Joint Modeling of NER and SA for Prompt-Based Alert Generation

Nicolás del Val, Miguel Ángel Huamani, Santiago Martínez and Jorge Ibinarriaga
Universidad Pontificia Comillas
https://github.com/ndelval/PracticaFinalDeepNLP/tree/master

Abstract

Effective text understanding often requires integrating both morphological information and syntactic structure. Morphological information provides lexical and contextual cues, while structural information captured by dependency trees models the relationships between words and enhances the overall sentence representation. In this work, we investigate how effectively combining both morphological features and dependency structure contributes to improved performance in text understanding. To assess this, we test our model on the task of automatically generating alerts from social media posts. We compare the full version of our model with two simplified variants and analyze which version achieves the best trade-off between performance and complexity.

1 Introduction

Information extraction and text understanding are central challenges in Natural Language Processing (NLP), particularly in tasks such as Sentiment Analysis (SA) and Named Entity Recognition (NER). While NER benefits from morphological information and character-level features, SA often relies on broader contextual cues provided by syntactic relationships between words. Traditional models typically focus on either word-level representations or syntactic structure, but rarely combine them effectively.

In this work, we investigate whether integrating both architectures for NER and SA into a hybrid neural model can lead to improved understanding and information extraction. NER aims to identify and classify key elements in the text, such as people or organizations; using a custom trained LSTM network. On the other hand, SA determines overall sentiment expressed in the text: positive or negative through a separate neural network that captures sentence-level context. Our approach combines the original input text with the outputs of both NER and SA to automatically generate well-structured alerts that reflect the underlying meaning of the text. The alert generation process in handled by a Large Language Model (LLM) based on two well defined prompts, responsible for selecting the most relevant alert candidates.

Our goal is to develop the simplest possible version of the model that can still accurately generate alerts. We compare the full hybrid architecture with two simplified variants: one excluding character-level features and another excluding syntactic structure. This comparison not only helps identify the most efficient configuration for alert generation, but also sheds light on which type of linguistic integration—morphological or syntactic—is more critical for accurate information extraction and text understanding. Determining the relative important of each feature type can contribute to the understanding of information extraction and text generation.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2 Prior related work

This work has been inspired by the research of Xu et al. Better Feature Integration for Named Entity Recognition (2021) for the task of Named Entity Recognition. The task of NER has traditionally relied on stacking the LSTM and graph neural network (GCN). While this approach can incorporate both sequential and structural features, the interaction between the two types of features is not clear and does not yield a significant performance gain. To overcome this, Tu et al. introduces the Synergized-LSTM (Syn-LSTM), a custom recurrent neural network architecture which integrates GCN derived syntactic information into the LSTM's internal computation. This is achieved allowing the cell to receive graph-encoded representation from the GCN output (as illustrated in Figure 1 of their paper). The result is a model where output step time depends both on sequential and structural features.

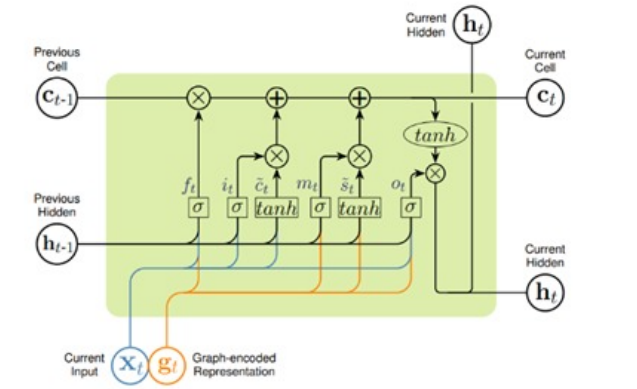

Figure 1: Syn-LSTM architecture

In addition, extensive experiments across multiple datasets and 4 different languages demonstrate that this architecture achieves significant performance gain while maintaining low parameter complexity. Moreover, their work shows that even shallow GCNs when combined properly with sequence models, can capture long term dependencies between entities, which play a key role in NER. Static analysis further supports that the new gating mechanism enables to model to regulate the most relevant structured information at each prediction.

Building on this foundation, our work explores how this architecture can be extended beyond NER. We propose a hybrid neural model that combines character-level encoding (CharBILSTM) and syntactic representation (GCN) and apply it to join Sentiment Analysis with NER in an automated alert generation from text. Our work further investigates how combining linguistic features plays a role in generative tasks and to what extent simplified architecture can retain effectiveness.

