Decoding Text Spans for Efficient and Accurate Named-Entity Recognition

Andrea Maracani, Savas Ozkan, Junyi Zhu, Sinan Mutlu, Mete Ozay
Samsung Research UK, Staines-upon-Thames, United Kingdom
{a.maracani, savas.ozkan, junyi.zhu, s.mutlu, m.ozay}@samsung.com

Abstract

Named Entity Recognition (NER) is a key component in industrial information extraction pipelines, where systems must satisfy strict latency and throughput constraints in addition to strong accuracy. State-of-the-art NER accuracy is often achieved by span-based frameworks, which construct span representations from token encodings and classify candidate spans. However, many span-based methods enumerate large numbers of candidates and process each candidate with marker-augmented inputs, substantially increasing inference cost and limiting scalability in large-scale deployments. In this work, we propose SpanDec, an efficient span-based NER framework that targets this bottleneck. Our main insight is that span representation interactions can be computed effectively at the final transformer stage, avoiding redundant computation in earlier layers via a lightweight decoder dedicated to span representations. We further introduce a span filtering mechanism during enumeration to prune unlikely candidates before expensive processing. Across multiple benchmarks, SpanDec matches competitive span-based baselines while improving throughput and reducing computational cost, yielding a better accuracy–efficiency trade-off suitable for high-volume serving and on-device applications.

1 Introduction

Named Entity Recognition (NER) is a core component of industrial information extraction pipelines, enabling applications such as clinical and biomedical coding, legal document parsing, and high-volume email understanding for reminder and task generation (Francis et al., 2019; Dozier et al., 2010; Anupama M. Nair, 2020). In production settings, NER models are often deployed as always-on services and must satisfy strict latency and throughput targets under cost constraints (e.g., GPU/CPU budget and total cost of ownership), making *accuracy–efficiency trade-offs* at least as important as peak F1/accuracy (Wang et al., 2020).

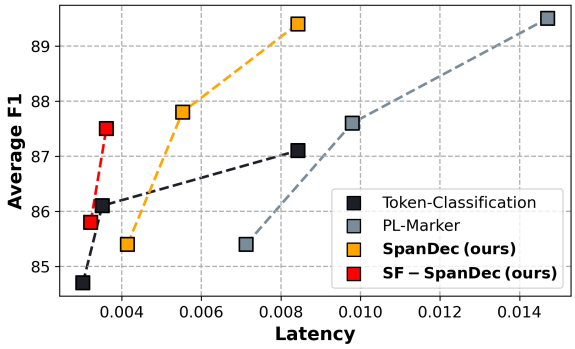

Figure 1: **F1 scores vs. Latency**. Our methods achieve a better accuracy–efficiency trade-off under realistic serving constraints. Same color markers represent encoder models of different sizes (Details in Sec. 6).

A widely used baseline is token classification with pre-trained transformer encoders, which performs per-token labeling with a lightweight prediction head and offers strong accuracy after fine-tuning (Devlin et al., 2019). While effective, token-level labeling can struggle to represent complete entity spans for ambiguous boundaries (and is less naturally compatible with span-level reasoning), motivating span-based formulations that explicitly construct and classify candidate spans (Zhong and Chen, 2021; Wang et al., 2021). Among span-based approaches, Ye et al. (2022) propose *Packed Levitated Markers* (PL-Marker), which inserts marker token pairs to focus the encoder on span representations and has become a strong and widely-adopted backbone for span-based NER (Ye et al., 2022).

Despite their accuracy benefits, marker-based span methods introduce non-trivial serving overhead. In contrast to token classification, which does not require enumerating and processing large numbers of candidate spans, span-based methods can consider $O(L^2)$ candidates, and even variants that cap the maximum span length may still incur substantial extra computation due to repeated process-

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track), pages 1025–1036

ing of marker-augmented inputs (see Sec. 3). In large-scale deployments, this overhead can become the dominant bottleneck and can force practitioners to reduce batch sizes, shorten context windows, or fall back to simpler models.

Recent LLM-based approaches also tackle NER via prompting or text generation, sometimes offering strong generalization and low-label adaptability; however, their autoregressive decoding and large parameter footprints typically yield prohibitively high latency and cost for high-throughput serving (Wang et al., 2025, 2023).

In this work, we target the practical bottleneck of PL-Marker-style NER and propose two complementary optimizations that preserve the modeling benefits of span representations while substantially reducing inference cost:

- **Decoupled span processing (`SpanDec`).** To reduce redundant computation, we decouple span-marker processing from the main encoder. Concretely, we replace the final encoder layer used by PL-Marker with a lightweight decoder dedicated to span representations (see Fig. 2c). The text tokens are encoded once as usual, while span-marker interactions are handled only in the decoder, avoiding repeated processing of marker tokens in earlier layers. We keep the overall parameter budget comparable to the original model by trading an encoder layer for the decoder. `SpanDec` delivers significant throughput gains without degrading accuracy (Sec. 4).

- **Early span filtering (`SF-SpanDec`).** We further improve end-to-end efficiency with a simple and effective span filtering mechanism: we train a lightweight binary classifier to predict tokens that contain no entity mentions (see Fig. 2d). At inference time, `SF-SpanDec` prunes the candidate set aggressively, often retaining only a small fraction of spans (e.g., $\sim$15%),so the decoder processes only likely entity spans (Sec. 4.1).

