 3.2), our decoder first maps each index in y^π to a span, $(t_{s_i}, t_{e_i}) = g_t(y_i^\pi)$, grounded on the representation H and then converts the span to an attention mask, M_0, to allow the model to learn to represent a span as a weighted sum of a segment of the contextualized word representations referred by the span,

$$Q = W_1^T H_{[CLS]} + \mathbf{b}_1 \quad \in R^{|y^\pi| \times d_m},$$
$$K = W_2^T H + \mathbf{b}_2 \quad \in R^{l_h \times d_m},$$
$$H_y = \text{softmax}\left(\frac{QK^T}{\sqrt{d_m}} + M_0\right) H \quad \in R^{|y^\pi| \times d_m},$$
$$M_0(i,j) = \begin{cases} 0, & t_{s_i} \leq j \leq t_{e_i} \\ -\infty, & \text{otherwise} \end{cases}$$

where $H_{[CLS]} \in R^{|y^\pi| \times d_m}$ is the $|y^\pi|$-times repeated hidden representation of the start of the sequence token, [CLS], from the text segment of H, and H_y is our final representation of the hybrid spans in y^π. $W_1, W_2, \mathbf{b}_1, \mathbf{b}_2$ are learnable parameters, and t_{s_i}, t_{e_i} are the start and the end position of the span thatwe are encoding. Note that for the type spans whose length is 1, the result of the *softmax* calculation will always be 1, which leads to its span representation to be exactly its embedding vector as we desired.

Traversal Embedding In order to distinguish the hybrid spans at different position in y^π, a naive way is to add a sinusoidal position embedding (Vaswani et al., 2017) to H_y. However, this approach treats the alternating sequence as an ordinary sequence and ignores the underlying graph structure it encodes. To alleviate this issue, we propose a novel traversal embedding approach which captures the traversal level information, the parent-child information and the intra-level connection information as a substitution of the naive position embedding. Our traversal embedding can either encode the BFS or DFS traversal pattern. As an example, we assume BFS traversal here and leave the details of DFS traversal embedding in Appendix D.

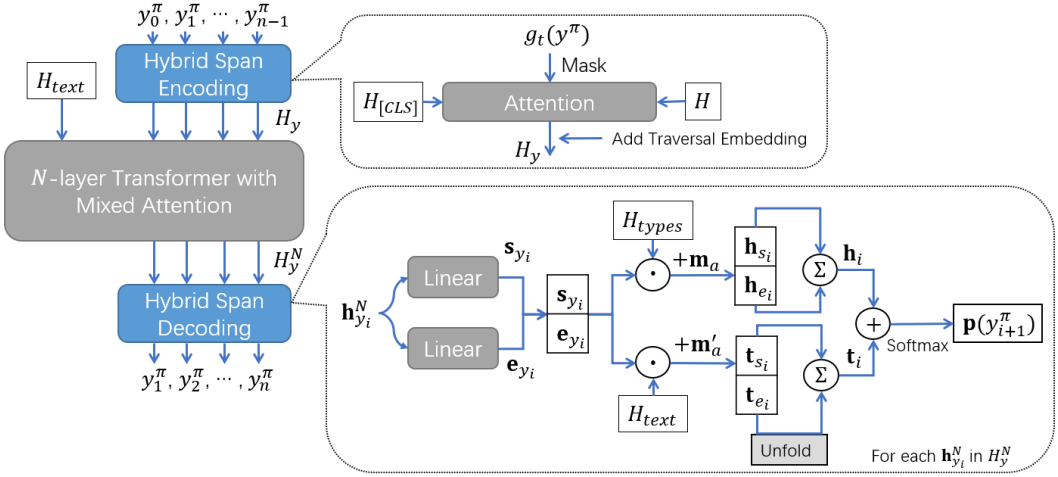

Figure 4: The architecture of our hybrid span decoder. N is the number of the decoder layers. $\oplus$ before the softmax function means the concatenation operator. H_y^N is the hidden representation of the sequence y^π from the last decoder layer. Our hybrid span decoder can be understood as an auto-regressive model that operates in a closed context space and output space defined by H.

