Master Thesis
Data Science

Radboud University
Faculty of Science

SpanMarker for Named Entity Recognition

Author:
Tom Aarsen

First supervisor/assesor:
prof. dr. Fermin Moscoso del Prado Martin
fermin.moscoso-del-prado@ru.nl

Daily supervisor:
dr. Daniel Vila Suero
daniel@argilla.io

Second assesor:
dr. Harrie Oosterhuis
harrie.oosterhuis@ru.nl

Nijmegen, June 2023

Abstract

This thesis presents SpanMarker, a span-level Named Entity Recognition (NER) model that aims to improve performance while reducing computational requirements. SpanMarker leverages special marker tokens and utilizes BERT-style encoders with position IDs and attention mask matrices to capture contextual information effectively. Compared to prior work, the model incorporates key modifications to improve computational efficiency, such as reducing token padding and simplifying the feature vector. Experimental results demonstrate that SpanMarker achieves state-of-the-art performance on benchmark datasets, including CoNLL03 and FewNERD, while significantly reducing training time compared to existing models. The SpanMarker library has been released as a user-friendly Python module, and is integrated into the Hugging Face Hub for easy sharing, testing, and deployment. This thesis additionally emphasizes the importance of reporting means and standard errors for empirical evaluation. The source code, training scripts, and published models are all available at `https://github.com/tomaarsen/SpanMarkerNER`.

Table of Contents

1. Introduction

Named Entity Recognition (NER) is a fundamental task in Natural Language Processing (NLP) that involves identifying and classifying named entities, such as person names, locations, organizations, and dates, within text documents. Accurate NER is crucial for numerous downstream NLP applications, including information extraction, question answering, sentiment analysis, and machine translation. Over the years, various NER models have been developed, ranging from rule-based systems to advanced deep learning architectures.

Traditionally, NER has been approached as a token-level problem, focusing on individual tokens without considering their relationships. However, in recent years, span-level NER has gained significance due to its ability to capture the contextual information around the named entities, leading to improved performance in downstream tasks. Existing approaches for span-level NER often involve the use of complex architectures [57] or sophisticated pretraining techniques [53], which are computationally expensive and require substantial amounts of training data.

This thesis presents a novel NER model and Python library called SpanMarker, which aims to address the challenges of span-level NER while maintaining a simpler architecture. SpanMarker leverages special marker tokens fed to an underlying encoder model that "observe" specific text spans. This approach considers numerous spans from a text per forward pass of the underlying encoder, all while using each marker token embedding for only one span. Much previous work uses token embeddings for multiple spans [57, 60], requiring the embeddings to represent different labels when combined with other token embeddings. The flexibility required by these embeddings in previous work limits the potential performance, and SpanMarker unlocks this improved performance by requiring each embedding to represent only one span and thus only one label.

The main contributions of this thesis are as follows:

1. I propose SpanMarker, a new approach for span-level Named Entity Recognition that achieves competitive performance while minimizing computational requirements.
2. I design, implement and open source my approach into a usable and approachable Python library, effectively reducing the gap between state of research and state of practice.
3. I integrate SpanMarker into the Hugging Face hub[1], allowing SpanMarker models to be easily shared, tested online, and deployed into production.
4. I evaluate SpanMarker on standard NER benchmark datasets and compare its performance both against state-of-the-art and state-of-practice NER models.
5. I compare the training times of SpanMarker models with those of prior state-of-the-art approaches.
6. I conduct a comprehensive ablation study to assess the impact of different components and design choices in the SpanMarker model. Additionally, I perform several experiments incorporating design choices from previous works.
7. I provide case studies to demonstrate the effectiveness of SpanMarker compared to commonly used alternatives.

[1] `https://huggingface.co/`

The remainder of this thesis is organized as follows: Section 1.1 formulates the NER task, followed by preliminaries in Section 1.2. Section 1.3 wraps up the introduction with a review of relevant related work. Chapter 2 presents the methodology of the SpanMarker model. Section 3.1 describes the experimental setup, including datasets, evaluation metrics, and baseline models for comparison. Section 3.2 discusses the experimental results, including a detailed analysis. Chapter 4 concludes the paper, followed by Chapter 5 which describes the published SpanMarker Python library. Finally, Chapter 6 proposes future directions for improving span-level NER using the SpanMarker model.

1.1 Task formulation

Given a text document or a sequence of words $X = \{x_1, x_2, \ldots, x_n\}$, the task of Named Entity Recognition (NER) involves identifying and classifying named entities within the text into predefined categories. The goal is to assign a label $Y = \{y_1, y_2, ..., y_m\}$ to spans (or n-grams) of words in the input sequence, indicating the specific entity category to which it belongs.

Formally, the task can be defined as follows:

- **Input:** Text sequence: $X = \{x_1, x_2, ..., x_n\}$, where x_i represents the i-th word in the text.
- **Output:** A list of entities each consisting of three elements:
  1. Span start index i,
  2. Span end index j with $j \geq i$,
  3. Label y_k representing the k-th label in Y, the predefined list of entity categories.

  For each entity in the output, the label y_k is assigned to the subsequence of words $\{x_i, x_{i+1}, \ldots, x_{j-1}, x_j\}$.

1.1.1 Objective

The objective of the NER task is to learn a mapping function f that takes the input text sequence X as input and produces the corresponding list of entities as output. The function f should accurately identify the boundaries of named entities and correctly assign entity labels to the corresponding tokens in the input text.

1.1.2 Training Data

A labeled dataset is required to train any NER model, consisting of a collection of texts, where each text is annotated with the corresponding entity labels. The annotations indicate the boundaries of the named entities within the text and their respective categories. Several standards for annotating these boundaries exist, many of which prefix the label categories with a specific character.

The most common format is the IOB2 labeling scheme, which adds an "`O`" (outside) label and prefixes all entity categories with "`I`" (inside) and "`B`" (begin). For example, when annotating for a NER problem containing the "`PER`" (person), "`ORG`" (organisation), "`LOC`" (location) and "`MISC`" (miscellaneous) labels, the IOB2 labeling scheme requires the following labels: "`O`", "`I-PER`", "`B-PER`", "`I-ORG`", "`B-ORG`", "`I-LOC`", "`B-LOC`", "`I-MISC`" and "`B-MISC`".

In practice, "`B`"-prefixed labels are used for the first words of all entities, while "`I`"-prefixed labels are used for all other words that are a part of a named entity. This labeling scheme allows two adjacent entities to be annotated with clarity on which words belong to which entity. If only the original labels was used, this would not be possible.

A sample that uses this labeling scheme may look like this:

```
John       B-PER
Smith      I-PER
works      O
at         O
Microsoft  B-ORG
Research   I-ORG
in         O
Redmond    B-LOC
.          O
```

Other labeling schemes are used in practice too, such as IOB [35], BIOES (Begin, Inside, Outside, End, Singular) and BILUO (Begin, Inside, Last, Unit, Outside) [13]. These are all equally descriptive as IOB2.

1.1.3 Evaluation

The performance of NER models is typically evaluated using precision, recall, and F1 score, calculated at the entity level, while ensuring correct span boundaries. This mirrors the evaluation used in the CoNLL 2003 [43] task. In practice, evaluation is performed using the `seqeval` [29] Python package, from which micro-F1 is most commonly reported.

1.2 Preliminaries

1.2.1 Encoder Models

Encoder models play a fundamental role in various tasks by transforming input text into meaningful representations. Encoder models are responsible for capturing the semantic, syntactic, and contextual information embedded in the input text. Examples include the BERT-based encoders, such as the original BERT [9] model or its variants, like RoBERTa [23] and DeBERTa [11, 12].

These are models that produce contextual word embeddings, also known as contextualized representations. Unlike traditional word embeddings (e.g., word2vec [27], GloVe [31], fastText [4, 16], or Senna [7]), which provide fixed representations for words, BERT produces word embeddings that are sensitive to the context in which the word appears. The contextual word embeddings capture fine-grained information about each word its meaning and how it interacts with the surrounding words in the sentence.

Notably, BERT-based encoders can be fine-tuned on specific downstream tasks. Fine-tuning involves training a pretrained model on task-specific labeled data, allowing it to adapt its learned representations to the task at hand. This leads to competitive performance on a wide range of NLP tasks, even with limited labeled data. For this reason, the majority of modern NER approaches rely on a BERT-style encoder model.

1.2.2 Words Versus Tokens

BERT-based encoder models typically operate at the token level rather than the word level. These models use a process called tokenization to break down the input text into smaller units called tokens. Tokens often correspond either to words or subwords, therefore tokens do not always align one-to-one with traditional linguistic words. Furthermore, BERT-style encoders work with token or input IDs, which are integer representations of the tokens.

1.2.3 Special Tokens

BERT-based encoders use special tokens to provide additional information to the model. These tokens include the "`[CLS]`" (classification) token, which is prepended to the input and represents the aggregate representation of the whole sequence for downstream tasks, and the "`[SEP]`" (separator) token, used to separate two different sentences in tasks like sentence pair classification or text generation. These special tokens help BERT understand the sentence structure and help with specific tasks during training and inference. When finetuning, users have the freedom to specify additional special tokens for custom use cases.

1.2.4 Position IDs

Position IDs are used to encode the positional information of tokens in a sequence. Since BERT operates on fixed-size input sequences, it needs to understand the relative positions of tokens within the sequence to capture their relationships. Position IDs assign a unique identifier to each token in the input sequence based on its position. These identifiers are typically integer values that range from 0 to the maximum sequence length. BERT-style models rely on these position IDs to learn contextual representations that capture positional dependencies.

1.2.5 Attention Mask Matrices

In BERT-based encoders, attention mask matrices can be used to affect the contextual relationships within the input sequence. These matrices represent the attention weights assigned to each pair of tokens, indicating the relevance and importance of one token to another.

