How does the integration of syntactic features (POS tags or dependency parsing) during BERT fine-tuning influence the performance of semantic parsing?

Jinrui Zhang, University of Twente, The Netherlands

BERT has achieved outstanding performance in many NLP tasks, but its implicit handling of syntax may limit its effectiveness in shallow semantic parsing. This study explores incorporating explicit syntactic features, such as POS tags and dependency parsing labels into BERT. This is completed through two approaches: addition or concatenation either at the input embedding layer or after transformer layers. Experiments have been conducted, and results show that the additive approach with dependency parsing labels at input embedding layer achieves the highest F1 score, improving performance by 1.24%. This work provides insights into the integration of syntactic features in BERT.

Additional Key Words and Phrases: BERT, Semantic Parsing, Large Language Model, BIO tagging, Dependency Parsing, POS tagging.

1 INTRODUCTION

1.1 Context and Motivation

Shallow semantic parsing identifies the roles of words or phrases in a sentence and represents them in a structured format [1]. In semantic parsing, capturing syntactic and semantic relationships, such as grammatical roles and word dependencies among words in a sentence, is essential to understanding its meaning [2]. BERT, a Large Language Model (LLM), learns syntactic and semantic meanings of sentences using self-attention mechanisms and contextualized embeddings from large datasets during pre-training [3]. Instead of incorporating syntactic features like grammatical roles and word dependencies, BERT implicitly learns syntactic relationships during pre-training [4]. While BERT has achieved impressive results, its design of not explicitly incorporating syntactic information may constrain its performance. Research has shown that incorporating explicit syntactic information with contextual embeddings can improve performance [2]. Therefore, adding explicit syntactic features to BERT may enhance its ability to capture syntactic and semantic relationships.

1.2 Specific Problem

Understanding grammatical roles and sentence structures are critical for shallow semantic parsing. The limitation of (learning syntax implicitly) can cause it to underperform in such tasks. For example, understanding whether a word is the subject or object is essential for identifying its role in a sentence.

POS tags and dependency parsers could help address this issue. POS gives each word in a sentence a grammatical category, such as nouns, verbs, adjectives, and adverbs [5]. Dependency parsers provide the dependencies between the words of a sentence such as conjunct, determinant, and nominal modifier [6].

POS tags and dependency parsers (explicit syntactic features) can complement BERT's implicit learning during fine-tuning.

This research explores the potential of integrating POS tags and dependency parsing labels into BERT to improve its ability to capture grammatical and semantic relationships in shallow semantic parsing.

1.3 Research Questions

This study addresses the following questions:

1. How does adding explicit syntactic features, such as POS tags or dependency labels, during BERT fine-tuning affect performance in shallow semantic parsing?
2. Which strategy for adding syntactic features (additive at input embeddings, additive post-transformer layers, or concatenation) works best for shallow semantic parsing tasks?
3. Which syntactic feature, POS tags or dependency parsing labels, has a greater impact on the performance of shallow semantic parsing?

2 RELATED WORK

The study *Syntax-Infused Transformer and BERT models for Machine Translation and Natural Language Understanding* discusses incorporating syntactic features like POS tags, case, and subword positions into the Transformer and adding explicit syntactic features (POS tags) to BERT's token embeddings. Two approaches are introduced for the integration of syntactic features like POS tags into BERT. One is through addition, the trainable POS embeddings with the same dimensions as the token embedding are added. The other is implemented by concatenating POS embeddings with the token embeddings. Unlike in addition, concatenation increases the size of the dimensions of the resulting embeddings. To resolve this, the resulting embeddings are passed to a trainable affine transformation that maps the dimensions back to the required

TScIT 42, January 31st, 2025, Enschede, The Netherlands

size. It has concluded that the approach achieved performance improvements in machine translation (MT) and natural language understanding (NLU) tasks. The study focused on low-resource settings (as few as 2.5k sentences) [7].

However, this study limited the potential benefits of incorporating syntactic features at different stages of the model such as after transformer layers. Additionally, the reliance on only POS tags may also limit the understanding of semantic structures and relationships since POS tags only help identify the roles of words in a sentence but cannot represent grammatical relationships or dependencies in a sentence [6].