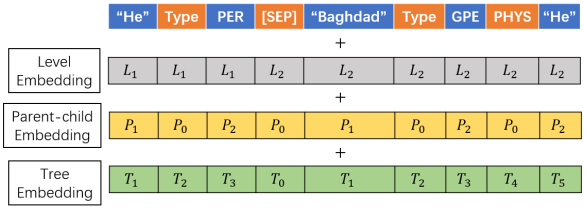

Figure 5: An example of BFS traversal embedding for an alternating sequence, ["He", Type, PER, [SEP], "Baghdad", Type, GPE, PHYS, "He"]. Our traversal embedding is the sum of the level embedding, the parent-child embedding and the tree embedding.

Our BFS traversal embedding is a pointwise sum of the level embedding, L, the parent-child embedding, P, and the tree embedding, T of a given alternating sequence, y,

$$\text{TravEmbed}(y) = L(y) + P(y) + T(y) \in R^{|y| \times d_m}$$

where the level embedding assigns the same embedding vector L_i for each position at the BFS traversal level i, and the value of the embedding vector is filled according to the non-parametric sinusoidal position embedding since we want our embedding to extrapolate to the sequence that is longer than any sequences in the training set. The parent-child embedding assigns different random initialized embedding vectors at the positions of the parent nodes and the child nodes in the BFS traversal levels to help model distinguish between these two kinds of nodes. For encoding the intra-level connection information, our insight is that the connection between each nodes in a BFS level can be viewed as a depth-3 tree, where the first depth takes the parent node, the second depth is filled with the edge types and the third depth consists of the corresponding child nodes for each of the edge types. Our tree embedding is then formed by encoding the position information of the depth-3 tree with a tree positional embedding (Shiv and Quirk, 2019) for each BFS level. Figure 5 shows a concrete example of how these embeddings function for a given alternating sequence. The obtained traversal embedding is then pointwisely added to the hidden representation of the alternating sequence H_y for injecting the traversal information of the graph structure.

Inner blocks With the input text representation H_{text} sliced from the hybrid representation H and the target sequence representation H_y, we apply an N-layer transformer structure with mixed-attention (He et al., 2018) to allow our model to utilize features from different attention layers when decoding the edges or the nodes of an alternating sequence. Note that our hybrid span decoder is perpendicular to the actual choice of the neural structures of the inner blocks, and we choose the design of mixed-attention transformer (He et al., 2018) because its layerwise coordination property is empirically more suitable for our heterogeneous decoding of two different kinds of sequence elements. The detailed structure of the inner blocks is explained in Appendix E.

Hybrid span decoding For the hybrid span decoding module, we first slice off the hidden rep-

resentation of the alternating sequence y^π from the output of the N-layer inner blocks and denote it as H_y^N. Then for each hidden representation $\mathbf{h}_{y_i}^N \in H_y^N, 0 \leq i < |y^\pi|$, we apply two different linear layers to obtain the start position representation, $\mathbf{s}_{y_i}$, and the end position representation, $\mathbf{e}_{y_i}$,

$$\begin{aligned}\mathbf{s}_{y_i} &= W_5^T \mathbf{h}_{y_i} + \mathbf{b}_5 \quad \in R^{d_m}, \\ \mathbf{e}_{y_i} &= W_6^T \mathbf{h}_{y_i} + \mathbf{b}_6 \quad \in R^{d_m},\end{aligned}$$

where $W_5, W_6 \in R^{d_m \times d_m}$ and $\mathbf{b}_5, \mathbf{b}_6 \in R^{d_m}$ are learnable parameters. Then we calculate the scores of the target spans separately for the types segment and the text segment of H, and concatenate them together before the final softmax operator for a joint estimation of the probabilities of text spans and type spans,