The attention matrices allow BERT to dynamically focus on relevant tokens, capture long-range dependencies, and model token interactions effectively. Additionally, it allows BERT-based encoders to prevent sections of input from interacting with each other through attention.

1.3 Related Work

1.3.1 Named Entity Recognition

The field of NLP has undergone a paradigm shift, moving away from static word embeddings such as word2vec [27], GLoVe [31], and fastText [4, 16], towards the adoption of context-sensitive embeddings [32] and pretrained encoders [6, 8, 9, 11, 12, 18, 23]. Furthermore, whereas NER was previously approached as a token-level sequence labeling task [1, 5, 9, 13, 14, 17, 26, 32, 34, 36, 45, 49, 59], recent advancements have shifted the focus towards span-level classification approaches [19, 20, 30, 53, 55, 57, 60].

Certain approaches in span-based methods initially extract spans through enumeration [25, 40, 55] or by identifying boundaries [41, 59], followed by classifying each individual span. The latter can be decomposed into two separate stages [37]. Moreover, external knowledge can be incorporated into models through various means, such as leveraging corpora like WordNet [28] [46], knowledge graphs [33, 58], or search engines [49]. Additionally, certain models adopt entity-related pretraining methods, which involve incorporating knowledge from Wikipedia's anchor texts [47, 53].

Various models also present unconventional solutions, such as considering NER as a dependency parsing task [56] or applying contrastive learning [21, 57]. NER can also be considered a generative sequence-to-sequence problem [42, 54] or solved via a diffusion process [39].

1.3.2 Span embeddings

Span-level models in the literature employ various approaches to generate span embeddings. The most widely adopted technique is the concatenation of token embeddings [15, 57, 60]. This

method typically involves concatenating the embeddings of the tokens at the start and end of the span, sometimes accompanied by a span length embedding [57, 60]. However, a critical limitation of this approach is the inability to capture the embeddings of tokens located between the start and end of the span, potentially leading to a decrease in performance, particularly for longer spans. Another variant of this technique involves aggregating all token embeddings within a span into a single embedding, commonly accomplished through mean or weighted mean calculations [15].

A different approach involves placing solid marker tokens [2] inside of the input text, one marker token before and one marker token after the span. A combination of the embeddings of these tokens are then used as the embedding of the span that they encompass. However, pretrained context-sensitive encoders will not be familiar with these tokens, and thus may have degraded language modeling performance. Moreover, this approach requires one forward pass of an encoder for each span, resulting in high computational costs.

Alternative approaches utilize levitated markers, which are tokens placed after the input text tokens [55, 60]. These methods employ a pair of levitated markers for each span, with their positions aligned to the start and end tokens of the respective span. By leveraging the attention mask matrix, these marker pairs can interact with both the text tokens and the pair partner, while remaining invisible to the text tokens and the other levitated marker pairs. This approach effectively addresses the limitations associated with the solid marker token strategy.

In their work, Ye et al. [55] propose the Packed Levitated Marker (PL-Marker) approach to pack levitated marker pairs together for all spans within a text. Because the BERT-style encoders utilized in PL-Marker suffer from quadratic complexity, it is important to keep the input size small. As a result, PL-Marker splits its spans into separate groups of 256 levitated tokens, each passed alongside the text tokens through the encoder.

While careful grouping has been employed in PL-Marker to enhance training and inference speeds, the model still suffers from time inefficiencies. This work proposes an adaptation of PL-Marker called SpanMarker to address these limitations. Key modifications are introduced, including reducing token padding and simplifying the feature vector, to enhance the efficiency of SpanMarker. Furthermore, SpanMarker is implemented in a Python library, making it accessible to practitioners working with diverse datasets. By addressing these challenges and providing an easy to use implementation, SpanMarker aims to offer an improved solution for efficient and practical NER tasks.

2. Methodology

2.1 Concept

In many prior approaches to NER, span embeddings are computed by concatenating token embeddings from the first and last tokens of all spans [57, 60]. While computationally efficient, this approach necessitates the use of all token embeddings in multiple spans, possibly with varying labels. Consequently, the downstream classifier must discern the intricate relationships between the token embeddings comprising each span embedding.

To address this limitation, SpanMarker introduces a restriction requiring each token embedding to be used in only one span embedding. This is achieved by introducing a pair of special "`<start>`" and "`<end>`" marker tokens for each individual span. The start marker token serves to observe the first token within the span, while the end marker token observes the last token of the last word in the span.

By employing this approach, the resulting span embeddings are not composed of the embeddings of the text tokens themselves, but rather the embeddings of the observing start and end markers. Importantly, these span embeddings offer improved classification capabilities. This is because the marker embeddings uniquely correspond to a single span with a single label, as opposed to being associated with multiple spans that may have different labels.

2.2 Preprocessing

Same as vanilla Transformer Encoder

SpanMarker is initialized using an encoder language model, such as BERT [9], RoBERTa [23], ELECTRA [6], or DeBERTa [11, 12]. During the preprocessing stage, the tokenizer associated with the chosen encoder is applied on a word sequence consisting of m words, denoted as w_1, w_2, $\ldots$, w_m. This process yields a token sequence comprising n tokens, represented as t_1, t_2, $\ldots$, t_n. It is important to note that some words require multiple tokens due to the limited vocabulary of these tokenizers, resulting in n often being greater than m.

Depending on the specific tokenizer utilized, the token sequence is wrapped with one special token at the beginning and end, such as "`[CLS]`" and "`[SEP]`", or "`<s>`" and "`</s>`". To illustrate, Example 2.2.1 introduces a recurring example to be used throughout this thesis.

Example 2.2.1 Consider the scenario where SpanMarker is initialized using a RoBERTa encoder and is given a word sequence "Andorra is located between France and Spain.", with 8 words including the period. Then, the corresponding tokenized sequence has a length of 12 and consists of these tokens: "`<s>`", " And", "or", "ra", " is", " located", " between", " France", " and", " Spain", ".", and "`</s>`". These tokens correspond to these input (token) IDs: `[ 0, 178, 368, 763, 16, 2034, 227, 1470, 8, 2809, 4, 2]`.

2.2.1 Spans

SpanMarker considers all possible spans (or n-grams) of words up until a predefined maximum entity length, for example up to 8 words. A span is considered valid if it does not contain more words than the maximum entity length allows. With a maximum entity length of 8 and the text from Example 2.2.1, SpanMarker considers 36 distinct spans, enumerated in Listing A.1.

Visit Page 26

2.2.2 Span Markers

One start marker and one end marker must be defined for each of the spans. Together, these are called a span marker pair. These markers are added as special tokens with texts "`<start>`" and "`<end>`" to the vocabulary of the tokenizer that corresponds with the current encoder.

2.2.3 Position IDs

It is crucial for the start and end markers from a span marker pair to be able to "observe" the tokens that correspond with the span. SpanMarker utilizes position IDs for this. Figure 2.1 shows how the position IDs are defined for Example 2.2.1. Please note the following:

1. For normal tokens, position IDs for RoBERTa range between `[2, num_tokens + 1]`, while 1 is used for padding [23]. Consequently, position IDs 0 and 1 are not used in the figure.
2. For example, the first span marker pair has position IDs 3 and 5, corresponding with " And" and "ra". Thus, this span marker pair refers to the span " Andorra". Subsequently, the second pair has position IDs of 3 and 6, referring to " And" and "is" and thus the span of " Andorra is". This repeats for all span marker pairs.
3. There are no start markers for position IDs 4 and 5 because these refer to tokens "or" and "ra", which are not starts of words. Similarly, there are no end markers for positions 3 and 4, which correspond to " And" and "or", as these are not ends of words.

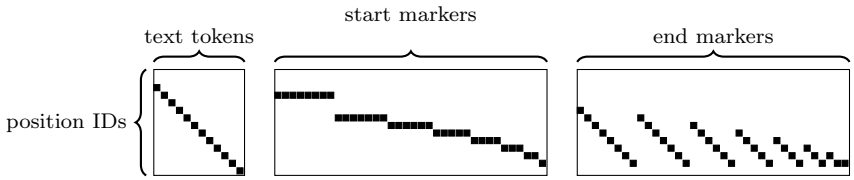


Figure 2.1: The position IDs of text tokens and span markers. Through the position IDs, the span markers can "observe" the token with the matching position ID. The y-position of the black squares determines the position ID value, so the figure shows a text token position ID vector of `[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]`. This sample is created using Example 2.2.1.

2.2.4 Distributing Tokens Efficiently

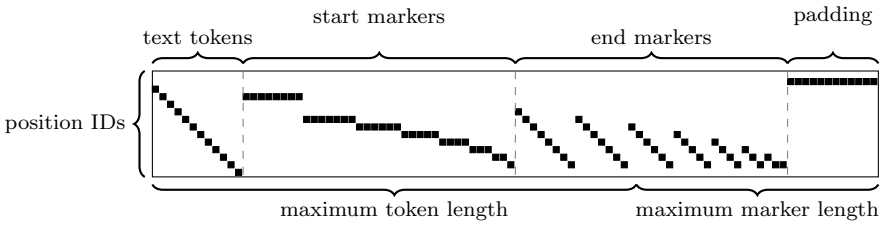
To preserve computational efficiency with the underlying encoder that scales quadratically to the size of the input length, a maximum length must be defined. This maximum length consists of two values:

1. Maximum token length: The number of tokens reserved for text tokens. Generally, this is equivalent to the maximum token length of the tokenizer. For example, for RoBERTa$_{base}$ this is 512 tokens. Crucially, unused tokens in this reserved space will be filled with span markers before padding is applied. This efficient use of space differs from previous approaches [55].
2. Maximum marker length: The number of tokens exclusively reserved for span markers or padding. If there are more text tokens than the maximum token length, the text tokens do not overflow into the span marker reserved section, but they are simply truncated. Common values of this maximum marker length are 128 or 256.

