Using Named Entity Recognition to Supplement The Ocean Cleanup's Global Beached Plastics Dataset

Stanford CS224N Custom Project

Mitty Yu
M.S. ICME Data Science
Stanford University
mittyu@stanford.edu

Stephen Peng
M.S. ICME General
Stanford University
stephen.peng@stanford.edu

Joe Jamison
M.S. Statistics Data Science
Stanford University
jamisonj@stanford.edu

Abstract

This project seeks to take advantage of recently emerging NLP capabilities and the proliferation of social media and internet news outlet text data to assist the global plastic modeling and cleanup efforts of The Ocean Cleanup. For this project, we built a Named Entity Extraction (NER) tool that, when provided unstructured text data about beach cleanup efforts from various internet sources, can identify and extract key data points from the text and transform them into a tabular format that is usable by The Ocean Cleanup for downstream analysis tasks. We experimented with several NER model architectures, fine-tuning them to perform on our task. Our best model is able to identify key data types for trash collection with an F1 score of 94.75, which is comparable to existing state-of-the-art NER model performance. This result gives us confidence that our tool can be used to reliably extract high-quality data from raw internet text for use by The Ocean Cleanup.

1 Key information to include

- **External collaborators:** Bruno Sainte-Rose (bruno.sainte-rose@theoceancleanup.com) framed the original research problem statement and The Ocean Cleanup provided the specifications for our data extraction target, but they did not contribute to our project work.
- **External mentor:** None
- **Sharing project:** CME 291 (ICME Xplore Research) - the focus for ICME Xplore is more functional, intended to provide an operationalizable web-scraping tool and model pipeline to collect new data, while the focus for CS224N is more investigative in nature (including all experimentation, performance analysis, relevant literature research, etc.)

2 Introduction

The ultimate goal of our project was to produce net-new, high-quality data on beach cleanup efforts for usage by The Ocean Cleanup. We achieved this goal by modifying the Stanza NER model architecture and fine-tuning it to reliably extract relevant information (e.g., date, location, type and quantity of trash) from natural language texts about beach cleanups sourced from press releases and Instagram posts. Due to a lack of high-quality, labeled data for our task, this model was trained with synthetic data generated by GPT-3. This project will support The Ocean Cleanup's larger goal of using the global beach cleanup dataset to perform further downstream analyses, refine their ocean plastic dispersal models, and thus optimize their ocean cleaning strategies. The main challenge of this task stemmed from the fact that some of our NER entities are more reliant on context than the meaning of their constituent tokens (e.g., in-context recognition of trash types and numerical weights of trash). Hence, we experimented to determine whether architectural changes can make our model more performant on this task than the default Stanza architecture.

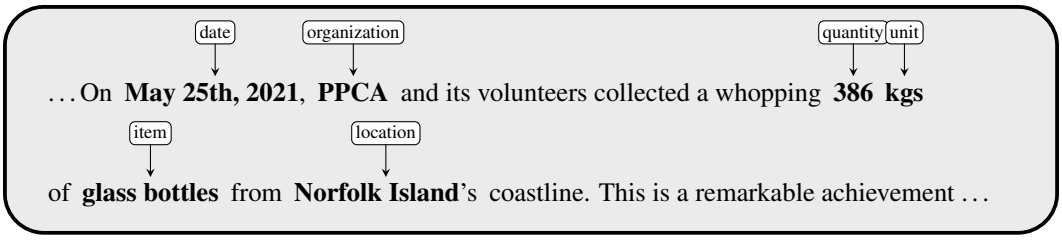


Figure 1: An example (abridged) input document with desired data attribute tagging.

Concretely, the model's task is to correctly identify entities (date, location, organization(s), trash type(s), weight, and unit) from raw press releases and Instagram captions. Our model to produces a vector $o_i \in (0, 1)^6$ for each token i in the document, with each element of o_i corresponding to the probability that token i is a member of the respective data attribute of interest. Figure 1 illustrates example input with output tags (after thresholding) that we expect our model to produce.

3 Related work

3.1 Stanza

We use the Stanza architecture and codebase (Qi et al., 2020) as a starting point for our modeling. However, we made significant modifications to allow for transfer learning and parameter freezing from the Stanza NER Tagger; we also implemented changes to the architecture ourselves.