Across multiple NER benchmarks, our approach matches or exceeds the accuracy of PL-Marker while increasing throughput by up to 2.7$\times$ and reducing computation (GFLOPs) by up to 8.2$\times$. Moreover, compared to standard token classification, our span-centric approach achieves higher average F1 (+1.8%) while remaining viable under high-throughput serving constraints.

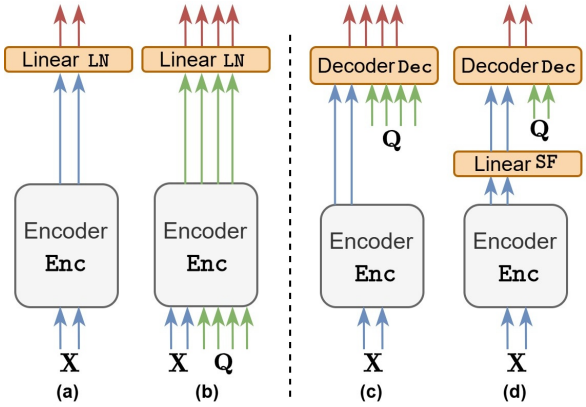

Figure 2: **Diagram of Model Flows. (a)** Token classification approach, **(b)** span classification approach, **(c)** `SpanDec` (ours), and **(d)** `SF-SpanDec` (ours).

2 Related Work

Table 1 summarizes common NER paradigms through an industrial lens, highlighting that span- and marker-based methods can improve boundary modeling at the cost of latency and scalability, while LLM-based prompting offers flexibility but remains expensive for high-throughput deployments.

Token classification for NER. The dominant practical baseline for NER is token-level sequence labeling with a transformer encoder and a lightweight classification head (Devlin et al., 2019). Token classification is appealing in production because it scales linearly with input length and maps cleanly to efficient batching and serving. Recent work also explores task-specialized encoders trained or adapted for NER, including pre-training with large-scale (possibly LLM-assisted) NER-style signals to improve data efficiency (Bogdanov et al., 2024).

Span-based and marker-based NER. Span-based NER replaces per-token labeling with explicit span enumeration and span classification, which can better capture whole-entity boundaries and support joint reasoning across spans (Zhong and Chen, 2021; Wang et al., 2021). Classic span-centric IE frameworks such as DyGIE++ enumerate candidate spans and iteratively refine span representations using contextual signals (Wadden et al., 2019). PL-Marker strategically inserts levitated markers to obtain stronger span representations from pre-trained encoders and has become a strong backbone for span-style NER (Ye et al., 2022). While effective, these approaches can incur sub-

Paradigm	Method example	Acc. ↑	Latency ↓	Scalability ↑	Typical deployment issue
Token classification	BERT tagging (2019)	Med	Low	High	Boundary / nesting limitations; weaker span modeling
Span enumeration	DyGIE++ (2019)	High	High	Low	Quadratic candidates; heavy compute/memory for long inputs
Marker-based span (encoder)	PL-Marker (2022)	High	High	Med–Low	Markers processed in all encoder layers; latency bottleneck
Generative, prompted (LLM)	InstructUIE (2023)	Med–High	V.High	Low	Autoregressive decoding cost; controllability / hallucination; cost
Decoupled span decoding (ours)	`SpanDec`, `SF-SpanDec`	High	Low	High	Extra module integration; thresholding trade-off

Table 1: Industrial view of NER paradigms: qualitative accuracy–latency–scalability trade-offs. Arrows indicate the preferred direction for deployment.

stantial computation due to (i) the number of candidate spans and (ii) repeated processing of marker-augmented inputs.

Improving efficiency in span-based extraction. Efficiency constraints have motivated both architectural and algorithmic changes. A common direction is to reduce the cost of the underlying encoder via compression techniques such as knowledge distillation (e.g., MiniLM) to improve latency without severe accuracy loss (Wang et al., 2020). Another direction is to reduce the number of candidates processed: span filtering or staged detection/classification can prune the search space before expensive classification. For example, SplitNER decomposes NER into a two-stage pipeline (span detection → span classification), casting both stages as Question Answering (QA) and training them as separate components (Arora and Park, 2023). Our work is complementary but different in nature: rather than reformulating NER into a multi-stage QA pipeline, we keep the PL-Marker formulation intact and reduce its serving-time overhead by (1) avoiding redundant marker computation in the encoder via a lightweight span decoder and (2) pruning spans early so the decoder processes only likely candidates.

LLMs and instruction-based extraction. Recent trends use LLMs and prompting/instruction tuning for information extraction, either directly (e.g., generative or prompted NER) or to improve supervision. GPT-NER adapts large language models to NER via prompting and verification strategies and reports competitive results in low-resource settings (Wang et al., 2025). InstructUIE proposes instruction-tuning to unify multiple IE tasks in a text-to-text framework (Wang et al., 2023). In parallel, compact "generalist" NER models such as GLiNER aim to support flexible schemas while keeping inference parallel and efficient compared to autoregressive generation (Zaratiana et al., 2024). Finally, verification-style post-processing with LLM reasoning has been explored to improve faithfulness in domain NER (e.g., biomedical) by revising model outputs using external knowledge (Kim et al., 2024). Compared to these lines of work, we focus on high-throughput deployment of span-based NER under tight latency constraints, providing a practical accuracy–efficiency improvement to PL-Marker-style systems.