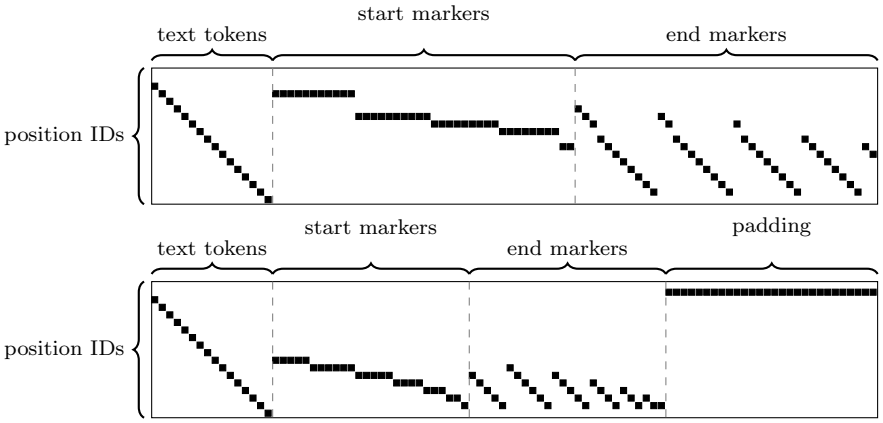
All input and position IDs used in SpanMarker will have a length of `max_tokens + max_markers`. Depending on the input text, the text tokens and all span markers may not fit within this space. Figure 2.2a demonstrates how tokens are distributed in samples when the text tokens fit comfortably within the reserved area, leaving space to accommodate span markers. In contrast, Figure 2.2b illustrates a situation where not all span markers can be represented in a single

sample. As a result, a second sample is created to place the remaining span markers. Notably, the text tokens remain identical across both samples.

Could have been intuitive
But Given it's a "THESIS", very much reasonable to add these



(a) Position IDs for a short text sample with a tiny maximum token length of 64 and a small maximum marker length of 32. The sample is created using Example 2.2.1.



(b) Position IDs for a longer text sample with a tiny maximum token length of 64 and a small maximum marker length of 32. Due to lack of space, this sample must be spread out between multiple vectors. The example text is "Andorra is a landlocked country located between France and Spain.".

Figure 2.2: Distribution of text and span marker tokens in samples that are ready to be passed to the SpanMarker model.

Document-level Context

While NER benchmarks commonly involve isolated sentences without any contextual information, real-world scenarios often require NER models to process sentences within a broader context. To support this, the SpanMarker library (introduced in Chapter 5) implements document-level context, an approach that incorporates adjacent sentences as context during training and inference to improve model performance.

When using document-level context, each sentence provided to the model during training and inference must be accompanied by two identifiers: a document identifier and a sentence identifier. The former serves to identify which document the sentence belongs to, while the latter indicates the position of the sentence within the document.

Only sentences originating from the same document are considered for contextual information. Users have the flexibility to set a limit on the number of sentences to be included as context. There is no limit imposed by default, allowing the inclusion of as much context as is available until the maximum token length is reached.

Even with the inclusion of additional text tokens, the span marker pairs are still only generated for each span within the original sentence. The extended text tokens, together with the start and end markers, are distributed in the samples using the same approach as when document-level context is not utilized.

2.2.5 Attention Mask Matrix

One-directional attention is required for span marker tokens to attend to the text tokens without the text tokens being able to attend to the span marker tokens. This is implemented via a `(max_tokens + max_markers) × (max_tokens + max_markers)` binary attention matrix, for example like Figure 2.3. This figure shows several sections of tokens that can attend to each other:

1. Top left square: All text tokens can attend to all text tokens.
2. Left rectangle: The start and end markers can attend to all text tokens, but not vice versa.
3. Diagonals: The start and end markers can attend to themselves (via the on-diagonal) and to the corresponding span marker pair partner (via the off-diagonal).

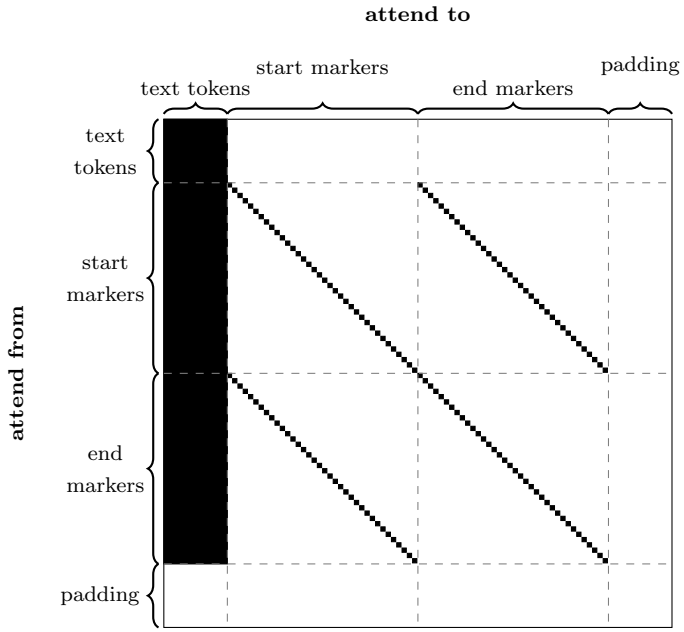


Figure 2.3: Attention mask matrix for the text tokens and span markers. Black indicates unmasked areas, i.e. where attention can occur, while white indicates masked areas. This attention mask matrix is created using Example 2.2.1.

2.3 Model Architecture

Figure 2.4 showcases the simple SpanMarker model architecture involving a BERT-based encoder to produce contextualized embeddings. The input consists of the input IDs, position IDs and attention mask matrix produced from the preprocessing. Upon passing the input through the encoder, the embeddings of the start and end markers are concatenated into one feature vector for each span marker pair. These vectors are fed through a linear layer to map each span to a logits vector storing the scores for each of the classes including "`O`", i.e., no entity.

Prior work also includes the embeddings of the text tokens used in each of the pairs in the feature vector, leading to feature vectors of size `4 * dimension_size` [55] as opposed to `2 * dimension_size`, where `dimension_size` is determined by the size of the BERT-style encoder. This thesis shows that the exclusion of these text token embeddings speeds up both training and inference without a loss in performance.

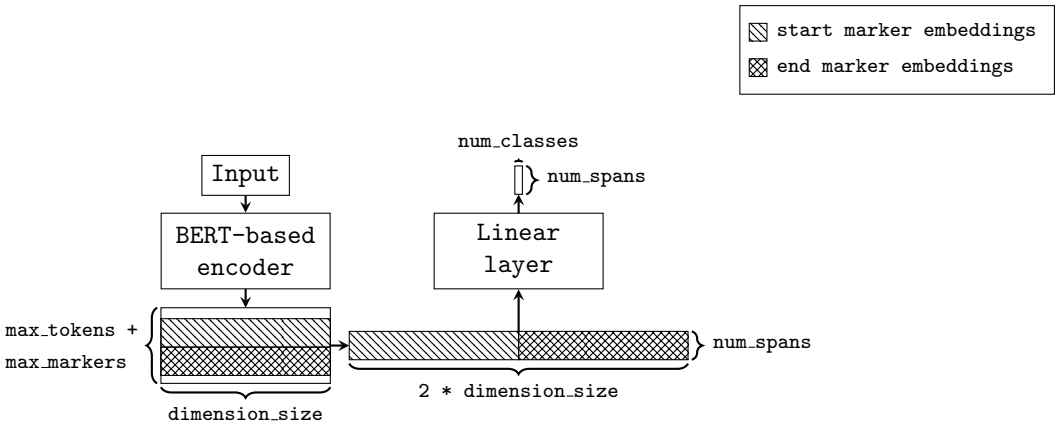


Figure 2.4: SpanMarker model architecture. Input samples are passed to the encoder, resulting in an embedding matrix. The start and end marker embeddings are concatenated and passed through a linear layer, outputting logits with shape `(num_spans, num_classes)`.

2.3.1 Training

Implementing batch processing in SpanMarker requires small changes to the model architecture. In particular, when adding the batch dimension, the linear layer output would be shaped like `(batch_size, num_spans, num_classes)`. However, `num_spans` differs for each of the samples in the batch. Consequently, `(batch_size, max_num_spans, num_classes)` is used for each batch by applying the required padding, where `max_num_spans` is the maximum possible number of spans that can be captured in a sample.

For training, standard cross entropy loss is used between the logits and integer labels that represent the entity class for each span. Due to superior experimental performance, each class is given equal weight when computing the loss, despite the significant class imbalance.

2.3.2 Inference

Performing inference given a sentence involves preprocessing the input into one or more samples and feeding them through the SpanMarker model. Each sample results in a logit matrix, which can be concatenated for all samples into one large logit matrix with shape `(num_spans, num_classes)`. A softmax operation is applied on the logit matrix to convert the logits into confidence scores. Then, an argmax operation is used to extract both the predicted entity class and confidence score for every span in the input sentence.

The process of identifying predicted entities involves iterating over all valid spans in descending order of prediction confidence. For each span, if the predicted entity class is not "`O`" and the span does not overlap with any previously predicted entity, it is added to a list of predicted entities. This criterion ensures that the model does not predict nested entities. The predicted list of entities is returned to the user, containing the text span, class label, confidence score, and start and stop indices for each entity.

3. Experiments

3.1 Experimental Setup

In these experiments, the mean and standard error obtained from 10 independent runs of the experiments are reported. The model with the highest performance on the test set for each benchmark is made available on the Hugging Face hub[1]. All experiments are conducted using consumer-grade hardware, specifically a RTX 3090 GPU, ensuring reasonable training times for practitioners.