$$\begin{aligned}\mathbf{h}_{s_i} &= H_{\text{types}}\, \mathbf{s}_{y_i} + \mathbf{m}_a && \in R^{l_p}, \\ \mathbf{h}_{e_i} &= H_{\text{types}}\, \mathbf{e}_{y_i} + \mathbf{m}_a && \in R^{l_p}, \\ \mathbf{h}_i &= \mathbf{h}_{s_i} + \mathbf{h}_{e_i} && \in R^{l_p}, \\ \mathbf{t}_{s_i} &= H_{\text{text}}\, \mathbf{s}_{y_i} + \mathbf{m}'_a && \in R^{n}, \\ \mathbf{t}_{e_i} &= H_{\text{text}}\, \mathbf{e}_{y_i} + \mathbf{m}'_a && \in R^{n}, \\ \mathbf{t}_i &= \text{unfold}(\mathbf{t}_{e_i}, m) + \mathbf{t}_{s_i} && \in R^{nm}, \\ \mathbf{p}(y_{i+1}^\pi) &= \text{softmax}(\mathbf{h}_i \oplus \mathbf{t}_i) && \in R^{nm+l_p},\end{aligned}$$

where $\mathbf{h}_i$ is the score vector of possible spans in the type segment of H, and $\mathbf{t}_i$ is the score vector of possible spans in the text segment of H. Since the type spans always have a span length 1, we only need an element-wise addition between the start position scores, $\mathbf{h}_{s_i}$ and the end position scores $\mathbf{h}_{e_i}$ to calculate $\mathbf{h}_i$. The entries of $\mathbf{t}_i$ contain the scores for the text spans, $t_{s_i,j} + t_{e_i,k}, \forall j \leq k, k - j < m$, which are calculated with the help of an *unfold* function which converts the vector $\mathbf{t}_{e_i} \in R^n$ to a stack of n sliding windows of size m, the maximum span length, with stride 1. The alternating masks $\mathbf{m}_a \in R^{l_p}, \mathbf{m}'_a \in R^n$ are defined as:

$$\mathbf{m}_a(j) = \begin{cases} 0, & y_i^\pi > l_e \cap j < l_e \\ -\infty, & \text{otherwise} \end{cases}$$

$$\mathbf{m}'_a(j) = \begin{cases} -\infty, & y_i^\pi > l_e \\ 0, & \text{otherwise} \end{cases}$$

where $l_e = |R| + |U|$ is the total number of edge types. In this way, while we have a joint model of nodes and edge types, the output distribution is enforced by the alternating masks to produce an alternating decoding of nodes and edge types, and this is the main reason why we call this decoder a hybrid span decoder.

4 Experiments

4.1 Experimental Setting

We test our model on the ACE 2005 dataset distributed by LDC[4], which includes 14.5k sentences, 38.3k entities (with 7 types), and 7.1k relations (with 6 types), derived from the general news domain. More details can be found in Appendix C.

Following previous work, we use F1 as an evaluation metric for both NER and RE. For the NER task, a prediction is marked correct when both the type and the boundary span match those of the gold entity. For the RE task, a prediction is correct when both the relation type and the boundaries of the two entities are correct.

4.2 Implementation Details

When training our model, we apply the cross-entropy loss with a label smoothing factor of 0.1. The model is trained with 2048 tokens per batch (roughly a batch size of 28) for 25000 steps using an AdamW optimizer (Loshchilov and Hutter, 2018) with a learning rate of $2e^{-4}$, a weight decay of 0.01, and an inverse square root scheduler with 2000 warm-up steps. Following the TabSeq model (Wang and Lu, 2020), we use RoBERTa-large (Liu et al.) or ALBERT-xxlarge-v1 (Lan et al., 2020) for the pretrained language model and slow its learning rate by a factor of 0.1 during training. A hidden state dropout rate of 0.2 is applied to RoBERTa-large while the rate of 0.1 for ALBERT-xxlarge-v1. A dropout rate of 0.1 is also applied to our hybrid span decoder during training. We set the maximum span length, $m = 16$, the hidden size of our model, $d_m = 256$, and the number of the decoder blocks, $N = 12$. Even though theoretically the beam-search should help us reduce the exposure bias, we do not observe any performance gain during grid search of the beam size and the length penalty on the validation set (detailed grid search setting is in Appendix A). Thus we set a vanilla beam size of 1 and the length penalty of 1, and leave this theory-experiment contradiction for future research. Our model is built with the FAIRSEQ toolkit (Ott et al., 2019) for efficient distributed training and all the experiments are conducted on two NVIDIA TITAN X GPUs.