Another study introduces a local attention mechanism that uses syntactic distances from dependency trees to restrict attention to syntactically relevant tokens. BERT with Syntax-Aware Local Attention (SLA) leads BERT to focus more on local syntactic relevance, meaning it emphasizes relationships between words that are close to each other in the syntactic structure of a sentence. The approach improves performance on single-sentence classification and sequence labeling tasks [8]. However, SLA separates local and global attention computations. The attention scores are later combined with a gated unit thus it may lead to a limitation of the synergy between local and semantic information [8]. In contrast, adding dependency parsers at the input embedding level of BERT directly into token embeddings enables better integration of syntactic and contextual information.

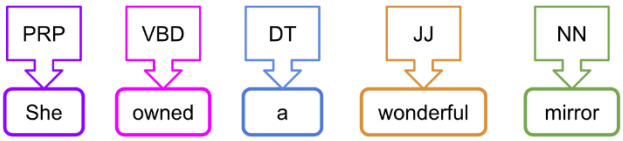
These limitations in the existing solutions address the gaps and highlight the need for other approaches.

3 BACKGROUND

3.1 BERT

BERT, or Bidirectional Encoder Representations from Transformers, is a large language model built on transformer architecture to process text. Its name indicates that BERT uses a bidirectional approach to analyze text. This approach allows BERT to consider a word's context by examining its preceding and succeeding words [3]. The model is pre-trained on massive text datasets, such as BooksCorpus and English Wikipedia. Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) are applied during pre-training. In Masked Language Modeling (MLM), 15% of the tokens in the input sequence are masked at random, and the model learns to predict them using the surrounding context. NSP, where the model is provided with pairs of sentences and learns to predict whether the second sentence follows the first logically [3]. The self-attention mechanism evaluates how much focus should be placed on each word relative to others in the sentence [9]. In self-attention, each word is converted into query, key, and value vectors first, then compared with the vectors of other words in the sentence [9]. The comparisons are used to compute attention scores. These scores show how strongly one word relates to another. BERT uses these scores to produce weighted representations of each word in the context of the sentence. This self-attention mechanism is applied across multiple layers in BERT and implicitly allows BERT to learn syntactic and semantic information [3]. BERT can be fine-tuned for specific downstream tasks, and a task-specific layer can be added to the model. For instance, a classification head can be added for token classification tasks such as shallow semantic parsing, where each token in the input sequence is assigned a label that represents its semantic role (e.g., predicate, argument, or modifier). Fine-tuning typically requires much less data than training, making BERT adaptable to various NLP tasks, including shallow semantic parsing [3].

3.2 POS Tagging



PRP: Personal Pronoun
VBD: Verb, Past Tense
DT: Determiner
JJ: Adjective
NN: Noun, Singular or Mass

Figure 1: An example of POS tagging.

Part-of-speech (POS) tagging assigns a tag to each word in a sentence. Each tag represents a word's grammatical role [10]. POS tagging is done by splitting the sentence into individual words and then analyzing each word using lexical information and context to determine the appropriate tag [10]. For example, in the sentence "She owned a wonderful mirror." each word gets assigned a tag based on its syntactic role in the sentence (See Figure 1).

3.3 Dependency Parsing

Dependency parsing identifies relationships between words, known as dependencies. These dependencies are shown in a dependency tree, where words are represented as nodes, and dependencies are represented as directed edges [10]. For example, in the sentence “She owned a wonderful mirror”, “owned” is the head of “she”, and the dependency between them is labeled as nsubj (nominal subject). The nsubj relationship indicates that "she" is the grammatical subject of the verb "owned". In other words, "she" is the entity performing the action of owning something (See Figure 2). In addition, dependency parsing builds on syntactic roles to identify grammatical dependencies in a sentence.

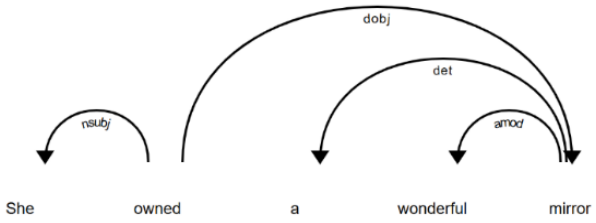


Figure 2: An example of dependency parsing.