3.1.1 Datasets

In order to provide evidence for the efficacy of SpanMarker, experiments are conducted on four flat English NER datasets: CoNLL03 [43], CoNLL++ [50], OntoNotes v5.0 [51] and supervised FewNERD [10]. The approaches for CoNLL03 and CoNLL++ are segregated based on whether they were trained using document-level context. See Table 3.1 for statistical information and Appendix B for additional information on these datasets.

3.1.2 Metrics

Strict evaluation metrics are applied, relying on both the correctness of the entity boundary and the entity class. In particular, micro-recall, micro-precision and micro-F1 scores are reported. For SpanMarker, the evaluations are computed using the `seqeval` [29] Python package, which is compatible with the evaluation script for the CoNLL-2003 [43] shared task.

3.1.3 Baselines

SpanMarker is evaluated against commonly used open-source Python NER modules (spaCy [13], transformers [52] models) as well as state-of-the-art research models such as LUKE [53]. Unless stated otherwise, the reported values are provided by the original authors of the respective approaches. To aid the comparison, the underlying encoder or embedding model used by each approach is specified.

Approaches that require training multiple models are not evaluated against, as they require

[1] `https://huggingface.co`

	CoNLL03 [43]			CoNLL++ [50]			OntoNotes v5.0 [51]			FewNERD [10]		
	Train	Dev	Test	Train	Dev	Test	Train	Dev	Test	Train	Dev	Test
#S	14041	3250	3453	14041	3250	3453	49706	13900	10348	131965	18824	37648
#E	23499	5942	5648	23499	5942	5702	128738	20354	12586	340247	48770	96902
ASL	14.50	15.80	13.45	14.50	15.80	13.45	24.94	20.11	19.74	24.49	24.61	24.47
#ME	20	20	31	20	20	31	32	71	21	50	35	49
#AE	1.67	1.83	1.64	1.67	1.83	1.65	2.59	1.46	1.22	2.58	2.59	2.57

Table 3.1: Statistics of the NER datasets used in the experiments. #S: the number of sentences, #E: the total number of entities, ASL: the average sentence length, #ME: the maximum number of entities in a sentence, #AE: the average number of entities in a sentence.

an unreasonable amount of training time and hardware capabilities. Consequently, no evaluation is performed against ACE [48], co-regularized [61] or CrossWeigh [50] models. Additionally, BERT-MRC+DSC [20] is not included in the experiments due to reproducibility concerns.

3.1.4 Implementation Details

Following prior work [36, 49, 55], the RoBERTa$_{large}$ [23] encoder is used for FewNERD and OntoNotes v5.0, and XLM-RoBERTa$_{large}$ [8] is utilized for CoNLL03 and CoNLL++. All SpanMarker models are exclusively trained on the training split, and its hyperparameters are based on the dataset statistics. For a benchmark that is representative of what a practitioner may experience, hyperparameter optimization or significant hyperparameter tuning is not performed. See Appendix C for details on the chosen hyperparameters, published models, and training scripts.

3.2 Results & Discussion

3.2.1 Results on CoNLL03 and CoNLL++

Table 3.2 shows the evaluation results for the CoNLL03 dataset under two conditions: with and without document-level context. Without document-level context, SpanMarker reaches a 92.9±0.0 F1, with 92.5±0.1 precision and 93.3±0.0 recall. The F1 scores range between 92.65 and 93.15 for the 10 independent models trained for this experiment. SpanMarker demonstrates superior performance to all baseline models that are trained solely on the training data.

When trained with document-level context, SpanMarker reaches a competitive 94.1±0.1 F1, 93.8±0.1 precision and 94.3±0.1 recall. The 10 individual evaluation models range between 93.54 and 94.44 F1. LUKE [53] reports 94.3 F1, but does not give standard errors or mention whether the result is a mean of various runs. Zhou and Chen [61] rerun LUKE and report a median of 93.9 F1 across 5 independent runs.

Out of the 16 baselines in this benchmark, only 3 report standard errors. I want to emphasize that most models experience notable performance variations when trained multiple times, even under identical settings. For a competitive benchmark such as CoNLL03, these differences have substantial consequences for the ranking of an approach on the leaderboard. Therefore, I strongly urge the NLP research community to report means and standard errors of sufficiently many runs to allow for proper comparison and evaluation.

Due to the lack of means and standard errors of many approaches, it is challenging to correctly compare them. However, given the averaged results from Zhou and Chen [61], it seems very likely that SpanMarker outperforms all competitors for CoNLL03 with document-level context.

	CoNLL03 [43]		
	Model	Encoder	F1
no document context	DiffusionNER [39]*	BERT_{large}	92.8
	Flair [1]*	GLoVe [31]	93.1±0.1
	Dep. Parsing [56]*	BERT_{large}	93.5
	spaCy [13]	RoBERTa_{base}	91.6
	Stanza [34]	word2vec [27]	92.1
	LUKE [53]†	RoBERTa_{large}	92.4
	tner [45]	RoBERTa_{large}	92.5
	FLERT [36]	$\text{XLM-RoBERTa}_{large}$	92.8±0.1
	CL-KL [49]⁂	$\text{XLM-RoBERTa}_{large}$	93.2
	SpanMarker	$\text{XLM-RoBERTa}_{large}$	92.9±0.0
document context	PIQN [38]*	BERT_{large}	92.9
	BERT [9]	BERT_{large}	92.8
	ASP [22]	T5_{large}	92.8
	XLM-RoBERTa [8]	$\text{XLM-RoBERTa}_{base}$	92.3
	XLM-RoBERTa [8]	$\text{XLM-RoBERTa}_{large}$	92.9
	BINDER [57]	RoBERTa_{large}	93.3
	Boundary Smoothing [62]	RoBERTa_{base}	93.7
	FLERT [36]	$\text{XLM-RoBERTa}_{large}$	93.8±0.2
	LUKE [53]‡	RoBERTa_{large}	93.9
	LUKE [53]	RoBERTa_{large}	**94.3**
	PL-Marker [55]	RoBERTa_{large}	94.0±0.1
	SpanMarker	$\text{XLM-RoBERTa}_{large}$	94.1±0.1

Table 3.2: F1 score (%) on the test set of CoNLL03. Bold indicates the highest reported number, regardless of statistical significance. †: mean F1 across 5 runs from Wang et al. [49], ‡: median F1 across 5 runs from Zhou and Chen [61], *: trained on training and evaluation set, ⁂: trained using external contexts.

Table 3.3 shows the evaluation results on CoNLL++, indicating that SpanMarker is only outperformed by LUKE. SpanMarker reaches 95.27±0.08 F1 with 95.74±0.09 precision and 94.81±0.08 recall, with individual models scoring between 94.89 and 95.59 F1. The upper bound of the individual SpanMarker models reach the median performance of LUKE for this benchmark.

	CoNLL++ [50]		
	Model	Encoder	F1
document context	LSTM-CRF [17]†	word2vec	91.47±0.15
	BiLSTM-CNN-CRF [26]†	Senna [7], word2vec	91.87±0.50
	BiLSTM-CRF+ELMo [32]†	Senna	93.42±0.15
	Flair [1]†	GLoVe	93.89±0.06
	Pooled-Flair [1]†	GLoVe	94.13±0.11
	CL-KL [49]	$\text{XLM-RoBERTa}_{large}$	94.81
	LUKE [53]‡	RoBERTa_{large}	**95.60**
	SpanMarker	$\text{XLM-RoBERTa}_{large}$	95.27±0.08

Table 3.3: F1 score (%) on the test set of CoNLL++. †: reported from Wang et al. [50], ‡: median F1 across 5 runs from Zhou and Chen [61].

3.2.2 Results on OntoNotes v5.0

Table 3.4 displays the experimental results on OntoNotes v5.0, which indicate that SpanMarker falls behind two baselines for this benchmark. Notably, the Boundary Smoothing (BS) approach is superior to SpanMarker on this benchmark, while SpanMarker outperforms BS on the CoNLL03 benchmark. The precision of SpanMarker has declined compared to PL-Marker, and although the exact cause is unclear, it warrants further investigation. The individual models score between 91.17 and 91.53 F1.

Once again, few approaches report standard errors alongside the mean. This hinders the evaluation of statistical significance and compromises the reliability of the reported results.

		OntoNotes v5.0 [51]		
Model	Encoder	Prec.	Rec.	F1
spaCy [13]	RoBERTa$_{base}$	-	-	89.8
Biaffine-NER [56]†	BERT$_{large}$	89.74	89.92	89.83
BERT-MRC [19]†	BERT$_{large}$	91.34	88.39	89.84
BARTNER [54]	BART$_{large}$	89.99	90.77	90.38
tner [45]	RoBERTa$_{large}$	90.51	91.21	90.86
FLERT [36]	XLM-RoBERTa$_{large}$	-	-	90.93
PIQN [38]	BERT$_{large}$	91.43	90.73	90.96
Dep. Parsing [56]	BERT$_{large}$	91.1	91.5	91.3
Boundary Smoothing [62]	RoBERTa$_{base}$	-	-	91.74
PL-Marker [55]	RoBERTa$_{large}$	**92.0**	91.7	**91.9**$\pm$0.1
SpanMarker	RoBERTa$_{large}$	90.96$\pm$0.06	**91.75$\pm$0.07**	91.35$\pm$0.06

Table 3.4: F1 score (%) on the test set of OntoNotes v5.0. †: reported from Yan et al. [54].

3.2.3 Results on FewNERD

Table 3.5 presents the evaluation results on the supervised FewNERD dataset. SpanMarker using RoBERTa$_{large}$ demonstrates comparable performance to PL-Marker, outperforming all other baseline models. The F1 scores achieved by the individual models range from 70.85 to 71.03 F1, with a 0.0187 standard error, denoting much more consistent results than PL-Marker.

