Syntactically Rich Discriminative Training: An Effective Method for Open Information Extraction

Frank Mtumbuka[3,1] **Thomas Lukasiewicz**[2,1]
[1] Department of Computer Science, University of Oxford, UK
[2] Institute of Logic and Computation, TU Wien, Austria
[3] Cardiff University, Cardiff, UK
firstname.lastname@cs.ox.ac.uk

Abstract

Open information extraction (OIE) is the task of extracting facts "(Subject, Relation, Object)" from natural language text. We propose several new methods for training neural OIE models in this paper. First, we propose a novel method for computing syntactically rich text embeddings using the structure of dependency trees. Second, we propose a new discriminative training approach to OIE in which tokens in the generated fact are classified as "real" or "fake", i.e., those tokens that are in both the generated and gold tuples, and those that are only in the generated tuple but not in the gold tuple. We also address the issue of repetitive tokens in generated facts and improve the models' ability to generate implicit facts. Our approach reduces repetitive tokens by a factor of 23%. Finally, we present paraphrased versions of the CaRB, OIE2016, and LSOIE datasets, and show that the models' performance substantially improves when trained on datasets augmented by such data. Our best model beats the SOTA of IMoJIE on the recent CaRB dataset, with an improvement of 39.63% in F_1 score.

1 Introduction

OIE (Banko et al., 2007) is a branch of information extraction (IE) that focuses on extracting structured information (Niklaus et al., 2018) from unstructured natural language text. This structured information is a set of tuples of the form "(Subject, Relation, Object)", also called facts in OIE. For instance, given the sentence "Machine learning is a subfield of AI.", the tuple ⟨*machine learning*, *is a subfield of*, *AI*⟩ can be extracted, where the relation phrase "is a subfield of" indicates the semantic relationship between the subject "Machine learning" and the object "AI". OIE is useful in many downstream natural language processing (NLP) tasks like natural language understanding (Mausam, 2016), multi-document question answering and summarization (Fan et al., 2019), and knowledge base construction from text (Soderland et al., 2010).

Several neural OIE methods in the literature approach OIE as either a sequence labeling (Stanovsky et al., 2018; Roy et al., 2019; Jiang et al., 2019; Zhan and Zhao, 2019; Hohenecker et al., 2020) or a sequence generation problem (Cui et al., 2018; Sun et al., 2018; Kolluru et al., 2020). Sequence labeling approaches label each token in the input text as either belonging to the subject, relation, or object, while sequence generation approaches generate facts one word at a time given the input text.

For the task of OIE, it is a common practice to make use of part-of-speech (PoS) and dependency tags in addition to the actual input text as a way of incorporating syntactic information (Stanovsky et al., 2018; Zhan and Zhao, 2019; Jia and Xiang, 2019; Sun et al., 2018; Hohenecker et al., 2020). In such work where these tags are used, the embeddings for the tags are just concatenated to the embeddings of the corresponding text tokens. This formulation does not fully use the rich syntactic information, especially the one that is expressed in the structure of dependency trees. Dependency trees' head-dependent relations provide a good approximation of the semantic relationships between predicates and their arguments (Jurafsky and Martin, 2009). As a result, they are directly applicable to a wide range of applications, including IE. In this work, we compute token representations based on the structure of dependency trees in order to benefit from their rich syntactic and semantic information. Furthermore, sequence generation approaches are susceptible to generating facts that often express redundant information and are also prone to generating repetitive text in facts. Sun et al. (2018) and Kolluru et al. (2020) looked into the issue of generating the same facts more than one time, which is redundant. No work has been done to control the generation of repetitive text in facts.

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 5972 - 5987
December 7-11, 2022 ©2022 Association for Computational Linguistics

Additionally, sequence generation approaches are capable of introducing words that are not in the input sentence into a generated fact. This capability enables such approaches to generate facts that are implicitly stated in the input sentence. However, the approaches employed by Cui et al. (2018) and Sun et al. (2018) are restrictive in how the models "pick" words when generating facts, which restrains the models' flexibility in generating implicit facts.

In this paper, our approach uses a sequence generation approach to generate facts from natural language text one word at a time. Unlike the previous approaches, we compute syntactically rich vector representations of input text tokens guided by the structure of dependency trees. For each input sentence, we construct a visibility matrix of its tokens based on the structure of its dependency tree by considering tokens that are directly related in the dependency tree as visible to each other. A graph neural network (GNN) (Zhou et al., 2018) encoder takes as input token embeddings and the corresponding visibility matrix to compute new token embeddings guided by the visibility matrix. Furthermore, we also introduce a new method for training neural OIE models. We add an extra module, the discriminator, that takes the generated tuple as input and classifies its tokens as either "real" or "fake", where "real" tokens are those that are in both the generated and gold tuples, while "fake" tokens are those that are only in the generated tuple but not in the gold tuple. To ensure that such a model does not generate repetitive text, we use a coverage vector to monitor the degree of coverage that words in the input text have received so far. When generating the next word at timestep t, the coverage vector at that instant is a sum of all attention distributions computed over the input text tokens from timesteps 0 to $t-1$. This coverage vector is used as part of the input when computing the next attention distribution. Intuitively, this informs the current attention mechanism's decision of the previous decisions and makes it easier to avoid repeatedly attending to the same words in the input text, hence avoiding generating repetitive text in a fact. As a means to explicitly guide the model's choice of either generating a word from a vocabulary or to "pick" a word from the input text, we explicitly compute the probability of picking a word from a vocabulary or input text using the model's context vectors. Lastly, we also investigate the effect of data-augmentation on the performance of our models. Figure 1 illustrates our approach. Our main contributions are briefly summarized as follows:

- We present several new methods for training neural OIE models. First, we propose a new method of computing syntactically rich vector representations of input tokens guided by the structure of dependency trees. This is done by using GNNs, as they are able to take into account the graph structure of a dependency tree.
- Second, we propose an additional module, the discriminator, on top of the model that generates facts. The additional module performs a binary classification of tokens in generated facts as either "real" or "fake". This new approach significantly improves the performance of sequence-to-sequence neural OIE models.
- Furthermore, we reduce repetitive words in generated facts by 23%, from 36% to 13%, when we jointly use the coverage vector and explicitly guide the model where to pick the next word when generating a fact.
- Finally, we present paraphrased versions of the CaRB (Bhardwaj et al., 2019), OIE2016 (Stanovsky and Dagan, 2016), and LSOIE (Solawetz and Larson, 2021) datasets, which we use to augment existing datasets.
- Our best performing model uses both the discriminator and the GNN encoder, and beats the state-of-the-art (SOTA) of IMoJIE on CaRB, with an improvement of 39.63% in F_1 score. This model also presents competitive results on OIE2016 and LSOIE.

2 Task Formulation

In this work, we approach OIE as a sequence generation task. As illustrated in Fig. 1, given an input text, we use a generator to generate a tuple one word at a time. Then, we pass the generated tuple through the discriminator, which classifies tokens in the tuple as either "real" or "fake", where "real" tokens are those that are both in the generated and gold tuples, while "fake" tokens are in the generated tuple but not in the gold tuple. Thus, the discriminator performs binary classification of the tokens in the generated tuple. We only consider binary extractions from sentences.

Given a sentence-tuple pair $\langle X, Y \rangle$, where $X = \langle w_1, \ldots, w_n \rangle$ and $Y = \langle y_1, \ldots, y_m \rangle$ are sequences of tokens in the input sentence and expected tuple, respectively, we define the generator's conditional

probability $P(Y|X) = \mathbb{P}\{Y|\langle w_1, \ldots, w_n\rangle\}$ as

$$\prod_i \mathbb{P}\{y_i \mid \langle w_1, \ldots, w_n\rangle; \langle y_1, \ldots, y_{i-1}\rangle\}, \quad (1)$$

where $P(y_i|\langle w_1, \ldots, w_n\rangle; \langle y_1, \ldots, y_{i-1}\rangle)$ is the probability of generating the i-th word, given the input sequence and the entire generated sequence by step i, and $P(Y|X)$ is the probability of generating an entire tuple Y, given the input sentence X.

