Question 5)

These are the five hyper parameters that I used in my both models, the rest are by default.

1. **per_device_train_batch_size**:
    - This hyperparameter determines the number of training samples processed in parallel on each device (GPU/TPU) during training.
    - A larger batch size can lead to faster training but may require more memory.
    - I chose a value of 16 based on experimentation and the available resources. It balances between training speed and memory consumption.

2. **per_device_eval_batch_size**:
    - Similar to **per_device_train_batch_size**, this hyperparameter controls the batch size during evaluation (validation/testing).
    - A smaller evaluation batch size can be used compared to training batch size as it doesn't involve backpropagation.
    - I chose a value of 8 for evaluation batch size to reduce memory consumption during evaluation while still ensuring reasonable evaluation performance.

3. **num_train_epochs**:
    - This hyperparameter determines the number of times the entire training dataset is passed through the model during training.
    - Too few epochs may result in underfitting, while too many epochs can lead to overfitting.
    - I chose a value of 3 after experimenting with different values and observing the validation performance to prevent overfitting.

4. **save_steps**:
    - This hyperparameter specifies how often the model checkpoints are saved during training.
    - Saving checkpoints too frequently can slow down training, while saving them too infrequently may result in losing progress in case of a crash.
    - I chose a value of 1875 based on the total number of steps per epoch and desired checkpoint frequency.

5. **save_safetensors**:
    - This hyperparameter determines whether to save intermediate tensors during training.
    - Saving tensors can be useful for debugging but may consume additional disk space.
    - I chose to set this parameter to False to conserve disk space as it is typically not needed during normal training.

Regarding the impact of batch size on the output:

- A larger batch size generally leads to faster convergence during training due to more stable gradients and better utilization of hardware resources.
- However, larger batch sizes require more memory, and excessively large batch sizes may lead to poor generalization and convergence to suboptimal solutions (i.e., sharp minima).
- On the other hand, smaller batch sizes may result in slower convergence but can generalize better and avoid sharp minima.
- Therefore, choosing an appropriate batch size involves trade-offs between training speed, memory consumption, and model generalization.

The output of the model

1. For Model :

<class 'list'> <class 'list'>

Class O:

Precision: 0.9652136699883842

Recall: 0.971748630516403

F1 Score: 0.9684701263648633

Class B-PER:

Precision: 0.8101430429128739

Recall: 0.7936305732484077

F1 Score: 0.8018018018018019

...

Macro F1 Score: 0.7620920902104906

2. **For manually annotated sentences :**

<class 'list'> <class 'list'>
Class B-LOC:
Precision: 0.8888888888888888
Recall: 0.8
F1 Score: 0.8421052631578948
Class B-MISC:
Precision: 1.0
Recall: 0.0
F1 Score: 0.0
Class B-ORG:
Precision: 0.7142857142857143
Recall: 1.0
F1 Score: 0.8333333333333333
Class B-PER:
Precision: 0.8333333333333334
Recall: 0.625
F1 Score: 0.7142857142857143
Class I-LOC:
Precision: 1.0
Recall: 0.2
F1 Score: 0.33333333333333337
Class I-MISC:
Precision: 1.0
Recall: 0.0
F1 Score: 0.0
...
Recall: 0.991044776119403
F1 Score: 0.9609261939218524

Macro F1 Score: 0.5715890331499544

3. For ChatGPT annotated sentences :

<class 'list'> <class 'list'>
Class B-MISC:
Precision: 1.0
Recall: 0.0
F1 Score: 0.0
Class B-PER:
Precision: 0.5
Recall: 0.5
F1 Score: 0.5
Class I-MISC:
Precision: 1.0
Recall: 0.0
F1 Score: 0.0
Class I-PER:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
Class O:
Precision: 0.928
Recall: 0.9747899159663865
F1 Score: 0.9508196721311476

Macro F1 Score: 0.4901639344262295