3 Background and Preliminaries of NER

Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L]$ represent an input sequence of L tokens, such as a sentence or document. Let $\mathcal{E} = \{e_0, e_1, \ldots, e_M\}$ is a pre-defined set of M entity types with an additional $e_0 =$ "`O`" tag, denoting the "outside" class (i.e., absence of any named entity). The objective of NER is twofold:

1. To identify the contiguous spans of tokens, with each span denoted by $\mathbf{x}_{[i,j]} = [\mathbf{x}_i, \ldots, \mathbf{x}_j]$, composing a single named entity.

2. To classify each identified $\mathbf{x}_{[i,j]}$ with one of the specific entity types $e_k \in \mathcal{E}$.

Token Classification. A prevalent approach to NER frames the task as a sequence labeling problem. Each token $\mathbf{x}_i$ in the input sequence $\mathbf{X}$ is first mapped to a contextualized vector representation $\mathbf{z}_i \in \mathbb{R}^d$ using a text encoder, $\mathsf{Enc}(\cdot)$ by

$$[\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_L] = \mathsf{Enc}([\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L]).$$

In this approach, to effectively describe entity boundaries and distinguish between adjacent entities of the same type, tagging schemes such as

BIO (Beginning, Inside, Outside) are commonly employed. In this scheme, the set of entity types $\mathcal{E}$ is expanded to $\mathcal{E}_{\texttt{BI}}$ that represents entity tags. Intuitively, for each entity type $e_k \in \mathcal{E} \setminus \{e_0\}$, two tags are created: $\texttt{B-}e_k$ (Beginning of entity e_k) and $\texttt{I-}e_k$ (Inside of entity e_k). The `O` tag remains for tokens outside any entity.

Subsequently, a classifier, typically a linear layer, $\texttt{LN}(\cdot)$ is applied to predict an entity tag $e_k^{\texttt{BI}} \in \mathcal{E}_{\texttt{BI}}$ for each $\mathbf{z}_i$.

Span Classification. An alternative framework for NER is the span classification. In this formulation, all possible contiguous subsequences of tokens $\mathbf{x}_{[i,j]}$ from the input sequence $\mathbf{X}$ are considered candidate named entities, and we refer to the ranges $[i, j]$ as *spans*. Each candidate span $[i, j]$ is represented as a vector $\mathbf{q}_{ij}, \forall i, j$, often derived from the contextualized embeddings of its constituent tokens (e.g., by concatenating the representations of its start and end tokens, or by a pooling operation over the span's tokens).

Span representations $\mathbf{Q} = [\mathbf{q}_{00}, \mathbf{q}_{01}, \ldots, \mathbf{q}_{LL}]$ and the input sequence $\mathbf{X}$ are fed into the encoder $\texttt{Enc}(\cdot)$ and linear layer $\texttt{LN}(\cdot)$ to predict corresponding entity types from $\mathcal{E}$, including the `O` type if it is not a valid entity. Span classification approaches consistently yield improved performance compared to token classification techniques, particularly for nested or complex text entity structures. However, they can also introduce greater computational complexity due to the enumeration of spans, which can be quadratic $O(L^2)$ for the exhaustive enumeration and/or can be linear $O(L)$ when the maximum length of span is fixed (L is the sequence length).

4 Our Method

Our proposed method builds upon the span classification framework, offering a novel method designed for simplicity and computational efficiency relative to existing span-based approaches.

Span Representation. We represent each candidate span $[i, j]$, with $1 \leq i \leq j \leq L$, using two special learnable vectors: a *start* marker $\mathbf{m}_{\text{start}}$ and an *end* marker $\mathbf{m}_{\text{end}}$. These markers are augmented with positional embedding of the corresponding input text tokens to encode the span boundaries by $\mathbf{q}_i = \mathbf{m}_{\text{start}} + \text{pos}(i)$ and $\mathbf{q}_j = \mathbf{m}_{\text{end}} + \text{pos}(j)$ where $\text{pos}(\cdot)$ denotes a positional encoding function. The pair $\mathbf{q}_{ij} = (\mathbf{q}_i, \mathbf{q}_j)$ serves as the query representation for the span of candidate entity $\mathbf{x}_{[i,j]}$.

Span Decoder. To classify the span, we propose a lightweight decoder $\texttt{Dec}(\cdot)$ composed of a prenorm cross-attention layer $\texttt{Att}(\cdot)$ followed by a multilayer perceptron $\texttt{MLP}(\cdot)$ and a classification layer $\texttt{LN}(\cdot)$. Each span pair $\mathbf{q}_{ij} = (\mathbf{q}_i, \mathbf{q}_j)$ attends to the encoder outputs $[\mathbf{z}_1, \ldots, \mathbf{z}_L]$ as well as to each others (but not to other span pairs). Specifically, for the span $\mathbf{x}_{[i,j]}$, the input key $\mathbf{K}_{ij}$ and value $\mathbf{V}_{ij}$ sequences in the cross-attention are defined by:

$$\mathbf{K}_{ij} = \mathbf{V}_{ij} = [\mathbf{q}_i, \mathbf{q}_j, \mathbf{z}_1, \ldots, \mathbf{z}_L].$$

