AIST3120 Final Project
Deep Learning Methods for Named Entity Recognition

QSSquared
(LEUNG, Chun Yui, HON, Kwan Shun Quinson, TAM, Ka Ho)

2024/04/24

Abstract

In this project report, we test the performance of three model architectures on the named entity recognition (NER) task - the SpaCy architecture using hash embeddings and transition-based prediction models, the Gemma 3 architecture based on the Transformer decoder, and the BERT architecture based on the Transformer encoder. Our results show that BERT models perform the best, improving by over 0.18 points in F1-score compared to Gemma 3 and over 0.24 points compared to SpaCy models in the CoNLL-2003 dataset. However, all models perform poorly on the WikiAnn dataset, with all models getting 0.39-0.51 in F1 scores. Our experiments show the effectiveness of BERT models compared to other deep learning methods, as well as give insights on what improves NER model performance.

1 Introduction

The named entity recognition task (NER) requires a machine to discover named entities within text, including names of people, organizations, locations, etc. In this project, we tested three different deep-learning models to complete the task: SpaCy's transition-based model [1], the BERT encoder model [2], and the recent Gemma 3 27B decoder model [3]. We also implement a custom implementation for aggregating token classification results and compare it with existing implementations to determine its effectiveness in NER.

2 Related Work

Named entity recognition was first defined in Grishman and Sundheim [4], with the goal of finding all people, organizations and locations in the MUC-6 dataset, consisting of 318 Wall Street Journal articles. Since then the CoNLL-2002 [5] and CoNLL-2003 [6] tasks have become popular datasets for evaluating the quality of NER methods. In our project, we use the CoNLL-2003 dataset for evaluating the performance of different NER systems.

Early attempts at NER often involved rule-based methods and statistical modeling [7]. However, these models use manually engineered features and do not implicitly learn nuanced patterns between words. Since then, deep learning has emerged as a viable approach in solving NER tasks with architectures, leveraging hidden representations to learn relationships between words. Architectures such as convolutional neural networks (CNNs) [8] and Long-Short Term Memory (LSTM) models [9] can be trained to generate different hidden representations that represent the meaning of text effectively, allowing models to better adapt to general text passages. These architectures have been used in NER in combination with statistical approaches such as Conditional Random Fields (CRF) [10] to some success. In this project, we use a SpaCy model [1] which combines feature engineering with a CNN for language understanding and NER.

The Transformer architecture [11] revolutionized natural language processing by introducing self-attention, a mechanism that helps language models capture long-distance relationships between words. The encoder-decoder architecture has since been modified with encoder-only variants such as BERT [2] and decoder-only variants such as GPT [12], both of which train their models on a large amount of general text. Brown, Mann, Ryder, *et al.* [13] later shows that pre-trained language models can adapt to tasks at inference time using few-shot learning. In this report, we use a variation of the BERT encoder model as the baseline and base model for finetuning. We also use the Gemma 3 27B [3] model, a modern pre-trained model that uses the GPT architecture, to test the effectiveness of few-shot learning for NER.

3 Methodology

3.1 SpaCy

SpaCy models adopt various techniques to perform NER. Some notable features include hash embeddings and its transition-based model.

3.1.1 Hash Embeddings in SpaCy

Rather than learning embeddings directly, SpaCy's embeddings use engineered features and hash tables to construct word embeddings. For each word in the sentence, orthographic features are extracted from word and are hashed several times to obtain different hash values, each value corresponding to an embedding. The vectors associated with each hash value of a feature are then aggregated to form the final word vector. A more detailed explanation for SpaCy's hash embeddings can be found in Appendix A.

3.1.2 Transition-based Models

Transition-based models for NER were first introduced in [14], proposed as an alternative to CRF models. Instead of predicting each word or token, the algorithm maintains a buffer of words that are used to construct the next entity. When processing the next word, the system looks at the entities, the buffer and the remaining context and does one of three operations:

- SHIFT: Move the next word into the buffer.
- REDUCE(label): Group all words in the buffer, mark the group with the given label and add it into the predicted entity list.
- OUT: Discard the next word.

In SpaCy, the model only looks at the last few words in the entity list, the first and last words of the current entity, and the next word to predict the next operation to perform. The process repeats until the entire context is processed, and the resulting list of entities forms the output.