Given a generated tuple, $Y = \langle y_1, \ldots, y_m\rangle$, and the input context vector, d^*, the discriminator seeks to label each token in Y as either "real" or "fake". The input context vector is weighted average of the vector representations of the input sequence. To that end, we define the discriminator's likelihood $P(L|Y)$ as

$$\prod_{i=1}^{m}[\sigma(v^T[\mathbf{y}_i, \mathbf{d}^*])^{(y_i^{real}=y_i)}(1-\sigma(v^T[\mathbf{y}_i, \mathbf{d}^*]))^{(y_i^{real}\neq y_i)}], \quad (2)$$

where $L = \{0, 1\}$ with 1 and 0 standing for "real" and "fake" tokens, respectively, σ is the sigmoid function, $\mathbf{y}_i$ is the vector representation of the i-th token y_i in the generated tuple, $\mathbf{d}^*$ is the input context vector, v is a set of learnable parameters, and "$[,]$" is the concatenation operation.

3 Our Approach

We now define and describe the modules that we designed to solve OIE as defined in Section 2 and illustrated in Fig. 1.

3.1 Embedding

The embedding block maps text, pre-processed into a sequence of tokens, to a corresponding sequence of embedding vectors. In this paper, we consider stacked layers of bidirectional LSTMs (BiLSTMs), a transformer encoder (Vaswani et al., 2017), pre-trained BERT (Devlin et al., 2018), a feedback transformer encoder (Fan et al., 2020), and a pre-trained ELECTRA model* (Clark et al., 2020) for the embedding block. Unlike transformers in which the representation at a given layer can only access representations from lower layers rather than the higher level representations already available, feedback transformers expose all previous representations to all future representations (Fan et al., 2020). That is, the lowest representation at the current timestep t is formed from the highest-level representations of the past. Unlike other pre-trained language models that are trained through masked language modelling (MLM), such as BERT, ELECTRA is pre-trained by detecting replaced tokens in the input sequence.

For a given sequence of input tokens, the embedding block computes a corresponding sequence of embedding vectors of the same length as the input sequence. In addition to input text, we also incorporate PoS and dependency tree tags obtained via the Spacy library†. We create embedding vectors for each PoS and dependency tree tag, and concatenate them to the vector representation of the corresponding input token. For words that are split into subwords, each subword token is attributed the same PoS and dependency tree tag as the parent word that it belongs to.

3.2 Encoding

The encoding block computes vector representations of the input sequence guided by the structure of the dependency tree. During the text prepossessing step, we construct a visibility matrix of tokens in the input sequence based on the proximity of tokens in the corresponding dependency tree. The encoder takes a sequence of vector representations from the embedding block and the visibility matrix as input, and gives a sequence of vector representations of the same length as the input sequence as output. The encoding block is optional. When the encoding block is not used, the encoder context vector is calculated using the embedder outputs.

To take into account the graph structure of the dependency tree when computing vector representations, we use a graph attention network (GAT) (Veličković et al., 2018) as encoder. GATs compute the vector representation of the current node by only attending to nodes in its neighborhood. In our case, we treat each token in the input sequence as a node and those that it is connected to in the dependency tree as its neighbors. Thus, the vector representation of the current node is computed by only attending to the vector representations of the nodes in the neighborhood as presented in the visibility matrix. For comparison, we also use a transformer encoder that does not take into account the visibility matrix when computing vector representations.

3.3 Decoding

During decoding, we adopt pointer-generator-networks (PGNs) (See et al., 2017) and make mod-

*https://huggingface.co/google/electra-small-discriminator

†https://spacy.io/usage/linguistic-features

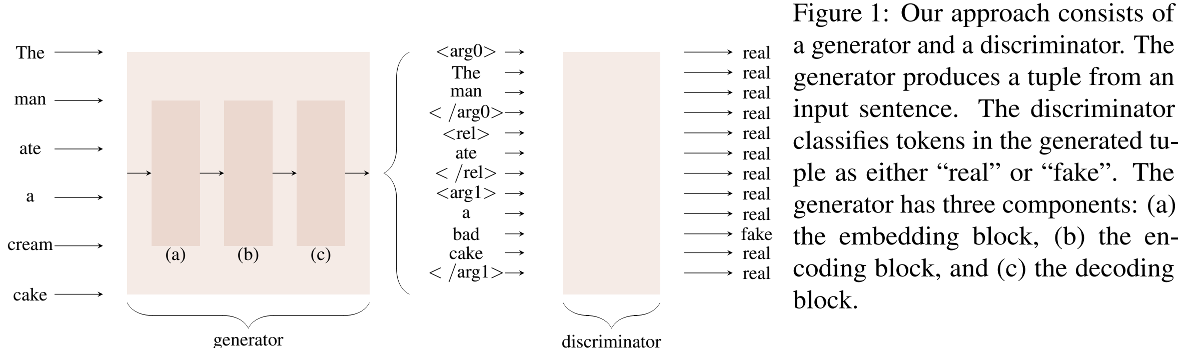

Figure 1: Our approach consists of a generator and a discriminator. The generator produces a tuple from an input sentence. The discriminator classifies tokens in the generated tuple as either "real" or "fake". The generator has three components: (a) the embedding block, (b) the encoding block, and (c) the decoding block.

ifications. For example, we calculate the distribution over the entire vocabulary from a weighted sum of the encoder and decoder context vectors, as opposed to the concatenation of the two. When generating the next token in a fact at timestep t, the embedding vectors of the entire generated sequence so far, $\langle y_1, \ldots, y_{t-1} \rangle$, are fed into either the BiLSTM network, a transformer, or feedback transformer to generate contextualized decoder hidden representations.

The decoder hidden representations $\langle d_1, \ldots, d_{t-1} \rangle$ are then used to calculate the decoder context vector, d_t^*, which is a weighted sum of decoder hidden representations (Bahdanau et al., 2015): $e_i^t = v^T Tanh(W_h d_i + b_{attn})$, where v, W_h, and b_{attn} are learnable parameters, and d_i are decoder hidden representations. Then, e^t is used to calculate attention scores:

$$a^t = Softmax(e^t). \quad (3)$$

The attention scores at timestep t, a^t, are used to calculate the decoder context vector d_t^* as weighted sum of decoder hidden representations:

$$d_t^* = \sum_{i=1}^{t-1} a_i^t d_i, \quad (4)$$

where d_i are decoder hidden representations, and a_t are attention scores over decoder hidden representations at timestep t.

Coverage mechanism. We use a coverage vector, c_t, to keep track of how much attention each word in the input text has received while generating a tuple one word at a time. The coverage vector is the sum of all attention distributions over encoder hidden representations up to timestep $t-1$: $c^t = \sum_{t'=0}^{t-1} \alpha^{t'}$, where $\alpha^{t'}$ is the attention distribution over encoder hidden representations. The coverage vector is initialized as a zero vector, as on the first timestep, no input token has been attended to.

Decoder-encoder attention. The decoder context vector d_t^* from Eq. 4, the encoder hidden representations from Section 3.2, and the coverage vector c^t are used to compute the encoder context vector h_t^*, using attention (Bahdanau et al., 2015):

$$e_i^t = v^T Tanh(W_h h_i + W_d d_t^* + W_c c^t + b_{attn}), \quad (5)$$

where v, W_h, W_d, W_c, and b_{attn} are learnable parameters, h_i are encoder hidden representations, and c^t is the coverage vector. The coverage vector is included in this computation in Eq. 5, so that the attention mechanism is informed of previous attention decisions, hence avoiding repeatedly attending to the same words in the input text. The attention distribution over input tokens, α^t, is calculated from e^t using Eq. 3. This α^t and encoder hidden states, $\langle h_1, \ldots, h_n \rangle$, are then used to compute the encoder context vector h_t^* using Eq. 4.

The attention distribution α^t obtained here can be interpreted as a probability distribution over the source words, which tells the model where to look to produce the next word.