[4] https://catalog.ldc.upenn.edu/LDC2006T06

IE Models		Space Complexity	Time Complexity	NER	RE
PointerNet (Katiyar and Cardie, 2017)		$O(n)$	$O(n^2)$	82.6	55.9
SpanRE (Dixit and Al-Onaizan, 2019)		$O(n)$	$O(n^2)$	86.0	62.8
Dygie++ (Wadden et al., 2019)		$O(n)$	$O(n^2)$	88.6	63.4
OneIE (Lin et al., 2020)		$O(n)$	$O(n^2)$	88.8	67.5
TabSeq (Wang and Lu, 2020)		$O(n^2)$	$O(n)$	89.5	67.6
HySPA (ours)	w/ RoBERTa	$O(n)$	$O(n)$	88.9	**68.2**
	w/ ALBERT			**89.9**	**68.0**

Table 1: Joint NER and RE F1 scores of the IE models on the ACE05 test set. Complexities are calculated for the entity and relation decoding part of the models (n is the length of the input text). The performance of the TabSeq model reported here is based on the same ALBERT-xxlarge (Lan et al., 2020) pretrained language model as ours.

Model	NER F1	RE F1
HySPA w/ RoBERTa	88.9	68.2
– Traversal-embedding	88.9	66.7
– Masking	88.1	64.8
– BFS	88.7	66.2
– Mixed-attention	88.6	64.7
– Span-attention	88.5	66.1

Table 2: Ablation study on the ACE05 test set. "– Traversal-embedding": we remove the traversal embedding and instead use sinusoidal position embedding, and the following ablations are based on the model after this ablation. "– Masking": we remove the alternating mask from the hybrid span decoder. "– BFS": we use DFS instead of BFS as traversal. "– Mixed-attention": we remove the mixed-attention layer and use a standard transformer encoder decoder structure. "– Span-attention": we remove the span attention in the span encoding module and instead average the words in the span.

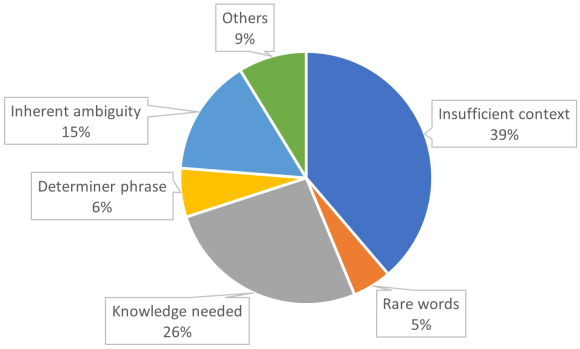

Figure 6: Distribution of remaining errors on the ACE05 test set.

4.3 Results

Table 1 compares our model with the previous state-of-the-art results on the ACE05 test set. Compared with the previous SOTA, TabSeq (Wang and Lu, 2020) with ALBERT pretrained language model, our model with ALBERT has significantly better performance for both NER score and RE score, while maintaining a linear space complexity which is an order smaller than TabSeq. Our model is the first joint model that has both linear space and time complexities compared with all previous joint IE models, and thus has the best scalability for large-scale real world applications.

4.4 Ablation Study

To prove the effectiveness of our approach, we conduct ablation experiments on the ACE05 dataset. As shown in Table 2, after we remove the traversal embedding the RE F1 scores drop significantly, which indicates that our traversal embedding can help encode the graph structure and improve relation predictions. Also if the alternating masking is dropped, the NER F1 and RE F1 scores both drop significantly, which proves the importance of enforcing the alternating pattern. We can observe that the mixed-attention layer contributes significantly for relation extraction. This is because the layer-wise coordination can help the decoder to disentangle the source features and utilize different layer features between the entity and the relation prediction. We can also observe that the DFS traversal has worse performance than BFS. We suspect that this is because the resultant alternating sequence from DFS is often longer than the one from BFS due to the nature of the knowledge graphs, and thus increases the learning difficulty.