3 Methodology

This section details the architecture and components of our main proposed system (Model A), inspired by the work of Xu et al. in *Better Feature Integration for Named Entity Recognition*. We also describe four variants (Model B to Model E), which allow us to isolate the effect of specific architectural blocks. All five variants of our model are instantiated from a shared base class `NNCRF`, with behavior controlled via three boolean parameters: `use_char_embs`, `use_separate_lstms`, and `use_syn_lstm`. This design allows switching architectural components on or off. To know how each model is derived by toggling these flags, please read the README file of our GitHub

3.1 Model Architecture (Model A)

Our system addresses the joint task of Named Entity Recognition (NER) and Sentiment Analysis (SA) by integrating diverse linguistic features into a hybrid neural architecture. The model includes the following components:

- **Character-level representation:** A CharBiLSTM processes each word as a sequence of characters. For each token, the final hidden states from the forward and backward LSTM are concatenated to produce a fixed-size embedding. This allows the model to capture subword information and better handle morphological variation.
- **Word embeddings:** Pretrained static word embeddings (Glove 100d) are used as the base representation for each token in the sentence.
- **Dependency-aware Graph Convolutional Network (GCN):** The syntactic structure of the sentence is injected through a GCN over dependency edges, built using `spaCy` parsing and represented as `torch_geometric` graph objects. Dependency relations are encoded via learned embeddings and propagated through the GCN.
- **SynLSTM (MyLSTM):** A customized LSTM cell that fuses sequential and syntactic information by integrating both word-level and GCN-enhanced representations at each time step. It maintains an internal memory for the standard sequence and modulates updates with an additional gate to incorporate GCN features.
- **BiLSTM for Sentiment Analysis:** In parallel, a standard BiLSTM is applied directly over the word embeddings to extract global sentence-level features for the SA task.
- **NER and SA output heads:** Two distinct MLP heads are used to produce predictions for each task. For NER, we apply a sequence-level classifier. For SA, we apply max pooling strategies (average, max, last token) over the BiLSTM output and feed the concatenated result into a classification head with focal loss.

3.1.1 Loss Functions

In our implementation, we chose not to use a Conditional Random Field (CRF) layer for decoding NER outputs, despite it being present in the original paper. The main reason for this decision was the incompatibility of loss magnitudes during joint training.

Specifically, we observed that the loss from the CRF head was several orders of magnitude higher (often $\sim$2000x larger) than the CrossEntropy or Focal Loss values typically used for sentiment classification. This resulted in the model ignoring the sentiment analysis task entirely, as its gradients became negligible during backpropagation.

To address this, we considered scaling the sentiment loss or normalizing the CRF output. However:

- Scaling the SA loss upwards led to unstable training and poor convergence.
- Normalizing the CRF loss prevented the NER model from learning altogether.

Therefore, we opted for a simpler and more stable approach: replacing the CRF with a standard CrossEntropy loss and using argmax for NER decoding. For both NER and sentiment analysis, we use the CrossEntropy loss function, which is widely adopted for classification tasks and ensures a balanced contribution of both objectives to the gradient updates.

The total loss used during training is defined as:

$$\mathcal{L}_{total} = \mathcal{L}_{NER}^{CE} + \mathcal{L}_{SA}^{CE}$$

3.2 Alert Generation

The alert generation class (AlertGenerator) combines **in-context learning** with a two-stage prompting strategy to transform the NER + SA output into well-defined alerts. The **Candidate Alert Prompt** ensures format constraints, examples, and step-by-step reasoning to generate multiple valid alerts in a loop, while the **Alert Selection Prompt** applies certain criteria to identify the best output using comparative analysis.

3.2.1 LLM selection

This task was achieved using the **DeepSeek-R1-Distill-Qwen-7B** large language model (LLM), which was selected after evaluating several candidate models, including GPT-2 and Mistral-7B, which were ultimately discarded due to limitations in output quality and poor reasoning.

3.2.2 Prompt Construction

The model alert generation is based on two main prompts:

Candidate Alert Prompt: The prompt design for the `generate_alert()` self function was carefully structured to guide the generative model to produce high-quality and relevant alerts. This prompt focused on, taking into account all the given inputs for implementing **in-context learning** through three key components:

- Format Instructions: Explicitly requires the "theme: entity" structure (1-3 word themes) to ensure concise, standardized outputs.
- Examples: Demonstrates the task of the model through various representative samples: "Health measures: European Union", "Diplomatic visit: Angela Merkel", "Economic warning: IMF"...
- Reasoning Steps: Guides the model to first identify the core event type, then select the most relevant entity, and finally generate a non-redundant theme.

This in-context approach enables the model to learn the alert generation task dynamically without fine-tuning, while maintaining output quality through structured reasoning.