This formulation allows the span representation to incorporate contextual information from the full input sequence while also enhancing the focus between the span boundaries. The two outputs of the cross-attention layer `Att` (denoted by $\mathbf{q}'_i$ and $\mathbf{q}'_j$) are passed through a shared `MLP`, concatenated and classified using a linear layer `LN`. As an output, a vector $\hat{\mathbf{y}}_{ij}$ containing logits of the classes of the entity types $\mathcal{E}$ is calculated by

$$\hat{\mathbf{y}}_{ij} = \texttt{LN}([\texttt{MLP}(\mathbf{q}'_i); \texttt{MLP}(\mathbf{q}'_j)]),$$

where $[\cdot;\cdot]$ denotes the concatenation operation in the channel dimension.

Computational Efficiency. Compared to vanilla span-based NER approaches such as PL-Marker, which appends span markers to the input of the model and propagate them through all layers of the encoder, our approach is more computationally efficient. Precisely, span representations are introduced only at the decoding stage in our method. Thereby, our method significantly reduces the computational overhead, especially when processing a large number of spans (e.g., longer input text).

4.1 Efficient Span Selection

To further improve the computational efficiency of our method while maintaining high performance, we introduce a lightweight Span Filtering (`SF`) mechanism based on binary token-level classification. Formally, before executing the span decoder `Dec`, we apply a linear classifier to contextualized vector representations $[\mathbf{z}_1, \ldots, \mathbf{z}_L]$ produced by the encoder `Enc`. The auxiliary `SF` classifier is trained to predict whether a token belongs to the `O` class (i.e., not part of any named entity) or to any entity class with the information estimated only from the encoder.

Importantly, the weights of the SF classifier and of the span decoder Dec are optimized jointly during the training phase. At inference time, we use the output of SF classifier to identify tokens that are likely not part of an entity by applying a confidence threshold. Spans that enclose tokens predicted as O are discarded and not passed to the decoder. As a result, the number of candidate spans processed by the span decoder is significantly reduced.

This filtering step introduces negligible overhead, but yields substantial gains in efficiency, as it limits the unnecessary computations in the decoder step with a much smaller set of potentially valid spans. Empirically, we observe that this barely causes negative impact on the overall performance while dramatically reducing the number of spans fed to the decoder, leading to a promising efficiency-accuracy trade-off.

5 Implementation and Setup

We conducted our experiments using the PyTorch framework and leveraged pre-trained models available through the Hugging Face Transformers library (Wolf et al., 2020). We implemented our proposed methodologies (SpanDec and SF-SpanDec) and the token-classification baseline, from scratch, using PyTorch Lightning. For comparison with PL-Marker, we utilized the SpanMarker (Aarsen, 2023) library, which is built upon Hugging Face components and designed for efficient performance. All experiments were performed on a cluster of 8 NVIDIA A40 GPUs. For testing and throughput evaluation, a single NVIDIA A40 GPU was used. Mixed-precision training was always employed.

Models. We employ and test three widely adopted text encoders of varying sizes: *MiniLM* (33M), *BERT-Base* (110M), and *RoBERTa-Large* (355M). For SpanDec , we remove the final layer of each encoder to maintain a comparable overall parameter size after integrating the decoder. With the goal of improving the computational efficiency for on-device applications, we study our Span Filtering model (SF-SpanDec) with lightweight encoders. Accordingly, we conduct experiments using MiniLM and also include results for BERT-Base to offer a broader comparison.

Datasets. We select four widely used NER benchmarks: (1) *CoNLL++* (Wang et al., 2019), a corrected version of the standard CoNLL03 dataset (Tjong et al., 2003); (2) *CrossNER* (Liu et al., 2020), a cross-domain dataset covering five diverse domains; (3) *OntoNotes v5* (English) (Hovy et al., 2006), a large-scale dataset comprising multiple genres of text; and (4) *BC5CDR* (Li et al., 2016), which contains PubMed articles annotated with chemical and disease entities (see Tab. 2).

	CoNLL++	CrossNER	OntoNotes5	BC5CDR
Entity Classes	5	40	19	3
Train Samples	4.5k	20k	28k	11k
Train Words	272k	364k	1547k	218k
Test Samples	817	2.5k	3.2k	5.9k
Test Words	50k	96k	179k	116k

Table 2: **Datasets.** Information about the datasets used in our experiments.

Metrics. Following prior work, we report the F1 score (using the evaluation techniques provided by the seqeval library (Nakayama, 2018)), the GFLOPs (see Sec. D) and the Throughput (samples processed per second) of different methodologies.

Training Procedure. Standard hyperparameters and training schedules were used for all models. MiniLM was trained for 100 epochs on each dataset with a learning rate of 5×10^{-5}. BERT-B and RoBERTa-L were trained for 25 epochs on all datasets, except for the CrossNER dataset, where 100 epochs were used. The learning rate of newly initialized weights (decoder and final linear layer) was multiplied by 10, a common practice. A OneCycle learning rate schedule with a warm-up ratio of 0.03 was applied, and a global batch size of 64 was used. Gradient clipping was set to 1.0. Following the SpanMarker implementation, we assumed a maximum entity length of 8 words. Entities exceeding this length were treated as errors, although there is just a negligible number of such cases in the datasets considered. Model weights were optimized using the AdamW optimizer with a weight decay of 0.01.