The Stanza NER architecture, illustrated in Figure 3(a), begins by concatenating word- and character-level embeddings for each token. The input embeddings are then fed through neuron and word dropout layers, the latter of which omits individual words with a small probability. Next, these embeddings are fed into a Bi-LSTM layer, before passing through a feed-forward network and a conditional random field layer which gives the final output. This conditional random field layer, introduced in Lafferty et al. (2001) and discussed for NER in Lample et al. (2016), takes in the Bi-LSTM output and maximizes the tag sequence probability rather than the element-wise probability that would result from a typical softmax.

3.2 Motivation for architectural experiments

We combed through relevant literature in search of potential experiments to better understand and improve upon the base Stanza NER architecture. The papers we referenced are documented here; further implementation details for each experiment are provided in (5).

3.2.1 Component analysis

Fisher and Vlachos (2019) performed ablation analysis on various layers of an NER model, which inspired us to do the same with the base Stanza architecture to further understand it. We did this by performing a set of experiments in which components of the network are removed, added, or replaced in order to measure the importance of these components relative to the model's performance.

We also experimented with downgrading the Bi-LSTM layer to a GRU, adding and removing hidden layers between Bi-LSTM and output layer, and varying the activation functions being used in those layers. One such activation function we tried was the Gaussian Error Linear Unit (GELU) proposed in Hendrycks and Gimpel (2016), which provides a smoother activation than ReLU and roughly weights its inputs based on the extent of its magnitude above other inputs.

3.2.2 Transformers

Following the publishing of "Attention is All You Need" (Vaswani et al., 2017), the NLP research community created many state-of-the-art models by replacing Bi-LSTM layers in previous models with transformer layers. Thus, we attempted to improve performance by: 1) adding a pre-trained

BERT model (Devlin et al., 2018) as part of the input embedding procedure, and 2) replacing the Bi-LSTM layer in the Stanza NER architecture with a transformer trained from scratch.

For experiments trained from the pre-trained BERT model in Devlin et al. (2018), we experimented with the linear probing then fine-tuning training procedure detailed in Kumar et al. (2022). The intuition is to train the model with the pretrained layers frozen as a first pass, to find good initial values for the downstream layers. Then, we fine-tune the whole model together with a suitably small learning rate.

3.2.3 1D Convolutions

In Fisher and Vlachos (2019), we also found an example of a convolution layer being applied over input embeddings for an NER task similar to ours. They found that additional local context improved performance when tagging nested NER types (e.g. "The Prime Minister of The United Kingdom"). As such, we thought that using an input convolution might improve our model's identification of the relationship between trash types and their surrounding words in the input text (e.g., using "picked up *XYZ* from the beach", even where *XYZ* was not seen in training).

3.2.4 spaCy and tri-gram CNN

We were also inspired by spaCy (Honnibal et al., 2020), another state-of-the-art NER system. While we did not replicate the core architecture of spaCy v2, which uses a transition-based parser, we drew inspiration from spaCy's alternative approach to including word context, referred to as a "tri-gram CNN" in Honnibal (2017). A tri-gram CNN layer concatenates the d-dimensional embeddings of three adjacent words and feeds it through a feed-forward network to condense the resulting $3d$-dimensional embedding back into $\mathbb{R}^d$.

4 Approach

Our approach for this project was to begin with the Stanza base architecture, fine-tune on our dataset, and then run a series of experiments as motivated in 3 in an attempt to improve performance on our NER task.

4.1 Baselines

Some of our NER entities are unique to our task, so there are no off-the-shelf models we can use as direct baseline comparisons. We expected our model to perform better than a 'dummy' baseline that simply predicts the 'O' (not-an-entity) tag for every token. We also evaluated the Stanza NER model off-the-shelf (without knowledge of our custom tags) on our data to compare it against the performance of our models. Given that the Stanza model was not trained on many of our entity types, we expected that it would perform substantially worse than any of our models which were specifically trained on our data. Finally, we sense-checked our models' performances by comparing them to those of similar NER models. While not a true baseline, we expected that our models should perform comparably to Stanza's reported NER F1 accuracy on other datasets (F1 scores of 88% to 92% for OntoNotes (Weischedel et al., 2013) and CoNLL03 (Sang and De Meulder, 2003), respectively).