However, LLM´s rarely return the same answer and some of them might be better than others. Due to this reason, the Alert Generator will provide the LLM with the same prompt multiple times, to generate various different alerts.

After, filtering all the alerts with a function in charge of discarding bad structure answers, the task of picking which one of the alerts is the best one will fall upon our LLM.

Alert Selection Prompt: This second prompt is more concise, but similar to an equally structured in-context learning approach. Even though it's shorter, it mantains the critical instructional components:

- Evaluation Criteria: The prompt establishes clear evaluation criteria by enumerating the essential qualities of an optimal alert.
- Examples: In-context learning is introduced through representative examples that show the expected selection format.
- Systematic Decision Process: The model should first analyze all of the options, taking into account the original text and the specified criteria, before choosing the best option.

However, when the input lacks of clear entities or has little sense, the model can fail to produce a coherent alert. There are two possible outcomes in this cases: the model either returns NO VALID ALERT (inserted manually when the input is poor and non-sense), or the model hallucinates a date as the theme (e.g., "2023-01-15: EU's veterinary committees"). This behavior occurs because of the pretraining of DeepSeek on large-scale web corpora. When uncertain the model mimics frequent patterns (date-prefixed headlines), inserting dates even if they aren't present in the input.

4 Data

We use an extended version of the CoNLL-2003 dataset, originally designed for Named Entity Recognition (NER), and adapt it for a multi-task setting that also includes Sentiment Analysis (SA). Each instance in the dataset corresponds to a sentence composed of tokens and their corresponding NER tags. The format of the raw data is plain text with one token per line, followed by its label. Sentences are separated by blank lines, and the dataset is split into three subsets: **training**, **validation**, and **testing**.

The dataset contains a total of 22,137 sentences annotated for both NER and sentiment analysis. For the sentiment task, labels were generated automatically using a pretrained model, resulting in a distribution of approximately 64% positive and 36% negative sentences—indicative of a moderate but manageable class imbalance.

Preprocessing steps. The raw dataset is first parsed to extract token sequences and their corresponding NER labels. For each sentence, a binary sentiment label (positive or negative) is automatically generated using the pre-trained model `siebert/sentiment-roberta-large-english` from HuggingFace's `pipeline`.

Subsequently, we construct the vocabulary using the training split only. This includes word-level and character-level indices, as well as dependency relation labels used later by the GCN. To extract syntactic structure, we rely on the `en_core_web_sm` parser from `spaCy`, which allows us to build dependency graphs for each sentence. These graphs are converted into tensors for edge indices and dependency types to be directly consumed by the graph-based module during training.

Finally, to speed up future runs and avoid redundant preprocessing, all the processed data is serialized and stored as a `.pkl` file. The dataset is partitioned into a training set of 14,987 sentences, a validation set of 3,466 sentences, and a test set of 3,684 sentences.

NER Label Distribution: There are 9 unique NER tags, including the non-entity label `O`, with the most common entity types (excluding `O`) being `B-LOC`, `B-ORG`, and `B-PER`. The table below shows the distribution of named entity labels across the training, validation, and test splits, excluding the majority class `O`.

NER Tag	Training	Validation	Testing	Total
B-LOC	7,140	1,837	1,668	10,645
B-MISC	3,438	922	702	5,062
B-ORG	6,321	1,341	1,661	9,323
B-PER	6,600	1,842	1,617	10,059
I-LOC	1,157	257	257	1,671
I-MISC	1,155	346	216	1,717
I-ORG	3,704	751	835	5,290
I-PER	4,528	1,307	1,156	6,991
TOTAL (excl. O)	**34,043**	**8,603**	**8,112**	**50,758**

Table 1: NER tag frequency per dataset split, excluding the `O` label.

Sentence Length Analysis: Sentence lengths range on average between 12 and 15 tokens across the splits, with medians between 8 and 10 tokens and a few outliers exceeding 100 tokens. But overall the three splits present the same distribution of lengths. Figure 2 shows the distribution of sentence lengths (in tokens) for each split.

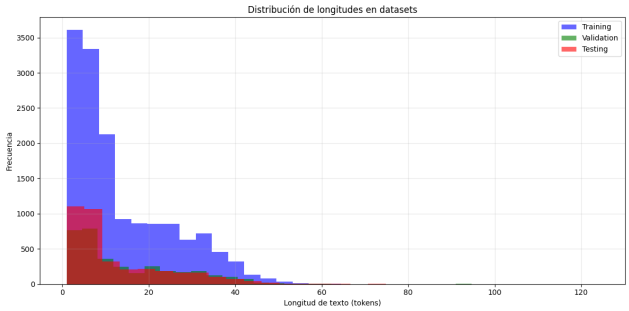

Figure 2: Distribución de longitudes de oración.

