Sentence-Incremental Neural Coreference Resolution

Matt Grenander  Shay B. Cohen  Mark Steedman
School of Informatics
University of Edinburgh
matt.grenander@ed.ac.uk
scohen@inf.ed.ac.uk,m.steedman@ed.ac.uk

Abstract

We propose a sentence-incremental neural coreference resolution system which incrementally builds clusters after marking mention boundaries in a shift-reduce method. The system is aimed at bridging two recent approaches at coreference resolution: (1) state-of-the-art non-incremental models that incur quadratic complexity in document length with high computational cost, and (2) memory network-based models which operate incrementally but do not generalize beyond pronouns. For comparison, we simulate an incremental setting by constraining non-incremental systems to form partial coreference chains before observing new sentences. In this setting, our system outperforms comparable state-of-the-art methods by 2 F1 on OntoNotes and 6.8 F1 on the CODI-CRAC 2021 corpus. In a conventional coreference setup, our system achieves 76.3 F1 on OntoNotes and 45.5 F1 on CODI-CRAC 2021, which is comparable to state-of-the-art baselines. We also analyze variations of our system and show that the degree of incrementality in the encoder has a surprisingly large effect on the resulting performance.

1 Introduction

Coreference Resolution (CR) is a task in which a system detects and resolves linguistic expressions that refer to the same entity. It is typically performed in two steps: in mention detection, the model predicts which expressions are referential, and in mention clustering, the model computes each mention’s antecedent. Many recently proposed systems follow a mention-pair formulation from Lee et al. (2017), in which all possible spans are ranked and then scored against each other. In particular, methods that augment this approach with large, pre-trained language models achieve state-of-the-art results (Joshi et al., 2019, 2020).

Despite impressive performance, these methods are computationally demanding. For a text with \( n \) tokens, they will score up to \( O(n^2) \) spans, followed by up to \( O(n^4) \) span comparisons. They also process documents non-incrementally, requiring access to the entire document before processing can begin. These properties present challenges when insufficient computational resources are available, or when the task setup is incremental, such as in dialogue (e.g. Khosla et al. 2021). From a cognitive perspective, these methods are also unappealing because research on “garden-path” effects show that humans resolve referring expressions incrementally (Altmann and Steedman, 1988).

These drawbacks have led to renewed interest in incremental coreference resolution systems, in which document tokens are processed sequentially. Some recent approaches use memory networks to track entities in differentiable memory cells (Liu et al., 2019; Toshniwal et al., 2020a). These models demonstrate proficiency at proper name and pronoun resolution (Webster et al., 2018). However, they seem unlikely to generalize to more complicated coreference tasks due to a strict interpretation of incrementality. Both Liu et al. (2019) and Toshniwal et al. (2020a) resolve mentions word-by-word, making coreference decisions possibly
before the full noun phrase has been observed. The approach is adequate for proper names and pronouns, but it may fail to distinguish entities who share the same phrase prefix. For example, in Figure 1, three mentions all begin with ‘Hong Kong’, though all belong to separate entities. In this case, it is difficult to see how a system using word-level predictions would resolve these mentions to different entities.

Motivated by this recent work, we propose a new system that processes a document incrementally at the sentence-level, creating and updating coreference clusters after each sentence is observed. The system addresses deficiencies in memory network-based approaches by delaying mention clustering decisions until the full mention has been observed. These goals are achieved through a novel mention detector based on shift-reduce parsing, which identifies mentions by marking left and right mention boundaries. Identified mention candidates are then passed to an online mention clustering model similar to Toshniwal et al. (2020b) and Xia et al. (2020). The model proposes a linear number of spans per sentence, reducing computational requirements and maintaining more cognitive plausibility compared to non-incremental methods.

In order to compare non-incremental and incremental systems on equal footing, we propose a new sentence-incremental evaluation setting. In this setting, systems receive sentences incrementally and must form partial coreference chains before observing the next sentence. This setting mimics human coreference processing more closely, and is a more suitable evaluation setting for downstream tasks in which full document access is generally not available (e.g. for dialogue (Andreas et al., 2020)).

Using the sentence-incremental setting, we demonstrate that our model outperforms comparable systems adapted from partly incremental methods (Xia et al., 2020) across two corpora, the OntoNotes dataset (Pradhan et al., 2012) and the recently released CODI-CRAC 2021 corpus (Khosla et al., 2021). Moreover, we show that in a conventional evaluation setting, where the model can access the entire document, our system retains close to state-of-the-art performance. However, the sentence-incremental setting is substantially outperformed by non-sentence-incremental systems. Analyzing the difference between these two settings reveals that the encoder is heavily dependent on how many sentences it can observe at a time. The analysis suggests better representations of the entities and their context may improve performance in the sentence-incremental setting. Nevertheless, our results provides new state-of-the-art baselines for sentence-incremental evaluation.

2 Related Work

Non-incremental mention-pair models have dominated the field in recent years, with many following the formulation presented by Lee et al. (2017). Several extensions have led to performance improvements, such as adding higher-order inference (Lee et al., 2018), and replacing the encoder with BERT and SpanBERT (Joshi et al., 2019, 2020). Extensions to this approach have looked at reformulating the problem as question-answering (Wu et al., 2020), simplifying span representations (Kirstain et al., 2021), and incorporating coherence signals from centering theory (Chai and Strube, 2022). Although our work is orthogonal to this line of research, we compare our system against this type of non-incremental model.