6 Results

Decoding Spans is All You Need. As shown in Tab 3, our method outperforms PL-Marker in terms of computational efficiency and achieves similar F1 score. For instance, with MiniLM and compared to the token-classification baseline, PL-Marker provides 42% of the throughput and 0.7% average accuracy increase, whereas our method provides a notable 73% of the throughput with an average performance boost of 1.8%. A similar trend is ob-

Model	Strategy	Throughput ↑	GFLOPs ↓	CoNLL++	CrossNER	OntoNotes5	BC5CDR	Avg.
MiniLM	Token Classif.	**330.8** (1×)	**1.9** (1×)	92.9	71.2	87.8	87.1	84.7
	PL-Marker	140.3 (0.42×)	15.8 (8.3×)	91.7	73.9	87.9	**88.0**	85.4
	SpanDec (ours)	241.3 (0.73×)	2.9 (1.5×)	**93.7**	**74.6**	**88.4**	**88.0**	**86.2**
BERT-B	Token Classif.	**299.1** (1×)	**7.5** (1×)	92.9	74.5	89.5	87.4	86.1
	PL-Marker	102.1 (0.34×)	62.5 (8.3×)	93.2	78.5	90.0	**88.5**	87.6
	SpanDec (ours)	180.9 (0.61×)	11.5 (1.5×)	**93.6**	**78.7**	**90.3**	88.4	**87.8**
RoBERTa-L	Token Classif.	**187.0** (1×)	**26.8** (1×)	94.3	76.1	89.9	88.2	87.1
	PL-Marker	68.0 (0.36×)	222.7 (8.3×)	**95.4**	**80.2**	**91.4**	**90.9**	**89.5**
	SpanDec (ours)	118.7 (0.64×)	33.9 (1.3×)	**95.4**	79.9	91.3	90.8	89.4

Table 3: **Results for SpanDec.** F1 scores on CoNLL++, CrossNER, OntoNotes5 and BC5CDR benchmarks using different encoder models. The best performance per dataset and encoder model is highlighted in **bold**. The *Throughput* shows the samples processed per second by models (relative value compared to token classification reported in parenthesis). For the *GFLOPs*, we take the input length as 44, which is the average value of all datasets.

Method	Params	Throughput ↑	BC5	CoNLL	O-Notes
LLaMA2-7B	7B	< 9 (0.08×)	-	76.53	52.78
LLaMA2-13B	13B	< 7 (0.06×)	-	68.24	38.32
LLaMA3-8B	8B	< 9 (0.08×)	-	85.37	22.34
SOLAR	10.7B	< 8 (0.07×)	-	76.03	64.84
Mistral-12B	12B	< 7 (0.06×)	-	74.23	59.30
SplitNER (2023)	0.35B	104.1 (0.88×)	-	-	90.9
InstructUIE (2023)	13B	< 5.8 (0.05×)	89.0	91.5	88.6
UniNER (2023)	7B	< 6.4 (0.05×)	89.0	93.3	89.9
GLiNER (2024)	0.35B	76.2 (0.64×)	88.7	92.5	88.1
GPT-NER (2025)	175B	< 5 (0.04×)	-	90.9	82.2
SpanDec (RoBERTa-L)	0.35B	**118.7** (1×)	**90.8**	**95.4**	**91.3**

Table 4: **Comparison with SOTA.** Top section: results for 1-shot generative inference with open LLMs from (Zhu et al., 2024). Mid section: results of SOTA.

Model	Strategy	Throughput ↑	GFLOPs ↓	Average F1
MiniLM	Token Classif.	**330.8** (1×)	**1.90** (1×)	84.7
	PL-Marker	140.3 (0.42×)	15.8 (8.3×)	85.4
	SpanDec (ours)	241.3 (0.73×)	2.91 (1.5×)	**86.2**
	SF-SpanDec (ours)	310.8 (0.92×)	1.92 (1.01×)	85.8
BERT-B	Token Classif.	**299.1** (1×)	**7.5** (1×)	86.1
	PL-Marker	102.1 (0.34×)	62.5 (8.3×)	87.6
	SpanDec (ours)	180.9 (0.61×)	11.5 (1.5×)	**87.8**
	SF-SpanDec (ours)	276.9 (0.93×)	7.6 (1.01×)	87.5

Table 5: **Results for SF-SpanDec**. Throughput (samples/s), GFLOPs and average F1 scores of different strategies for MiniLM and Bert-B.

served with BERT-Base: our approach not only improves performance, but also maintains better efficiency compared to PL-Marker. Even with RoBERTa-Large, our method achieves competitive performance comparable to PL-Marker while being notably more efficient during inference.

These results support the superiority of our design, demonstrating that a lightweight decoder alone is sufficient for processing and classifying spans both effectively and efficiently.

Comparison with SOTA. In Tab. 4 we compare SpanDec (RoBERTa-L encoder) with other SOTA models in terms of model parameters, throughput and F1 score on three benchmark datasets. Notably, our methodology is superior to those approaches while being significantly more efficient. CrossNER results are not reported in the table because they are not available for competitors.