Sentiment Distribution: The sentiment labels generated by the pre-trained model are relatively balanced across all dataset splits, though slightly skewed toward the positive class. On average, approximately 64% of the sentences are labeled as positive. We believe this moderate imbalance does not critically affect the model's performance, as it is still exposed to a sufficient number of negative examples during training.

5 Experiments

Our initial setup followed the optimizer and scheduler strategy suggested in the paper, using a standard SGD optimizer with a StepLR scheduler. While the model was able to learn the NER task reasonably well, the sentiment analysis (SA) head failed to converge. We hypothesize this is due to the shallow architecture of the SA, which made SGD unsuitable.

To address this, we switched to the AdamW optimizer, which provided more stable gradients and faster convergence for both tasks. The SA performance in particular improved significantly after this change.

However, new challenges emerged during multitask training:

- **Unstable training in early epochs:** Although the model was mostly stable, we noticed occasional large gradient values from the recurrent and GCN components. To be safe and prevent potential instability, we applied max-norm gradient clipping during each optimizer step.
- **Disparity in gradient magnitudes:** The sentiment branch, having fewer parameters and shallower structure, produced smaller gradients than the NER model. This made it difficult for the optimizer to balance learning. To mitigate this, we introduced two separate parameter groups in the optimizer, assigning a $3\times$ larger learning rate to the SA head.
- **Overfitting in long training runs:** Especially in NER, we noticed that training could continue even after validation performance had plateaued. To avoid this, we implemented task-specific early stopping with a patience of 5 epochs and a minimum delta of 0.001. Additionally, we applied weight decay (0.05) to penalize large weights and included dropout layers (with rates of 0.3 and 0.4) in both NER and SA heads.
- **Loss stagnation for SA:** Initial experiments with fixed-step learning rate schedulers did not yield satisfactory enough results. Our hypothesis is that the sentiment labels, being automatically generated by a pretrained model, are inherently noisy. To address this, we applied a `ReduceLROnPlateau` scheduler that monitors the SA F1 score and adaptively lowers the learning rate when progress slows down.

These modifications were gradually introduced as we identified specific issues during training. Each design decision aimed to improve either optimization stability, task balance, or generalization capacity.

A summary of the key training parameters and technical choices is provided in Table 2.

Parameter	**Value**
Batch size	64 (balanced with gradient accumulation)
Effective batch size	128 (via 2-step gradient accumulation)
Optimizer	AdamW with 2 param groups
Learning rate	1×10^{-3} (base), 3×10^{-3} (SA head)
Weight decay	0.05
Gradient clipping	Max-norm 1.0
Epochs	Max 30
Scheduler (NER)	None
Scheduler (SA)	ReduceLROnPlateau
Early stopping	Patience = 5, $\Delta = 0.001$ (independent for each task)
Dropout	0.3 (NER head), 0.4 (SA head)

Table 2: Training hyperparameters and system-level decisions.

6 Results

In this section, we present the evaluation of our proposed multitask learning models for Named Entity Recognition (NER) and Sentiment Analysis (SA). Our primary goal was to assess the impact of incorporating syntactic dependency features in the NER module, as prior work by Xu suggested substantial improvements using such features. To conduct this comparison, a standard LSTM was trained without syntactic dependency information (Model D) and compared against a synergized LSTM (SynLSTM) model (Model C).

Additionally, we explored the effect of separating the architectures for NER and SA by using two distinct LSTMs—one bidirectional LSTM for SA predictions and a SynLSTM for NER predictions (Models A and B). Finally, we examined the contribution of character-level embeddings by enabling or disabling them in selected models.

- **Model A:** Two separate LSTMs (SynLSTM for NER and two-layer BiLSTM for SA) with character embeddings.
- **Model B:** Same as Model A but without character embeddings.
- **Model C:** A single shared SynLSTM for both tasks.
- **Model D:** A single shared standard LSTM for both tasks.
- **Model E:** Same as Model C but with character embeddings.

6.1 Evaluation Metrics

Table 3 reports the F1 scores obtained for NER and the accuracy scores for SA on the test set. Additionally, Figure 3 visualizes the model performance across both tasks to facilitate a comparative analysis.

Table 3: Test set performance of the multitask models for NER (F1 score) and Sentiment Analysis (Accuracy).