3.2 BERT

With the success of the Transformer architecture, the encoder-based BERT architecture [15] has been the state of the art in language understanding, which makes it a natural choice for the NER task. Encoder-only transformer architectures, such as BERT, generally produce an embedding vector for each input token, while taking into account both the left and right contexts. In Named Entity Recognition, an extra linear classification layer is typically used to map feature embeddings to NER classes.

Unlike transition-based models, BERT-based models produce token-level class predictions. To differentiate consecutive entities, each entity type (such as `PER` for person, `ORG` for organization, `LOC` for location and `MISC` for others) is represented by two token classes, with one indicating the beginning of an entity (class names preceded by `B-`), and one for the remaining parts of an entity (class names preceded by `I-`).

Furthermore, while letter casing provides important cues in NER for distinguishing named entities, we fine-tune specifically the *cased* version of the pretrained BERT model `bert-base-cased` on Hugging Face in this project.

3.2.1 Aggregation Strategies

When tokenizing with BERT, individual words may be split into multiple tokens. As a result, aggregation of the outputs become crucial, particularly for reconstructing word-level entity predictions more accurately from fragmented token-level outputs. We implement a custom aggregation scheme to ensure that token-level predictions are aggregated on the word level to make sure its predictions match up with the dataset. Comparisons between our method with existing methods in Hugging Face Transformers [16] can be found in Appendix B.

3.2.2 Masking

In this project, we also investigate the effects of masking. We hypothesize that the nature and classification of the entity heavily depend on its surrounding context. By encouraging the model to focus specifically on the context around potential entities, we expect an improvement in the model's capability to generalize to unseen entities, especially the names of persons and organizations, which can often be too unique to be present in the datasets.

To implement this, we applied a masking probability of 15% specifically to named entities in the dataset. Instead of using the recommended fine-tuning setup of $N = 3$ epochs with a learning rate of $\eta = 2 \times 10^{-5}$, we fine-tuned `BERT-base-cased` for $5N$ epochs using a reduced learning rate of $\eta/5$ to increase the model's exposure to varied masking scenarios and enabling it to better learn contextual dependencies.

3.3 Decoder LLMs

In recent years, decoder LLMs like the GPT series [12] have experienced significant advancements, and have since been widely used in a wide range of general-purpose tasks, such as question-answering, translation, programming, etc. These models, which are often extremely large in model size, ranging from a billion to a few hundred billion parameters, are trained on diverse text data on the internet, meaning that they possess not just linguistic patterns, but also a vast amount of real-world knowledge.

While traditional NER models primarily learns entity patterns from the dataset they are trained on, decoder LLMs can identify entities based on real-world knowledge, potentially making them better at NER tasks with less common entities or complex, potentially ambiguous language. Therefore, we have decided to add a decoder LLM to our experiment, in order to investigate the performance of these models in NER when compared to state-of-the-art methods and potentially discover new insights from the experiments.

3.3.1 Prompting Strategies

Applying decoder LLMs to NER fundamentally differs from traditional approaches. Instead of classifying each token in a sequence (as with BERT), decoder LLMs approach the task as a text generation task. We include the task definition and entity descriptions in the context of the model, and guide it to answer in a structured JSON format with prompt engineering techniques. The model output is then parsed to extract the entities for evaluation.

Few-shot prompting is known to have great potential in improving the capabilities of LLMs [13]. To examine the influence of task-specific examples on this rather simple NER task, we performed two tests with zero-shot and few-shot prompting, respectively. The zero-shot prompt only includes the task description, output instructions and the input text, while the few-shot prompt includes a small number of example outputs as well. We manually select examples from the training set that are rich enough to give the model a good idea about the patterns within entities.

4 Experimental Setup

4.1 Datasets

We evaluate NER models on two datasets: CoNLL-2003 [6] and WikiAnn [17]. Models are trained on the train split of the CoNLL-2003 dataset, where it must discover the Person (PER), Organization (ORG), Location (LOC), and Miscellaneous (MISC) entity types in text. Models are evaluated but not trained on the WikiAnn dataset, where it only needs to identify PER, ORG and LOC entities.

4.2 Models

4.2.1 SpaCy Models

Two pre-trained SpaCy NER models [18] are evaluated: `en_core_web_md` (31 MB) and `en_core_web_lg` (382 MB).

4.2.2 BERT Models

We adopt `bert-base-NER` [19] as the baseline, a version of BERT fine-tuned by a developer on Hugging Face specifically on the CoNLL-2003 dataset for NER tasks.

We chose two pre-trained models as the base for fine-tuning on the NER task: `bert-base-cased` [2], the standard BERT model with a cased vocabulary, and `distilbert-base-cased`, a smaller, distilled version of BERT.

