Comparison Of Both Models & Learning on Naamapadam Dataset

Here are some points based on my observations:

Comparison of Models:

Indic-NER, being specifically designed for NER tasks, may have an advantage over Indic-BERT in terms of handling NER tasks out-of-the-box. This could explain why Indic-NER is giving less error or higher overall F1 score compared to Indic-BERT.

Indic-BERT, on the other hand, is a general-purpose language model fine-tuned for various tasks, including NER. However, since you are fine-tuning it again for the NER task, it might not perform as well as Indic-NER, which is already specialized for NER.

We can see that from below images of overall F1 Score of both the models

First photo overall macro F1 Score of Indic-BERT and second is of overall F1 Score of Indic-NER on Naamapadam Dataset

```
{'Precision': 0.6710362047440699, 'Recall': 0.6171067738231917, 'macro-F1 Score': 0.6429425837320574}
```

```
{'Precision': 0.7756801776790672, 'Recall': 0.8019517795637199, 'macro-F1 Score': 0.7885972339825007}
```

Comparison Of Both Models & Learning on 25 sentences

Labeling Issue:

Both models do not have the labels B-MISC and I-MISC, which means they are unable to identify those specific entity types in the 25 sentences I gave. This lack of specific labeling could be reducing the accuracy and overall F1 score for both models on this particular dataset.

Manual Annotations and Tokenization:

The 25 sentences were annotated by a me as Person, so there may be some manual mistakes in the annotations, which could also affect the performance comparison between the two models.

The way the sentences are tokenized and how NER entities are identified can vary between the models and human annotators, which can also impact the results. For example, models may not consider certain punctuation marks, sandhi, and vibhaktis as separate NER entities, whereas a person might tag them differently.

In conclusion, the differences in performance between Indic-BERT and Indic-NER could be due to various factors, including the nature of the models, the dataset, and the annotation process. It's important to consider these factors when interpreting the results and to take into account the specific requirements of your NER task when choosing a model**.**

First photo overall macro F1 Score of Indic-BERT and second is of overall F1 Score of Indic-NER on 25 given sentences

```
Sum of Overall Macro F1 Scores: 0.30711048211048214
```

```
Sum of Overall Macro F1 Scores: 0.41959208882285803
```

Conclusion:-

In terms of performance IndicNER is better than Indic-bert for both Naamapadam and also for 25 given sentences

Comparison Between Manual Answer and ChatGPT Answer

ChatGPT does not have built-in support for named entity recognition (NER) in any language, including Gujarati. NER requires specialized models and training data, which are not part of the standard capabilities of language models like ChatGPT.

Model Capability: ChatGPT is trained primarily for language generation tasks, such as conversation, text completion, and question answering. It does not have specific modules or training for NER, so it lacks the specialized knowledge and parameters required for accurate entity recognition.

Context Understanding: While ChatGPT can understand and generate text based on context, it may not be able to identify named entities accurately without specific training. NER models are trained on annotated data to recognize entities based on context, which is different from the language generation task of ChatGPT.

Fine-tuning: To adapt ChatGPT for NER, you would need to fine-tune it on a NER dataset, which requires access to annotated data and training resources. Without this fine-tuning, ChatGPT would not have the necessary knowledge or patterns to recognize named entities.

If you **explicitly provide instructions to ChatGPT** to perform Named Entity Recognition (NER), it would **attempt to follow those instructions** based on its existing understanding of language and context. However, there are **limitations** to this approach:

Lack of NER-specific Training: ChatGPT does not have specific training for NER, so its ability to accurately recognize named entities would be limited. It may not perform as well as specialized NER models trained on annotated data.

Contextual Understanding: While ChatGPT can understand and generate text based on context, its ability to accurately identify named entities may be suboptimal compared to models specifically designed for NER.

Ambiguity and Errors: Without NER-specific training, ChatGPT may misinterpret or misidentify entities, leading to errors or ambiguity in the output.

Complexity and Specificity: NER tasks can be complex and require a deep understanding of language and context, as well as knowledge about specific entities and their relationships. ChatGPT's general-purpose training may not be sufficient for these tasks.

In summary, while you can provide instructions to ChatGPT to perform NER, its performance may be limited compared to using a specialized NER model , That we can see from results that we are getting from Q-4.