Toshniwal et al. (2020b) and Xia et al. (2020) adapt the non-incremental system of Joshi et al. (2020) so that mention clustering is performed incrementally. Their resulting models achieve similar performance to the original non-incremental one. However, in their formulation, document encoding, mention detection and certain clustering decisions still fully depend on Joshi et al. (2020). The resulting model still requires access to the full document in order to compute coreference chains. Yu et al. (2020b) similarly present an incremental mention clustering approach where mention detection is performed non-incrementally as Lee et al. (2017).

Memory network-based approaches identify co-referring expressions by writing and updating entities into cells within a fixed-length memory (Liu et al., 2019; Toshniwal et al., 2020a). These models demonstrate how fully incremental coreference systems can be achieved. However, the formulation operates on token-level predictions, and does not easily extend to either nested mentions or certain multi-token mentions (e.g. in Figure 1).

Cross-document coreference resolution (CDCR) requires systems to compute coreference chains across documents, raising scalability challenges as the number of documents increases. Given these challenges, incremental CDCR systems are crucial (Allaway et al., 2021; Logan IV et al., 2021) due to lower memory requirements. However, these
works are not directly comparable to ours since they assume gold mentions are provided as input.

Other, earlier, incremental coreference systems also often ignore or diminish the role of mention detection. For example, Webster and Curran (2014) use an external parser for mention detection, requiring an additional model. Klenner and Tuggener (2011) assume gold mentions as input.

Recently, Liu et al. (2022) propose a coreference resolution system based a seq2seq formulation using hidden variables. Although their focus is on adding structure to seq2seq models, their system can also be viewed as transition-based like ours.

Our incremental mention detector bears similarities to certain models for nested named-entity recognition (NER). In particular, Wang et al. (2018) present an incremental neural model for nested NER based on a shift-reduce algorithm. Their deduction rules differ greatly from ours as they model mention spans using complete binary trees, and are aimed at NER rather than mention detection.

Recent work has also explored incremental transformer architectures (Katharopoulos et al., 2020; Kasai et al., 2021), and adapting these architectures to NLU tasks (though not coreference resolution) (Madureira and Schlangen, 2020; Kahardipraja et al., 2021). In this work, we focus on the simpler sentence-incremental setting, believing it to be sufficient for downstream tasks.

3 Method

Given a document, the goal is to output a set of clusters \( C = \{ C_1, \ldots, C_K \} \), where mentions within each cluster are co-referring. We assume mentions may be nested but otherwise do not overlap. This assumption allows us to model mentions using a method analogous to shift-reduce, where shifting corresponds to either incrementing the buffer index or marking a left mention boundary, and reducing corresponds to marking a right boundary and resolving the mention to an entity cluster.

3.1 Shift-Reduce Framework

The main idea is to mark mention boundaries using Push, Pop or Peek actions, or to pass over a non-boundary token with the Advance action. After Pop or Peek actions, a mention candidate is created using the current top-of-stack and buffer elements. The resulting mention candidate is then either resolved to an existing cluster or initialized as a new entity cluster.

<table>
<thead>
<tr>
<th>Initial</th>
<th>[∅, 0, ∅, ∅]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final</td>
<td>[∅, n, A, C]</td>
</tr>
<tr>
<td>Push</td>
<td>([S, i, A, C]\rightarrow [S[i, w_i, A]Push, C])</td>
</tr>
<tr>
<td>Advance</td>
<td>([S, i, A, C]\rightarrow [S[i + 1, A]Advance, C])</td>
</tr>
<tr>
<td>Pop</td>
<td>([S, i, A, C]\rightarrow [S[i, A]Pop, C</td>
</tr>
<tr>
<td>Peek</td>
<td>([S, i, A, C]\rightarrow [S[i, A]Peek, C</td>
</tr>
</tbody>
</table>

Figure 2: Deduction rules for our coreference resolver. \([S, i, A, C]\) denotes the stack \( S \), buffer index \( i \), action history \( A \), and cluster set \( C \). The Coref function indicates that span \((v, w_i)\) is clustered and added to \( C \).

We represent the state as \([S, i, A, C]\), where \( S \) is the stack, \( i \) is the buffer index, \( A \) is the action history and \( C \) is the current set of clusters. At each time step, one of four actions is taken:

- **Push**: Place the word at buffer index \( i \) on top of the stack, marking a left mention boundary.
- **Advance**: Move the buffer index forward.
- **Pop**: Remove the top element from \( S \) and create a mention candidate using this element and the current buffer element. Score the candidate against existing clusters and resolve it (or create a new cluster).
- **Peek**: Create a mention candidate using the top element on the stack and the current buffer element. Score the candidate against existing clusters and resolve it (or create a new cluster).

The Peek action does not alter the stack but is otherwise identical to Pop. This action is critical for detecting mentions sharing a left boundary.

Several hard action constraints ensure that only valid actions are taken and the final state is always reached. For example, Push can only be called once per token, or else the model would be marking the left boundary multiple times. The full list of constraints is described in the appendix.
where \( f_M \) is a two-layer neural network, and \( w_{a_t} \) is a column vector selecting action \( a_t \).

The action history at \( t \) is encoded with an LSTM over previous action embeddings:

\[
a_t = \text{LSTM}[a_0, \ldots, a_{t-1}].
\]

Then, the parser state is represented by the concatenation of buffer, stack, action history and additional mention features \( \phi_M \):

\[
p_t = [b_t; s_t; a_t; \phi_M(b_t, s_t)],
\]

where \( \phi_M \) denotes learnable embeddings corresponding to useful mention features such as span width and document genre. For span width, we use embeddings measuring the distance from the top of the stack to the current buffer token (i.e. \( i - s_L \)), or 0 if the stack is empty.