The Hugging Face `transformers` library [16] is used for training. We trained both models with a batch size of 32 with the AdamW optimizer [20]. The `bert-base-cased` model is trained for 3 epochs with a learning rate of 2×10^{-5} and a weight decay of 0.01. The `distilbert-base-cased` model is trained for 15 epochs with a learning rate of 4×10^{-6} and a weight decay of 0.002.

Google's Gemma 3 27B model [3] is used for the decoder LLM experiments. Inference was performed using default settings provided by the inference provider at the time of this report. The system prompts can be found in Appendix C.

4.3 Evaluation

The performance of all models was evaluated using standard entity-level metrics: precision, recall, and F1-score. The `seqeval` framework [21] is used for performing the evaluation.

5 Results

5.1 Aggregation Method Comparison

To evaluate the impact of various aggregation strategies on the NER performance of the fine-tuned BERT models, we compared our custom strategy against several built-in aggregation methods, including `simple`, `first`, `average` and `max`, provided by the Hugging Face `pipeline` library.

Table 1 showed the results evaluated on the same pre-trained model `bert-base-NER`, which was specifically fine-tuned over the CoNLL dataset. It can be observed that while the built-in aggregation methods lead to poor results, our custom approach showed significant improvement from the built-in approaches and achieved an F1 score of 0.9157.

Aggregation Strategy	Precision	Recall	F1
Simple	0.5804	0.7350	0.6486
First	0.8318	0.8890	0.8595
Average	0.8368	0.8791	0.8574
Max	0.8359	0.8817	0.8582
Custom	**0.9124**	**0.9189**	**0.9157**

Table 1: Model Performance

Model	Precision	Recall	F1
CoNLL-2003 (Test)			
BERT-base-cased (no mask)	0.8927	**0.9032**	**0.8979**
BERT-base-cased (masked)	**0.8929**	0.8948	0.8939
WikiAnn (Test) (Not trained on)			
BERT-base-cased (no mask)	**0.4603**	**0.5134**	**0.4854**
BERT-base-cased (masked)	0.4577	0.5080	0.4816

Table 2: Comparison of Performances between masked and unmasked training

5.2 Masking Comparison

We also tested the effects of our masking approach by conducting a controlled experiment on whether random masking over the named entities is applied to the training dataset. Table 2 showed that, contrary to our hypothesis, masking failed to further improve the results.

5.3 Method Comparison

Table 3 showed a comprehensive comparison of all different techniques applied over the NER task. With the pre-trained `bert-base-NER` model from Hugging Face being a baseline for our results, it can be observed that BERT-based models consistently outperformed the others, with an F1 score of 0.9157 (model from library) and 0.8979 (fine-tuned ourselves) respectively.

In contrast, SpaCy's models performed noticeably worse, with F1 scores of 0.6158 for the medium model, and 0.6573 for the large model. While the larger SpaCy model resulted in higher accuracy compared to the smaller one, both models fall short of the BERT-based fine-tuned models.

While decoder LLMs are typically excellent at language understanding, Gemma 3 27B displayed a similar level of accuracy as SpaCy. and that few-shot prompting improved the performance slightly to an F1 score of 0.7125 from the zero-shot case of 0.6879, both models are still underperforming compared to encoder-based models that are specifically fine-tuned over the NER task.

Model	**Precision**	**Recall**	**F1**
bert-base-NER	**0.9124**	**0.9189**	**0.9157**
distilbert-base-cased fine-tuned (no mask)	0.8927	0.9032	0.8979
SpaCy Medium	0.6618	0.5758	0.6158
SpaCy Large	0.6850	0.6317	0.6573
Gemma 3 27B (zero-shot)	0.6361	0.7489	0.6879
Gemma 3 27B (few-shot)	0.6689	0.7620	0.7125

(a) CoNLL-2003 Dataset

Model	**Precision**	**Recall**	**F1**
bert-base-NER	**0.4786**	**0.5287**	**0.5024**
distilbert-base-cased fine-tuned (no mask)	0.4603	0.5134	0.4854
Spacy Medium	0.4002	0.3905	0.3953
Spacy Large	0.4051	0.3987	0.4019

(b) WikiAnn Dataset

Table 3: Comparison of NER Model Performance on Different Datasets

6 Analysis of Results

While SpaCy's hash embeddings are interesting, the backbone of the model still uses the less expressive CNN architecture, and only a fixed window size is used for predicting entity transitions rather than the full context of the sentence. This means that the model cannot capture all context of the sentence by default, which puts it at a significant disadvantage over other methods. We also note that SpaCy reports their models having scores at or above 0.85 in precision, recall and F1-scores on trained datasets such as OntoNotes and WordNet. The fact that we are getting lower metric scores suggests that CNNs have limited ability to generalize to other datasets such as CoNLL-2003.