Improve Performance Of FineTuning on Indic-Bert & IndicNER

The **hyperparameters** I have tuned and their significance in my code are as follows:

per_device_train_batch_size and per_device_eval_batch_size: These parameters control the batch size for training and evaluation, respectively, per device (e.g., per GPU). A larger batch size can lead to faster training but may require more memory.

A larger batch size can speed up training by processing more examples simultaneously, leveraging parallel computation. However, this also increases memory requirements, as each example and its associated computations must be stored. Finding the right balance between batch size and memory usage is crucial for efficient training.

num_train_epochs: This parameter specifies the number of times the model will iterate over the entire training datasets. Increasing the number of epochs can improve model performance, but too many epochs can lead to over fitting.

evaluation_strategy: This parameter determines when evaluation is performed during training. Setting it to "epoch" means that evaluation will be performed after each training epoch. This allows you to monitor the model's performance and potentially early stop training if the performance does not improve.

learning_rate: The learning rate parameter determines the step size taken during the training process. A higher learning rate can result in faster convergence, as the model adjusts more quickly to the data. However, if the learning rate is too high, the model may overshoot optimal solutions and perform poorly. On the other hand, a low learning rate can slow down the convergence process but may lead to a more stable and accurate model in the end.

The optimal values I have chosen are:

per_device_train_batch_size and per_device_eval_batch_size: 16, which is a common batch size used in training deep learning models for starting.

evaluation_strategy: "epoch", which means evaluation will be performed after each training epoch.

num_train_epochs: 3, which means the model will iterate over the entire training datasets three times , for more number of epoch the time taking is also very high

learning_rate: When fine-tuning pretrained models like BERT, a commonly used **learning rate is 2e-5**. Due to system constraints, Therefore, I am using the same learning rate for my experiments starting.

Precision , Recall , F1 Score of fine tune on Indic-Bert

```
batch_size=16
args=TrainingArguments(
    output_dir='output_dir',
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    evaluation_strategy = "epoch",
    learning_rate=2e-5)
```

Epoch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	0.427800	0.366698	0.696838	0.528974	0.601412	1208	0.585526	0.419811	0.489011	1060	0.699793	0.627864	0.661880	1615	0.671145	0.540304	0.598659	0.889683
2	0.327900	0.319343	0.653044	0.648179	0.650602	1208	0.542135	0.546226	0.544173	1060	0.723871	0.694737	0.709005	1615	0.650773	0.639712	0.645195	0.900889
3	0.302600	0.310825	0.689346	0.637417	0.662366	1208	0.571719	0.530189	0.550171	1060	0.730893	0.704644	0.717528	1615	0.675417	0.636106	0.655172	0.905397

```
batch_size=8
args=TrainingArguments(
    output_dir='output_dir',
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    evaluation_strategy = "epoch",
    learning_rate=2e-5)
```

Epoch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	0.516500	0.360590	0.691924	0.574503	0.627770	1208	0.425656	0.550943	0.480263	1060	0.685473	0.654489	0.669623	1615	0.596119	0.601339	0.598718	0.887873

```
batch_size=8
args=TrainingArguments(
    output_dir='output_dir',
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    evaluation_strategy = "epoch",
    learning_rate=5e-6)
```

Epoch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	0.282200	0.317143	0.700535	0.650662	0.674678	1208	0.543961	0.531132	0.537470	1060	0.709385	0.678638	0.693671	1615	0.660454	0.629668	0.644693	0.901873
2	0.276800	0.308957	0.710811	0.653146	0.680759	1208	0.566077	0.513208	0.538347	1060	0.712267	0.686687	0.699243	1615	0.673098	0.628895	0.650246	0.902889
3	0.262800	0.308113	0.699115	0.653974	0.675791	1208	0.556680	0.518868	0.537109	1060	0.720598	0.686687	0.703234	1615	0.669675	0.630698	0.649602	0.903492

```
batch_size=12
args=TrainingArguments(
    output_dir='output_dir',
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    evaluation_strategy = "epoch",
    learning_rate=5e-6)
```