O-Classification for Efficiency. In Tab. 5, we present the results for our efficient filtering variant, SF-SpanDec. Compared to the standard SpanDec, it leads to a slight performance drop; however, it still prominently outperforms the token classification baseline and remains competitive with PL-Marker. The efficiency gains of SF-SpanDec are substantial and it delivers a strong performance boost with only a minimal increase in computational cost over standard classification. This makes SF-SpanDec a compelling candidate for on-device and resource-constrained applications, where balancing the performance and efficiency trade-off is critical.

Additional results. In the appendix we report detailed results on the throughput (Sec. A), the complete results for SF-SpanDec (Sec. B), an ablation study on the number of encoder/decoder blocks (Sec. C) and the calculation of FLOPs (Sec. D).

7 Conclusion

In this work, we study improvement of the efficiency of span-based frameworks. More precisely, we convert an encoder only architecture to an encoder for text with a lightweight decoder in order to optimize the utilization of span representation generation process. Additionally, deferring the span representation generation enables us to introduce a span filtering mechanism, which further reduces the computational cost of span representations. With these novel solutions, our method achieves competitive performance on NER tasks, while running fast and demonstrating a promising trade-off between accuracy and efficiency.

8 Limitations

The hyperparameters of our method were not exhaustively optimized, leaving room for potential performance gains through further tuning. This work primarily aims to demonstrate its superiority with standard configurations in a general setting.

Furthermore, as shown in Tab. 5, `SF-SpanDec` is so efficient that requires the same GFLOPs of naive token-classification, providing a notable boost in performance. Nevertheless, our implementation achieved 92-93% of its throughput. This suggests possible enhancements in our software implementation, that will be addressed in future work based on the specific hardware and application.

References

Tom Aarsen. 2023. Spanmarker for named entity recognition. https://github.com/tomaarsen/SpanMarkerNER.

Arjun Ramesh Binu Rajan M. R. Anupama M. Nair, Anusha Aji Justus. 2020. Event extraction from emails. *International Journal of Computer Applications*, 176(41):1–8.

Jatin Arora and Youngja Park. 2023. Split-NER: Named entity recognition via two question-answering-based classifications. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Sergei Bogdanov, Alexandre Constantin, Timothée Bernard, Benoit Crabbé, and Etienne P Bernard. 2024. NuNER: Entity recognition encoder pre-training via LLM-annotated data. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11829–11841, Miami, Florida, USA. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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, Minneapolis, Minnesota. Association for Computational Linguistics.

Christopher Dozier, Ravikumar Kondadadi, Marc Light, Arun Vachher, Sriharsha Veeramachaneni, and Ramdev Wudali. 2010. *Named Entity Recognition and Resolution in Legal Text*, pages 27–43. Springer Berlin Heidelberg, Berlin, Heidelberg.

Sumam Francis, Jordy Van Landeghem, and Marie-Francine Moens. 2019. Transfer learning for named entity recognition in financial and biomedical documents. *Information*, 10(8).

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. OntoNotes: The 90% solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 57–60, New York City, USA. Association for Computational Linguistics.

Seoyeon Kim, Kwangwook Seo, Hyungjoo Chae, Jinyoung Yeo, and Dongha Lee. 2024. VerifiNER: Verification-augmented NER via knowledge-grounded reasoning with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Jiao Li, Yueping Sun, Robin J. Johnson, Daniela Sciaky, Chih-Hsuan Wei, Robert Leaman, Allan Peter Davis, Carolyn J. Mattingly, Thomas C. Wiegers, and Zhiyong Lu. 2016. Biocreative V CDR task corpus: a resource for chemical disease relation extraction. *Database J. Biol. Databases Curation*, 2016.

Zihan Liu, Yan Xu, Tiezheng Yu, Wenliang Dai, Ziwei Ji, Samuel Cahyawijaya, Andrea Madotto, and Pascale Fung. 2020. Crossner: Evaluating cross-domain named entity recognition.

Hiroki Nakayama. 2018. seqeval: A python framework for sequence labeling evaluation. Software available from https://github.com/chakki-works/seqeval.

Kim Sang Tjong, F. Erik, and Fien De Meulder. 2003. 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.

David Wadden, Ulme Wennberg, Yi Luan, and Hannaneh Hajishirzi. 2019. Entity, relation, and event extraction with contextualized span 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)*.

Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, Guoyin Wang, and Chen Guo. 2025. GPT-NER: Named entity recognition via large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4257–4275, Albuquerque, New Mexico. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Xingyao Wang, Wenxuan Zhou, Chao Zu, Haoran Xia, Tian Chen, Yong Zhang, Rui Zheng, Jing Ye, Qianying Zhang, Tao Gui, Jian Kang, Junfeng Yang, Sha Li, and Chao Du. 2023. Multi-task instruction tuning for unified information extraction. *Preprint*, arXiv:2304.08085.

Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021. 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, Online. Association for Computational Linguistics.

Zihan Wang, Jingbo Shang, Liyuan Liu, Lihao Lu, Jiacheng Liu, and Jiawei Han. 2019. 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 5157–5166.

Thomas Wolf, Lysandre Debut, Victor Sanh, and 1 others. 2020. 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.