4.5 Error Analysis

After analyzing 80 remaining errors, we categorize and discuss common cases below (Figure 6 plots the distribution of error types). These may require additional features and strategies to address.

Insufficient context. In many examples, the answer entity is a pronoun that cannot be accurately typed given the limited context: in *"We notice they said they did not want to use the word destroyed, in fact, they said let others do that"*, it's difficult to correctly classify *We* as an organization. This could be mitigated by using entire documents as input, leveraging cross-sentence context.

Rare words. The rare word issue is when the word in test set rarely appeared in the training set and often not termed in the dictionary. In the sentence *"There are also Marine FA-18s and Marine Heriers at this base"*, the term *Heriers* (a vehicle incorrectly classified as person by the model) neither appeared in the training set, nor understood well by pre-trained language model; the model, in this case, can only rely on subword-level representation.

Background knowledge required Often the sentence mentions entities that are difficult to infer from the context, but are easily identified by consulting a knowledge base: in *"but critics say Airbus should have sounded a stronger alarm after a similar incident occurred in 1997"*, our model incorrectly predicts the *Airbus* to be a vehicle while the *Airbus* here refers to the European aerospace corporation. Our system also separated *United Nations Security Council* into two entities *United Nations* and *Security Council*, generating a non-existing relation triple (*Security Council* part-of *United Nations*). Such mistakes could be avoided by consulting a knowledge base such as DBpedia (Bizer et al., 2009) or by performing entity linking.

Inherent ambiguity Many examples have inherent ambiguity, e.g. *European Union* can be typed as organization or political entity, while some entities (e.g., military bases) can be both locations and organizations, or facilities.

5 Related Work

NER is often done jointly with RE in order to mitigate error propagation and learn inter-relation between tasks. One line of approaches is to treat the joint task as a squared table filling problem (Miwa and Sasaki, 2014; Gupta et al., 2016; Wang and Lu, 2020), where the i-th column or row represents the i-th token. The table has diagonals indicating sequential tags for entities and other entries as relations between pairs of tokens. Another line of work is by performing RE after NER. In the work by Miwa and Bansal (2016), the authors used BiLSTM (Graves et al., 2013) for NER and consequently a Tree-LSTM (Tai et al., 2015) based on dependency graph for RE. Wadden et al. (2019) and Luan et al. (2019), on the other hand, takes the approach of constructing dynamic text span graphs to detect entities and relations. Extending on Wadden et al. (2019), Lin et al. (2020) introduced OneIE, which further incorporates global features based on cross subtask and instance constraints, aiming to extract IE results as a graph. Note that our model differs from OneIE (Lin et al., 2020) in that our model captures global relationships automatically through autoregressive generation while OneIE uses feature engineered templates; Moreover, OneIE needs to do pairwise classification for relation extraction, while our method efficiently generates existing relations and entities.

While several Seq2Seq-based models (Zhang et al., 2020; Zeng et al., 2018, 2020; Wei et al., 2019; Zhang et al., 2019) have been proposed to generate triples (i.e., node-edge-node), our model is fundamentally different from them in that: (1) it is generating a BFS/DFS traversal of the target graph, which captures dependencies between nodes and edges and has a shorter target sequence, (2) we model the nodes as the spans in the text, which is independent of the vocabulary, so even if the tokens of the nodes are rare or unseen words, we can still generate spans on them based on the context information.

6 Conclusion

In this work, we propose the Hybrid Span Generation (HySPA) model, the first end-to-end text-to-graph extraction model that has a linear space and time complexity at the graph decoding stage. Besides its scalability, the model also achieves state-of-the-art performance on the ACE05 joint entity and relation extraction task. Given the flexibility of the structure of our hybrid span generator, abundant future research directions remain, e.g. incorporating the external knowledge for hybrid span generation, applying more efficient sparse self-attention, and developing better search methods to find more globally plausible graphs represented by the alternating sequence.