To compute the distribution over the entire vocabulary P_{vocab}, we first calculate a weighted average of the encoder context vector h_t^* and decoder context vector d_t^*, and then feed the resulting vector into a linear layer:

$$P_{vocab} = Softmax(V[h_t^*, d_t^*] + b_{vocab}), \quad (6)$$

where V and b_{vocab} are learnable parameters, and P_{vocab} is the probability distribution over all words in the vocabulary. P_{vocab} gives the final probability of predicting the next word w from the vocabulary: $P(w) = P_{vocab}(w)$.

Generation probability. The generation probability at timestep t, P_{gen}, is calculated from the encoder and decoder context vectors, h_t^* and d_t^*, respectively: $P_{gen} = \delta(w_{h^*}^T h_t^* + w_{d^*}^T d_t^* + b_{pgen})$, where w_{h^*}, w_{d^*}, and b_{pgen} are learnable parameters, δ is a sigmoid function, and $P_{gen} \in [0, 1]$.

P_{gen} controls whether the next word should be generated from the vocabulary by sampling from P_{vocab} (Eq. 6) or copied from the input tokens by

sampling from the attention distribution over input tokens α^t. This explicitly guides the model on where to look for the next word. For each input batch, there is an extended vocabulary that is the union of the vocabulary and all tokens in the input batch. As in (See et al., 2017), this yields the following probability distribution over the extended vocabulary:

$$P(w) = P_{gen}P_{vocab} + (1-P_{gen}) \sum_{i:w_i=w} \alpha_i^t. \quad (7)$$

This allows the model to generate out-of-vocabulary (OOV) words, because if w is an OOV word, then P_{vocab} is zero. Similarly, if the word is not in the input batch, then $\sum_{i:w_i=w} \alpha_i^t = 0$.

Fig. 2 summarises the workflow in the generator.

3.4 Discriminator

Once the tuple has been generated, it is passed through the discriminator along with the input context vector, which classifies the tokens in the tuple as either "real" or "fake", where "real" tokens are those that are in both the generated tuple and the gold tuple, while "fake" tokens are those that are in the generated tuple but not in the gold tuple. The input context vector is a weighted average of the vector representations of the tokens in the input sequence. Thus, the discriminator performs binary classification of the tokens in the generated tuple while being informed of the input sequence via the input context vector. We consider a discriminator that is composed of a transformer encoder and a sigmoid layer. The transformer encoder computes vector representations of the generated tuple. The vector presentations are then passed through the sigmoid layer for classification as "real" or "fake" as in Eq. 2.

Although this approach is similar in its design to generative adversarial networks, the generator in this approach is trained with maximum likelihood rather than adversarially to fool the discriminator.

3.5 Training loss

During training, the model loss is the sum of the generator loss and discriminator loss, defined as follows.

Generator loss. We use both the probability of generating the next word $P(w)$ (Eq. 7) and the coverage mechanisms to calculate the generator's loss. The loss at timestep t, $loss_t$, is the sum of the negative log likelihood of the target word and the coverage loss:

$$loss_{gen} = -\log P(w_t) + \lambda \sum_i min(\alpha_i^t, c_i^t)\,.$$

The model is penalised by the coverage loss, $\lambda \sum_i min(\alpha_i^t, c_i^t)$, if it repeatedly attends to the same locations. We believe that a specific token from the source sentence can only appear once in a generated fact. As a result, if the final coverage vector is greater or less than one, then it is penalised.

Discriminator loss. For the discriminator, compute the negative log-likelihood of Eq. 2:

$$loss_{disc} = -\sum_{i=1}^{m} [(y_i^{real} = y_i) \log(\sigma(w^T[\mathbf{y}_i, \mathbf{d}^*])) + (y_i^{real} \neq y_i) \log(1 - \sigma(w^T[\mathbf{y}_i, \mathbf{d}^*]))].$$

4 Experiments and Results

In this section, we discuss the various experiments that we conducted. We present descriptions of the OIE datasets that we used and the data preparation routines involved. We also discuss the evaluation framework that was used and finally present the results of each experiment in terms of the F_1 and area under the precision-recall curve (AUC-PR) values.

Experiments, training, and evaluation. We set up different combinations of the neural network (NN) modules in blocks discussed in Section 3, and each unique combination resulted in a different model. Thus, we had 15 unique embedder-decoder combinations, and we present results for each combination.

We run experiments for all 15 models[‡] in 6 experimental setups: (a) *default* setup with embedders and decoders only; (b) + *discriminator*, where we add a discriminator to the default setup; (c) + *transformer encoder*, where we add a transformer-based encoder to the default setup; (d) + *GNN encoder*, where we add a GNN-based encoder to the default setup; (e) + *transformer encoder* + *discriminator*, where we add both a transformer-based encoder and discriminator to the default setup; and (f) + *GNN encoder* + *discriminator*, where we add both a GNN-based encoder and discriminator to the default setup.

For the models with BERT and ELECTRA embedders, we used pre-trained[§] `bert-base-uncased`

[‡] https://gitlab.com/Frank-Mtumbuka/oie_package

[§] https://huggingface.co/google/electra-small-discriminator

and `electra-small-discriminator` versions, respectively. Due to resource constraints[¶], the parameters of the pretrained models were frozen. All configurations for different module combinations are in Appendix E.

All models were trained by minimizing the loss defined in Section 3.5. During training, we define a teacher-forcing ratio and randomly choose whether to use teacher forcing or not depending on the value of the randomly generated number compared to the teacher-forcing ratio. The confidence of the extracted fact was calculated by multiplying the probabilities of all tokens in the fact. The checkpoints that were saved after each epoch were then evaluated on the test partition of each dataset using the CaRB (Bhardwaj et al., 2019) evaluation framework. The saved checkpoints did not include the discriminator, as the discriminator was only being used during training.

Datasets. Our models were trained and evaluated on three benchmark OIE datasets: OIE2016 (Stanovsky and Dagan, 2016), CaRB (Stanovsky and Dagan, 2016), and LSOIE (Solawetz and Larson, 2021). In Appendix A , we provide detailed descriptions of the datasets and their statistics.

Results. In Table 1, we evaluate our models on CaRB under the experimental setups described in the experiments subsection. We note that in each experimental setup, the model resulting from the combination of an ELECTRA embedder and a feedback transformer decoder outperforms other combinations. The best model on CaRB has an ELECTRA embedder, GNN encoder, feedback transformer, and a discriminator. This model achieves an F_1 value of 0.747 and an AUC-PR value of 0.740.

Based on the results in Appendix C, we note that the best model on CaRB also achieves the best results on OIE2016. The model achieves an F_1 value of 0.619 and an AUC-PR value of 0.639. Furthermore, our best model achieves an F_1 value of 0.517 and an AUC-PR value of 0.525 on LSOIE. The detailed results and analyses of similar experiments on OIE2016 and LSOIE are in Appendices C and D, respectively. Note that we cannot directly compare our results to the previous works by Zhan and Zhao (2019) and Solawetz and Larson (2021) on OIE2016 and LSOIE, respectively, as they use different evaluations. Solawetz and Larson (2021) report the best F1 and AUC-PR values on LSOIE of only 0.380 and 0.220, respectively.

Results on augmented datasets. We trained the best model from Table 1 on paraphrased and mixed versions of CaRB. When training the model on the mixed dataset, each batch was made of equal portions of the paraphrased and original versions. After training, the model was evaluated on the original test set of CaRB dataset. We note that the model performs poorly when trained only on the paraphrased dataset. However, when trained on the mixed dataset, the model performs better than when trained on just the original dataset. Table 2 presents the results.

Comparison to other systems. In Table 3, we compare our best performing model to previous OIE systems on CaRB. The results from other systems are quoted directly from previous works, which share the same experimental settings and can directly be compared with. We compare our best performing model to results reported by Kolluru et al. (2020) (IMoJIE) under the same settings. From this comparison, our best performing model beats the SOTA model IMoJIE with 39.63% in F_1 value on CaRB.

5 Results Analysis and Ablation Studies

In this section, we discuss the results presented in Section 4. We also present a detailed analysis of the contributions of each experimental setup.