Additionally, SpanMarker has been trained with the smaller BERT$_{base}$ encoder to allow proper comparisons against the other baselines. Using this encoder, SpanMarker reaches 70.44$\pm$0.0179 F1. The individual models range between 70.36 and 70.55 F1, demonstrating that SpanMarker consistently and considerably outperforms the other BERT$_{base}$ baselines, even in the worst-case scenario.

		FewNERD [10]		
Model	Encoder	Prec.	Rec.	F1
BERT-Tagger [10]	BERT$_{base}$	65.56	68.87	67.13
Locate&Label [37]†	BERT$_{base}$	64.69	70.87	67.64
Seq2Set [42]†	BERT$_{base}$	67.37	69.12	68.23
PIQN [38]	BERT$_{base}$	70.16	69.18	69.67
PL-Marker [55]	RoBERTa$_{large}$	71.2	70.6	70.9$\pm$0.1
SpanMarker	BERT$_{base}$	70.92$\pm$0.03	69.97$\pm$0.03	70.44$\pm$0.02
SpanMarker	RoBERTa$_{large}$	**71.21$\pm$0.04**	**70.65$\pm$0.04**	**70.93$\pm$0.02**

Table 3.5: F1 score (%) on the test set of FewNERD. †: reported results from Shen et al. [38].

3.3 Training Time

For NER models to be practical and appealing to practitioners, it is crucial that they can be trained within a reasonable timeframe using consumer-grade hardware. Table 3.6 highlights that SpanMarker achieves competitive state-of-the-art performance while requiring minimal training time, making it an attractive option for practitioners.

	CoNLL03 (no context)	CoNLL03 (context)	OntoNotes	FewNERD (RoB.$_{large}$)	FewNERD (BERT$_{base}$)
Samples/sec	20.16	13.35	25.48	24.03	235.43
Input size	256	768	512	512	512
Max entity len.	6	8	10	8	8
Num. epochs	3	3	4	3	3
Batch size	4	4	8	8	32
Training time					
- SpanMarker	0:46:13±00:04	1:03:43±00:12	3:13:11±01:14	5:35:05±01:57	1:39:6±00:07
- PL-Marker[1]	-	∼2:27:00	-	∼10:48:00	-
- LUKE[2]	-	∼2:01:00	-	-	-

Table 3.6: Metrics relating to training times for all experimental benchmarks. Results for CoNLL++ match that of CoNLL03 with document-level context. Input size is equivalent to `max_tokens + max_markers`, i.e. the size of the input passed to the underlying encoder.
[1]: The estimated PL-Marker training times are under the same parameters as SpanMarker and computed using the provided estimated total training time after 5 minutes of training.
[2]: The estimated LUKE training times are taken from the LUKE paper [53] and extrapolated down from 5 epochs to 3 epochs for fairer comparison. Note that significantly stronger hardware was used for LUKE: two Intel Xeon E5-2698 v4 CPUs and eight V100 GPUs.

Additionally, Table 3.3 SpanMarker shows significant improvements in training time efficiency compared to PL-Marker on both CoNLL03 and supervised FewNERD datasets. Specifically, for the CoNLL03 dataset, SpanMarker achieves a remarkable reduction in training time of 56.6%, equivalent to a speedup of 130.7%. Similarly, when applied to the supervised FewNERD dataset, SpanMarker showcases a substantial training time reduction of 48.2%, resulting in an impressive speedup of 93.3%.

SpanMarker even trains twice as fast as LUKE on CoNLL03, which was trained with considerably stronger hardware. These findings highlight that SpanMarker allows for faster training compared to prior research approaches without compromising on model performance.

The underlying encoder, number of epochs, batch size, model max length, marker max length and entity max length hyperparameters from Appendix C can be tuned to heavily impact the training time. This is evident from the speed difference of training SpanMarker on FewNERD with BERT$_{base}$ compared to RoBERTa$_{large}$. The ability to tune these hyperparameters allows practitioners to determine the tradeoff between training- and inference-time versus performance that best suits their use case.

3.4 Case Studies

Three text snippets inspired by various Wikipedia articles will be considered as case studies to compare SpanMarker with RoBERTa$_{large}$ trained on OntoNotes v5.0 against several commonly used spaCy models: `en_core_web_sm`, `en_core_web_lg` and `en_core_web_trf`. These text snippets were used in their original state to ensure a fair assessment of the performance of SpanMarker. Note that the spaCy `trf` model uses the smaller RoBERTa$_{base}$ encoder, and that the `sm` and `lg` models are convolutional neural networks.

SpanMarker	"Leonardo di ser Piero da Vinci (PERSON) painted the Mona Lisa (WORK_OF_ART) based on Italian (NORP) noblewoman Lisa del Giocondo (PERSON)."
spaCy (sm)	"Leonardo (PERSON) di ser Piero da Vinci (PERSON) painted the Mona Lisa based on Italian (NORP) noblewoman Lisa del Giocondo (PERSON)."
spaCy (lg)	"Leonardo (PERSON) di ser Piero da Vinci (PERSON) painted the Mona Lisa based on Italian (NORP) noblewoman Lisa del Giocondo (PERSON)."

Table 3.7: Case study of SpanMarker trained on OntoNotes v5.0 compared to spaCy with `en_core_web_sm` and `en_core_web_lg`. Not shown here, spaCy with `en_core_web_trf` gives equivalent results to SpanMarker. Red text indicates differences compared to reasonable human annotations.

In Table 3.7, the spaCy (`sm`, `lg`) models do not recognize "the Mona Lisa", nor that "Leonardo di ser Piero da Vinci" is one name. On the other hand, the output given by SpanMarker matches the gold annotations I would provide for this sample. For this case, SpanMarker is superior to spaCy `sm` and `lg`.

SpanMarker	"Amelia Earhart (PERSON) flew her single engine Lockheed (ORG) Vega 5B (PRODUCT) across the Atlantic (LOC) to Paris (GPE)."
spaCy (sm)	"Amelia Earhart (PERSON) flew her single engine Lockheed (ORG) Vega 5B across the Atlantic (LOC) to Paris (GPE)."
spaCy (lg)	"Amelia Earhart (PERSON) flew her single engine Lockheed Vega 5B (PRODUCT) across the Atlantic (LOC) to Paris (GPE)."

Table 3.8: Case study of SpanMarker trained on OntoNotes v5.0 compared to spaCy with `en_core_web_sm` and `en_core_web_lg`. Not shown here, spaCy with `en_core_web_trf` gives equivalent results to SpanMarker.

Table 3.8 shows that the SpanMarker and spaCy (`sm`, `lg`) models disagree on how to annotate "Lockheed Vega 5B". Both the output of SpanMarker and spaCy (`lg`) are sensible. SpanMarker surpasses spaCy `sm` in performance, and depending on personal preference, it also outperforms spaCy `lg`.

SpanMarker	"Cleopatra VII (PERSON), also known as Cleopatra the Great (PERSON), was the last active ruler of the Ptolemaic Kingdom of Egypt (GPE). She was born in 69 BCE (DATE) and ruled Egypt (GPE) from 51 BCE (DATE) until her death in 30 BCE (DATE)."
spaCy (sm)	"Cleopatra VII, also known as Cleopatra the Great (WORK_OF_ART), was the last active ruler of the Ptolemaic Kingdom of Egypt (GPE). She was born in 69 (CARDINAL) BCE (ORG) and ruled Egypt (GPE) from 51 (CARDINAL) BCE (ORG) until her death in 30 (CARDINAL) BCE (ORG)."
spaCy (lg)	"Cleopatra VII (PERSON), also known as Cleopatra the Great (WORK_OF_ART), was the last active ruler of the Ptolemaic Kingdom of Egypt (GPE). She was born in 69 BCE (TIME) and ruled Egypt (GPE) from 51 BCE (TIME) until her death in 30 BCE (TIME)."
spaCy (trf)	"Cleopatra VII (PERSON), also known as Cleopatra the Great (PERSON), was the last active ruler of the Ptolemaic Kingdom (GPE) of Egypt (GPE). She was born in 69 BCE (DATE) and ruled Egypt (GPE) from 51 BCE (DATE) until her death in 30 BCE (DATE)."

Table 3.9: Case study of SpanMarker trained on OntoNotes v5.0 compared to spaCy with `en_core_web_sm`, `en_core_web_lg` and `en_core_web_trf`.

Table 3.9 highlights a case with two sentences. The spaCy `sm` model struggles considerably on this case, missing entities, mispredicting a person as a work of art, and incorrectly considering dates as cardinals and organisations. The spaCy `lg` model performs better, but also considers "Cleopatra the Great" as a work of art and incorrectly predicts that the dates are times.

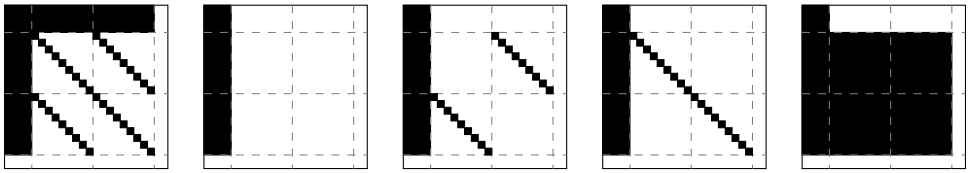
Lastly, the spaCy `trf` (RoBERTa) model performs similarly to SpanMarker, with only one difference: spaCy splits up "the Ptolemaic Kingdom of Egypt" into two GPE entities. Both the predictions of SpanMarker and spaCy (`trf`) are reasonable, although my gold annotations would match the SpanMarker output. Like before, SpanMarker outperforms spaCy `sm` and `lg`, and depending on preference, also the spaCy `trf` model.

The case studies presented offer further evidence that SpanMarker surpasses or demonstrates comparable performance to commonly employed approaches for NER.