For decoder LLMs, the pretrained LLM seems to perform better with few-shot examples, which matches the typical behaviour of decoder LLMs. This could be because that the examples give the model a better idea of what and how entities are marked, leading to better performance. However, they seem to have subpar performance compared to BERT models, which have significantly smaller size at about 100 million parameters compared to Gemma 3's 27 billion. This suggests that while pretrained models have high general capabilities in question-answering and language understanding, they are not the most efficient or effective method for solving established tasks such as NER.

As for BERT models, our results show that designing aggregation strategies properly according to the dataset can improve the model's performance. While we attempted to apply a masking strategy on the named entities to achieve better results, the results did not improve. We believe that a possible reason is because when an entity is masked, the model cannot check whether the named entity is in title case, which can be very helpful for finding named entities. Despite these drawbacks, BERT itself is still an effective model for NER as it uses the attention mechanism for better understanding, makes predictions in parallel rather than in sequence as in decoder-based models, and can be easily fine-tuned on datasets to achieve good performance with small models, making it the most cost-effective architecture for the task out of the three options we tested.

7 Limitations of the Experiment

In the SpaCy tests, we did not attempt to fine-tune the model, as the model makers recommend that users "make use of pre-trained embeddings distributed by spaCy" [1]. However, there have been reports of users obtaining F1-scores comparable to that of BERT by training directly on the CoNLL-2003 dataset, achieving an F1-score of 0.89 [22]. This suggests that the SpaCy's pretrained embeddings may not be suitable to be used out-of-the-box and need to be finetuned as well. While we do not attempt to finetune the SpaCy models in this project due to complexity of SpaCy's development tools, we believe that a CNN SpaCy model trained on CoNLL will perform similarly compared to a fine-tuned BERT model on CoNLL, though it may not generalize well to other datasets.

In decoder LLMs, we only used instructions and few-shot prompts that directly produce named entity predictions in the format used for evaluation. However, this format may not be the most useful way to help decoder LLMs generate predictions. Moreover, the cost of executing decoder LLM queries and the difficulty of creating parallelized LLM requests for speed means that we were unable to run tests with the WikiAnn dataset.

In BERT models, we did not include common techniques to improve training, such as using data augmentation to improve the genralization ability of models, and applying better sampling procedures to avoid bias in predictions. With these techniques, it should be possible to replicate the results in publicly available fine-tuned models such as bert-base-NER.

8 Conclusion

In this project, we tested SpaCy's hash-embedding and transition-based models, finetuned BERT models and off-the-shelf decoder LLM models for named entity recognition on the CoNLL-2003 dataset and the WikiAnn dataset. We found that SpaCy's models struggle to generalize beyond the training set despite the use of hash embeddings and transition-based models. We also found that decoder LLMs are inefficient for solving the established task of NER, even though few-shot examples can help them produce better results. Finally, while we were able to improve NER performance with BERT models by implementing custom aggregation strategies outside of what the Transformers library provides, we were unable to replicate the performance found in BERT models finetuned by others. Despite this, we believe that our experiments show BERT's capabilities in the NER task and allow us to better identify its advantages over other deep learning approaches.

References

[1] L. J. Miranda, Á. Kádár, A. Boyd, S. V. Landeghem, A. Søgaard, and M. Honnibal, *Multi hash embeddings in spacy*, 2022. arXiv: 2212.09255 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2212.09255.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805.

[3] G. Team, A. Kamath, J. Ferret, *et al.*, *Gemma 3 technical report*, 2025. arXiv: 2503.19786 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2503.19786.

[4] R. Grishman and B. Sundheim, "Message Understanding Conference- 6: A brief history," in *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996. [Online]. Available: https://aclanthology.org/C96-1079/.

[5] E. F. Tjong Kim Sang, "Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition," in *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*, 2002. [Online]. Available: https://www.aclweb.org/anthology/W02-2024.

[6] E. F. Tjong Kim Sang and F. 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*, 2003, pp. 142–147. [Online]. Available: https://aclanthology.org/W03-0419/.