### 3.2.2 Mention Clustering Model

The mention clustering is similar to previous online clustering methods (Toshniwal et al., 2020b; Xia et al., 2020; Webster and Curran, 2014), though we take care to avoid dependence on non-incremental pre-trained language models which have already been fine-tuned to this task.

Given a mention candidate’s span representation \( v \), we score \( v \) against the existing entity cluster representations \( m_1, \ldots, m_k \):

\[
\begin{align*}
s_C(v) &= [f_C(m_1, v), \ldots, f_C(m_k, v), \alpha] \\
f_C(m_i, v) &= \text{MLP}([v, m_i, v \odot m_i, \phi_C(v, m_i)]) \\
i^* &= \arg \max_{i \in \{1, \ldots, k+1\}} s_C(v)
\end{align*}
\]

where \( f_C \) is two-layer neural network, \( \alpha \) is a threshold value for creating a new cluster, \( v \odot m_i \) is the element-wise product and \( \phi_C \) encodes useful features between \( v \) and \( m_i \); the number of entities in \( m_i \), mention distance between \( v \) and \( m_i \), the previous coreference action and document genre.

If the scores between \( v \) and all cluster representations \( m_1, \ldots, m_k \) are below some threshold value \( \alpha \) (i.e. \( i^* = k + 1 \)), we initialize a new entity cluster with \( v \). Otherwise, we update the cluster representation \( m_{i^*} \) via a weighted average using the number of entities represented by \( m_{i^*} \):

\[
m_{i^*} \leftarrow \beta \cdot m_{i^*} + (1 - \beta) \cdot v,
\]

where \( \beta = \frac{|m_{i^*}|}{|m_{i^*}| + 1} \) is the weighting term.
3.2.3 Training

Training is done via teaching forcing. At each time step, the model predicts the gold action given the present state. The state is then updated using the gold action. At each step, we compute mention detection loss $L_M$ and coreference loss $L_C$.

The mention detection loss $L_M$ is calculated using the cross-entropy between the predicted mention detection action and gold action $a_t^* \in V(S, i, A, C)$:

$$L_M = - \sum_t \log p_M(a_t^* \mid p_t),$$

where $t$ sums over time steps across all documents.

Similarly, the coreference loss $L_C$ is defined by the cross entropy between the highest-scoring coreference action and the gold coreference decision $z_{j^*} \in A_k$:

$$L_C = - \sum_j \log p_C(z_{j^*} \mid m_j),$$

where $j$ sums over mentions across all documents. The entire network is then trained to optimize the sum of the two losses, $L_M + L_C$. During inference, we predict actions using greedy decoding, updating the state solely with predicted actions.

Figure 4 presents a summary of the various components and the overall algorithm.

4 Experiments

4.1 Datasets

We train and evaluate our system on the OntoNotes 5.0 dataset (Weischedel et al., 2013), using the same setup described in the CoNLL-2012 Shared Task (Pradhan et al., 2012). OntoNotes includes 7 document genres and does not restrict mention token length; annotations cover pronouns, noun phrases and heads of verb phrases. We evaluate using the MUC (Vilain et al., 1995), B$^3$ (Bagga and Baldwin, 1998) and CEAF$\phi_4$ (Luo, 2005) metrics and their average (the CoNLL score), using the official CoNLL-2012 scorer.

We also test models on the recently released CODI-CRAC 2021 corpus (Khosla et al., 2021). This dataset annotates coreference (and other anaphora-related tasks) for 134 documents across 4 separate dialogue corpora (Light, Urbanek et al. 2019, AMI, Carletta 2006, Persuasion, Wang et al. 2019 and Switchboard, Godfrey et al. 1992). The dataset suits incremental systems well since dialogue can be naturally presented as incremental utterances. Given the small dataset size, we use it for evaluation only, using models trained on OntoNotes. Since OntoNotes marks document genre (which systems often use as a feature), we associate CODI-CRAC documents with OntoNotes’ ‘telephone conversation’ genre, since it is the most similar. We remove singleton clusters due to lack of annotation in the training set. We again evaluate using MUC, B$^3$ and CEAF$\phi_4$, using the official Universal Anaphora scorer (Yu et al., 2022).

4.2 Model Components

4.2.1 Document Encoder

Recent models on coreference resolution often use SpanBERT (Joshi et al., 2020) for word embeddings (Wu et al. (2020); Toshniwal et al. (2020b); Xia et al. (2020); Xu and Choi (2020), among others), since Joshi et al. (2020) demonstrate Span-
Figure 4: A summary of the overall algorithm. After document encoding, the mention detector predicts transition actions PUSH, POP, PEEK or ADVANCE using the parser state $p_t$. If a mention is predicted, the coreference resolver then clusters it to an existing cluster representation or creates a new cluster. Clustering a mention implies a coreference relation with mentions in the cluster. The steps can all be performed incrementally, assuming the document encoder is also incremental.

BERT’s proficiency for entity-related tasks such as coreference resolution. However, SpanBERT is unsuitable for incremental applications because it expects all its input simultaneously and cannot partially process text while waiting for future input. Instead, we turn to XLNet (Yang et al., 2019), which extends the earlier Transformer-XL (Dai et al., 2019). XLNet differs from typical pre-trained language models as it can efficiently cache and reuse its previous outputs. The caching mechanism allows for recurrent computation to be performed efficiently. Cached outputs provide a context to the current sentence being processed.