3.5 Ablation Study and Experiments Best part that they shared: What didn't work

Critical sections of SpanMarker, such as the position IDs and attention mask matrix, are modified as an extensive ablation study. Additionally, approaches inspired by various previous works are experimented with. SpanMarker using BERT_{base} trained on FewNERD from Section 3.2.3 is used as a baseline, as that experiment yielded the lowest standard error. Three models are trained for the ablations and experiments, from which the mean and standard error are reported. Additionally, statistical significance tests are computed with $p < 0.05$ to determine whether the results are significantly worse or better than the baseline.

1. **Position IDs**: Section 2.2.3 deliberately corresponds the position IDs of marker tokens with the text tokens that the markers should "observe". For this ablation, the position IDs for all span markers are randomized.
2. **Attention Mask Matrix**: Section 2.2.5 introduces an intentionally designed attention mask matrix. This matrix allows the text to attend to other parts of the text, the markers to attend to the text, and the marker pairs to attend to their corresponding partners. In this ablation study, different modifications to the attention mask matrix are explored.



(a) Text token to marker attention. (b) No diagonal attentions. (c) No diagonal self-attention. (d) No diagonal cross-attention. (e) Full marker attention.

Figure 3.1: Variations of the attention mask matrix from Figure 2.3 to use in the ablation study.

(a) **Text tokens to marker attention**: As shown in Figure 3.1a, in this ablation the text tokens can attend to the marker tokens.

(b) **No diagonal attention**: As Figure 3.1b shows, for this ablation the markers are disallowed from attending to themselves or their corresponding pair partner.

(c) **No diagonal self-attention**: As indicated by Figure 3.1c, marker self-attention is disallowed in this ablation.

(d) **No diagonal cross-attention**: For this ablation, Figure 3.1d shows markers are disallowed from attending with their corresponding pair partner.

(e) **Full marker attention**: Lastly, as shown in Figure 3.1e, in this ablation all markers can attend to all markers.

3. **Span length embedding**: Previous studies concatenate a span length embedding to the span embedding [57, 60]. This approach allows the classifier to consider the number of words in the span as a relevant factor. For this ablation, a 128-dimensional span length embedding based on the span word count is added to the SpanMarker feature vector.

4. **PL-Marker**:

   (a) **Expanded feature vectors**: Section 2.3 shows that the SpanMarker feature vectors consists of a concatenation of the start marker embeddings and the end marker embeddings. PL-Marker also includes the start token embeddings and the end token embeddings in these feature vectors. In this experiment, SpanMarker adopts the extended feature vector that is used in PL-Marker.

   (b) **Span marker initialization**: In PL-Marker, the start and end marker tokens are initialized using the embeddings of the `[MASK]` and `entity` tokens, respectively. For this experiment, this initialization approach is adopted in SpanMarker.

	FewNERD [10]		
Model	Prec.	Rec.	F1
SpanMarker ($BERT_{base}$)	70.92±0.03	69.97±0.03	70.44±0.02
1 Random position IDs	0.00 †	0.00 †	0.00 †
2a Text tokens to marker attention	70.73±0.14 †	69.78±0.09 †	70.25±0.09 †
2b No diagonal attention	70.94±0.08	69.32±0.03 †	70.12±0.05 †
2c No diagonal self-attention	70.93±0.06	69.91±0.09	70.41±0.08
2d No diagonal cross-attention	70.95±0.04	69.45±0.03 †	70.19±0.03 †
2e Full marker attention	70.96±0.04	69.45±0.03 †	70.20±0.03 †
3 Span length embedding	70.95±0.07	70.01±0.05	70.48±0.04
4a Extended feature vector	70.85±0.02	69.96±0.08	70.40±0.04
4b Span marker initialization	70.90±0.03	69.88±0.04	70.39±0.03

Table 3.10: Ablation study comparing SpanMarker with $BERT_{base}$ on FewNERD. †: statistically significantly worse than the baseline with $p < 0.05$. None of the results from this table are statistically significantly better than the baseline with $p < 0.05$.

The ablation study results presented in Table 3.10 indicate that precise alignment between the position IDs of markers and the tokens they observe is crucial (1). Additionally, it shows that the model performs statistically significantly worse if the text tokens can observe the marker tokens (2a). Consequently, the one-directional marker to text attention is critical to reach the best performance.

Moreover, ablating all attention between all markers (2b) significantly reduced the recall, leading to significantly worse F1. The precision is notably unaffected. If only the self-attention (2c) is ablated, then the performance is similar to the baseline. However, between the three runs for this ablation, the results ranged between 70.32 and 70.56 F1, whereas the 10 baseline runs ranged between 70.36 and 70.55 F1. Further experimentation may be warranted, but it seems that the diagonal self-attention increases the stability and consistency of the results.

Ablating the diagonal cross-attention (2d) reduces the recall while keeping the precision the same, much like ablating all attention between markers. This is indicative that the attention from the marker to its pair partner is more important than the marker self-attention. Lastly, allowing attention between all markers (2e) seems to introduce too much noise, leading to similar results as not allowing attention between marker pairs at all.

As for the experiments, including a span length embedding (3) like prior work [57, 60] does not affect the performance, but does increase the training time by approximately 5%. This justifies the exclusion of such a span length embedding in SpanMarker. Similarly, the extended feature vector (4a) as used in PL-Marker does not impact performance, while it does increase the training time by roughly 9%.

Lastly, the span marker initialization (4b) from PL-Marker does not influence performance. My intuition is that using this style of initialization is equivalent in performance to SpanMarker its random initialization, but superior to using the embeddings from `madeupword0000` and `madeupword0001`, which is what PL-Marker was using as a baseline.

Throughout this ablation study, the position IDs and attention mask matrix are shown to play a crucial role in the performance of SpanMarker. Furthermore, these experiments have demonstrated that incorporating additional elements from previous work would only yield increased training time and model complexity, without resulting in performance improvements.

4. Conclusion

In this thesis, I present SpanMarker, a span-level Named Entity Recognition model designed to leverage special span marker token pairs with position IDs, and attention mask matrices from BERT-style encoders. SpanMarker adapts the PL-Marker [55] model with key modifications such as reducing token padding (Section 2.2.4) and simplifying the feature vectors (Section 2.3) in order to improve the computational efficiency.

The experimental results presented in Section 3.2 demonstrate that SpanMarker achieves state-of-the-art performance across various benchmark datasets, such as CoNLL03 and FewNERD. Additionally, Section 3.3 highlights the effectiveness of the improvements made to SpanMarker through the notable reduction in training time compared to PL-Marker.

SpanMarker has been released as a consumer-ready and user-friendly NER library, accompanied by training scripts and top-performing models for all benchmarks. The library has been integrated into the Hugging Face Hub, allowing SpanMarker models to be easily shared, tested and deployed into production.

Finally, this thesis emphasizes the significance of reporting means and standard errors as a crucial aspect of empirical evaluation.

5. SpanMarker Library

SpanMarker[1] has been released as a consumer-ready Python library that allows practitioners to train research-grade NER models with minimal effort. Crucially, SpanMarker can easily be initialized with any BERT-based encoder on the Hugging Face Hub, such as all BERT, RoBERTa or DeBERTa models[2]. Moreover, SpanMarker models can be trained using datasets annotated with different label schemes, such as IOB, IOB2, BIOES, BILOU, or no label scheme at all. Training and performing inference with document-level context is also supported without requiring text preprocessing.

Built on top of the familiar `transformers` [52] library, SpanMarker inherits a wide range of powerful functionalities, such as easily loading and saving models, hyperparameter optimization, automatic logging in various tools, checkpointing, callbacks, mixed precision training, 8 bit inference, and more.

Additionally, the SpanMarker library has been integrated with the Hugging Face Hub and the Hugging Face Inference API. For example, see the SpanMarker documentation on Hugging Face[3] or browse all SpanMarker models on the Hugging Face Hub[4]. Through the Inference API integration, users can test any SpanMarker model on the Hugging Face Hub for free using a widget on the model page[5]. Furthermore, each public SpanMarker model includes a free API for fast prototyping and can be deployed to production using Hugging Face Inference Endpoints.

[1] `https://github.com/tomaarsen/SpanMarkerNER`
[2] As long as the model supports position IDs and attention mask matrices.
[3] `https://huggingface.co/docs/hub/span_marker`
[4] `https://huggingface.co/models?library=span-marker`
[5] `https://huggingface.co/tomaarsen/span-marker-bert-base-fewnerd-fine-super`

6. Future Work

1. **Nested NER and Relationship Extraction (RE)**
   In future research, there is potential for extending SpanMarker to support nested NER and RE, taking inspiration from the achievements of PL-Marker. PL-Marker has demonstrated state-of-the-art performance in these tasks, and due to the similarity between SpanMarker and PL-Marker as displayed in Section 3.2, SpanMarker may also perform competitively.

   This extension is especially valuable as the codebase of PL-Marker lacks the flexibility for users to apply the approach on their own datasets, which is one of the aspects where SpanMarker excels.

2. **Candidate Span Filtering**
   As explained in Section 2.2.2, a span marker pair is created for every single span in the input text. Depending on the chosen maximum entity length, this leads to a large number of span markers, that may need to be distributed between multiple samples, as shown in Section 2.2.4. Future research could improve the computational efficiency of SpanMarker further by reducing the number of spans to classify. Several feasible approaches can be considered.

   (a) One potential approach is exact boundary identification. This decomposes the approach into two phases, where the first phase identifies spans that it believes are entities, and the latter classifies them. Notably, unlike the default SpanMarker implementation, the classifier here cannot classify the "`O`" (outside) label.

   Separate from the discussion on computational efficiency, this approach could be used with two SpanMarker models. Contrary to the normal SpanMarker approach, all non-outside labels for the boundary-identifying SpanMarker model could be normalized to "`ENTITY`". When trained like normal, this SpanMarker model returns a list of all spans it considers entities.