4 NEW ARCHITECTURE

4.1 Objective

The objective of this task is to perform shallow semantic parsing by predicting semantic roles in a sentence, represented using BIO tagging. BIO tagging assigns tags to tokens or words with their corresponding semantic roles in a sentence. Ultimately, the aim is to identify the main action and determine who did what to whom, where, and when, represented in BIO format.

In BIO tagging:
B-TAG: Indicates that the token is the first word of a specific entity or semantic role chunk.
I-TAG: Indicates that the token is inside the entity or chunk and follows a "B-" tag.
O-TAG: Indicates that the token does not belong to any semantic role or entity chunk.

4.2 Approach 1: Incorporating Syntactic Features into the Input Embedding Layer

Integrating POS/DEP embeddings alongside traditional input embeddings extends the standard BERT model for token classification (See Figure 3). This integration adds an additional embedding layer designed to encode Part-of-Speech (POS) or dependency parsing (DEP) labels (Each child in the dependency tree gets the corresponding dependency relation to its parent word.). The embedding layer maps POS tags or DEP labels into dense representations with the same dimensionality as BERT embeddings (D=768). Each token is augmented with syntactic information derived from POS/DEP embeddings, which helps the model better understand grammatical relationships in the sentence.
The '+' symbol indicates two methods of combining syntactic information with token embeddings. The first is the additive approach, where POS/DEP embeddings are element-wise added to token embeddings. For example, if the token embeddings are (0.7, 0.3, 0.2) and the POS/DEP embeddings are (0.1, 0.8, 0.3), the result is (0.8, 1.1, 0.5). This approach keeps the dimensionality the same (D). The second method is concatenation, where the embeddings are simply joined. For example, token embeddings (0.7, 0.3, 0.2) concatenated with POS/DEP embeddings (0.1, 0.8, 0.3) result in (0.7, 0.3, 0.2, 0.1, 0.8, 0.3), doubling the dimensionality (D×2). To handle this increase, a trainable linear projection layer reduces the dimensionality back to D.
The enriched embeddings are then combined with segment and positional embeddings to form the final input embeddings, which are passed into BERT's encoder layers (12 layers for BERT Base). Within the encoders, these enriched embeddings are processed through 12 transformer layers to generate contextualized embeddings. In the BERT for token classification model, these embeddings combine the semantic information learned during BERT's fine-tuning with the syntactic information provided by POS/DEP embeddings. After passing through the encoder, the contextualized embeddings are fed into a linear classification layer. This layer maps each token's contextualized embedding to a task-specific set of logits. These logits are converted into probabilities using softmax for prediction.
For example, if there are 20 BIO tags in total, the linear layer generates 20 logits (one for each BIO tag) for every token. These logits are then passed through softmax to predict the most likely BIO tag for each token.
For instance, in the example 'LABEL_1', 'score': 0.69566524, 'word': 'I', the word 'I' receives the highest score for LABEL_1 among all BIO tags, meaning it is classified as the BIO tag corresponding to LABEL_1.

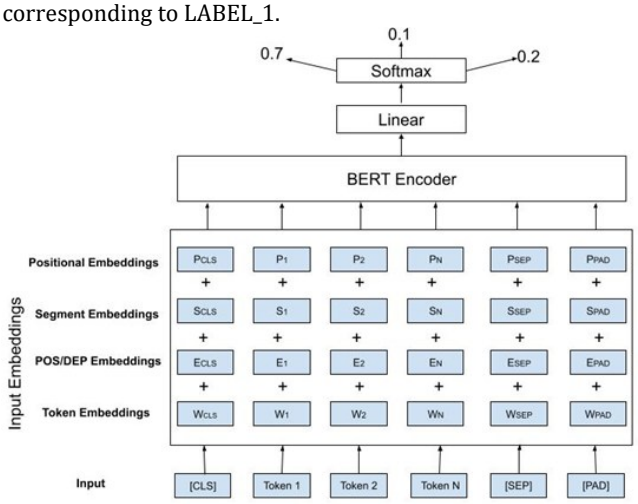


Figure 3: The BERT base model for token classification + POS/dependency parsing embeddings at the input embedding layer.