Performance across experimental setups. We considered 6 experimental setups, which we have defined in Section 4. Table 5 presents the average performance of all models in each setup on CaRB based on the results presented in Table 1. We note that each experimental setup introduced significant[‖] improvement in performance compared to the control experimental setup, *default*. The setup with the GNN encoder only, + *GNN encoder*, achieves the best average performance in F_1. It improves on the *default* setup with 11.69%. The setup with both the GNN encoder and discriminator, + *GNN encoder* + *discriminator*, achieves the best average performance in AUC-PR. It improves on the *default* setup with 6.53%. Furthermore, we note that models in the + *GNN encoder* setup perform better than models in the + *transformer encoder* setup with 1.56% and 2.19% improvements in average F_1 and AUC-PR values, respectively. To that end,

[¶]The models were trained on one GeForce GTX TITAN XP GPU.

[‖]Significance tests for the results in Table 1 are presented in Appendix B.

		BiLSTM		Transformer		Feedback Transformer		BERT		ELECTRA	
		F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
(a)	**BiLSTM**	0.395	0.410	0.442	0.423	0.453	0.477	0.433	0.434	0.445	0.444
	Transformer	0.438	0.389	0.508	0.475	0.599	0.549	0.567	0.553	0.685	0.670
	Feedback Transformer	0.504	0.400	0.620	0.602	0.706	0.672	0.701	0.673	**0.717**	**0.691**
(b)	**BiLSTM**	0.452	0.420	0.463	0.435	0.474	0.457	0.460	0.458	0.456	0.463
	Transformer	0.632	0.399	0.661	0.482	0.680	0.552	0.675	0.569	0.700	0.696
	Feedback Transformer	0.541	0.404	0.692	0.676	0.711	0.700	0.710	0.699	**0.726**	**0.719**
(c)	**BiLSTM**	0.410	0.415	0.451	0.435	0.466	0.457	0.467	0.459	0.478	0.463
	Transformer	0.648	0.401	0.669	0.484	0.683	0.562	0.687	0.565	0.699	0.651
	Feedback Transformer	0.516	0.419	0.701	0.682	0.718	0.685	0.723	0.690	**0.728**	**0.704**
(d)	**BiLSTM**	0.418	0.432	0.465	0.445	0.476	0.469	0.469	0.464	0.468	0.466
	Transformer	0.656	0.412	0.684	0.498	0.703	0.572	0.699	0.568	0.709	0.693
	Feedback Transformer	0.527	0.423	0.714	0.699	0.728	0.695	0.731	0.698	**0.739**	**0.714**
(e)	**BiLSTM**	0.471	0.439	0.483	0.455	0.485	0.468	0.488	0.480	0.475	0.482
	Transformer	0.457	0.417	0.528	0.482	0.610	0.563	0.614	0.589	0.719	0.715
	Feedback Transformer	0.560	0.423	0.640	0.622	0.722	0.711	0.737	0.715	**0.745**	**0.738**
(f)	**BiLSTM**	0.473	0.441	0.484	0.456	0.495	0.478	0.489	0.481	0.477	0.484
	Transformer	0.459	0.419	0.529	0.483	0.620	0.573	0.615	0.590	0.721	0.717
	Feedback Transformer	0.562	0.425	0.641	0.623	0.732	0.721	0.738	0.716	**0.747**	**0.740**

Table 1: Results from different module combinations on the CaRB dataset. The columns and rows represent embedders and decoders , respectively. (a) *default* setup, (b) + *discriminator* setup, (c) + *transformer encoder* setup, (d) + *GNN encoder* setup, (e) + *transformer encoder* + *discriminator* setup, and (f) + *GNN encoder* + *discriminator* setup. The results in bold indicate the best performance in each setup.

Dataset	F_1	**AUC-PR**
Para-phrased	0.509	0.524
Original	0.747	0.740
Mixed	**0.753**	**0.751**

Table 2: Performance of our best model from Table 1, (ELECTRA + GNN encoder + discriminator), when trained on paraphrased, original and mixed versions of CaRB.

System	**Metric**	
	F_1	**AUC-PR**
Stanford-IE	0.230	0.134
OLLIE	0.411	0.225
OpenIE-4	0.516	0.295
OpenIE-5	0.485	0.257
ClausIE	0.451	0.224
CopyAttention	0.354	0.204
IMoJIE	0.535	0.333
Our best model	**0.747**	**0.740**

Table 3: How our best performing model compares to other systems on the CaRB dataset and evaluation framework. The results of previous systems are directly quoted from (Kolluru et al., 2020), because they share the same experimental settings and can be directly compared. Results in bold indicate the best performance.

we see that the GNN encoder that computes embeddings of the input sequence based on the structure of the dependency tree performs better than the transformer encoder, which does not take into account the structure of the dependency tree when computing embeddings. Additionally, the best performance is achieved when the GNN encoder is used jointly with the discriminator.

Taking syntactic information into account when computing embeddings not only improves model performance but also improves extraction in some cases. For example, given a sentence *"The twins who came are identical but are uniquely mannered."*, the best model with the transformer encoder, which does not consider the topology of the dependency tree, yields ⟨*who*, *are*, *uniquely mannered*⟩, whereas the extraction with the GNN encoder, which does consider the topology of the dependency tree, yields ⟨*The twins*, *are*, *uniquely mannered*⟩. This example shows that syntax can be useful in OIE. Thus, we conclude that taking into account the dependency tree's structure when computing embeddings and the discriminative training approach are the main cause of the high boost in the performance of our models. This is consistently shown on all three benchmark datasets.

Performance of modules. We also looked into the average performance of the modules in the embedder and decoder blocks. Table 6 summarises the analysis based on results in Table 1. Considering all models, we note that models with a pre-trained ELECTRA model achieve the best average values of 0.653 and 0.625 in F_1 and AUC-PR, respectively. Furthermore, models with the feedback transformer decoder achieve the best average values of 0.676 and 0.636 in F_1 and AUC-PR, respectively. The feedback transformer performs well, and this is largely owing to its ability to capture the input's sequential property, which is critical for tuple generation. However, the other components that we introduce in this paper boost performance significantly as well. For instance, in F_1, the *+GNN encoder* setup achieves the best average performance by 11.69% over the default setup. In AUC-PR, the + *GNN encoder* + *discriminator* setup achieves the best average performance by 6.53% over the *default*

Model	$S \cap T$	Repetition in T
-(*coverage mechanism + gen prob*)	89%	36%
+(*coverage mechanism + gen prob*)	65%	13%

Table 4: Results on our best model's ability to introduce new words from the vocabulary into the generated tuple, and to avoid generating repetitive tokens on CaRB. -(*coverage mechanism + gen prob*) is the best model without both *generation probability* and *coverage mechanisms* and +(*coverage mechanism + gen prob*) is the opposite.

Experimental Setup	Avg. F_1	Avg. AUC-PR
Default	0.548	0.522
+ Transformer Encoder	0.603	0.538
+ Discriminator	0.602	0.542
+ GNN Encoder	**0.612**	0.550
+ Transformer Encoder + Discriminator	0.582	0.553
+ GNN Encoder + Discriminator	0.585	**0.556**

Table 5: Average performance of all models under different experimental setups on CaRB. Default represents the setup where all models are comprised of just embedders and decoders.

	Module	Avg. F_1	Avg. AUC-PR
(i)	**BiLSTM**	0.506	0.416
	Transformer	0.576	0.525
	BERT	0.612	0.578
	Feedback Transformer	0.615	0.574
	ELECTRA	**0.635**	**0.625**
(ii)	**BiLSTM**	0.462	0.452
	Transformer	0.629	0.543
	Feedback Transformer	**0.676**	**0.636**

Table 6: Average performance different modules in different blocks for all experiments on CaRB. **(i)** for the embedders, and **(ii)** for the decoders. The best average performance for each block is shown in **bold**.

setup.