   The classification SpanMarker model exclusively considers these few spans, each of which it will classify as one of the original labels, except "`O`". Conceptually, this approach allows all spans to be classified in one sample, rather than spread out between multiple samples. Although decomposed approaches like this often perform worse, Ye et al. [55] showed that packing more markers into one sample leads to improved entity recognition performance.

   (b) Another approach is high recall span identification, which aims to identify a subset of spans that maximizes the inclusion of spans corresponding to entities while minimizing the size of the subset. If the number of selected spans can be decreased significantly while retaining a large portion of the spans that correspond to entities, then it is reasonable to anticipate an improvement in computational efficiency at a small cost in performance.

   In the context of SpanMarker, this can again be done by employing two SpanMarker models. The first one may use a small encoder, like the tiny BERT [3, 44] encoder, and considers the same binarized "`O`" and "`ENTITY`" labels as with the first approach. Unlike before, the "`ENTITY`" class is given a higher weighing in the cross-entropy loss, causing the model to favour recall over precision. This weight allows for tuning between high recall and small subset size. Preliminary experiments resulted in a recall of 99 (%) with a subset half of the original span set, in just a few minutes of training.

Intuitively, this should allow for slightly faster training and inference at the cost of 1% of performance.

3. **Analysis of Hyperparameter Effects on Computational Efficiency**
   As described in 2.2.4, span markers are distributed across multiple samples to prevent individual samples from becoming too long. This helps with computational efficiency due to the quadratic time complexity of the BERT-style encoder. However, shrinking the area dedicated to span markers too much may mean that an exorbitant number of samples need to be created to distribute all span markers, especially for longer input texts.

   This results in a complex relationship between the maximum entity length, maximum token length, maximum marker length, and the computational efficiency. Further research could investigate this relationship and, given a dataset, suggest recommended hyperparameters for optimal computational efficiency without a reduction in performance.

A. Example Spans

```
1  Andorra
2  Andorra is
3  Andorra is located
4  Andorra is located between
5  Andorra is located between France
6  Andorra is located between France and
7  Andorra is located between France and Spain
8  Andorra is located between France and Spain.
9  is
10 is located
11 is located between
12 is located between France
13 is located between France and
14 is located between France and Spain
15 is located between France and Spain.
16 located
17 located between
18 located between France
19 located between France and
20 located between France and Spain
21 located between France and Spain.
22 between
23 between France
24 between France and
25 between France and Spain
26 between France and Spain.
27 France
28 France and
29 France and Spain
30 France and Spain.
31 and
32 and Spain
33 and Spain.
34 Spain
35 Spain.
36 .
```

Listing A.1: All possible spans (or n-grams) in Example 2.2.1.

B. Datasets

B.1 CoNLL03

The CoNLL03 dataset [43], derived from a shared task on language-independent named entity recognition (NER), has become the most widely utilized dataset for NER tasks, particularly in English. The dataset specifically focuses on four categories of named entities: persons, locations, organizations, and miscellaneous entities not belonging to the aforementioned groups.

The English portion of the dataset was sourced from the Reuters Corpus, which consists of a collection of news stories. The training, validation and testing splits were taken from different timespans of this corpus.

B.2 CoNLL++

The CoNLL++ dataset, proposed by Wang et al. [50], serves as an amendment to the CoNLL03 dataset. In their work, the authors discovered labeling errors in approximately 5.38% of all test sentences, which is noteworthy considering that the state-of-the-art models achieve 94 F1 [53, 55].

By rectifying these labeling mistakes through manual correction of the test set, the CoNLL++ dataset aims to provide a more accurate and reliable benchmark for evaluating the performance of named entity recognition models. As shown in Table 3.1, the two datasets are very similar, with the only differences being in the test set.

B.3 OntoNotes v5.0

The OntoNotes v5.0 dataset [51] is a large-scale, multilingual dataset consisting of numerous genres of text (news, conversational telephone speech, weblogs, usenet newsgroups, broadcast, talk shows) in multiple languages. The dataset contains 18 different entity types: `CARDINAL`, `DATE`, `PERSON`, `NORP` (nationalities or religious or political groups), `GPE` (geopolitical entities), `LAW`, `PERCENT`, `ORDINAL`, `MONEY`, `WORK_OF_ART`, `FAC` (buildings, airports, highways, bridges, etc.), `TIME`, `QUANTITY`, `PRODUCT`, `LANGUAGE`, `ORG` (organisations), `LOC` (locations), `EVENT`. The diversity in both text genres and entity types makes it an attractive benchmark.

B.4 FewNERD

FewNERD, also known as the Few-shot Named Entity Recognition Dataset [10], is a large-scale annotated dataset primarily designed for few-shot NER tasks, although it can also be utilized for standard supervised NER. It stands out from older benchmarks such as CoNLL03 and OntoNotes v5.0 by offering both coarse-grained and fine-grained entity types, providing a more nuanced entity classification.

The dataset covers 8 coarse-grained categories: `art`, `building`, `event`, `location`, `organization`, `other`, `person`, and `product`. Each of these coarse-grained types include various fine-grained subcategories such as `art-broadcastprogram`, `art-film`, `art-music`, `art-other`, `art-painting`, `art-writtenart`, and many more. In total, FewNERD consists of 66 fine-grained types, which often require a comprehensive understanding of contextual cues to accurately distinguish between them. The full 66 fine-grained types are used when evaluating with this dataset.

Derived from human annotations of Wikipedia pages, FewNERD offers a substantial dataset for training and evaluating NER models. The inclusion of fine-grained entity types enriches the complexity of the task and encourages models to capture subtle contextual nuances.

C. Implementation Details

Please refer to Table C.1 and Table C.2 for the hyperparameters on the CoNLL03 and CoNLL++ experiments and the OntoNotes v5.0 and FewNERD experiments, respectively. The highest scoring models for each experiment have been published on the Hugging Face Hub, the links of which are in Table C.3. Each model also includes a simplified training script that is equivalent to the one used for training the respective model. All of these training scripts are also uploaded to GitHub[1].

Please take note that `model_max_length` corresponds to the maximum token length, while `marker_max_length` is half of the maximum marker length as described in Section 2.2.4. `entity_max_length` is the maximum number of words that the model will consider a candidate span.

Hyperparameter	Document context	No document context
SpanMarker version	1.1	1.1
Encoder	`xlm-roberta-large`	`xlm-roberta-large`
`model_max_length`	512	128
`marker_max_length`	128	64
`entity_max_length`	8	6
`max_prev_context`	`None`	`None`
`max_next_context`	`None`	`None`
Optimizer	AdamW [24]	AdamW
Scheduler	Linear	Linear
Learning rate	1e-5	1e-5
Batch size	4	4
Gradient accumulation steps	2	2
Number of epochs	3	3
Weight decay	0.01	0.01
Warmup ratio	0.1	0.1
bf16	`True`	`True`

Table C.1: Hyperparameters used for the experiments with CoNLL03 and CoNLL++. The top parameters are passed to the `SpanMarkerModel.from_pretrained` method, while the bottom parameters are provided to `TrainingArguments` [52]. Unspecified hyperparameters take the defaults from `TrainingArguments` at version 4.28.1 of `transformers`.

[1] `https://github.com/tomaarsen/SpanMarkerNER/tree/main/training_scripts`

Hyperparameter	OntoNotes v5.0	FewNERD (large)	FewNERD (base)
SpanMarker version	1.1	1.1	1.1
Encoder	`roberta-large`	`roberta-large`	`bert-base`
`model_max_length`	256	256	256
`marker_max_length`	128	128	128
`entity_max_length`	10	8	8
`max_prev_context`	`None`	`None`	`None`
`max_next_context`	`None`	`None`	`None`
Optimizer	AdamW	AdamW	AdamW
Scheduler	Linear	Linear	Linear
Learning rate	1e-5	1e-5	5e-5
Batch size	8	8	32
Gradient accumulation steps	2	1	1
Number of epochs	4	3	3
Weight decay	0.01	0.01	0.01
Warmup ratio	0.1	0.1	0.1
bf16	`True`	`True`	`True`

Table C.2: Hyperparameters used for the experiments with OntoNotes 5.0 and FewNERD. The top parameters are passed to the `SpanMarkerModel.from_pretrained` method, while the bottom parameters are provided to `TrainingArguments` [52]. Unspecified hyperparameters take the defaults from `TrainingArguments` at version 4.28.1 of `transformers`.

Experiment	**Link to best model**
CoNLL03 (no context)	tomaarsen/span-marker-xlm-roberta-large-conll03
CoNLL03 (context)	tomaarsen/span-marker-xlm-roberta-large-conll03-doc-context
CoNLL++ (context)	tomaarsen/span-marker-xlm-roberta-large-conllpp-doc-context
OntoNotes v5.0	tomaarsen/span-marker-roberta-large-ontonotes5
Fine-grained FewNERD	tomaarsen/span-marker-bert-base-fewnerd-fine-super
Fine-grained FewNERD	tomaarsen/span-marker-roberta-large-fewnerd-fine-super

Table C.3: The published best models for each set of experiments.

Bibliography

[1] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649. Association for Computational Linguistics, August 2018. URL https://aclanthology.org/C18-1139.

[2] Livio Baldini Soares, Nicholas FitzGerald, Jeffrey Ling, and Tom Kwiatkowski. Matching the blanks: Distributional similarity for relation learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2895–2905. Association for Computational Linguistics, July 2019. URL https://aclanthology.org/P19-1279.

[3] Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. Generalization in NLI: Ways (not) to go beyond simple heuristics. In *Proceedings of the Second Workshop on Insights from Negative Results in NLP*, pages 125–135. Association for Computational Linguistics, November 2021. URL https://aclanthology.org/2021.insights-1.18.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. URL https://aclanthology.org/Q17-1010.

[5] Jason P.C. Chiu and Eric Nichols. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016. URL https://aclanthology.org/Q16-1026.