Deming Ye, Yankai Lin, Peng Li, and Maosong Sun. 2022. 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, Dublin, Ireland. Association for Computational Linguistics.

Urchade Zaratiana and 1 others. 2024. GLiNER: Generalist model for named entity recognition using bidirectional transformer. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Zexuan Zhong and Danqi Chen. 2021. 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, Online. Association for Computational Linguistics.

Wenxuan Zhou, Sheng Zhang, Yu Gu, Muhao Chen, and Hoifung Poon. 2023. Universalner: Targeted distillation from large language models for open named entity recognition. *arXiv preprint arXiv:2308.03279*.

Dengya Zhu, Sirui Li, Nik Thompson, and Kok Wai Wong. 2024. Open-source large language models excel in named entity recognition. In *International Conference on Neural Information Processing*, pages 313–326. Springer.

A Throughput

We evaluate the throughput (samples/s) for each method on all the considered dataset and we report these numbers in Table 6. To provide comparable results, we use the same setup and inference parameters for all the models (single Nvidia A40 GPU, mixed precision, etc.). The results are the average of 3 runs.

B Full Results for SF-SpanDec

Additionally, in Table 7, we report the full results of our efficient model (SF-SpanDec) that are previously reported average results in Tab. 5, compared to our standard model (SpanDec), PL-Marker and token-classification baseline

C Ablation Study: Number of Encoder and Decoder Blocks

Table 8 presents an ablation study investigating the impact of varying the number of encoder and decoder layers within our SpanDec model architecture. This study is conducted using the MiniLM model on the CoNLL++ dataset. Two sets of experiments are performed:

1. **Encoder Reduction, Fixed Decoder:** In the first set, we progressively remove layers from the encoder, starting from the final layer, while maintaining a fixed single-layer decoder.

2. **Constant Total Layers:** In the second set, for each layer remove from the encoder, a corresponding layer is added to the decoder. This maintains a constant total number of 12 encoder and decoder layers.

The results from the first experiment indicate that reducing the number of encoder layers leads to a modest decrease in performance. Crucially, this reduction also corresponds to a decrease in computational cost, measured in GFLOPs, suggesting this configuration could be beneficial for applications with computational constraints.

Conversely, the second experiment demonstrates that increasing the number of decoder layers does not yield significant performance improvements while simultaneously increasing computational complexity. This suggests that a simple model, single-layer decoder, is sufficient for the Named Entity Recognition task with this architecture.

D FLOPs Formulas for Transformer Encoder

Profiling tool, such as `torch.profiler`, can be inaccurate in estimating the floating-point operations (FLOPs) of the attention mechanism, e.g. when fused kernels are used to optimize the computation device performance. Therefore, we conduct the following analytic approach to estimate the FLOPs of different models and approaches adopted in this work.

Since the computation of the input embedding functions, the output linear classifier, activation functions and normalization layers are negligible (in total $\lesssim 2\%$) compared to the computation of the attention mechanism and the FFN (feed-forward network) linear projection, our analysis will focus on the last two operations.

Parameters of the calculation formulas are defined as follows:

- L: Number of Transformer layers.
- S: Input sequence length.
- M: Span marker lengths, M indicates $M/2$ pairs of span markers.
- H_{dim}: Hidden dimension size.
- A: Number of attention heads.
- $H_{head} = H_{dim}/A$: Dimension size per attention head.
- F_{dim}: FFN intermediate dimension size (typically $4 \times H_{dim}$)

Given an architecture, these parameters are fixed except S and M. We summarize the parameters corresponding to the models adopted in our experiments in Tab. 9.

Next we first present the FLOPs calculation for the text tokens when bidirectional self-attention is applied. Note that 1 MAC (multiply-accumulate) operation is generally counted as 2 FLOPs.

1. Bidirectional Self-Attention for Text Tokens ($Attn_{text}$ per block). The self-attention mechanism of text tokens mainly consists of:

- Input/Output Projections (for Q, K, V, and the final output projection; 4 matrices of $H_{dim} \times H_{dim}$ applied to S tokens): $4 \times (2 \cdot S \cdot H_{dim} \cdot H_{dim}) = 8SH_{dim}^2$.

Model	Strategy	CoNLL++	CrossNER	OntoNotes5	BC5CDR	Average
MiniLM	Token Classif.	247.6	297.3	350.0	428.1	330.8
	PL-Marker	86.0	109.1	123.6	242.6	140.3
	SpanDec (ours)	185.7	221.0	243.2	315.3	241.3
	SF-SpanDec (ours)	258.5	266.6	319.1	399.0	310.8
BERT-B	Token Classif.	206.3	240.5	330.9	418.6	299.1
	PL-Marker	64.1	78.1	84.3	182.0	102.1
	SpanDec (ours)	114.3	163.8	161.4	283.9	180.9
	SF-SpanDec (ours)	230.1	212.2	282.0	383.3	276.9
RoBERTa-L	Token Classif.	131.4	127.0	221.2	268.5	187.0
	PL-Marker	40.2	59.5	60.8	111.5	68.0
	SpanDec (ours)	71.2	92.9	110.4	200.4	118.7

Table 6: **Throughput Results.** Throughput (samples/s) evaluated using a single Nvidia-A40 GPU with a reference batch size of 8.