Token repetition. In this work, the core tools that control the repetition of tokens during decoding are the *generation probability* and *coverage mechanisms* that we have defined in Section 3.3. As such, to check whether these tools are effective, we run our best model with and without the tools and the compute the average percentage of repetitive tokens and the model's ability to introduce new words from the vocabulary. When the best model is run on 50 samples from the test partition with and without both *generation probability* and *coverage mechanisms*. We compute the number of tokens that are both in the source sentence S and the generated tuple T, $S \cap T$, in both settings to measure the model's ability to introduce words from the vocabulary. Similarly, we also compute a percentage of repetitive tokens in T. Table 4 summarizes the results. From Table 4, we note that the percentage of tokens in $|S \cap T|$ drops by 24% when using both *generation probability* and *coverage mechanisms*. This implies that the model copies fewer tokens from the source sentence, and introduces more tokens from the vocabulary, hence the improved ability to generate implicit facts. Furthermore, the repetitive tokens reduce by 23%. To that end, the results confirm that both *generation probability* and *coverage mechanisms* are effective in controlling repetitive tokens, and improving the models' ability to generate implicit facts. This is the first work to look into redundancy at fact level. Sun et al. (2018) and Kolluru et al. (2020) look into redundant information as an issue of generating the same facts multiple times.

6 Related Work

In the recent past, neural OIE models have been developed, and they tackle OIE as either sequence labeling or sequence generation. The former (Stanovsky et al., 2018; Roy et al., 2019; Jiang et al., 2019; Zhan and Zhao, 2019; Hohenecker et al., 2020) involves tagging each word in the input text with an appropriate tag that indicates whether the word belongs to either the subject, relation, or object. Methods that approach OIE as sequence generation generate facts one word at a time. These include CopyAttention (Cui et al., 2018), Logician (Sun et al., 2018), and IMoJIE (Kolluru et al., 2020). CopyAttention is an encoder-decoder model enhanced with copying and attention mechanisms. Logician is another encoder-decoder model that uses coverage attention and gated-dependancy attention to extract facts from Chinese text. IMoJIE outputs a variable number of different facts per sentence. The next fact from a sentence is best determined in context of all other facts extracted from it so far. Hence, IMoJIE uses a decoding strategy that generates facts in a sequential fashion, one after another, each one being aware of all the ones generated prior to it.

Our work substantially differs from previous OIE works. Firstly, we are the first to compute embeddings of input sequences based on the structure of the corresponding dependency trees. This is done by first forming a visibility matrix based on which words are directly related in the dependency tree. Then, a GNN encoder computes syntactically rich embeddings of the input sequences controlled by the formed visibility matrices. This formulation is different from previous approaches where the embeddings of PoS and dependency tags are just

concatenated to the embeddings of the according text tokens at embedding level. Secondly, this is the first work that uses an additional module that classifies tokens in the generated tuple as either "real" or "fake" during training. Thirdly, unlike previous neural approaches that tackle OIE as sequence generation, we consider the entire generated sequence by timestep t, $\langle y_1, \ldots, y_{t-1} \rangle$, to compute a decoder context vector, as discussed in Eqs. 3–4 in Section 3.3. Additionally, Sun et al. (2018) and Kolluru et al. (2020) consider redundant information as the issue of generating the same facts multiple times, while we consider redundancy at token level in generated facts. Furthermore, Cui et al. (2018) and Sun et al. (2018) are restrictive in how the models "pick" words when generating facts, which restrains the models' flexibility in generating facts. Cui et al. (2018) only limit the sample space only to the source sentence, and Sun et al. (2018) only consider the source sentence and keywords, while we consider the source sentence and the entire vocabulary.

7 Summary and Outlook

This work has shown that computing syntactically rich embeddings of text based on the structure of dependency trees significantly improves the performance of neural OIE models and the quality of extractions. Furthermore, the novel discriminative training method for OIE models boosts models' performance. Additionally, we show that *generation probability* and *coverage mechanisms* substantially improve the models' ability to introduce new words into generated facts and reduce repetitive tokens in generated facts. Finally, we have presented paraphrased versions of OIE2016, CaRB, and LSOIE, and shown that the models' performance substantially improves when trained on datasets augmented by such data. Our approach, syntactically rich discriminative training, beats the SOTA of IMoJIE on the latest CaRB benchmark dataset: the model improves the F_1 score of IMoJIE by 39.63%. This model also presents competitive results on other benchmark datasets: OIE2016 and LSOIE.

In future work, we will build on this work and investigate further whether the proposed approaches could also be beneficial to sequence labelling approaches for OIE. Furthermore, models trained on original OIE datasets perform poorly on paraphrased versions of the datasets. This is clear from the results shown in Table 2. Under normal conditions, one would expect to identify the same sets of facts from all paraphrased versions of the same sentence. This is not true of the OIE models, and the phenomenon shows that models do not understand sentence meaning. Future studies could build on this to look into how OIE models can better capture sentence meanings.

8 Limitations

Like any other research, the work described in this paper has some space for improvement. First, we limited ourselves to binary relations that are described in a single sentence. However, in the real world, we are constantly confronted with content that spans many sentences and relationships that span multiple sentences. In future research, the concepts and approaches outlined in this paper could be expanded to include n-ary relations that span many sentences. Second, models trained on original datasets perform poorly on paraphrased datasets. This is clear from the results shown in Table 2. Under normal conditions, one would expect to identify the same sets of facts from all paraphrased versions of the same sentence. This is not true of the OIE models, and the phenomenon shows that models do not understand sentence meaning. Future studies could build on this to look into how OIE models can better capture sentence meanings. Finally, we used the formulation for constructing syntactically rich vector representations as well as the new discriminative training approach to only train sequence generation OIE models. It could be examined further in future research whether the proposed methodologies could also be advantageous to OIE sequence labelling approaches.

Acknowledgements

This work was partially supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1, by the AXA Research Fund, by the EPSRC grant EP/R013667/1, and by the ESRC grant "Unlocking the Potential of AI for English Law". We also acknowledge the use of Oxford's ARC facility, of the EPSRC-funded Tier 2 facilities JADE (EP/P020275/1) and JADE II (EP/T022205/1), and of GPU computing support by Scan Computers International Ltd. Frank Mtumbuka was supported by the Rhodes Trust under a Rhodes Scholarship.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. 2007. Open information extraction from the Web. In *IJCAI*, pages 2670–2676.

Sangnie Bhardwaj, Samarth Aggarwal, and Mausam Mausam. 2019. CaRB: A crowdsourced benchmark for open IE. In *EMNLP-IJCNLP*, pages 6262–6267.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators.

Lei Cui, Furu Wei, and Ming Zhou. 2018. Neural open information extraction. In *ACL (Volume 2: Short Papers)*, pages 407–413.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Angela Fan, Claire Gardent, Chloé Braud, and Antoine Bordes. 2019. Using local knowledge graph construction to scale Seq2Seq models to multi-document inputs. In *EMNLP-IJCNLP*, pages 4186–4196.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. 2020. Addressing some limitations of transformers with feedback memory. *CoRR*, abs/2002.09402.

Nicholas FitzGerald, Julian Michael, Luheng He, and Luke Zettlemoyer. 2018. Large-scale QA-SRL parsing. In *ACL (Volume 1: Long Papers)*, pages 2051–2060.

Patrick Hohenecker, Frank Mtumbuka, Vid Kocijan, and Thomas Lukasiewicz. 2020. Systematic comparison of neural architectures and training approaches for open information extraction. In *EMNLP*, pages 8554–8565.

Shengbin Jia and Yang Xiang. 2019. Chinese user service intention classification based on hybrid neural network. *Journal of Physics: Conference Series*, 1229.

Zhengbao Jiang, Pengcheng Yin, and Graham Neubig. 2019. Improving open information extraction via iterative rank-aware learning. In *ACL*, pages 5295–5300.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Keshav Kolluru, Samarth Aggarwal, Vipul Rathore, Mausam, and Soumen Chakrabarti. 2020. IMoJIE: Iterative memory-based joint open information extraction. In *ACL*, pages 5871–5886.

Mausam Mausam. 2016. Open information extraction systems and downstream applications. In *IJCAI*.

Christina Niklaus, Matthias Cetto, André Freitas, and Siegfried Handschuh. 2018. A survey on open information extraction. In *COLING*, pages 3866–3878.

Arpita Roy, Youngja Park, Taesung Lee, and Shimei Pan. 2019. Supervising unsupervised open information extraction models. In *EMNLP-IJCNLP*, pages 728–737.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *ACL (Volume 1: Long Papers)*, pages 1073–1083.