4.3 Approach 2: Incorporating Syntactic Features after Transformer Layers

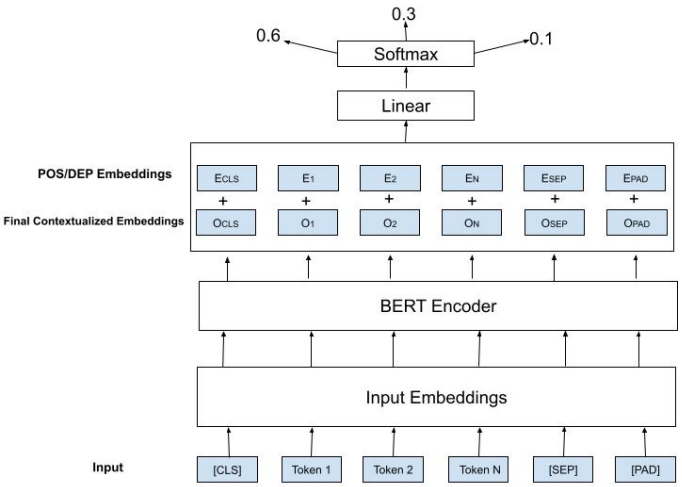


Figure 4: The BERT base model for token classification + POS/dependency parsing embeddings after transformer layers.

Tokens are first mapped to their respective input embeddings by the embedding layer of BERT model for token classification. Next, these embeddings are fed into 12 transformer layers of BERT base to get contextualized embeddings. These embeddings are provided by the last hidden layer, represented as O1, O2, etc. (See Figure 4). These contextualized embeddings are then enriched by combining token embeddings with POS/DEP embeddings through either addition or concatenation, ensuring syntactic information is incorporated.
The embeddings are integrated in the same way as described in approach 1. These enriched contextualized embeddings are then fed into a linear classification layer that generates logits for each token. Finally, the logits are converted into probabilities using a softmax layer to predict the most likely BIO tag for each token.

5 METHODOLOGY

This study applies a quantitative methodology to measure and compare the performance of semantic parsing using BERT under different conditions:

- Baseline Condition: Fine-tuning BERT without incorporating explicit syntactic features.
- Experimental Condition: Fine-tuning BERT with the integration of syntactic features, such as Part-of-Speech (POS) tags or Dependency Parsing.

The steps for each condition will be explained below. The tools and frameworks used during the process will also be addressed.

5.1 Frameworks and Tool

The frameworks and tools used are as follows:

- PyTorch: Used to customize the model imported by the Hugging Face Transformers.
- Hugging Face Transformers: Provide a Pre-trained BERT model and tools necessary for training (optimizer, Trainer, Training Arguments, scheduler).
- SpaCy: Used for extracting syntactic features (POS tags and dependency parsing labels).
- Regex (re): Used to extract entities and tags in the dataset.
- SeqEval: Used to calculate precision, recall, F1-score, and accuracy for predictions.
- NumPy: Used to process logits and align predictions during evaluation.

5.2 Process Overview

5.2.1 Baseline Condition

Figure 5: Baseline Condition Process Overview.

Baseline Condition Preprocessing:
In the baseline condition, the dataset is preprocessed without incorporating syntactic features like POS tags or dependency parsing labels. Sentences are converted into tokenized inputs aligned only with BIO tags for shallow semantic parsing. Each sentence in the database follows the format [TAG: entity], such as [Actor: He] [Action: sat down] [Location: on a stone]. The raw sentences are processed using regular expressions to extract entities and their corresponding tags. After processing, clean sentences are produced, such as "He sat down on a stone." and BIO tags are aligned at the word level, for example, [B-Actor: He] [B-Action: sat] [I-Action: down] [B-Location: on][I-Location: a] [I-Location: stone] [O: .]. Next, the BERT tokenizer tokenizes words into subword tokens to ensure compatibility with the BERT model. For example, the word "ebony" might be split into subwords ['e', '##bon', '##y'], and each subword gets the same BIO tag as the original word. Note that not all the words will be split into subwords, only rare words that are not in the tokenizer's vocabulary. Special tokens like [CLS], [SEP], and [PAD] are assigned a value of -100 for BIO tags ensure that these tokens are excluded from loss computation because they are not part of the actual input that requires tagging [3]. Doing so prevents irrelevant tokens from having an effect on model learning.