Model	Strategy	CoNLL++	CrossNER	OntoNotes5	BC5CDR	Average f1
MiniLM	Token Classif.	92.9	71.2	87.8	87.1	84.7
	PL-Marker	91.7	73.9	87.9	**88.0**	85.4
	SpanDec (ours)	**93.7**	**74.6**	**88.4**	**88.0**	**86.2**
	SF-SpanDec (ours)	93.4	73.8	88.3	87.6	85.8
BERT-B	Token Classif.	92.9	74.5	89.5	87.4	86.1
	PL-Marker	93.2	78.5	90.0	**88.5**	87.6
	SpanDec (ours)	**93.6**	**78.7**	**90.3**	88.4	**87.8**
	SF-SpanDec (ours)	93.4	78.2	90.1	88.2	87.5

Table 7: **Full Results for SF-SpanDec.** F1 score on CoNLL++, CrossNER, OntoNotes5 and BC5CDR benchmarks using different encoder models. The best performance per dataset and encoder model is highlighted in **bold**.

Encoder Blocks	Decoder Blocks	GFLOPs	F1
11 (31.5M)	1 (1.8M)	2.9	93.6
10 (29.7M)	1 (1.8M)	2.8	93.4
9 (27.9M)	1 (1.8M)	2.6	93.2
8 (26.1M)	1 (1.8M)	2.4	92.9
7 (24.3M)	1 (1.8M)	2.3	92.3
10 (29.7M)	2 (1.8M)	3.9	93.4
9 (27.9M)	3 (3.6M)	4.9	93.2
8 (26.1M)	4 (5.4M)	5.9	93.2
7 (24.3M)	5 (9.0M)	7.0	92.4

Table 8: **Ablation study: Impact of Encoder/Decoder Layers.** In the top section, we report the base configuration for SpanDec (11 encoder layers and 1 encoder layer). In the middle section, we remove the final layers of the encoder. In the bottom section, we keep constant the total number of layers to 12.

- Attention Score Calculation (QK^T across A heads): $A \cdot (2 \cdot S \cdot H_{head} \cdot S) = 2S^2(A \cdot H_{head}) = 2S^2 H_{dim}$.

- Weighted Sum with V (applying attention scores to V, across A heads): $A \cdot (2 \cdot S \cdot H_{head} \cdot S) = 2S^2(A \cdot H_{head}) = 2S^2 H_{dim}$.

Parameter	MiniLM	BERT-B	RoBERTa-L
L (Transformer blocks)	12	12	24
H_{dim} (Hidden size)	384	768	1024
A (Attention heads)	12	12	16
F_{dim} (FFN int. size)	1536	3072	4096

Table 9: **Architecture Details.** Configuration details for different models.

The total FLOPs for self-attention per block is:

$$Attn_{text} = 8SH_{dim}^2 + 4S^2 H_{dim} \quad (1)$$

2. FFN Linear Projection for Text Tokens (FFN_{text} per block) Bert-like model's FFN typicall includes two linear layers:

- First Linear Layer (expansion: $H_{dim} \rightarrow F_{dim}$): $2 \cdot S \cdot H_{dim} \cdot F_{dim}$.

- Second Linear Layer (contraction: $F_{dim} \rightarrow H_{dim}$): $2 \cdot S \cdot F_{dim} \cdot H_{dim}$.

The total FLOPs for the FFN per block is:

$$FFN_{text} = 4SH_{dim}F_{dim} \quad (2)$$

Then, we present the FLOPs calculation for the span markers. In particular, queries of a pair of marker tokens matches the keys of each other (omitted in this analysis for simplicity) plus the keys of input text tokens, resembling the cross-attention mechanism. Note that the computation of span tokens is additional to the previous computation of text tokens. It consists of the following:

Cross-Attention for Span Markers ($Attn_{marker}$ per block) The cross-attention mechanism mainly consists of:

- Input/Output Projections (for Q, K, V, and the final output projection; 4 matrices of $H_{dim} \times H_{dim}$ applied to M tokens): $4 \times (2 \cdot M \cdot H_{dim} \cdot H_{dim}) = 8MH_{dim}^2$.
- Attention Score Calculation (QK^T across A heads): $A \cdot (2 \cdot S \cdot H_{head} \cdot M) = 2SM(A \cdot H_{head}) = 2SMH_{dim}$.
- Weighted Sum with V (applying attention scores to V, across A heads): $A \cdot (2 \cdot S \cdot H_{head} \cdot M) = 2SM(A \cdot H_{head}) = 2SMH_{dim}$.

The total FLOPs for cross-attention per block is:

$$Attn_{marker} = 8MH_{dim}^2 + 4SMH_{dim} \quad (3)$$

FFN Span Markers (FFN_{marker} per block) Following the calculation of FFN's cost for text tokens, we can derive the total FLOPs per block is:

$$FFN_{marker} = 4MH_{dim}F_{dim} \quad (4)$$

Given the parameters in Tab. 9, the number of layers involving span marker computation, and the lengths of input text and span marker, we can calculate the FLOPs based on Eqs. (1) to (4). Based on the average input length of the datasets adopted in the work, we set $S = 44$ and $M = 324$ for the FLOPs computation.