Stephen Soderland, Brendan Roof, Bo Qin, Shi Xu, Mausam Mausam, and Oren Etzioni. 2010. Adapting open information extraction to domain-specific relations. *AI Magazine*, 31(3):93–102.

Jacob Solawetz and Stefan Larson. 2021. LSOIE: A large-scale dataset for supervised open information extraction. In *EACL: Main Volume*, pages 2595–2600.

Gabriel Stanovsky and Ido Dagan. 2016. Creating a Large Benchmark for Open Information Extraction. In *EMNLP*.

Gabriel Stanovsky, Julian Michael, Luke Zettlemoyer, and Ido Dagan. 2018. Supervised open information extraction. In *NAACL-HLT, Volume 1 (Long Papers)*, pages 885–895.

Mingming Sun, Xu Li, Xin Wang, Miao Fan, Yue Feng, and Ping Li. 2018. Logician: A unified end-to-end neural approach for open-domain information extraction. In *WSDM*, pages 556–564.

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 *NeurIPS*, volume 30, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR*.

Junlang Zhan and Hai Zhao. 2019. Span model for open information extraction on accurate corpus. *CoRR*, abs/1901.10879.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434.

A Datasets' Descriptions and Statistics.

OIE2016 has a total of 5,078 training samples; it is small, which makes it hard to train models that generalize to unseen problem instances. For training purposes, OIE2016 was augmented with samples from another dataset created by Cui et al. (2018). This resulted in a huge dataset of more than 36M training samples. This augmented dataset was then trimmed to 1.7M training samples using preprocessing steps presented in (Hohenecker et al., 2020). LSOIE is constructed from QA-SRL BANK 2.0 (FitzGerald et al., 2018) and contains over 70K sentences and over 150K extraction tuples. CaRB is an improved dataset compared to OIE2016 and is crowdsourced. It was annotated by NLP experts, and it is more accurate than OIE2016 (Bhardwaj et al., 2019). In addition to binary and explicit extractions, CaRB also comes with implicit as well as n-ary extractions. The n-ary extractions were truncated to bear some resemblance to the binary extractions that we considered in this work. For example, in the sentence "A year later, he was appointed Attorney-General for Ireland and on this occasion was sworn of the Privy Council of Ireland", we have the subject "he", relation "was appointed", and objects "Attorney-General", "for Ireland", and "A year later". From this, we formulate a truncated tuple ⟨*he*, *was appointed*, *Attorney-General for Ireland a year later*⟩.

In each extraction, we introduced special tokens into the vocabulary to mark the beginning and end of either subject, relation, or object. "$\langle arg0\rangle$" and "$\langle /arg0\rangle$" were used to indicate the start and end of a subject span, respectively, while "$\langle rel\rangle$" and "$\langle /rel\rangle$" were used to indicate the start and end of a relation span, respectively, and "$\langle arg1\rangle$" and "$\langle /arg1\rangle$" were used to indicate the start and end of an object span, respectively.

In addition to the original versions of the CaRB, OIE2016, and LSOIE datasets, we also generate paraphrased versions, which we use to train our models. We only paraphrase the training sets of the datasets and keep the test sets. We use Parrot[**] to paraphrase samples. Parrot generates paraphrased versions that are adequate and fluent, while being as different as possible from the original versions on the surface lexical form. We paraphrased the datasets, so that we get more and diverse data for training our models. Furthermore, as paraphrased samples convey the same meaning as the original samples but with a different lexical form, models trained on this data have a better ability to capture implicit relations.

[**] https://github.com/PrithivirajDamodaran/Parrot_Paraphraser

B Significance of Various Experimental Setups on CaRB

We further investigated whether the improvements brought about by each experimental configuration in Table 1 are substantial rather than a random occurrence. For this, we perform a paired t-test on each new experimental configuration and the default setup, taking into account all paired F_1 and AUC-PR values.

First, the paired t-test produced p values smaller than .05, $p < .05$, in both F_1 and AUC-PR when the + *transformer encoder* setup was compared to the *default* setup. This means that the *default* setup has a less than 5% chance of outperforming the + *transformer encoder* setup. Second, when the + *discriminator* and + *transformer encoder* + *discriminator* setups are individually matched with the *default* setup, both F_1 and AUC-PR have p values of less than .03, $p < .03$. This means that the *default* setup has a less than 3% chance of outperforming either the + *discriminator* setup or the + *transformer encoder* + *discriminator* setup. Finally, when the + *GNN encoder* and + *GNN encoder* + *discriminator* setups are individually paired with the *default* setup, both F_1 and AUC-PR produce p values of less than .01, $p < .01$. This means that the *default* setup has less than a 1% chance of outperforming either the + *GNN encoder* or + *GNN encoder* + *discriminator* setup.

Based on the above deliberations, each experimental setup significantly improved the *default* setup in terms of F_1 and AUC-PR compared to the *default* setup. Additionally, the + *GNN encoder* setup outperforms the + *transformer encoder* setup. The + *GNN encoder* setup generates syntactically richer embeddings than the + *transformer encoder* setup. Furthermore, the + *GNN encoder* + *discriminator* setup produces the best results, which can be attributed to the formulation for computing embeddings based on the dependency tree's structure as well as the discriminative training technique. Finally, these patterns may be seen in the results shown in Tables 7 and 12, which were derived from the OIE2016 and LSOIE datasets, respectively. These advancements were not coincidental.

C OIE2016 Results and Analysis

In this section, we present results of our models on the OIE2016 dataset. All models were trained and evaluated in all six experimental setups discussed in Section 4. We used the CaRB evaluation framework.

Results. Table 7 presents the results of evaluating our models on the OIE2016 dataset under the experimental setups described in the experiments subsection. Note that in each experimental setup, the model resulting from the combination of an ELECTRA embedder and a feedback transformer decoder outperforms other combinations. The best model on the OIE2016 dataset has an ELECTRA embedder, GNN encoder, feedback transformer, and a discriminator. The model achieves an F_1 value of 0.619 and an AUC-PR value of 0.639. Note that we cannot directly compare our results to the previous work by Zhan and Zhao (2019), because the evaluation framework used is different.

Performance across experimental setups. Table 8 presents the average performance of all models in each experimental setup on OIE2016 based on the results presented in Table 7. We note that each experimental setup introduced considerable improvement in performance compared to the control experimental setup, *default*. The setup with the GNN encoder only, + *GNN encoder*, achieves the best average performance in F_1. It improves on the *default* setup with 18.90%. The setup with both the GNN encoder and discriminator, + *GNN encoder* + *discriminator*, achieves the best average performance in AUC-PR. It improves on the *default* setup with 13.74%. Furthermore, we note that models in the + *GNN encoder* setup perform better than models in the + *transformer encoder* setup with 2.32% and 4.08% improvements in average F_1 and AUC-PR values, respectively. To that end, we see that the GNN encoder that computes embeddings of the input sequence based on the structure of the dependency tree performs better than the transformer encoder that does not take into account the structure of the dependency tree when computing embeddings. Additionally, the best performance is achieved when the GNN encoder is used jointly with the discriminator. Thus, we conclude that taking into account the dependency tree's structure when computing embeddings and the discriminative training approach are the main cause of the high boost in the performance of our models.

Performance of modules. We also looked into the average performance of the modules in the embedder and decoder blocks. Table 9 summarises the analysis based on results in Table 7. Considering all models, the models with a pre-trained ELECTRA model achieve the best average values of 0.545 and 0.550 in F_1 and AUC-PR, respectively. Furthermore, models with the feedback transformer decoder achieve the best average values of 0.563 and 0.564 in F_1 and AUC-PR, respectively.

Token repetition. Our best model from Table 7 is run on 50 samples from the test partition of the OIE2016 dataset with and without both *generation probability* and *coverage mechanisms*. We compute the number of tokens that are both in the source sentence S and the generated tuple T, $S \cap T$, in both settings to measure the model's ability to introduce words from the vocabulary. Similarly, we also compute a percentage of repetitive tokens in T. Table 10 summarises the results. From Table 10, we note that that the percentage of tokens in $|S \cap T|$ drops by 18% when using both *generation probability* and *coverage mechanisms*. This implies that the model copies fewer tokens from the source sentence, and introduces more tokens from the vocabulary, hence the improved ability to generate implicit facts. Furthermore, the repetitive tokens reduce by 23.70%. To that end, the results confirm that both *generation probability* and *coverage mechanisms* are effective in controlling repetitive tokens, and improving the models' ability to generate implicit facts.