[6] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators, 2020. URL https://arxiv.org/abs/2003.10555.

[7] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch, 2011. URL https://arxiv.org/abs/1103.0398.

[8] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451. Association for Computational Linguistics, July 2020. URL https://aclanthology.org/2020.acl-main.747.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, June 2019. URL https://aclanthology.org/N19-1423.

[10] Ning Ding, Guangwei Xu, Yulin Chen, Xiaobin Wang, Xu Han, Pengjun Xie, Haitao Zheng, and Zhiyuan Liu. Few-NERD: A few-shot named entity recognition dataset. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3198–3213. Association for Computational Linguistics, August 2021. URL https://aclanthology.org/2021.acl-long.248.

[11] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. DeBERTa: Decoding-enhanced bert with disentangled attention, 2021. URL https://arxiv.org/abs/2006.03654.

[12] Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing, 2023. URL https://arxiv.org/abs/2111.09543.

[13] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python. 2020. URL https://spacy.io.

[14] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging, 2015. URL https://arxiv.org/abs/1508.01991.

[15] Zhengbao Jiang, Wei Xu, Jun Araki, and Graham Neubig. Generalizing natural language analysis through span-relation representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2120–2133. Association for Computational Linguistics, July 2020. URL https://aclanthology.org/2020.acl-main.192.

[16] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. 2016. URL https://arxiv.org/abs/1612.03651.

[17] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270. Association for Computational Linguistics, June 2016. URL https://aclanthology.org/N16-1030.

[18] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880. Association for Computational Linguistics, July 2020. URL https://aclanthology.org/2020.acl-main.703.

[19] Xiaoya Li, Jingrong Feng, Yuxian Meng, Qinghong Han, Fei Wu, and Jiwei Li. A unified MRC framework for named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5849–5859. Association for Computational Linguistics, July 2020. URL https://aclanthology.org/2020.acl-main.519.

[20] Xiaoya Li, Xiaofei Sun, Yuxian Meng, Junjun Liang, Fei Wu, and Jiwei Li. Dice loss for data-imbalanced NLP tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 465–476. Association for Computational Linguistics, July 2020. URL https://aclanthology.org/2020.acl-main.45.

[21] Yongqi Li and Tieyun Qian. Type-aware decomposed framework for few-shot named entity recognition, 2023. URL https://arxiv.org/abs/2302.06397.

[22] Tianyu Liu, Yuchen Eleanor Jiang, Nicholas Monath, Ryan Cotterell, and Mrinmaya Sachan. Autoregressive structured prediction with language models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 993–1005. Association for Computational Linguistics, December 2022. URL https://aclanthology.org/2022.findings-emnlp.70.

[23] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach, 2019. URL https://arxiv.org/abs/1907.11692.

[24] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[25] Yi Luan, Dave Wadden, Luheng He, Amy Shah, Mari Ostendorf, and Hannaneh Hajishirzi. A general framework for information extraction using dynamic span graphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3036–3046. Association for Computational Linguistics, June 2019. URL https://aclanthology.org/N19-1308.

[26] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074. Association for Computational Linguistics, August 2016. URL https://aclanthology.org/P16-1101.

[27] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL https://arxiv.org/abs/1301.3781.

[28] George A. Miller. WordNet: A lexical database for English. In *Human Language Technology: Proceedings of a Workshop*, March 1994. URL https://aclanthology.org/H94-1111.

[29] Hiroki Nakayama. seqeval: A python framework for sequence labeling evaluation, 2018. URL https://github.com/chakki-works/seqeval.

[30] Hiroki Ouchi, Jun Suzuki, Sosuke Kobayashi, Sho Yokoi, Tatsuki Kuribayashi, Ryuto Konno, and Kentaro Inui. Instance-based learning of span representations: A case study through named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6452–6459. Association for Computational Linguistics, July 2020. URL https://aclanthology.org/2020.acl-main.575.

[31] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, October 2014. doi: 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

[32] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics, June 2018. URL https://aclanthology.org/N18-1202.

[33] Matthew E. Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54. Association for Computational Linguistics, November 2019. URL https://aclanthology.org/D19-1005.

[34] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 101–108. Association for Computational Linguistics, July 2020. URL https://aclanthology.org/2020.acl-demos.14.

[35] Lance Ramshaw and Mitch Marcus. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*, 1995. URL https://aclanthology.org/W95-0107.

[36] Stefan Schweter and Alan Akbik. FLERT: Document-level features for named entity recognition, 2021. URL https://arxiv.org/abs/2011.06993.

[37] Yongliang Shen, Xinyin Ma, Zeqi Tan, Shuai Zhang, Wen Wang, and Weiming Lu. Locate and label: A two-stage identifier for nested named entity recognition. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2782–2794. Association for Computational Linguistics, August 2021. URL https://aclanthology.org/2021.acl-long.216.

[38] Yongliang Shen, Xiaobin Wang, Zeqi Tan, Guangwei Xu, Pengjun Xie, Fei Huang, Weiming Lu, and Yueting Zhuang. Parallel instance query network for named entity recognition. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 947–961. Association for Computational Linguistics, May 2022. URL https://aclanthology.org/2022.acl-long.67.

[39] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. DiffusionNER: Boundary diffusion for named entity recognition, 2023. URL https://arxiv.org/abs/2305.13298.

[40] Mohammad Golam Sohrab and Makoto Miwa. Deep exhaustive model for nested named entity recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2843–2849. Association for Computational Linguistics, October-November 2018. URL https://aclanthology.org/D18-1309.

[41] Chuanqi Tan, Wei Qiu, Mosha Chen, Rui Wang, and Fei Huang. Boundary enhanced neural span classification for nested named entity recognition. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):9016–9023, Apr. 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6434.

[42] Zeqi Tan, Yongliang Shen, Shuai Zhang, Weiming Lu, and Yueting Zhuang. A sequence-to-set network for nested named entity recognition, 2021. URL https://arxiv.org/abs/2105.08901.

[43] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003. URL https://aclanthology.org/W03-0419.

[44] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. 2019. URL http://arxiv.org/abs/1908.08962.

[45] Asahi Ushio and Jose Camacho-Collados. T-NER: An all-round python library for transformer-based named entity recognition. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 53–62. Association for Computational Linguistics, April 2021. URL https://aclanthology.org/2021.eacl-demos.7.

[46] Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Jianshu Ji, Guihong Cao, Daxin Jiang, and Ming Zhou. K-Adapter: Infusing Knowledge into Pre-Trained Models with Adapters. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1405–1418. Association for Computational Linguistics, August 2021. URL https://aclanthology.org/2021.findings-acl.121.

[47] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021. URL https://aclanthology.org/2021.tacl-1.11.

[48] Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. Automated concatenation of embeddings for structured prediction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2643–2660. Association for Computational Linguistics, August 2021. URL https://aclanthology.org/2021.acl-long.206.

[49] Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. Improving named entity recognition by external context retrieving and cooperative learning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1800–1812. Association for Computational Linguistics, August 2021. URL https://aclanthology.org/2021.acl-long.142.

[50] Zihan Wang, Jingbo Shang, Liyuan Liu, Lihao Lu, Jiacheng Liu, and Jiawei Han. CrossWeigh: Training named entity tagger from imperfect annotations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5154–5163. Association for Computational Linguistics, November 2019. URL https://aclanthology.org/D19-1519.

[51] Ralph Weischedel, Martha Palmer, Mitchell Marcus, Hovy Eduard, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, and Ann Houston. OntoNotes Release 5.0, 2022. URL https://doi.org/10.5683/SP2/KPKFPI.

[52] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics, October 2020. URL https://aclanthology.org/2020.emnlp-demos.6.

[53] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. LUKE: Deep contextualized entity representations with entity-aware self-attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6442–6454. Association for Computational Linguistics, November 2020. URL https://aclanthology.org/2020.emnlp-main.523.

[54] Hang Yan, Tao Gui, Junqi Dai, Qipeng Guo, Zheng Zhang, and Xipeng Qiu. A unified generative framework for various NER subtasks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5808–5822. Association for Computational Linguistics, August 2021. URL https://aclanthology.org/2021.acl-long.451.

[55] Deming Ye, Yankai Lin, Peng Li, and Maosong Sun. Packed levitated marker for entity and relation extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4904–4917. Association for Computational Linguistics, May 2022. URL https://aclanthology.org/2022.acl-long.337.

[56] Juntao Yu, Bernd Bohnet, and Massimo Poesio. Named entity recognition as dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6470–6476. Association for Computational Linguistics, July 2020. URL https://aclanthology.org/2020.acl-main.577.

[57] Sheng Zhang, Hao Cheng, Jianfeng Gao, and Hoifung Poon. Optimizing bi-encoder for named entity recognition via contrastive learning. 2022. URL https://arxiv.org/abs/2208.14565.

[58] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451. Association for Computational Linguistics, July 2019. URL https://aclanthology.org/P19-1139.

[59] Changmeng Zheng, Yi Cai, Jingyun Xu, Ho-fung Leung, and Guandong Xu. A boundary-aware neural model for nested named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 357–366. Association for Computational Linguistics, November 2019. URL https://aclanthology.org/D19-1034.

[60] Zexuan Zhong and Danqi Chen. A frustratingly easy approach for entity and relation extraction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 50–61. Association for Computational Linguistics, June 2021. URL https://aclanthology.org/2021.naacl-main.5.

[61] Wenxuan Zhou and Muhao Chen. Learning from noisy labels for entity-centric information extraction. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5381–5392. Association for Computational Linguistics, November 2021. URL https://aclanthology.org/2021.emnlp-main.437.

[62] Enwei Zhu and Jinpeng Li. Boundary smoothing for named entity recognition. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7096–7108. Association for Computational Linguistics, May 2022. URL https://aclanthology.org/2022.acl-long.490.