4.2 Architecture

As our architecture varies substantially between experiments, we refer the reader to (3.1) for a description of the base architecture and (5) for in-depth descriptions of the various architectural modifications we made across different experiments.

5 Experiments

5.1 Data

Due to a lack of labeled data, we generated a corpus of synthetic Instagram posts and press releases using GPT-3. We prompted GPT-3 to generate an Instagram caption or a press release for a beach cleanup with randomly sampled data attributes from our key entity types (i.e., those in Figure 1).

To simulate real data, we varied the number and specificity of prompted trash types, omitting each attribute with a small probability. We divided our data into around 8k tagged sentences for training, 1k for development, and 1k for testing. See A.1 for further information on synthetic data generation.

5.2 Evaluation method

We measured the performance of our models using F1 Score on a test set of $\sim$1k sentences containing the custom tags described above. F1 Score was chosen as it balances precision and recall, both of which are important to our task. Once we evaluated on test data, we scraped new texts from Instagram captions and manually annotated a subset of the texts to assess if our model generalizes well to true human-generated data. Lastly, we subjectively analyzed those annotated texts to try to understand if and why our model was able or unable to extract the desired information.

5.3 Experimental details

The base architecture of our model comes from Stanza (Qi et al., 2020), discussed in 3.1. We make various adjustments to the training and architecture in hopes of increasing test performance. Further description of our major experimental classes is provided in the sections below.

For each of our experiments except the Transformer experiments (5.3.3), the time required to train on $\sim$8k sentences was $\sim$45 minutes. The non-Transformer experiments used an SGD optimizer with an initial learning rate of 0.1 and the PyTorch `ReduceLROnPlateau` scheduler with a patience of 3 and decay of 0.5. Other hyperparameters include a batch size of 32, a dropout rate of 0.5, and a word dropout rate of 0.01. Other experiment-specific hyperparameters are detailed in their respective sections; 5.3.3 outlines the hyperparameters used to train the Transformer experiments.

5.3.1 Training of base Stanza architecture

We ran several experiments on the Stanza base architecture, modifying the training procedure to achieve different results. Our base experiment was implemented without transfer learning, where we trained the whole architecture aside from the pretrained word and character embeddings (i.e., random initialization of all layers inclusive of the first linear layer in Figure 3(a)). We then modified the code to leverage transfer learning, initializing our parameters with the learned values from an off-the-shelf Stanza NER model for existing tags. In later experimental rounds, we also tried allowing and disallowing gradient updates ("freezing") for these transferred parameters, in addition to adding an additional hidden layer between the Bi-LSTM and output layers to compensate for the freezing.

5.3.2 Component analysis

Component analysis of the base Stanza architecture allowed us to investigate the contribution of different components of our deep learning model to its overall performance. This analysis can help us understand the most performant activation functions, number of hidden layers, and recurrent layer type (Bi-LSTM vs. GRU), among other things.

To investigate the importance of the three LSTM gates– forget, update, and output– we replace the Bi-LSTM network with a bi-directional GRU network (Cho et al., 2014), which only has two gates– reset and update. Compared to LSTM, GRU has fewer parameters and is faster to train, making it a popular choice for smaller datasets or tasks with limited computational resources. Here, we wanted to investigate the improvement in performance attributed to using Bi-LSTM over GRU, due to its ability to maintain longer-term context.

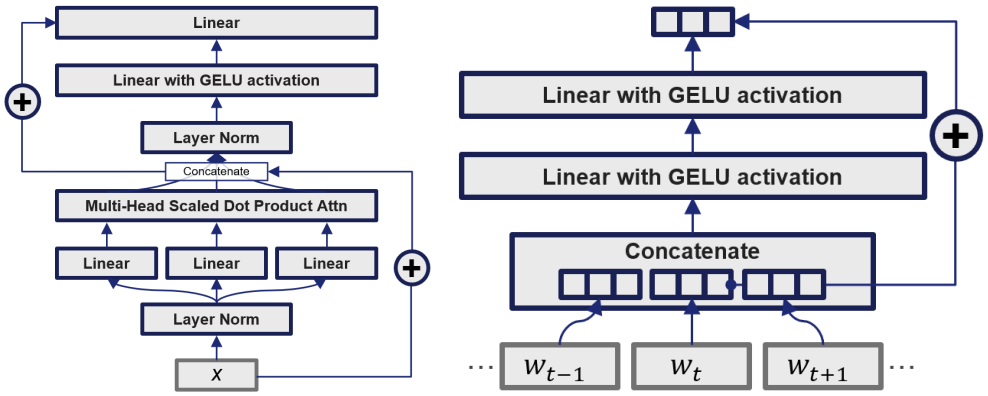
Different numbers of hidden layers and different choices of activation function also affect the model performance. We also added different numbers of hidden layers (with different activation functions– GELU, ReLU, and sigmoid) between Bi-LSTM and the output layer to see how these changes affect performance. Finally, we tried replacing the linear layer between the Bi-LSTM and output layer with an attention layer of varying numbers of parallel attention heads.