Stage	Step
Dataset Preprocessing	See Baseline Condition Preprocessing.
Dataset Splitting	Step 1: Shuffle the tokenized dataset with seed 42 to ensure the dataset is randomized in a consistent and reproducible manner.
	Step 2: Split the tokenized dataset into training, validation, and test sets (70%, 15%, 15%).
Model Setup	Step 3: Load the pre-trained BERT model (bert-base-cased) for token classification.
	Step 4: Define the number of output labels (BIO tags).
Training	Define Training Arguments (epochs, learning rate etc.)
	Step 5: Train the model on the training dataset and validate it on the validation dataset.
Evaluation	Step 6: Evaluate the model on the test dataset using the seqeval metric.
	Step 7: Compute performance metrics (precision, recall, F1-score, accuracy).

5.2.2 Experimental Condition

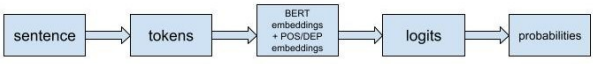


Figure 6: Conditional Condition Process Overview.

Experimental Condition Preprocessing:
In the experimental condition, the preprocessing is extended to include syntactic features like POS tags or dependency parsing labels. After processing the raw sentences and aligning BIO tags as described earlier, the clean sentences are passed through SpaCy to extract dependency tags or POS tags, which are mapped to unique IDs. For example, if "ebony" is assigned a BIO tag of 14 and a POS tag of 26, the subwords ['e', '##bon', '##y'] will also get assigned the same BIO tag and POS tag. The BERT tokenizer tokenizes the words, aligning the BIO tags and POS/dependency labels to subword tokens. Special tokens like [CLS], [SEP], and [PAD] are assigned a value of -100 for BIO tags and 0 for POS or dependency parsing labels [3].

- Approach 1: Incorporating Syntactic Features into the Input Embedding Layer

Stage	Step
Dataset Preprocessing	See Experimental Condition Preprocessing.
Dataset Splitting	Step 1: Shuffle the tokenized dataset with seed 42 to ensure the dataset is randomized in a consistent and reproducible manner.
	Step 2: Split the tokenized dataset into training, validation, and test sets (70%, 15%, 15%).
Model Customization	Step 3: Add an embedding layer for dependency parsing labels/POS tags.
	Step 4: Add or concatenate DEP/POS embeddings to token embeddings.
(Only needed for concatenation)	Step 5: Define a projection layer to reduce concatenated embeddings back to the dimensions of BERT.
Model Setup	Step 6: Load the customized BERT model (bert-base-cased) for token classification.
	Step 7: Define the number of output labels (BIO tags).
Training	Step 8: Define training arguments.
	Step 9: Use a custom optimizer with different learning rates for BERT and dependency label embeddings/POS tags.
	Step 10: Use a linear learning rate scheduler to gradually adjust the learning rate during training.
	Step 11: Train the model on the training dataset and validate it on the validation dataset.
Evaluation	Same as baseline.

- Approach 2: Incorporating Syntactic Features after Transformer Layers

Stage	Step
Dataset Preprocessing	See Experimental Condition Preprocessing.
Dataset Splitting	Step 1: Shuffle the tokenized dataset with seed 42 to ensure the dataset is randomized in a consistent and reproducible manner.
	Step 2: Split the tokenized dataset into training, validation, and test sets (70%, 15%, 15%).
Model Customization	Step 3: extract the contextualized embeddings from the last hidden layer.
	Step 4: Add or concatenate DEP/POS embeddings to contextualized embeddings.
(Only needed for concatenation)	Step 5: Define a projection layer to reduce concatenated embeddings back to the dimensions of BERT.
Model Setup	Step 6: Load the customized BERT model (bert-base-cased) for token classification.
	Step 7: Define the number of output labels (BIO tags).
Training	Step 8: Define training arguments.
	Step 9: Use a custom optimizer with different learning rates for BERT and dependency label embeddings/POS tags.
	Step 10: Use a linear learning rate scheduler to gradually adjust the learning rate during training.
	Step 11: Train the model on the training dataset and validate it on the validation dataset.
Evaluation	Same as baseline.