Acknowledgments

This work is supported by Agriculture and Food Research Initiative (AFRI) grant no. 2020-67021-32799/project accession no.1024178 from the USDA National Institute of Food and Agriculture.

References

Jimmy Ba, J. Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *ArXiv*, abs/1607.06450.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009. Dbpedia-a crystallization point for the web of data. *Journal of web semantics*, 7(3):154–165.

Yee Seng Chan and Dan Roth. 2011. Exploiting syntactico-semantic structures for relation extraction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 551–560, Portland, Oregon, USA. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Kalpit Dixit and Yaser Al-Onaizan. 2019. Span-level model for relation extraction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5308–5314, Florence, Italy. Association for Computational Linguistics.

Markus Eberts and Adrian Ulges. 2019. Span-based joint entity and relation extraction with transformer pre-training. *arXiv preprint arXiv:1909.07755*.

Radu Florian, Hongyan Jing, Nanda Kambhatla, and Imed Zitouni. 2006. Factorizing complex models: A case study in mention detection. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 473–480, Sydney, Australia. Association for Computational Linguistics.

Radu Florian, John Pitrelli, Salim Roukos, and Imed Zitouni. 2010. Improving mention detection robustness to noisy input. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 335–345, Cambridge, MA. Association for Computational Linguistics.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2007. Multi-dimensional recurrent neural networks. In *International conference on artificial neural networks*, pages 549–558. Springer.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee.

Pankaj Gupta, Hinrich Schütze, and Bernt Andrassy. 2016. Table filling multi-task recurrent neural network for joint entity and relation extraction. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2537–2547.

Tianyu He, Xu Tan, Yingce Xia, Di He, Tao Qin, Zhibo Chen, and Tie-Yan Liu. 2018. Layer-wise coordination between encoder and decoder for neural machine translation. In *Advances in Neural Information Processing Systems*, volume 31, pages 7944–7954. Curran Associates, Inc.

Jing Jiang and ChengXiang Zhai. 2007. A systematic exploration of the feature space for relation extraction. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 113–120, Rochester, New York. Association for Computational Linguistics.

Arzoo Katiyar and Claire Cardie. 2017. Going out on a limb: Joint extraction of entity mentions and relations without dependency trees. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 917–928, Vancouver, Canada. Association for Computational Linguistics.

Keshav Kolluru, Samarth Aggarwal, Vipul Rathore, Soumen Chakrabarti, et al. 2020. Imojie: Iterative memory-based joint open information extraction. *arXiv preprint arXiv:2005.08178*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Qi Li and Heng Ji. 2014. Incremental joint extraction of entity mentions and relations. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 402–412, Baltimore, Maryland. Association for Computational Linguistics.

Qi Li, Heng Ji, Yu Hong, and Sujian Li. 2014. Constructing information networks using one single model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1846–1851, Doha, Qatar. Association for Computational Linguistics.

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, pages 7999–8009, Online. Association for Computational Linguistics.

Y Liu, M Ott, N Goyal, J Du, M Joshi, D Chen, O Levy, M Lewis, L Zettlemoyer, and V Stoyanov. Roberta: A robustly optimized bert pretraining approach. arxiv 2019. *arXiv preprint arXiv:1907.11692*.

Yue Liu, Tongtao Zhang, Zhicheng Liang, Heng Ji, and Deborah L McGuinness. 2018. Seq2rdf: An end-to-end application for deriving triples from natural language text. *arXiv preprint arXiv:1807.01763*.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Yi Luan, Dave Wadden, Luheng He, Amy Shah, Mari Ostendorf, and Hannaneh Hajishirzi. 2019. A general framework for information extraction using dynamic span graphs. *arXiv preprint arXiv:1904.03296*.

Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using lstms on sequences and tree structures. *arXiv preprint arXiv:1601.00770*.