5.3.3 Transformers

As motivated in 3.2.2, we experimented with Transformers in two ways. First, we included the results of a pre-trained BERT model (`bert-base-cased` (Devlin et al., 2018)) by concatenating them with

our other input embeddings before feeding into the Bi-LSTM layer. We also experimented with removing the Bi-LSTM layer– adding our feed-forward and conditional random field layers directly to the model to investigate whether the pre-trained model rendered the Bi-LSTM layer redundant. For each of these experiments, we trained one model using the same hyper-parameters detailed in 5.3 and another one inspired by the linear probing then fine-tuning approach suggested in Kumar et al. (2022), where we froze the BERT layers and trained our Bi-LSTM and classifier layers from scratch using the hyper-parameters from 5.3 before fine-tuning the whole model for 50k steps using the ADAM optimizer with an initial learning rate of 1e$-$5, β_1 of 0.9, and β_2 of 0.999.

We also experimented with implementing our own Transformers from scratch, before replacing the Bi-LSTM layer in the original (non-BERT) architecture with 1 to 8 Transformer layers. Figure 2(a) details the architecture of our custom-implemented Transformers. These experiments required considerably more training time, up to 8 hours for $\sim$100k iterations. We used an ADAM optimizer with an initial learning rate of 0.003, β_1 of 0.9, and β_2 of 0.999. We set our batch size to 256. Utilizing grid search, we found that 2 layers of 16-head Transformers worked best to replace the Bi-LSTM, which is the result reported in 5.4.



(a) Diagram of single Transformer block (b) Diagram of tri-gram CNN layer

Figure 2: Diagram of custom-implemented layers

5.3.4 1D convolutional layers

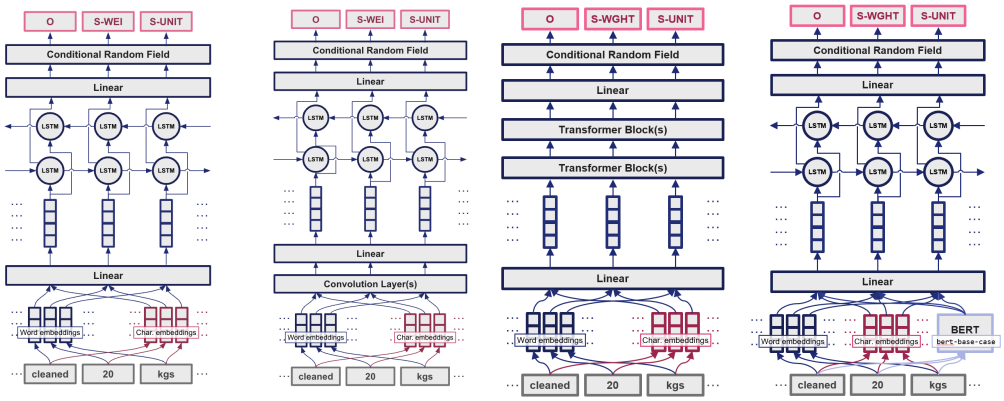
Next, we included a 1D convolution layer over the input embeddings. We padded the front and back of each sentence to ensure the output was the same length as the input. We tested this architecture with a kernel size of 3 and 5, finding that as the kernel size increased, our model performance decreased compared to the base architecture.

Figure 3(b) illustrates the updated architecture of our base model for those experiments including a convolutional layer. "Convolution Layer(s)" represents the location of both the 1D Convolution layer and the Tri-gram CNN layers, included in distinct experiments.