6 EVALUATION

6.1 Evaluation Metrics

SeqEval, a Python framework, is used here to calculate precision, recall, F1-score, and accuracy for predictions. In addition, three confusion matrices are visualized to analyze how well the model predicts each tag and to evaluate the impact of grammar on the predictions.
Since the database is imbalanced, with a higher proportion of "O" tags compared to the other tags, F1 is preferred for evaluation in this task. Unlike accuracy which reflects only overall correctness, F1 balances precision and recall making it more suitable for evaluating the performance of the minority classes [10].

6.2 Experimental Setup

6.2.1 Data Split

The dataset consists of a total of 2041 samples, split as follows:

- 70% train data
- 15% validation data
- 15% test data

To ensure the reliability of the results, the dataset is shuffled before splitting, and the training, validation, and test dataset remain consistent across all approaches.

6.2.2 Hyperparameters

Table1: Training Hyperparameters of BERT without syntactic features

Hyperparameters	Values
Epochs	10
Learning Rate	0.00001
Training Batch Size	32
Evaluation Batch Size	16
LR Warmup	0.2
LR Scheduler	linear
Dropout	0.1

Table2: Training Hyperparameters of BERT with syntactic features

Hyperparameters	Values
Epochs	10
Learning Rate BERT	0.00002
Learning Rate POS/DEP	0.0001
Training Batch Size	32
Evaluation Batch Size	16
LR Warmup	0.2
LR Scheduler	linear
Dropout	0.1

When fine-tuning a model, hyperparameters like batch size, learning rate, and number of epochs significantly impact model's performance and convergence speed. The optimization of hyperparameters is crucial for effective fine-tuning [11]. In this case, all hyperparameters are consistent across approaches except for learning rates. Since BERT parameters are pre-trained and POS/DEP embeddings are randomly initialized, learning rates are adjusted to train syntactic embeddings more quickly.

6.3 Results

6.3.1 SeqEval Metrics

Table1: Performance of BERT base model for token classification.

Precision (%)	Recall (%)	F1 (%)	Accuracy (%)
81.64	85.49	83.52	88.07

Table2: Performance of BERT base model for token classification with POS/DEP using different integration approaches.

Approach	Syntactic Features	Precision (%)	Recall (%)	F1 (%)	Accuracy (%)
Addition at input embedding	POS	82.74	84.75	83.73	88.57
	DEP	83.36	86.17	84.74	89.46
Concatenation at input embedding	POS	69.16	72.94	71.00	78.20
	DEP	69.76	73.95	71.80	77.90
Addition after transformers	POS	82.05	84.82	83.41	88.40
	DEP	83.18	85.76	84.45	88.77
Concatenation after transformers	POS	81.48	84.01	82.72	87.44
	DEP	82.79	84.41	83.60	87.94

Two approaches incorporating syntactic features outperform BERT without explicit grammar, while other approaches show minor improvements to be worth discussing. Adding dependency parsing at the input embedding layer achieves an F1 score of 84.74%, showing an improvement of 1.24%. Likewise, adding dependency parsing after the transformer layers achieves an F1 score of 84.45%, with an improvement of 0.93%. Among all approaches, the additive approach with dependency parsing at the input embedding layer shows the best performance. In contrast, concatenating syntactic features at the input embedding layer leads to a significant decrease in performance. Moreover, dependency parsing consistently outperforms POS tags across all approaches.