Makoto Miwa and Yutaka Sasaki. 2014. Modeling joint entity and relation extraction with table representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1858–1869.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Jeffrey Pennington, Richard Socher, and Christopher 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, Doha, Qatar. Association for Computational Linguistics.

C. Shi, Y. Li, J. Zhang, Y. Sun, and P. S. Yu. 2017. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37.

Vighnesh Shiv and Chris Quirk. 2019. Novel positional encodings to enable tree-based transformers. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Ang Sun, Ralph Grishman, and Satoshi Sekine. 2011. Semi-supervised relation extraction with large-scale word clustering. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 521–529, Portland, Oregon, USA. Association for Computational Linguistics.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

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

David Wadden, Ulme Wennberg, Yi Luan, and Hannaneh Hajishirzi. 2019. Entity, relation, and event extraction with contextualized span representations. *arXiv preprint arXiv:1909.03546*.

Jue Wang and Wei Lu. 2020. Two are better than one: Joint entity and relation extraction with table-sequence encoders. *arXiv preprint arXiv:2010.03851*.

Zhepei Wei, Jianlin Su, Yue Wang, Yuan Tian, and Yi Chang. 2019. A novel cascade binary tagging framework for relational triple extraction. *arXiv preprint arXiv:1909.03227*.

Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306.

Daojian Zeng, Haoran Zhang, and Qianying Liu. 2020. Copymtl: Copy mechanism for joint extraction of entities and relations with multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9507–9514.

Xiangrong Zeng, Daojian Zeng, Shizhu He, Kang Liu, and Jun Zhao. 2018. Extracting relational facts by an end-to-end neural model with copy mechanism. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 506–514.

Haoran Zhang, Qianying Liu, Aysa Xuemo Fan, Heng Ji, Daojian Zeng, Fei Cheng, Daisuke Kawahara, and Sadao Kurohashi. 2020. Minimize exposure bias of seq2seq models in joint entity and relation extraction. *arXiv preprint arXiv:2009.07503*.

Meishan Zhang, Yue Zhang, and Guohong Fu. 2017. End-to-end neural relation extraction with global optimization. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1730–1740, Copenhagen, Denmark. Association for Computational Linguistics.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. Broad-coverage semantic parsing as transduction. 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 3786–3798, Hong Kong, China. Association for Computational Linguistics.

A Hyperparameters

We use 100-dimensional GloVe word embeddings trained on 6B tokens as intialization [5], and freeze its update during training. The character embedding has 30-dimension with LSTM encoding [6] and the Glove Embeddings for the out of vocabulary tokens are replaced with randomly initialized vectors following Wang and Lu (2020). We use gradient clipping of 0.25 during training. The number of heads for our mixed attention is set to 8. The beam size and length penalty is decided by a grid-search on the validation set of the ACE05 dataset, and the range for the beam size is from 1 to 7 with a step size of 1 and the length penalty is from 0.7 to 1.2 with a step size of 0.1. We choose the best beam size and length penalty based on the metric of relation extraction F1 score.

B Training Details

Our model has 236 million parameters with the ALBERT-xxlarge pretrained language model. On average, our best performing model with ALBERT-xxlarge can be trained distributedly on two NVIDIA TITAN X GPUs for 20 hours.

C Data

The Automatic Content Extraction (ACE) 2005 [7] dataset contains English, Arabic and Chinese training data for the 2005 Automatic Content Extraction (ACE) technology evaluation, providing entity, relation, and event annotations. We follow Wadden et al. (2019) [8] for preprocessing and data splits. The preprocessed data contains 7.1k relations, 38k entities, and 14.5k sentences. The split contains 10051 samples for training, 2424 samples for development, and 2050 for testing.