We experiment using XLNet in two settings: in the Sentence-Incremental (Sent-Inc) setting, each sentence is processed sequentially, and partial coreference clusters are computed before the next sentence is observed. After each sentence is processed, we accumulate XLNet’s outputs (up to a cutoff point) and reuse them when processing the next sentence. We limit the number of cached tokens so that the cached and ‘active’ tokens do not exceed 512, so that our work remains comparable to other recent works. Although the mention detector is token-incremental and the mention clustering component is span-incremental, the document encoder is sentence-incremental, so overall we describe the system as sentence-incremental.

In the Part-Incremental (Part-Inc) setting, we allow XLNet to access multiple sentences simultaneously. We experiment both with and without the cache mechanism, using up to a total of 512 tokens at a time. This setting is comparable to experiments in Xia et al. (2020) and Toshniwal et al. (2020b), where document encoding is also non-incremental. In our case, both mention detection and mention clustering components remain incremental as in the Sentence-Incremental setting. In this way, we can isolate the effect of sentence-incrementality on the document encoder (XLNet).

4.2.2 Span Representation

We use a similar span representation to Lee et al. (2017): for a span $(i,j)$, we concatenate word embeddings $(x_i, x_j)$, an attention-weighted average $\bar{x}$ and learnable embeddings for span width and speaker ID (the speaker for $(i,j)$). We use 20-dimensional learned embeddings for all features (span width, speaker ID, document genre, action history, mention distance and number of entities in each cluster).

4.2.3 Training

We use Adam to train task-specific parameters, and AdamW for XLNet’s parameters (Kingma and Ba, 2015; Loshchilov and Hutter, 2019). The gradient is accumulated across one document before updating model weights. We use a learning rate scheduler with a linear decay, and additionally warmup SpanBERT’s parameters for the first 10% update steps. For the mention detector, we balance the loss weights based on the frequency of each action in the training set. This step is important because most tokens do not correspond to mention boundaries, meaning the ADVANCE action is by far the most prevalent in the training set.

Training converges within 15 epochs. The model is implemented in PyTorch (Paszke et al., 2019). A
complete list of hyperparameters is included in the appendix.

4.3 Comparisons

We compare against several recent works with varying degrees of incrementality. Table 1 summarizes their differences in incrementality compared to ours, as well as the span complexity. Joshi et al. (2020) is a non-incremental formulation: it adopts the *end-to-end* and *coarse-to-fine* formulations from Lee et al. (2017) and Lee et al. (2018), replacing the LSTM encoder with their novel SpanBERT architecture.

<table>
<thead>
<tr>
<th>Model</th>
<th>Incremental Components</th>
<th>Span Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpanBERT</td>
<td>None</td>
<td>(O(n^4))</td>
</tr>
<tr>
<td>longdoc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICoref</td>
<td>Mention Clustering</td>
<td>(O(n^2m))</td>
</tr>
<tr>
<td></td>
<td>Mention Clustering</td>
<td>(O(n^2m))</td>
</tr>
<tr>
<td>Part-Inc (Ours)</td>
<td>Mention Detection +</td>
<td>(O(nm))</td>
</tr>
<tr>
<td></td>
<td>Mention Clustering</td>
<td></td>
</tr>
<tr>
<td>ICoref-inc</td>
<td>All</td>
<td>(O(n^2m))</td>
</tr>
<tr>
<td>Sent-Inc (Ours)</td>
<td>All</td>
<td>(O(nm))</td>
</tr>
</tbody>
</table>

Table 1: The list of systems we compare, alongside their incrementality (on a sentence-level) and span complexity. ‘All Components’ means document encoding, mention detection and mention clustering. \(n\) is the number of tokens and \(m\) is the number of entities.

The longdoc (Toshniwal et al., 2020b) and ICoref (Xia et al., 2020) systems adapt Joshi et al. (2020) so that mention clustering is done incrementally. However, both models avoid modifying the non-incremental document encoding and mention detection steps from Joshi et al. (2020), and the resulting systems are only partly incremental. Since Toshniwal et al. (2020b) and Xia et al. (2020) only experiment with SpanBERT-large, we re-train their implementations with SpanBERT-base to fairly compare against our own systems.

Xia et al. (2020) also provide a truly sentence-incremental version of their system, which we call ICoref-inc\(^3\). This version is trained by encoding tokens and proposing mentions sentence-by-sentence, independently processing each sentence as it is observed while maintaining entity clusters across sentences. Since ICoref-inc is fully sentence-incremental, it provides the fairest comparison to our own Sentence-Incremental setting. Having more incremental components results in increased difficulty on the coreference task, as the system must rely on partial information when making clustering decisions.

We do not compare against Liu et al. (2019) and Toshniwal et al. (2020a)’s token-incremental models. Besides being generally unsuitable for span-based coreference, they also do not handle nested mentions. Roughly 11% of OntoNotes’ mentions are nested, meaning that training these systems on OntoNotes is infeasible.