6.3.2 Confusion Matrix

The entries of diagonal down in the Confusion Matrix represent the correctly predicted BIO tags. Figure 8 displays the results of the best-performing approach, where 14 out of 27 tags show improved predictions. In contrast, Figure 9 represents the results of the worst-performing approach, where only two tags, 'O' and 'B-Action' are predicted more accurately compared to BERT without grammar (See Figure 7), while most other predictions show a decrease.

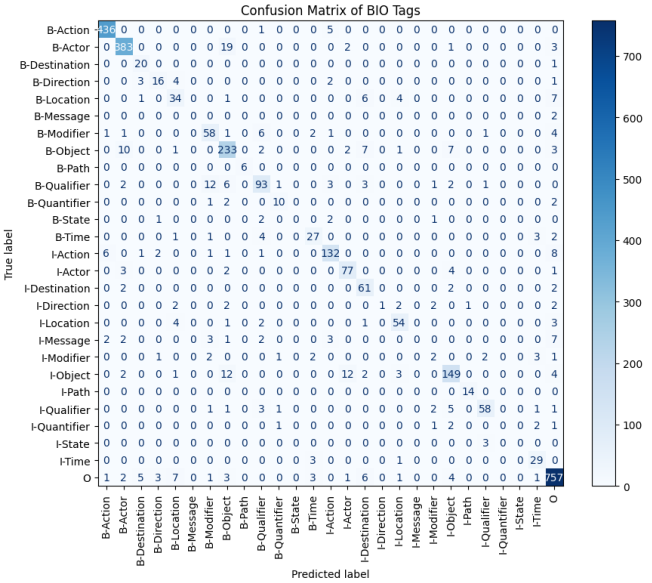


Figure 7: Confusion Matrix of BIO tags for BERT without syntactic features.

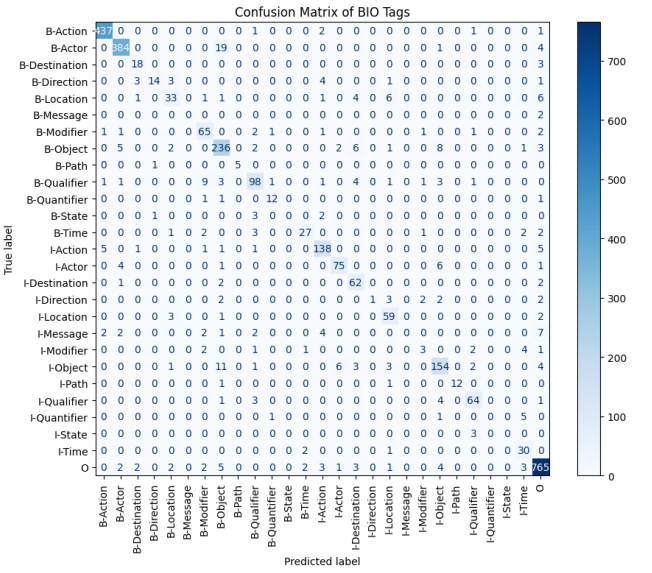


Figure 8: Confusion Matrix of BIO tags for BERT with Dependency Parsing added at input embedding.

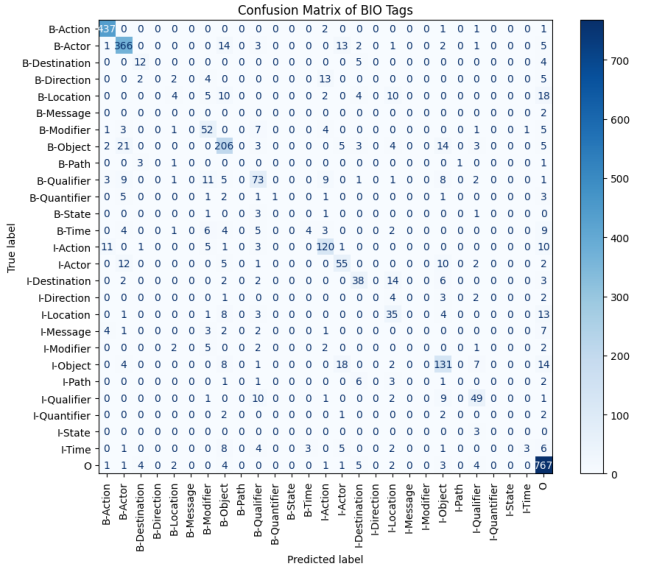


Figure 9: Confusion Matrix of BIO tags for BERT with POS concatenated at input embedding.