Model	**NER F1 (%)**	**SA Acc (%)**
Model A: 2 LSTMs + SynLSTM + Char	94.05	84.28
Model B: 2 LSTMs + SynLSTM	92.59	83.31
Model C: Shared SynLSTM	92.55	83.50
Model D: Shared Standard LSTM	92.84	83.06
Model E: Shared SynLSTM + Char	94.20	84.55

6.2 Analysis

As shown in Table 3, all models share similar F1 scores, model C slightly surpasses model B with a accuracy of (83.50% vs. 83.31%) in SA and suggests that a simpler model works better to prevent overfitting that occurs with more complex models with independent LSTMs for NER and SA. Additionally, model C achieved comparable results with fewer parameters as model B with a F1 score of 0.9257%, almost identical to the models mentioned above. This could be attributed to shared sequence patterns that benefit both NER and SA tasks.

Figure 3 it is observed how character embeddings had an impact on NER metrics, with a 0.94% F1 score. Furthermore, it simultaneously affects the SA in comparison to the other models by increasing the accuracy, which seems to find productive the character information for the SA predictions. This is interesting because among all the other variables explored in the analysis, the character embeddings seem to cause the most notable change in the NER and in the SA too.

Interestingly, the NER F1 scores among models without character embeddings are marginally higher in the model without a synergized-LSTM (92.84%) and Model C (92.55%). This remarks that there may be a bigger dependency for the LSTM to interpret the syntactic dependencies with the character embeddings.

Finally, we see the benefits of having a simpler model with model E and also the surprising improving impact on the metrics of the inclusion of the character embeddings.

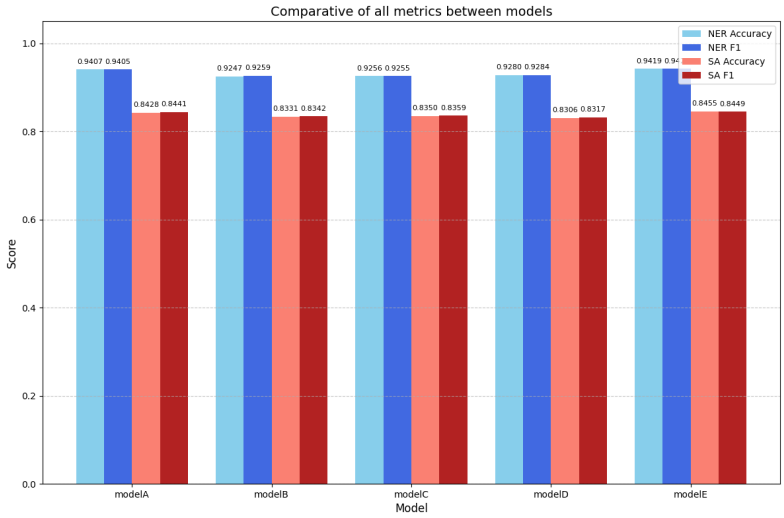

Figure 3: F1 and accuracy scores for NER and Sentiment Analysis across model configurations.

7 Conclusion

The results presented demonstrate that while all models achieved reasonably strong performance on both NER and SA, the inclusion of character embeddings along with using the syntactical dependencies graph and the GNC layer, had the most significant positive impact. Specifically, Model E, which integrates character embeddings into a shared SynLSTM, achieved the highest F1 score for NER (0.9420) and the best SA accuracy (0.8455), indicating that character-level information enhances sequence encoding in both tasks.

Interestingly, while models with independent LSTMs (A and B) performed competitively, the shared SynLSTM architecture in Model C attained comparable results with fewer parameters, suggesting that a shared sequence encoder can effectively capture patterns beneficial for both NER and SA without sacrificing performance. Moreover, the simplest model, Model D, using a standard LSTM, delivered slightly lower scores, emphasizing the value of syntactic dependency features in multitask learning scenarios.

Another notable observation is that among models without character embeddings, Model D achieved a marginally higher NER F1 than Model C, implying that when syntactic dependencies are not reinforced with character-level information, a standard LSTM may generalize better for entity recognition.

Overall, the findings confirm that character embeddings substantially benefit NER performance and slightly improve SA predictions, and that a shared SynLSTM encoder strikes a balance between performance and architectural efficiency.

References

[1] Xu, L., Jie, Z., Lu, W., and Bing, L. (2021). Better Feature Integration for Named Entity Recognition. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. `https://aclanthology.org/2021.naacl-main.271.pdf`

[2] DeepSeek AI. (2024). DeepSeek-R1-Distill-Qwen-7B. *GitHub Repository*. `https://github.com/deepseek-ai/DeepSeek-VL`