4.3.1 Span Complexity

Table 1 also compares the *span complexity* between systems, in terms of how many spans must be scored and compared. This comparison is analytic and not runtime-based, and so ignores handcrafted memory-saving techniques such as eviction and span pruning. Joshi et al. (2020) score all possible spans and compare them pairwise, meaning their system runs in \(O(n^4)\), where \(n\) is the number of tokens. Toshniwal et al. (2020b) and Xia et al. (2020) reduce the complexity to \(O(n^2m)\), where \(m\) is the number of entities, by incrementally clustering mentions. Finally, we claim our systems’ span complexity is \(O(nm)\). Our mention detector proposes \(O(n)\) spans, as we can show each action is linearly bounded in the number of tokens. Our reduced complexity speaks to its increased cognitive plausibility compared to the part- and non-incremental systems, which consider a quadratic number of spans.

Note that the runtime is not comparable because non-incremental methods process the entire document in parallel, whereas ours is not parallelizable and therefore slower. We also note that this comparison does not have any bearing on memory requirements, since Toshniwal et al. (2020b) and Xia et al. (2020) both maintain constant memory through eviction strategies.

5 Results

5.1 OntoNotes

The main results for OntoNotes are shown in Table 2. First, SpanBERT (Joshi et al., 2020), being non-incremental, unsurprisingly outperforms other systems, both part and sentence incremental.

Within partly incremental systems, the ICoref model (Xia et al., 2020) performs best, below SpanBERT by 0.4 F1. Our Part-Inc model performs comparably to longdoc (Toshniwal et al., 2020b), only trailing ICoref by 0.7 F1 points.

\(^3\)Specifically, this system is the “Train 1-sentence / Inference 1-sentence” model from Xia et al. (2020)’s Table 4.
Table 2: Main results on the OntoNotes 5.0 test set with the CoNLL 2012 Shared Task metrics and the average F1 (the CoNLL F1 score). The ‘SI’ column denotes the sentence-incrementality of each system, summarizing details in Table 1. The top four systems are not directly comparable to ours, since they train with a ‘large’ encoder (either SpanBERT or Longformer (Beltagy et al., 2020)). Note that scores for Xia et al. (2020) and Toshniwal et al. (2020b) differ from their reported results because we re-train them with SpanBERT-base instead of large.

Table 3: Main results for the CODI-CRAC 2021 corpus. All scores denote the CoNLL F1 score (average of MUC, B^3 and CEAF_{φ}). Here, SpanB=SpanBERT, and IC-inc=ICoref-inc.

The advantages of our method are more evident in the sentence-incremental evaluation. Since ICoref-inc relies on SpanBERT to encode tokens and score mentions, its performance suffers considerably when evaluated in the sentence-incremental setting. In contrast, the Sent-Inc model effectively uses the history of previous processed sentences and outperforms ICoref-inc by 2 F1 points. Still, both systems suffer considerably when compared to their part-incremental counterparts: ICoref drops by 9 F1 points and our model by 6.3 F1. In Section 6, we explore the main causes of this drop.

5.2 CODI-CRAC

The results on the CODI-CRAC corpus are shown in Table 3. We observe many of the same trends as in OntoNotes: the non-incremental SpanBERT again surpasses other models, achieving 2.9 F1 higher than ICoref.

Within partly incremental systems, our Part-Inc system trails ICoref by 1.2 F1. We omit the long-doc results from this table, after finding its performance surprisingly plummets when evaluated on CODI-CRAC. On all subsets, it scores below 2 F1, indicating issues with model transfer. Other works have explored this topic in depth (Toshniwal et al., 2021), and we do not investigate further here.

In the Sentence Incremental setting, although our Sent-Inc model again outperforms Xia et al. (2020)’s ICoref-inc, the performance difference is much larger here: 7 F1 compared to 2 F1 in OntoNotes. The gap between the Sent-Inc and Part-Inc is also much smaller: only 2.5 F1 points compared to 6.3 F1 on OntoNotes. The difference in performances between the two datasets may suggest our model is better suited to the inherent incrementality in a dialogue setting.

6 Analysis

The dramatic performance gap between the Sent-Inc and Part-Inc settings may be surprising. Since coreference resolution is primarily processed incrementally by humans, why does access to future tokens affect the Sent-Inc model so heavily?

To investigate this issue deeper, we design additional k-Sentence-Incremental settings. In each setting, the system accesses k sentences ($S_1,\ldots,S_k$) at a time as active input, and $512 - \sum_{i=1}^{k} |S_k|$ tokens as memory. In each setting, the model observes the same number of tokens (512), but varies the amount of active input vs. memory. The mention detection and mention clustering steps remain the same and are still incremental; the only change is in the encoder.
Varying $k$ in this way allows us to test the effect of more or less incrementality on the system. When $k = 1$, we recover the original Sent-Inc model. When $k$ is large enough (in practice, 24), we get the Part-Inc model. For each $k \in \{1, 4, 8, 12, 16, 20, 24\}$, we fully train the corresponding model on OntoNotes as described in Section 4, and evaluate on the dev set.

![Figure 5: The CoNLL performance (average of MUC, $B^3$ and CEAF$_{\phi_4}$) of each $k$-Sentence-Incremental model on the OntoNotes dev set.](image)

The results are shown in Figure 5. There are a few notable characteristics. The first is that as $k$ increases, we see a much more dramatic lift when $k$ is small (e.g. moving from 1 to 4 sentences) compared to when $k$ is large. This effect corresponds to the intuition that coreferring expressions are usually close to their antecedent. The more coreference chains the model can observe simultaneously, the better it is at resolving them.