7 DISCUSSION

Additive approaches generally achieve better performance than concatenation approaches. Concatenation at the input embedding layer requires projecting the combined embeddings back to the dimensions compatible with BERT. This process creates noise and results in this approach performing the worst. However, concatenation after the transformer layers has minimal impact on performance because the embeddings are already contextualized by BERT before adding the syntactic features. Additionally, inconsistency in the dataset may limit the performance of the models. For example, in the sentence "He made up his mind…", the word 'up' should be considered as part of the action. However, it is labeled as a direction in the dataset that can confuse the models. Moreover, the size of the dataset is not large enough for efficient fine-tuning.

8 CONCLUSION

In this study, the impact of incorporating explicit syntactic features, such as POS tags and dependency parsing labels into BERT has been investigated. Various approaches have been implemented and evaluated for shallow semantic parsing in the form of BIO tagging. The results highlight the importance of selecting and integrating syntactic features carefully to enhance shallow semantic parsing performance. There are two main approaches: incorporating syntactic features (POS tags or dependency parsing labels) either through addition or concatenation at BERT's input embedding layer or incorporating them after the transformer layers.

Among the approaches tested, the additive approach at the input embedding layer with dependency parsing labels shows an improvement of 1.24%, Adding dependency parsing after the transformer layers achieves a 0.93% improvement. Concatenating syntactic features at the input embedding layer

significantly worsens the performance. While incorporating explicit syntactic features has potential, their impact depends heavily on the integration method and specific features used. Further research could explore alternative ways to incorporate syntactic information for more significant improvements and ensure the consistency of data.

REFERENCES

[1] Gildea, D., and Jurafsky, D. 2002. Automatic labeling of semantic roles. *Computational Linguistics* 28, 3 (Sept. 2002), 245 – 288. https://doi.org/10.1162/089120102760275983.

[2] Strubell, E., Verga, P., Andor, D., Weiss, D., and McCallum, A. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (Brussels, Belgium, Oct. 2018), Association for Computational Linguistics, 5027 – 5038. https://doi.org/10.18653/v1/D18-1548.

[3] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. 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)* (Minneapolis, MN, June 2019), Association for Computational Linguistics, 4171 – 4186. https://doi.org/10.18653/v1/N19-1423.

[4] Hewitt, J., and Manning, C. D. 2019. A structural probe for finding syntax in word representations. 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)* (Minneapolis, MN, June 2019), Association for Computational Linguistics, 4129 – 4138. https://doi.org/10.18653/v1/N19-1419.

[5] Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics* (Edmonton, Canada, May 2003), Association for Computational Linguistics, 252 – 259. https://doi.org/10.3115/1073445.1073478.

[6] Dozat, T., and Manning, C. D. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the 5th International Conference on Learning Representations* (Toulon, France, Apr. 2017). https://openreview.net/forum?id=Hk95PK9le.

[7] Sundararaman, D., Subramanian, V., Wang, G., Si, S., Shen, D., Wang, D., and Carin, L. 2019. Syntax-infused transformer and BERT models for machine translation and natural language understanding. *arXiv preprint arXiv:1911.06156*. https://arxiv.org/abs/1911.06156.

[8] Li, Z., Zhou, Q., Li, C., Xu, K., and Cao, Y. 2021. Improving BERT with syntax-aware local attention. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021* (Online, Aug. 2021), Association for Computational Linguistics, 645 – 653. https://doi.org/10.18653/v1/2021.findings-acl.57.

[9] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 6000 – 6010. https://arxiv.org/abs/1706.03762.

[10] Jurafsky, D., and Martin, J. H. 2009. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Pearson Prentice Hall.

[11] Oliver, M., and Wang, G. 2024. Crafting Efficient Fine-Tuning Strategies for Large Language Models. arXiv preprint arXiv:2407.13906. https://arxiv.org/abs/2407.13906.