Epoch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	0.748900	0.532607	0.501174	0.353477	0.414563	1208	0.361963	0.111321	0.170274	1060	0.509128	0.466254	0.486749	1615	0.488521	0.334278	0.396942	0.846889

```
batch_size=4
args=TrainingArguments(
    output_dir='output_dir',
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    evaluation_strategy = "epoch",
    learning_rate=5e-6)
```

Epoch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	0.408500	0.404823	0.614639	0.472682	0.534394	1208	0.445370	0.453774	0.449533	1060	0.648704	0.588854	0.617332	1615	0.576403	0.515838	0.544441	0.875873
2	0.373600	0.378933	0.667770	0.500828	0.572375	1208	0.494802	0.449057	0.470821	1060	0.687055	0.598142	0.639523	1615	0.625229	0.527170	0.572027	0.883302
3	0.353100	0.371481	0.630455	0.562086	0.594311	1208	0.530172	0.464151	0.494970	1060	0.670895	0.622291	0.645679	1615	0.621182	0.560391	0.589223	0.886984

Observations:

Learning Rate Impact:

A higher learning rate (2e-5) generally led to better performance when paired with a larger batch size (16), achieving an accuracy of 0.90.
However, a lower learning rate (5e-6) performed equally well when paired with a smaller batch size (8), also achieving an accuracy of 0.90.

Batch Size Impact:

Larger batch sizes (16) tended to perform better with a higher learning rate (2e-5), achieving the highest accuracy of 0.90.
Smaller batch sizes (8 and 4) performed better with a lower learning rate (5e-6), achieving accuracies of 0.88 and 0.88 respectively.

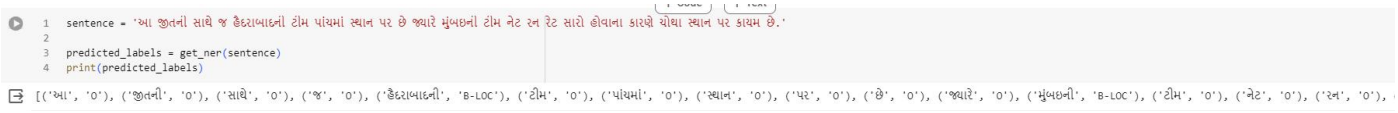
Trade-offs:

There seems to be a trade-off between batch size and learning rate. Higher learning rates may require larger batch sizes for optimal performance, while lower learning rates may perform better with smaller batch sizes. Smaller batch sizes can sometimes lead to better performance, but they may require more training iterations, which can increase training time.

Conclusion:-

Based on the results shown in the screenshot, the best accuracy was achieved using a **batch size of 8**, a **learning rate of 5e-6**, and a total of **3 training epochs**. This combination resulted in the highest accuracy compared to other combinations of batch sizes, learning rates, and training epochs tested.

Output of Indic-Bert Model:-



Precision , Recall , F1 Score of fine tune on IndicNER

```
batch_size=16
args=TrainingArguments(
    output_dir='output_dir',
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    evaluation_strategy = "epoch",
    learning_rate=2e-5)
```

Epoch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	0.496000	0.190977	0.789976	0.822020	0.805680	1208	0.718021	0.725472	0.721727	1060	0.809322	0.827864	0.818488	1615	0.778643	0.798094	0.788249	0.941492
2	0.168400	0.190141	0.796016	0.826987	0.811206	1208	0.705349	0.733962	0.719371	1060	0.804790	0.832198	0.818265	1615	0.774826	0.803760	0.789028	0.941460
3	0.148600	0.197174	0.795563	0.831126	0.812955	1208	0.716543	0.727358	0.721910	1060	0.806743	0.829721	0.818071	1615	0.778945	0.802215	0.790409	0.941905

```
batch_size=8
args=TrainingArguments(
    output_dir='output_dir',
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    evaluation_strategy = "epoch",
    learning_rate=5e-6)
```

Epoch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	0.232700	0.208042	0.757364	0.808775	0.782226	1208	0.696179	0.704717	0.700422	1060	0.782634	0.820433	0.801088	1615	0.751726	0.785218	0.768107	0.938508
2	0.187700	0.197407	0.789641	0.820364	0.804710	1208	0.692722	0.727358	0.709618	1060	0.793802	0.824768	0.808989	1615	0.764706	0.796807	0.780426	0.940381
3	0.173000	0.196559	0.789849	0.824503	0.806804	1208	0.705775	0.726415	0.715946	1060	0.797370	0.826006	0.811436	1615	0.770186	0.798352	0.784016	0.941365

```
batch_size=8
args=TrainingArguments(
    output_dir='output_dir',
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    evaluation_strategy = "epoch",
    learning_rate=5e-4)
```