The second noteworthy trend is that increasing $k$ improves recall (9.7%) far more than precision (3.8%). Although not shown here, we observe this trend across all three metrics within the CoNLL score ($MUC$, $B^3$ and CEAF$_{\phi_4}$). The result means that finding and resolving true coreference links (i.e. reducing false negatives) is a far more serious obstacle for the Sent-Inc model than for Part-Inc. Since the only difference in these models is how many embeddings are cached, the result suggests caching or not caching embeddings plays a large role in finding and correctly resolving mentions.

7 Future Work

A major goal would be to elevate incremental coreference resolvers to the same level as non-incremental ones. As we showed in Section 6, a large part of the performance difference occurs because the XLNet encoder does not effectively handle incremental input. A simple strategy therefore may be to swap out that encoder for a more powerful one. However, few pre-trained language models targeted at NLU tasks are naturally incremental. One candidate is GPT-J (Wang and Komatsuzaki, 2021) but its size is prohibitively large.

Other ways to bridge this gap may come from improving the mention detection component. A similar task is nested named entity recognition, where the system must identify named entity boundaries and coarsely classify them. Recent nested NER systems such as Katiyar and Cardie (2018) or Yu et al. (2020a) may provide directions for improving mention detection in our incremental formulation.

8 Conclusion

We propose a sentence-incremental coreference resolution model using a shift-reduce formulation. The model delays mention clustering until the full span has been observed, alleviating a key flaw with previous incremental systems. It efficiently processes text, and avoids scoring a quadratic number of spans during mention detection.

In a sentence-incremental setting, our method outperforms strong baselines adapted from state-of-the-art systems. When access to the full document is allowed, the proposed system achieves similar performance to state-of-the-art methods while maintaining a higher level of incrementality. We investigate why this relaxation has such a dramatic effect, finding that the document encoder does not make effective use of its memory cache.

Our sentence-incremental results suggest an important point: non-incremental methods are not effective tools when they must be used incrementally. Creating new, incremental coreference resolvers that perform at the same level as non-incremental ones is a challenging but meaningful goal. Achieving this result would make a significant impact in downstream applications where text is received incrementally, such as dialogue systems or conversational question answering (e.g. Andreas et al. 2020; Martin et al. 2020). Our proposal demonstrates an important step towards highly effective, incremental coreference resolution systems.
Limitations

In this work, we have experimented with training neural networks on OntoNotes and evaluating on other datasets (the CODI-CRAC 2021 corpus). Several recently published papers have explored the difficulties of coreference resolution model transfer (Subramanian and Roth, 2019; Xia and Van Durme, 2021; Toshniwal et al., 2021; Yuan et al., 2022). These works have noted generalization problems with models trained on OntoNotes, with one particular difficulty that OntoNotes does not annotate singleton clusters, or ‘markable’ mentions.

Several recent works have addressed generalization issues by training on additional resources (Subramanian and Roth, 2019; Xia and Van Durme, 2021; Toshniwal et al., 2021; Yuan et al., 2022). In particular, Toshniwal et al. (2021) augment OntoNotes with pseudo-singletons: a fully trained coreference resolver scores all spans in the text, and the top-scoring spans outside of gold mentions are regarded as singletons. The authors show that adding pseudo-singletons to the OntoNotes training data improves (1) coreference resolution metrics on OntoNotes and (2) generalization capabilities. Pseudo-singletons are especially helpful for transfer learning from OntoNotes because other coreference datasets will often annotate singletons.

In our experiments, we attempted to use their published pseudo-singletons, but faced difficulties because the pseudo-singletons do not respect the “non-crossing bracketing” structure in OntoNotes, and overlap arbitrarily (not only nested). Our mention detector assumes mentions may be nested but otherwise do not overlap, and determining which pseudo-singletons to filter out without redoing the whole experiment was not feasible. We leave the problem for future work, but we agree that models trained on OntoNotes without heuristically added singletons are limited in their generalization capabilities.

Our experiments have focused on the base versions of XLNet and SpanBERT due to resource requirements. Training our models requires a GPU with 16 GB of memory; we used NVIDIA Tesla V100 16 GB cards. Greater memory efficiency could be achieved by extending the memory to be more dynamic. In the current system, entities are added but never evicted. Ideally, when a referent is no longer relevant to the context, it should be detected and removed. This concept has been explored with memory network-based systems (Liu et al., 2019; Toshniwal et al., 2020a), and also recent partly incremental systems (Xia et al., 2020; Toshniwal et al., 2020b). Memory-based systems using dynamic eviction strategies appear in other NLP tasks as well, such as semantic parsing (Jain and Lapata, 2021).

Ethical Considerations

NLP systems such as ours must be employed with special consideration that they do not demonstrate unwanted patterns towards protected groups. Previously, systems have been shown to learn harmful associations from training corpora. For example, Bolukbasi et al. (2016) show that word embeddings trained on a news corpus exhibit gender stereotypes, such as associating “receptionist” with “female”.

Coreference resolution systems in particular may learn gender biases, and methods exist to counter this effect (Rudinger et al., 2018; Zhao et al., 2018). Our system is trained on OntoNotes, which include data from a diverse set of sources such as Wall Street Journal articles, telephone conversations and Bible passages. Our final trained model may therefore reflect undesirable content from these texts.

Any off-the-shelf deployment of our model should first check whether the model is harmful towards any protected group, and appropriate mitigation should be taken. For example, evaluating on specialized datasets such as Webster et al. (2018) may indicate whether the system unfairly predicts certain labels based on gender.