[7] M. Munnangi, *A brief history of named entity recognition*, 2024. arXiv: 2411.05057 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2411.05057.

[8] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds., Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. DOI: 10.3115/v1/D14-1181. [Online]. Available: https://aclanthology.org/D14-1181/.

[9] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. arXiv: 1308.0850. [Online]. Available: http://arxiv.org/abs/1308.0850.

[10] Z. Huang, W. Xu, and K. Yu, "Bidirectional LSTM-CRF models for sequence tagging," *CoRR*, vol. abs/1508.01991, 2015. arXiv: 1508.01991. [Online]. Available: http://arxiv.org/abs/1508.01991.

[11] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[12] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[13] T. B. Brown, B. Mann, N. Ryder, *et al.*, “Language models are few-shot learners,” *CoRR*, vol. abs/2005.14165, 2020. arXiv: 2005.14165. [Online]. Available: https://arxiv.org/abs/2005.14165.

[14] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. 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*, K. Knight, A. Nenkova, and O. Rambow, Eds., San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 260–270. DOI: 10.18653/v1/N16-1030. [Online]. Available: https://aclanthology.org/N16-1030/.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1810.04805.

[16] T. Wolf, L. Debut, V. Sanh, *et al.*, “Transformers: State-of-the-art natural language processing,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[17] A. Rahimi, Y. Li, and T. Cohn, “Massively multilingual transfer for NER,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds., Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 151–164. DOI: 10.18653/v1/P19-1015. [Online]. Available: https://aclanthology.org/P19-1015/.

[18] E. AI, *English · spaCy Models Documentation*, MIT License, Accessed: 2025-04-21, 2024. [Online]. Available: https://spacy.io/models/en.

[19] dslim, *dslim/bert-base-NER at main - Hugging Face*, Accessed: 2025-04-21, 2024. [Online]. Available: https://huggingface.co/dslim/bert-base-NER.

[20] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7.

[21] H. Nakayama, *seqeval: A Python framework for sequence labeling evaluation*, Software available from https://github.com/chakki-works/seqeval, 2018. [Online]. Available: https://github.com/chakki-works/seqeval.

[22] JINHXu, *CoNLL03_SpaCy_v3: Train SpaCy v3 NER models (English and German) with CoNLL-2003 data*, Accessed: April 18, 2025, 2025. [Online]. Available: https://github.com/JINHXu/CoNLL03_SpaCy_v3.

Appendices

A Detailed Explanation of SpaCy's Hash Embeddings

Rather than learning embeddings directly, SpaCy's embeddings use engineered features and hash tables to construct word embeddings. For each word in the sentence, four orthographic features are extracted from the word:

- NORM: A lowercase version of the word, with additional operations on punctuation, currency characters and alternative spellings.
- PREFIX: The first character.
- SUFFIX: The last three chracters.
- SHAPE: Uppercase letters replaced with 'X', lowercase letters replaced with 'x', numbers replaced with 'd', and truncated characters that repeat more than four times.

After creating the features, the strings representing each feature are hashed four times. The hashed values are used to construct a multi-hot embedding vector $\mathbf{b}$, where $\sum_j \mathbf{b}_j = 4$. Then, the multi-hot embedding is passed through a typical embedding matrix $\mathbf{E}^\top \mathbf{b}$ to obtain the dense vector for the feature. Finally, the dense features are concatenated and processed with an additional Maxout-layer to obtain the final embedding:

$$\Psi(\mathbf{x}) = \max\left(\mathbf{W}_1^T \mathbf{x} + \mathbf{b}_1, \mathbf{W}_2^T \mathbf{x} + \mathbf{b}_2, \mathbf{W}_3^T \mathbf{x} + \mathbf{b}_3\right)$$

Hash embeddings allow many words to be encoded using a small table of hash embeddings. Consider a hash table of size n, which means that each string is mapped to any hash table entry with probability $\frac{1}{n}$. If we use k hash functions to construct representations for a dictionary of size m where $k << m$, the probability of two strings sharing the same multi-hot embedding is approximately $\frac{1}{n^k}$. Therefore, the probability that some string features share the same embedding is approximately

$$p = 1 - \prod_{j=0}^{m-1} \left(1 - \frac{j}{n^k}\right) \leq 1 - \left(1 - \frac{m(m-1)}{n^k}\right) = \frac{m(m-1)}{n^k}$$

For n = 5,000, m = 50,000 and k = 4, $p \leq 4 \times 10^{-6}$, which means that the probability that two different string representations will share the same embedding is very low. Therefore, the smaller dictionary is still expressive enough to produce unique representations for every possible string feature.