Epoch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	0.150700	0.200805	0.782437	0.818709	0.800162	1208	0.700549	0.721698	0.710967	1060	0.796429	0.828483	0.812140	1615	0.766105	0.796292	0.780907	0.939746
2	0.145500	0.201643	0.797265	0.820364	0.808650	1208	0.681338	0.730189	0.704918	1060	0.790725	0.823529	0.806794	1615	0.762128	0.797064	0.779204	0.939619
3	0.140600	0.202021	0.792363	0.824503	0.808114	1208	0.695652	0.724528	0.709797	1060	0.797726	0.825387	0.811321	1615	0.768105	0.797579	0.782565	0.940413

```
batch_size=16
args=TrainingArguments(
    output_dir='output_dir',
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=4,
    evaluation_strategy = "epoch",
    learning_rate=5e-7)
```

Epoch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	6.738900	2.881739	0.000172	0.000828	0.000285	1208	0.020333	0.010377	0.013741	1060	0.137566	0.193189	0.160700	1615	0.037578	0.083441	0.051819	0.051778
2	1.801200	0.748460	0.000000	0.000000	0.000000	1208	0.000000	0.000000	0.000000	1060	0.247294	0.183901	0.210938	1615	0.172674	0.076487	0.106015	0.798603
3	0.759500	0.599956	0.000000	0.000000	0.000000	1208	0.028807	0.006604	0.010744	1060	0.339776	0.413622	0.373080	1615	0.255006	0.173835	0.206738	0.812762
4	0.569500	0.564811	0.000000	0.000000	0.000000	1208	0.033210	0.008491	0.013524	1060	0.357075	0.468731	0.405355	1615	0.274650	0.197270	0.229616	0.816762

Observations:

Batch Size Impact:

A batch size of 16 with a learning rate of 2e-5 achieved the highest accuracy of 0.94.
Decreasing the batch size to 8 with a lower learning rate of 5e-6 also achieved an accuracy of 0.94, showing that a smaller batch size can be as effective as a larger one with the right learning rate.

Learning Rate Impact:

A lower learning rate of 5e-6 with a batch size of 8 achieved the same high accuracy of 0.94, indicating that a lower learning rate can compensate for a smaller batch size.
Increasing the learning rate to 5e-4 with the same batch size of 8 resulted in a slightly lower accuracy of 0.93, suggesting that too high a learning rate can negatively impact performance.

Trade-offs:

When using a larger batch size, it's often necessary to decrease the learning rate to prevent the model from diverging or oscillating. This is because larger batch sizes provide more accurate estimates of the gradient, which means that smaller updates are needed.
Conversely, when using a smaller batch size, a higher learning rate may be more effective, as the noise in the gradient estimates can benefit from more frequent updates to help the model converge.

Conclusion:-

Based on the results shown in the screenshot, the best accuracy was achieved using a **batch size of 16**, a **learning rate of 2e-5**, and a total of **3 training epochs**. This combination resulted in the highest accuracy compared to other combinations of batch sizes, learning rates, and training epochs tested.

Output of IndicNER Model:-

```
1  sentence = 'આ જીતનાં સાથે જ હૈદરાબાદનો ટીમ પાંચમાં સ્થાન પર છે જ્યારે મુંબઇનો ટીમ નેટ રન રેટ સારો હોવાના કારણે ચોથા સ્થાન પર કાયમ છે.'
2
3  predicted_labels = get_ner(sentence)
4  print(predicted_labels)
```

```
[('આ', 'O'), ('જીતનાં', 'O'), ('સાથે', 'O'), ('જ', 'O'), ('હૈદરાબાદનો', 'B-LOC'), ('ટીમ', 'O'), ('પાંચમાં', 'O'), ('સ્થાન', 'O'), ('પર', 'O'), ('છે', 'O'), ('જ્યારે', 'O'), ('મુંબઇનો', 'B-LOC'), ('ટીમ', 'O'),
```