5.3.5 Tri-gram CNN layers

As discussed in 3.2.4, a tri-gram CNN layer concatenates the d-dimensional embeddings of three adjacent words and feeds it through a feed-forward network with dropout ($p = 0.5$) to condense the resulting $3d$-dimensional embedding back into $\mathbb{R}^d$. A residual connection is also added from the input to the output of the layer, which among other things encourages the output space of the layer to be similar to the input / embedding space (Honnibal, 2017). Figure 2(b) illustrates the basic architecture of a single tri-gram CNN layer.

We implemented the tri-gram CNN ourselves in PyTorch and experimented with inserting multiple layers before the Bi-LSTM (to include more contextual information into the embedding given to the Bi-LSTM) as in Figure 3(b), as well as with replacing the Bi-LSTM outright with tri-gram CNN layers.



(a) Base Stanza architecture (b) Placement of conv. layer(s) (c) Transformers in place of Bi-LSTM (d) BERT additions

Figure 3: Model Architecture for Various Experiments

5.4 Results (highest F1 highlighted)

Our results on our held-out test set are better than we expected. Our best model, which leverages pre-trained BERT encodings as additional representation of the tokens, achieves an F1 score of 94.75 on the synthetic test set. While re-training the default Stanza architecture performed quite well already (F1 score of 94.31), we were able to improve performance slightly by changing certain components (e.g., replacing the final classification layer with multi-head attention (94.45 F1 Score) or adding a tri-gram CNN layer (details in 5.3.5), which improved F1 score to 94.51.

Architecture	Experimental Method	F1 Score (Synthetic[1])
Stanza	No transfer learning	94.31
	Transfer learning w/o param. freezing	93.04
	Transfer learning w/ freezing + hidden layer	90.67
	Transfer learning w/ freezing	88.24
Component analysis	Add 1x 10-head attn. and 1x linear layer w/o act.	94.33
	Add 1x 10-head attn. and 1x linear layer w/ GELU	94.45
	Add 1 linear layer w/o act.	94.31
	Add 1 linear layer with GELU	94.43
	Add 1 linear layer with ReLU	94.36
	Add 1 linear layer with Sigmoid	94.02
	Replace Bi-LSTM with bidirectional GRU	94.06
BERT	**BERT added to Stanza arch., 2-stage training**	**94.75**
	BERT added to Stanza arch., 1-stage training	94.51
	BERT replacing Bi-LSTM, 2-stage training	94.10
	BERT replacing Bi-LSTM, 1-stage training	93.69
Transformers	2x 16-head Transformers replacing Bi-LSTM	90.34
Tri-gram CNN	3 layers added to Stanza arch.	94.51
	5 layers replacing Bi-LSTM	94.41
	5 layers added to Stanza arch.	94.07
1D Conv.	Single conv. layer (kernel = 3) added to Stanza arch.	93.57
	Single conv. layer (kernel = 5) added to Stanza arch.	92.96

Baseline / Comparison Method	F1 Score
Reported Stanza NER Performance (OntoNotes Dataset)	88.8
Dummy predictions on test set (always predict 'O' tag)	81.5
Stanza NER off-the-shelf (without custom tags)	23.51

[1]F1 Score evaluated on a held-out test set generated by GPT-3 as described in 5.1

6 Analysis

Our models' performance on a held-out synthetic testing set exceeded that of all of our baselines and comparisons, including the published Stanza results on their NER task and datasets. This level of performance may be due to the relative simplicity of identifying entity types in a reduced domain space (only beach cleanups as compared to all text domains) or the relative homogeneity of our synthetic dataset compared to all natural language. In addition, adding hidden layers on top of the base model was not helpful for predicting our tags, indicating that the base model already had sufficient capacity to approximate our restricted domain space. Interestingly, the Stanza base model confuses organizations and locations (two tags it was trained on) more frequently than our models do, suggesting that the fine-tuning taught our model distinguishable characteristics about beach cleanup organizations specifically, which often contain locations in their names. We found these high-level observations to be true across the range of our experiments, but will provide further analysis below on our primary experimentation classes.

6.1 Component analysis

We begin our network component analysis by adding hidden layers and attention layers. Though adding a hidden layer did not improve performance by much, adding an attention layer slightly improved the model performance. The latter result may be an effect of the attention layer weights helping the model focus on the most relevant parts of the input data.