B Comparison of Aggregation Strategies

When tokenizing with BERT, individual words may be split into multiple tokens. As a result, aggregation of the outputs become crucial, particularly for reconstructing predictions more accurately from fragmented token-level outputs:

> Input Tokens: `["Greg", "Blewett", ...]`
> Tokenization from BERT: `["Greg","Blew","#ett",...]`
> Prediction: `[B-PER, I-PER, I-PER, ...]`
> Expected entity: `Greg Blewett`

The Hugging Face `pipeline` library for the NER task provides built-in aggregation strategies, such as `simple`, `first` and `max`, to combine results from subword token-level output. However, built-in strategies may fail to preserve the integrity of input words when subword predictions within a single word are inconsistent, which can lead to fragmented entity recognition:

> **Case 1.**
> Input Tokens: `["Greg", "Blewett", ...]`
> Tokenization from BERT: `["Greg","Blew","#ett",...]`
> Prediction: `[B-PER, I-PER, O, ...]`
>
> Built-in Aggregation: `[("PER", "Greg Blew")]`
> Our Aggregation: `[("PER", "Greg Blewett")]`
>
> **Case 2.**
> Input Tokens: `["Pau-Orthez", ...]`
> Tokenization from BERT: `["Pau","-","Orthez",...]`
> Prediction: `[B-PER, O, B-PER, ...]` (imperfections)
>
> Built-in Aggregation: `[("PER", "Pau"),("PER", "Orthez")]`
> Our Aggregation: `[("PER", "Pau-Orthez")]`

Our method addresses by ensuring that the entire word is correctly aggregated, and each input phrase is not fragmented in the output entity prediction.

C Prompts for Decoder LLMs

When performing NER with a zero-shot prompt, we use the following prompt:

```
You are a named entity recognition (NER) system. Your task is to
identify and classify named entities in the text.
The entities can be of the following types: PERSON, ORGANIZATION,
LOCATION, and MISCELLANEOUS.
You will receive a text input, and you need to return the entities in
```

```
json format:
{
    "entities": [
        {"label": "PERSON", "text": "entity_name"},
        {"label": "ORGANIZATION", "text": "entity_name"},
        {"label": "LOCATION", "text": "entity_name"},
        {"label": "MISCELLANEOUS", "text": "entity_name"}
        ...
    ]
}
```

When performing NER with few-shot examples, we add the following text to the end of the zero-shot prompt:

```
 Below are some more example inputs and outputs:
    text: "A Mujahideen Khalq statement said its leader Massoud Rajavi met in Baghdad
        the Secretary-General of the Kurdistan Democratic Party of Iran ( KDPI ) Hassan
         Rastegar on Wednesday and voiced his support to Iran 's rebel Kurds ."
    output: {
        "entities": [
            {"label": "ORGANIZATION", "text": "Mujahideen Khalq"},
            {"label": "PERSON", "text": "Massoud Rajavi"},
            {"label": "LOCATION", "text": "Baghdad"},
            {"label": "ORGANIZATION", "text": "Kurdistan Democratic Party of Iran"},
            {"label": "ORGANIZATION", "text": "KDPI"},
            {"label": "PERSON", "text": "Hassan Rastegar"},
            {"label": "LOCATION", "text": "Iran"},
            {"label": "MISCELLANEOUS", "text": "Kurds"}
        ]
    }

    text: "Ireland midfielder Roy Keane has signed a new four-year contract with
        English league and F.A. Cup champions Manchester United ."
    output: {
        "entities": [
            {"label": "LOCATION", "text": "Ireland"},
            {"label": "PERSON", "text": "Roy Keane"},
            {"label": "MISCELLANEOUS", "text": "English"},
            {"label": "MISCELLANEOUS", "text": "F.A. Cup"},
            {"label": "ORGANIZATION", "text": "Manchester United"}
        ]
    }

    text: "I have stayed out of the way and let them get on with the job ."
    output: {
        "entities": []
```

```
}

text: "TENNIS - AUSTRALIANS ADVANCE AT CANADIAN OPEN ."
output: {
    "entities": [
        {"label": "MISCELLANEOUS", "text": "AUSTRALIANS"},
        {"label": "MISCELLANEOUS", "text": "CANADIAN OPEN"}
    ]
}
```