Acknowledgements

We would like to thank Ratish Puduppully and Miloš Stanojević as well as the anonymous reviewers for their helpful discussions and feedback. This work used the Cirrus UK National Tier-2 HPC Service at EPCC (http://www.cirrus.ac.uk) funded by the University of Edinburgh and EPSRC (EP/P020267/1), as well as the Baskerville Tier 2 HPC service (https://www.baskerville.ac.uk/). Baskerville was funded by the EPSRC and UKRI through the World Class Labs scheme (EP/T022221/1) and the Digital Research Infrastructure programme (EP/W032244/1) and is operated by Advanced Research Computing at the University of Birmingham. The work was partly supported by ERC Advanced Fellowship GA 742137 SEMANTAX and the University of Edinburgh Huawei Laboratory.
References


Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. 2018. Gender bias in


A Action Constraints

To ensure the final state is always reached, it is necessary to enforce a set of rules during mention detection:

1. ADVANCE can only be called on the final token if the stack is empty.

2. POP and PEEK can only be called if the stack is non-empty.

3. PUSH can only be called once per token, ensuring that left boundaries are only marked once.

4. PUSH cannot directly follow POP or PEEK. Allowing this action sequence would either admit multiple paths to the same mention or non-nested overlapping mentions.

5. POP cannot directly follow PEEK, or else the same mention would be proposed twice.

6. PEEK cannot be called on the final token. This action would imply the stack is non-empty on the final token, and that POP must be called.

B k-Sentence-Incremental Mention Detection

We repeat the experiment in Section 6 for mention detection. For each k-Sentence-Incremental setting, we evaluate on the dev set and record the mention detection recall, precision and F1.

The results are shown in Figure 6. Certain trends remain the same as for the CoNLL score, namely that performance rises more when k is small compared to when it is large. However, we do not see the same dramatic difference in recall between k = 1 to k = 24 settings as in Section 6. Here, the difference in recall between the two settings is around 3%, whereas in Figure 5 it is 9.7%.

Overall, the reduced severity between k = 1 and k = 24 settings compared to Figure 5 most likely indicate that XLNet’s caching deficiencies affect mention clustering (particularly false negatives) more seriously than mention detection.

C Partitioning Document Clusters

We explore the deficiency in the previous section further, guided by the hypothesis that XLNet relies on active inputs and cannot effectively use its memory. We partition each document into segments of k sentences, and call the number of sentences in...
each segment the ‘partition size’. Within each segment, we maintain the original coreference links. However, we remove the coreference links between segments. For each \( k \)-Sentence-Incremental model, we similarly partition their coreference predictions, and evaluate against the partitioned gold labels on the OntoNotes dev set.

Each segment therefore is independent from the other, and we can measure how reliant the model is on its active inputs by observing performance change across partition sizes. For example, when the partition size is 1, the only coreference links are intra-sentential ones. In this case, models are only evaluated on their intra-sentential coreference resolution ability. When the partition size is large, the models are evaluated on the documents’ original coreference chains. Since the previous experiments demonstrated shifts in incrementality heavily affected recall, we measure the mention detection recall and CoNLL recall score.

The results are shown in Figure 7. All models do well when the partition size is one, reflecting the fact that intra-sentential coreference is generally simpler than distantly linked mentions. As the partition size increases, model performance decreases as the coreference chains become more spread apart and raise the task difficulty. Crucially, we notice an upward performance bump whenever the partition size matches the \( k \)-Sentence-Incremental setting, for both coreference performance and mention detection. When \( k \) matches the partition size, the model observes coreference chains that are always within the active input window. This performance bump therefore indicates XLNet is much better at mention detection and coreference when the coreference chain occurs within its active inputs. Performance suffers whenever the model must rely more on its memory (whenever \( k \) is not equal to the partition size). In particular, these results suggest that more powerful pre-trained language models, in particular ones that can take better advantage of cached representations, may be more successful at incremental coreference resolution.

D Speaker Embeddings

The ICoref-inc model from Xia et al. (2020) is an important comparison point as the only baseline in the sentence-incremental setting. While ICoref-inc does not rely on speaker embeddings, our own models (both Part-Inc and Sent-Inc) do. Given the important role of speaker identity in a dialogue setting, it is useful to know the effect of removing these embeddings in our models.

We compare the Sent-Inc model with and without speaker embeddings in Table 4 for OntoNotes, and Table 5 for CODI-CRAC. We find that speaker embeddings play little to no role in coreference performance. In OntoNotes, removing speaker
embeddings improves CoNLL F1 by 0.1, and in CODI-CRAC, it decreases performance by 0.2 F1. In both cases, the results are unlikely to be statistically significant. The finding indicates that Sent-Inc’s advantage over ICRef-inc is not simply due to feature selection but a true modelling advantage. It also suggests that further performance improvements are possible if speaker identity can be better represented, since Sent-Inc effectively ignores the speaker embeddings. One possibility, from Wu et al. (2020), is to preprocess the text with speaker tags directly included in the input, rather than including it separately. This way, the document encoder directly learns how to handle speakers, instead of relying on a separate embedding in downstream classifiers.

### E XLNet in Non-Incremental Baselines

Choosing XLNet as the document encoder is motivated by the fact that XLNet can efficiently cache and reuse input, making it suitable for incremental processing. However, XLNet can also be used non-incrementally in the same way as SpanBERT. In particular, we can train Joshi et al. (2020)’s coreference system using XLNet instead of SpanBERT. This experiment allows us to compare how the choice of pre-trained language model affects performance.