Performance on augmented dataset. We trained the best model from Table 7 on paraphrased and mixed versions of the OIE2016 dataset. After training, the model was evaluated on the original test set of the OIE2016 dataset. We note that the performance of the model improves when trained on the mixed dataset. Table 11 summarises the results.

D LSOIE Results and Analysis

In this section, we present results of our models on the LSOIE dataset. We considered all experimental setups discussed in Section 4, and we used the CaRB evaluation framework.

Results. In Table 7, we present the results of evaluating our models on the LSOIE dataset under all our experimental setups. We note that in each experimental setup, the model resulting from the combination of an ELECTRA embedder and a feedback transformer decoder outperforms other combina-

		BiLSTM		Transformer		Feedback Transformer		BERT		ELECTRA	
		F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
(a)	**BiLSTM**	0.411	0.438	0.420	0.441	0.421	0.445	0.420	0.446	0.425	0.449
	Transformer	0.439	0.446	0.440	0.447	0.469	0.466	0.471	0.469	0.485	0.471
	Feedback Transformer	0.472	0.499	0.487	0.498	0.491	0.502	0.494	0.515	**0.497**	**0.521**
(b)	**BiLSTM**	0.431	0.450	0.460	0.455	0.467	0.466	0.469	0.468	0.472	0.470
	Transformer	0.497	0.451	0.510	0.488	0.549	0.523	0.560	0.555	0.595	0.602
	Feedback Transformer	0.546	0.554	0.568	0.572	0.577	0.579	0.599	0.585	**0.600**	**0.626**
(c)	**BiLSTM**	0.469	0.468	0.470	0.471	0.485	0.475	0.489	0.477	0.491	0.483
	Transformer	0.504	0.470	0.510	0.476	0.522	0.502	0.562	0.520	0.687	0.660
	Feedback Transformer	0.531	0.527	0.554	0.536	0.575	0.550	0.599	0.584	**0.601**	**0.607**
(d)	**BiLSTM**	0.470	0.471	0.471	0.476	0.488	0.487	0.492	0.489	0.493	0.491
	Transformer	0.528	0.482	0.531	0.489	0.570	0.523	0.581	0.556	0.597	0.604
	Feedback Transformer	0.548	0.556	0.569	0.576	0.585	0.590	0.594	0.597	**0.616**	**0.633**
(e)	**BiLSTM**	0.444	0.463	0.452	0.466	0.472	0.469	0.481	0.477	0.486	0.487
	Transformer	0.491	0.453	0.501	0.472	0.535	0.494	0.551	0.501	0.562	0.567
	Feedback Transformer	0.541	0.526	0.551	0.540	0.566	0.540	0.582	0.563	**0.599**	**0.592**
(f)	**BiLSTM**	0.454	0.472	0.472	0.477	0.479	0.488	0.487	0.485	0.493	0.489
	Transformer	0.521	0.473	0.529	0.481	0.573	0.524	0.581	0.567	0.592	0.607
	Feedback Transformer	0.568	0.556	0.569	0.579	0.586	0.590	0.602	0.596	**0.619**	**0.639**

Table 7: Results from different module combinations on the OIE2016 dataset. The columns and rows represent embedders and decoders, respectively. (a) *default* setup, (b) + *discriminator* setup, (c) + *transformer encoder* setup, (d) + *GNN encoder* setup, (e) + *transformer encoder* + *discriminator* setup, and (f) + *GNN encoder* + *discriminator* setup. The results in bold indicate the best performance in each setup.

Experimental Setup	Avg. F_1	Avg. AUC-PR
Default	0.456	0.470
+ Transformer Encoder	0.530	0.514
+ Discriminator	0.527	0.523
+ GNN Encoder	**0.542**	0.533
+ Transformer Encoder + Discriminator	0.521	0.507
+ GNN Encoder + Discriminator	0.542	**0.535**

Table 8: Average performance of all models under different experimental setups on OIE2016. Default represents the setup where all models are comprised of just embedders and decoders.

	Module	Avg. F_1	Avg. AUC-PR
(i)	**BiLSTM**	0.492	0.487
	Transformer	0.504	0.496
	BERT	0.534	0.526
	Feedback Transformer	0.523	0.512
	ELECTRA	**0.545**	**0.550**
(ii)	**BiLSTM**	0.464	0.470
	Transformer	0.431	0.508
	Feedback Transformer	**0.563**	**0.564**

Table 9: Average performance different modules in different blocks for all experiments on OIE2016. **(i)** for the embedders, and **(ii)** for the decoders. The best average performance for each block is shown in **bold**.

Model	$S \cap T$	Repetition in T
-(*coverage mechanism* + *gen prob*)	85%	39.50%
+(*coverage mechanism* + *gen prob*)	67%	15.80%

Table 10: Results on our best model's ability to introduce new words from the vocabulary into the generated tuple, and to avoid generating repetitive tokens on OIE2016. -(*coverage mechanism* + *gen prob*) represents the best model without both *generation probability* and *coverage mechanisms* and +(*coverage mechanism* + *gen prob*) represents the opposite.

Dataset	F_1	AUC-PR
Para-phrased	0.419	0.449
Original	0.619	0.639
Mixed	**0.701**	**0.699**

Table 11: Performance of our best model from Table 7, (ELECTRA + GNN Encoder + Discriminator), when trained on paraphrased, original, and mixed versions of the OIE2016 dataset.

tions. The best model on LSOIE has an ELECTRA embedder, GNN encoder, feedback transformer, and a discriminator. The model achieves an F_1 value of 0.517 and an AUC-PR value of 0.525.

Performance across experimental setups. Table 13 presents the average performance of all models in each experimental setup on OIE2016 based on the results in Table 12. We note that each experimental setup introduced considerable improvement in performance compared to the control experimental setup, *default*. The setup with the GNN encoder only, + *GNN encoder*, achieves the best average performance in F_1. It improves on the *default* setup with 7.73%. The setup with both the GNN encoder and discriminator, + *GNN encoder* + *discriminator*, achieves the best average performance

		BiLSTM		Transformer		Feedback Transformer		BERT		ELECTRA	
		F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
(a)	**BiLSTM**	0.375	0.381	0.380	0.391	0.391	0.415	0.405	0.421	0.412	0.429
	Transformer	0.428	0.433	0.431	0.438	0.438	0.446	0.448	0.550	0.452	0.457
	Feedback Transformer	0.456	0.467	0.457	0.472	0.462	0.480	0.475	0.492	**0.487**	**0.501**
(b)	**BiLSTM**	0.410	0.397	0.416	0.413	0.429	0.423	0.434	0.431	0.440	0.433
	Transformer	0.428	0.440	0.435	0.448	0.446	0.455	0.451	0.458	0.456	0.462
	Feedback Transformer	0.435	0.470	0.458	0.476	0.474	0.486	0.480	0.500	**0.496**	**0.519**
(c)	**BiLSTM**	0.423	0.399	0.432	0.401	0.435	0.422	0.441	0.434	0.446	0.450
	Transformer	0.433	0.421	0.439	0.435	0.447	0.444	0.463	0.455	0.479	0.471
	Feedback Transformer	0.459	0.461	0.463	0.467	0.472	0.475	0.490	0.485	**0.595**	**0.504**
(d)	**BiLSTM**	0.430	0.413	0.445	0.432	0.439	0.455	0.444	0.454	0.450	0.456
	Transformer	0.438	0.443	0.451	0.456	0.480	0.467	0.493	0.475	0.499	0.485
	Feedback Transformer	0.469	0.479	0.471	0.499	0.481	0.489	0.499	0.496	**0.509**	**0.524**
(e)	**BiLSTM**	0.413	0.401	0.424	0.414	0.437	0.425	0.445	0.439	0.460	0.449
	Transformer	0.427	0.431	0.452	0.446	0.455	0.457	0.459	0.461	0.462	0.463
	Feedback Transformer	0.465	0.463	0.467	0.468	0.473	0.479	0.499	0.481	**0.507**	**0.502**
(f)	**BiLSTM**	0.427	0.415	0.438	0.436	0.449	0.446	0.452	0.449	0.469	0.466
	Transformer	0.439	0.445	0.454	0.458	0.458	0.470	0.463	0.481	0.471	0.486
	Feedback Transformer	0.471	0.475	0.474	0.492	0.487	0.492	0.502	0.496	**0.517**	**0.525**

Table 12: Results from different module combinations on the LSOIE dataset. The columns and rows represent embedders and decoders, respectively. (a) *default* setup, (b) + *discriminator* setup, (c) + *transformer encoder* setup, (d) + *GNN encoder* setup, (e) + *transformer encoder* + *discriminator* setup, and (f) + *GNN encoder* + *discriminator* setup. The results in bold indicate the best performance in each setup.