With respect to activation function, we believe GELU performs the best due to its smoothness compared to ReLU, which can make it more effective at learning complex patterns in the data. Specifically, ReLU can suffer from problems when a significant proportion of neurons "die" (become zero and are not updated due to the zero gradient). By contrast, GELU is smooth near zero and has nonzero gradient at zero, allowing these neurons to be "revived".

Our 0.25 decrease in F1 Score when switching from Bi-LSTM to Bi-GRU implementations confirms our expectations that LSTMs perform better than GRU on sufficiently large datasets, due to the GRU's lower expressivity.

6.2 Transformers

As discussed in 5.3.3, we concatenate pre-trained BERT encodings to the word- and character-level embeddings in the base Stanza NER architecture. A naive training of this addition improved upon the base model F1 Score by 0.20; however, we were able to improve the score by a further 0.24 using the linear probing then fine-tuning approach motivated in Kumar et al. (2022). This is intuitive, as freezing the BERT layers initially allows the downstream layers to settle on parameter values that can reasonably leverage the BERT encoding as-is; then, when we fine-tune, the model is effectively initialized to reasonable values throughout which leads to more cohesive parameter updates.

Meanwhile, when we attempt to implement our own Transformer layers from scratch to replace the Bi-LSTM in Stanza, we noticed a significant (3.0-point) drop in F1 Score. We posit this is mostly due to the relatively small amounts of data we had generated– on the order of thousands of sentences rather than the millions in Vaswani et al. (2017). Furthermore, the maximum batch size we could fit on our single-GPU system was 256, while the Vaswani et al. (2017) work was able to leverage 8 times the GPU resources and use a batch size of 4096, per Vaswani et al. (2018). This much larger batch size leads to less dithering in parameter updates, which is crucial given the large number of parameters present in a Transformer layer.

6.3 Convolutional layers

We experimented with convolutional layers in two ways– both using a traditional 1D convolutional layer (5.3.4) and the spaCy tri-gram CNN layer (3.2.4). We presume that since the Bi-LSTM layers already provide some notion of 'memory' in context, the additional 1D convolution layers obfuscated the input signal of the word embeddings rather than providing additional information to the model, decreasing F1 score in all our 1D convolution experiments.

We had more success with the tri-gram CNN layers, achieving the highest non-BERT F1 Score (94.51) when inserting 3 layers before the Bi-LSTM layer. We posit that the improvement in performance

over the traditional 1D convolution and base architecture stems from the tri-gram CNN's ability to represent non-linear contextual relationships. Furthermore, the residual connection in the layer coerces the output to be in a similar space to the input, which may better suit the Bi-LSTM. These layers' incremental value is surprising given that the Bi-LSTM can already retain and represent context from neighboring tokens. We hypothesize that the value of the tri-gram CNN stems from its explicit prioritization of adjacent tokens, which when combined with the importance of short-term context to our task, acts as a sort of inductive bias which nudges the model towards leveraging that short-term context.

6.4 Generalization to scraped data

Lastly, we note that even our best models do not seem to generalize well to human-written texts scraped from Instagram (F1 Score of $\sim$ 67-73 [2] for our best models). It appears that while GPT-3 produced texts with perceived linguistic variation, its output still did not vary as much as human-generated texts. In terms of content, our synthetic tests set included only posts and press releases that actually contain information about a beach cleanup, while the vast majority of the scraped posts with the hashtag "#OceanCleanup" do not speak about completed cleanup events, instead advertising planned cleanups, lamenting the need for cleanups, or sharing thoughts wholly unrelated to beach cleanup efforts. Orthographically, GPT-3 also produces text with fewer spelling mistakes and more formal language than is found in our scraped texts. In summary, while resource limitations forced us to train on a synthetic dataset, it is not a sound assumption to claim that our training data and the scraped test data come from the same distribution. This distribution shift likely contributed significantly to the drop in F1 score between our synthetic and scraped evaluations.

7 Conclusion

In conclusion, our models work well on the NER task and distribution of data they were trained on, but our limitations in access to reliable, labelled, human-generated training data prevent us from achieving a similar performance when predicting on real data. As potential future work, it would greatly benefit this project to compile such a dataset for training and testing, or to improve the prompting method of GPT-3 (or even experiment with GPT-4) in an attempt to produce synthetic data with greater verisimilitude.