Table 6 shows results of training Joshi et al. (2020) with an XLNet encoder instead of SpanBERT on the OntoNotes dev set. XLNet significantly underperforms compared to SpanBERT, scoring almost 7 CoNLL F1 points lower. Surprisingly, XLNet is an effective document encoder for our Part-Incremental formulation (achieving 76.3 F1 on the OntoNotes test set), but ineffective when used in Joshi et al. (2020)’s non-incremental setup. We do not attempt swapping the fine-tuned XLNet into Toshniwal et al. (2020b) or Xia et al. (2020) as it seems unlikely to yield useful results.

### F Hyperparameters and Other Model Details

The main hyperparameters are listed in Table 7. The bottom four rows refer to the maximum number of learned embeddings we use for each feature. Additionally:

- The top performing Part-Inc model uses 20 sentences as active input, with the remainder as memory (up to 512 tokens total).
- During training, the Sent-Inc model accumulates gradients after every 32 sentences to ensure that the memory used does not exceed capacity.

Our implementation is based off of Xu and Choi (2020)’s codebase. We find their model hyperparameters are already extremely well-tuned, and so we do not explore further hyperparameter tuning for these cases. Regarding new hyperparameters introduced in this work, we follow previous work in choosing sensible values. For example, the StackLSTM and Action History LSTM hidden sizes follow Dyer et al. (2015)’s recommendations.

We train all models using NVIDIA Tesla V100 16 GB cards on an HPC cluster. Training convergence takes approximately 24 hours. Both Sent-Inc and Part-Inc models contain around 140 million parameters.

### G Dataset Details

For all datasets, we follow standard preprocessing steps such as tokenization, mapping subword units to token IDs, and adding segment boundary tokens (such as [CLS] and [SEP]). Since our algorithms rely on teacher forcing, we compute gold actions for both mention detection and mention clustering steps.

The English portion of OntoNotes 5.0 contains 3493 documents, divided into 2802/343/348
Table 6: Results from non-incremental methods on the OntoNotes dev set with different pretrained language models. e2e-coref refers to the non-incremental formulation from Lee et al. (2018), which is adapted in Joshi et al. (2020).

<table>
<thead>
<tr>
<th>Model</th>
<th>MUC</th>
<th>B$^3$</th>
<th>CEAF$_{\phi_4}$</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rec.</td>
<td>Prec.</td>
<td>F1</td>
<td>Rec.</td>
</tr>
<tr>
<td>SpanBERT+e2e-coref</td>
<td>82.5</td>
<td>84.3</td>
<td>83.4</td>
<td>75.8</td>
</tr>
<tr>
<td>XLNet+e2e-coref</td>
<td>70.3</td>
<td>80.8</td>
<td>75.2</td>
<td>63.7</td>
</tr>
</tbody>
</table>

Table 7: Hyperparameters used during training.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder Learning Rate</td>
<td>2e-5</td>
</tr>
<tr>
<td>Task Learning Rate</td>
<td>1e-4</td>
</tr>
<tr>
<td>Adam Eps</td>
<td>1e-6</td>
</tr>
<tr>
<td>Adam Weight Decay</td>
<td>1e-2</td>
</tr>
<tr>
<td>Gradient Clipping Norm</td>
<td>1</td>
</tr>
<tr>
<td>Dropout Rate</td>
<td>0.3</td>
</tr>
<tr>
<td>StackLSTM Hidden Size</td>
<td>200</td>
</tr>
<tr>
<td>Action History Hidden Size</td>
<td>30</td>
</tr>
<tr>
<td>$f_M$ Hidden Size</td>
<td>1000</td>
</tr>
<tr>
<td>$f_C$ Hidden Size</td>
<td>3000</td>
</tr>
<tr>
<td>New Cell Threshold ($\alpha$)</td>
<td>0.0</td>
</tr>
<tr>
<td>$\phi_C$ Max Entity Count</td>
<td>10</td>
</tr>
<tr>
<td>$\phi_C$ Max Mention Distance</td>
<td>10</td>
</tr>
<tr>
<td>$\phi_M$ Max Span Width</td>
<td>30</td>
</tr>
<tr>
<td>Max Speaker Number</td>
<td>20</td>
</tr>
</tbody>
</table>

H Evaluation Metrics

As mentioned in Section 4, we evaluate using the MUC, $B^3$ and CEAF$_{\phi_4}$ metrics, together with their average — commonly referred to as the CoNLL score after its usage in the CoNLL 2012 Shared Task (Pradhan et al., 2012).

None of the metrics have a trivial definition. Each metric measures different aspects of the predicted coreference chains. We do not provide a full definition of each metric here, but note that MUC corresponds to a link-based measure, $B^3$ is a mention-based measure, and CEAF$_{\phi_4}$ is an entity-based measure. We refer the reader to the CoNLL 2012 Shared Task paper (Pradhan et al., 2012) for an overview of all three metrics and to the original papers for a full description (Vilain et al., 1995; Bagga and Baldwin, 1998; Luo, 2005).

The CODI-CRAC 2021 corpus contains 134 English documents, split into 60 documents for validation and 74 documents for testing. Since we use the corpus solely for evaluation, we only use the test set. The corpus is available at https://codalab.lisn.upsaclay.fr/competitions/614, although a license from the Linguistic Data Consortium (LDC) is needed for the Switchboard portion. The official scorer can be found at: https://github.com/juntaoy/universal-anaphora-scorer. As detailed in Section 4.1, we remove singletons due to lack of training data.