Experimental Setup	Avg. F_1	Avg. AUC-PR
Default	0.433	0.452
+ Transformer Encoder	0.461	0.448
+ Discriminator	0.446	0.454
+ GNN Encoder	**0.467**	0.468
+ Transformer Encoder + Discriminator	0.456	0.452
+ GNN Encoder + Discriminator	0.465	**0.469**

Table 13: Average performance of all models under different experimental setups on LSOIE. Default represents the setup where all models are comprised of just embedders and decoders.

in AUC-PR. It improves on the *default* setup with 3.81%. Furthermore, we note that models in the + *GNN encoder* setup perform better than models in the + *transformer encoder* setup with 1.19% and 4.48% improvements in average F_1 and AUC-PR values, respectively. To that end, we see that the GNN encoder that computes embeddings of the input sequence based on the structure of the dependency tree performs better than the transformer encoder that does not take into account the structure of the dependency tree when computing embeddings. Additionally, the best performance is achieved when the GNN encoder is used jointly with the discriminator. Thus, we conclude that taking into account the dependency tree's structure when computing embeddings and the discriminative training approach are the main cause of the high boost in the performance of our models.

Performance of modules. We also looked into the average performance of the modules in the embedder and decoder blocks. Table 14 summarises the analysis based on results in Table 12. Considering all models, the models with a pre-trained ELECTRA model achieve the best average values of 0.478 and 0.477 in F_1 and AUC-PR, respectively. Furthermore, models with the feedback transformer decoder achieve the best average values of 0.482 and 0.487 in F_1 and AUC-PR, respectively.

	Module	Avg. F_1	Avg. AUC-PR
(i)	**BiLSTM**	0.435	0.435
	Transformer	0.444	0.447
	BERT	0.463	0.470
	Feedback Transformer	0.453	0.457
	ELECTRA	**0.478**	**0.0.470**
(ii)	**BiLSTM**	0.430	0.426
	Transformer	0.452	0.458
	Feedback Transformer	**0.482**	**0.487**

Table 14: Average performance different modules in different blocks for all experiments on LSOIE. **(i)** for the embedders, and **(ii)** for the decoders. The best average performance for each block is shown in **bold**.

Token repetition. Our best model from Table 12 is run on 50 samples from the test partition of the LSOIE dataset with and without both *generation probability* and *coverage mechanisms*. We compute the number of tokens that are both in the source sentence S and the generated tuple T, $S \cap T$, in both settings to measure the model's ability to introduce words from the vocabulary. Similarly, we also com-

Model	$S \cap T$	Repetition in T
-(*coverage mechanism* + *gen prob*)	82%	34%
+(*coverage mechanism* + *gen prob*)	68%	11%

Table 15: Results on our best model's ability to introduce new words from the vocabulary into the generated tuple, and to avoid generating repetitive tokens on LSOIE. -(*coverage mechanism* + *gen prob*) represents the best model without both *generation probability* and *coverage mechanisms* and +(*coverage mechanism* + *gen prob*) represents the opposite.

Dataset	F_1	AUC-PR
Para-phrased	0.401	0.424
Original	0.517	0.525
Mixed	**0.610**	**0.605**

Table 16: Performance of our best model from Table 12, (ELECTRA + GNN encoder + discriminator), when trained on paraphrased, original, and mixed versions of the LSOIE dataset.

pute a percentage of repetitive tokens in T. Table 15 summarizes the results. From Table 15, we note that that the percentage of tokens in $|S \cap T|$ drops by 14% when using both *generation probability* and *coverage mechanisms*. This implies that the model copies fewer tokens from the source sentence, and introduces more tokens from the vocabulary, hence the improved ability to generate implicit facts. Furthermore, the repetitive tokens reduce by 23%. To that end, the results confirm that both *generation probability* and *coverage mechanisms* are effective in controlling repetitive tokens, and improving the models' ability to generate implicit facts.

Performance on augmented dataset. We trained the best model from Table 12 on paraphrased and mixed versions of the LSOIE dataset. After training, the model was evaluated on the original test set of the LSOIE dataset. We note that the performance of the model improves when trained on the mixed dataset. Table 16 summarises the results.

E Hyperparameters

Table 17 summarizes the hyperparameters for all the modules that were used in our experiments. These were determined over a number of initial experiments, and kept constant throughout all training runs conducted for this paper.

model block	number of layers
BiLSTM decoder	3
BiLSTM embedder	3
Discriminator	6
ELECTRA embedder	12
Feedback Transformer	6
GAT encoder	3
Transformer	6
hyperparameter	**value**
batch size	128
dropout rate	0.3
dependency tag embedding size	100
embedder/decoder hidden size	256
feedback transformer heads	8
feedback transformer hidden size	512
learning rate	0.0005
maximum gradient norm	2
weight decay λ	10^{-5}
PoS embedding size	100
transformer heads	6
transformer query/key/value size	50
transformer encoder hidden size	512
vocabulary size	30522

Table 17: The hyperparameters that were used throughout all our experiments.

F Generator's Workflow

Fig. 2 summarises the workflow in the generator module.

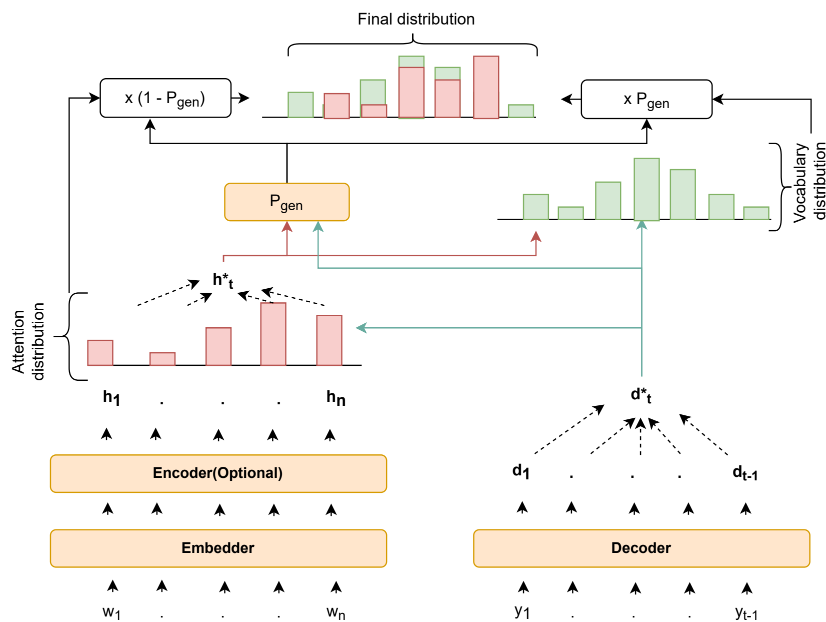

Figure 2: Illustration of the workflow in the generator module discussed in Sections 3.1–3.3. As the encoder is optional, the output of the embedder is used to compute the encoder context vector h_t^*, when it is not used. The decoder context vector d_t^* is computed by attending to the vector representations of the tokens in the entire generated tuple by timestep t. Both the encoder and decoder context vectors are used to compute the generation probability and the distribution over the entire vocabulary as discussed in Section 3.3.