Given more time and resources, we would like to increase our models' functionality to support additional international languages, as keeping our oceans clean is a global effort. This expansion would allow us to build a more versatile tool for The Ocean Cleanup and would offer opportunities for analyses on linguistic and cultural differences that may affect NER performance across languages. Additionally, we might consider investigating ways to help our model generalize better to various types of text documents, by implementing a more diverse training set, hyperparameter tuning, or further architectural changes. We would also like to complete a more holistic evaluation of our model on a diverse evaluation set, including various text genres and additional noise or low-quality data, to better understand its limitations and identify areas for improvement. By pursuing these avenues of research, we would hope to develop a more robust and effective NER model and pipeline that can be applied to a wide range of languages and texts to assist the global plastic modeling and cleanup efforts of The Ocean Cleanup.

[2] F1 Score evaluated on a manually annotated subset of scraped Instagram captions

References

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches.

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.

Joseph Fisher and Andreas Vlachos. 2019. Merge and label: A novel neural network architecture for nested ner. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5840–5850, Online. Association for Computational Linguistics.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

Matthew Honnibal. 2017. Spacy's entity recognition model: incremental parsing with bloom embeddings & residual cnns.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. 2022. Fine-tuning can distort pretrained features and underperform out-of-distribution.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher Manning. 2020. Stanza: A python natural language processing toolkit for many human langauges. In *Association for Computational Linguistics (ACL)*.

Erik Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shard task; language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2tensor for neural machine translation. *CoRR*, abs/1803.07416.

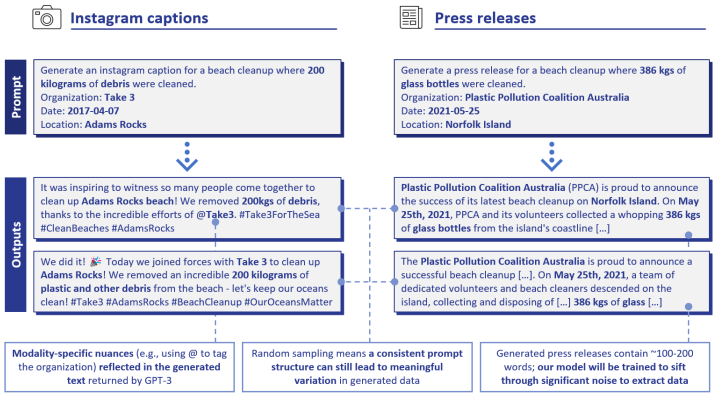
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Ralpha Weischedel, Martha Palmer, Mitchess Marcus, Eduard Hovy, Sameer Pradhan, et al. 2013. Ontonotes release 5.0. In *Linguistic Data Consortium*.

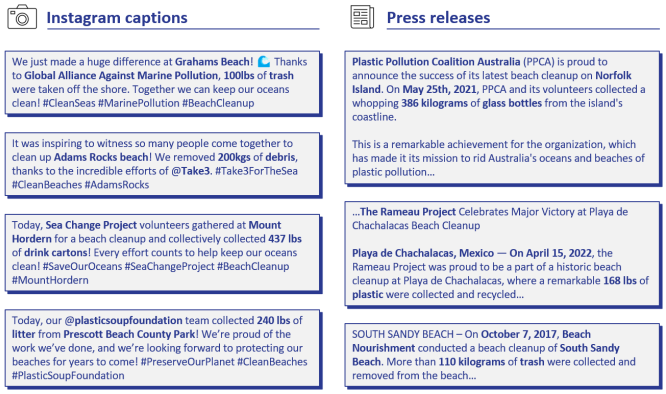
A Appendix

A.1 Synthetic data generation details

As mentioned in 5.1, we leveraged GPT-3 to generate synthetic data for our task. We did so by sampling desired features (location, organization, date, trash item, trash weight, weight unit), before prompting GPT-3 via the Python API to generate synthetic natural language documents of the format requested, as shown in Figure 4(a).



(a) Example prompts and generated data



(b) Further examples of generated data

Figure 4: Illustrative GPT-3 Prompting Process for Synthetic Data Generation