D DFS Traversal Embedding

Since the parent-child information is already contained in the intra-level connections of DFS traversal, we only have the sum of the level embedding and the connection embedding for DFS traversal

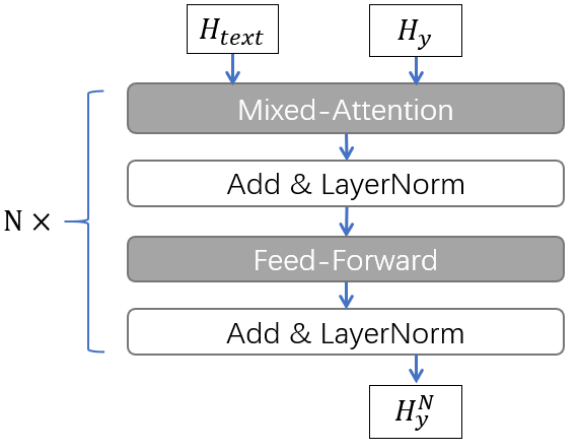

Figure 7: The general model architecture of the mixed-attention transformer.

embedding. Similar to BFS embedding, the DFS level embedding assigns the same embedding vector L_i for each position at the DFS traversal level i, but the value of the embedding vector is randomly initialized instead of filled with the non-parametric sinusoidal position embedding, since the proximity information does not exist between the traversal levels of DFS. However, we do have clear distance information for the elements in a DFS level, *i,e.*, for a DFS level D = [A, B, C, ..., [sep]], the distance from A to the elements [A, B, C, ..., [sep]] is $[0, 1, 2, 3, ..., |D| - 1]$. We encode this distance information with the sinusoidal position embedding which becomes our connection embedding that captures the intra-level connection information.

E Transformer with Mixed-attention

We first slice off the hidden representation of the input text from the hybrid representation H, and denote it as H_{text}, then the input text representation H_{text} and the output from the Hybrid Span Encoding H_y gets fed into a stack of N mixed-attention/feedforward blocks that have the following structure (as shown in Figure 7):

Since generating the node and edge types may need features from different layers, we use mixed attention (He et al., 2018), which allows our model to utilize the features from different attention layers when encoding the text segment, H_{text}, and the

[5] https://nlp.stanford.edu/projects/glove/

[6] https://github.com/LorrinWWW/two-are-better-than-one/blob/master/layers/encodings/embeddings.py

[7] https://www.ldc.upenn.edu/collaborations/past-projects/ace

[8] https://github.com/dwadden/dygiepp/tree/master/scripts/data/ace05/preprocess

target features, H_y,

$$\text{MixedAtt}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_m}} + M_1\right)V \in R^{l_m \times d_m},$$

$$M_1(i, j) = \begin{cases} 0, & j < n \cup j \leq i + n \\ -\infty, & \text{otherwise} \end{cases}$$

where $n = |x|$ is the length of the input text, $l_m = |x| + |y^\pi|$ is the total length of the source and the target features. Denoting the concatenation of the source features, H_{text}, and the target features, H_y, as H_0, a source/target embedding (He et al., 2018) is also added to H_0 before the first layer of the mixed attention to allow the model to distinguish the features from the source and the target sequences. The mixed-attention layer is combined with a feed-forward layer to form a decoder block:

$$\begin{aligned} \text{FFN}(x) &= \max(0, xW_3 + b_3)W_4 + b_4, \\ Q &= W_q^T H_0 + \mathbf{b}_q, \\ K &= W_k^T H_0 + \mathbf{b}_k, \\ V &= W_v^T H_0 + \mathbf{b}_v, \\ H_0' &= \text{LayerNorm}(\text{MixedAtt}(Q, K, V) + H_0), \\ H_1 &= \text{LayerNorm}(\text{FFN}(H_0') + H_0'), \end{aligned}$$

where $W_{q,k,v}, \mathbf{b}_{q,k,v}, W_3 \in R^{d_m \times 4d_m}, W_4 \in R^{4d_m \times d_m}, \mathbf{b}_3, \mathbf{b}_4$ are the learnable parameters, and LayerNorm is the Layer Normalization layer (Ba et al., 2016). The decoder block is stacked N times to obtain the final hidden representation H_N, and output the final representation of the target sequence, H_y^N. The mixed-attention has a time complexity of $O(n^2)$ when encoding the source features, but we can cache the hidden representation of this part when generating the target tokens due to the causal masking of the target features, and thus maintain a time complexity of $O(n)$ for each decoding step.