Project Title: Named Entity Recognition on Medical Data using Pre-Trained LLM

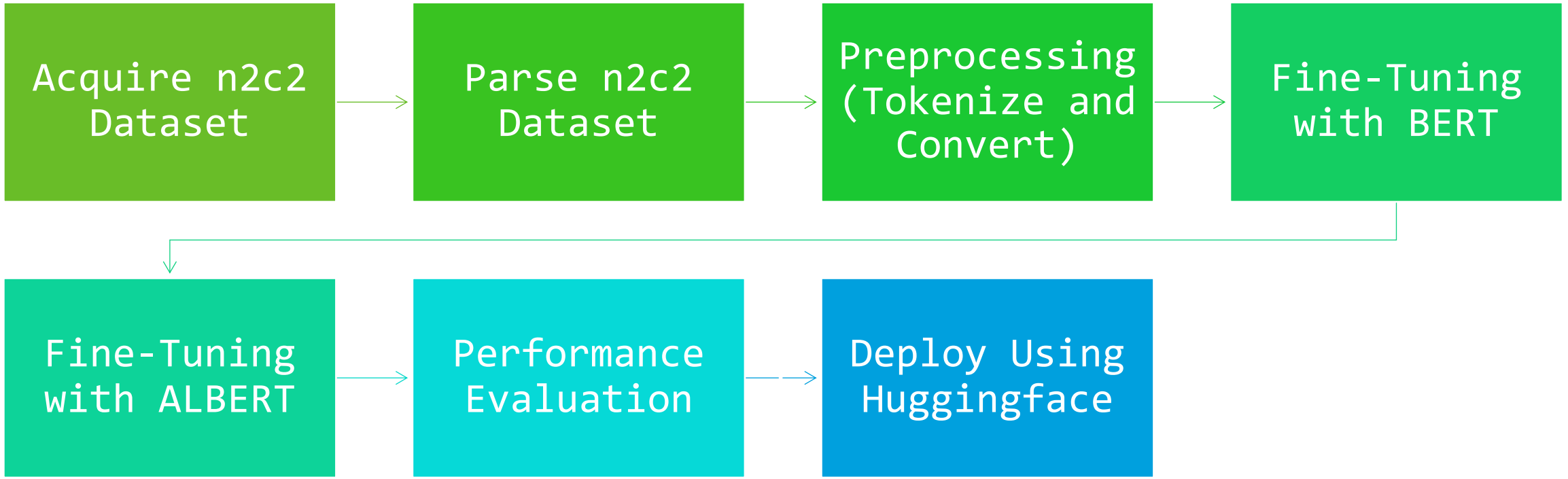
PROJECT IDEA BRIEF

- Named Entity Recognition (NER) is the task of identifying and extracting named entities from text.
- Named entities in the medical domain include diseases, symptoms, treatments, and medications.
- Medical data can come in various forms, such as clinical notes, electronic health records, and medical literature.
- Clinical notes are particularly useful for NER, as they contain unstructured text that is rich in medical terminology and often reflects the patient's medical history, symptoms, and treatment.
- The goal is to accurately identify and extract the relevant named entities from the text, in order to facilitate downstream tasks such as information retrieval and clinical decision support.

DATA

- Data: We have obtained educational usage permission to use i2b2/n2c2 dataset.
- The dataset contains clinical notes from patients with various medical conditions and has annotations for named entities such as diseases, symptoms, treatments, and medications, which will allow for training and evaluating NER models.
- The dataset consists of 1,000 discharge summaries, 100 test notes, and 400 training notes.
- The dataset has been preprocessed to remove identifying information such as patient names and addresses.
- We are focused on the following tags: **Person, Problem, Pronouns, Test, and Treatment**
- **The split of data is 70 percent for Training (Train-85% + Validation-15%) and 30% Testing.**

PROJECT FLOWCHART



PROJECT TASK METHODOLOGY

S. NO	TASK DESCRIPTION
1	Acquire the i2b2 dataset
2	Parsing the i2b2 dataset
	a. Convert i2b2 format (.con & .txt) to a more suitable format for NER tasks
	b. Split the dataset into training, validation, and test sets
3	Preprocess the data for input into the language model
	a. Tokenize the text
	b. Convert the text into a format compatible with the chosen large-scale language models (BERT & ALBERT)
4	Fine-tune the chosen language models on the i2b2 dataset on BERT and ALBERT models
5	Evaluate the performance of the fine-tuned models on the test set
	a. Measure precision, recall, accuracy, and F1-score
	b. Compare performance with benchmark models such as BIOELECTRA.
	c. Perform error analysis to identify common types of mistakes
6	Deploy the fine-tuned NER model using Huggingface
7	Conclusion, Improvements & Future Implementation

IMPLEMENTATION-> DATA PARSING

- Retrieves and processes clinical data from concept and text files, creating annotation and entry corpora.
- Converts corpora into dataframes, merges them, and fills missing values.
- Computes POS tags, creates NP and VP chunks, and combines chunk tags for all tokens in the dataset.
- Inserts blank rows at sentence boundaries and splits the dataset into training, validation, and test sets.
- Saves the resulting sets as .txt and .csv files in the output directory.

```
columnsOfEntries = ["id", "row", "offset", "word"]
dataframesOfEntries = pand.DataFrame(columns=columnsOfEntries)

columnsForAnnotations = ["id", "NER_tag", "row", "offset", "length"]
dataframesOfAnnotations = pand.DataFrame(columns=columnsForAnnotations)
```

```
# Chunk noun phrases

ruleInit = ChunkRule("<DT>?<JJ.*>*<NN.*>+", "More complete chunk NP sequences")
parserChunkNoun = RegexpChunkParser([ruleInit],chunk_label='NP')
resultTreeNoun = parserChunkNoun.parse(positionOfText)

chunkTagNoun = []

for i in resultTreeNoun:
    if isinstance(i, Tree):
        for j in range(0, len(i)):
            if j == 0:
                # print("B-" + i.label())
                chunkTagNoun.append("B-" + i.label())
            else:
                chunkTagNoun.append("I-" + i.label())
                # print("I-" + i.label())
    else:
        # print("O")
        chunkTagNoun.append("O")

len(chunkTagNoun) == dataframeRES.shape[0]  # check that chunk col has same length
```

'BERT' FOR NER

- **Pre-trained language model:** BERT, or Bidirectional Encoder Representations from Transformers, is a pre-trained language model that provides deep bidirectional contextualized representations, making it highly effective for various natural language processing tasks, including Named Entity Recognition (NER) in the medical domain.
- **Transfer learning:** BERT's transfer learning capability allows leveraging its pre-trained knowledge to fine-tune the model on specific medical datasets, significantly improving the accuracy of NER tasks in the medical field, such as identifying diseases, medications, and symptoms.
- **Contextual understanding:** BERT's bidirectional context-awareness helps in accurately identifying and disambiguating medical entities in text, which is crucial for tasks like information extraction, question answering, and patient data analysis.

'ALBERT' FOR NER

- **Efficient architecture:** ALBERT (A Lite BERT) is an optimized version of BERT that employs parameter sharing and factorized embedding to reduce the model size and training time, making it more efficient for NER tasks in medical data.
- **Comparable performance:** Despite its smaller size, ALBERT still provides comparable or even better performance compared to BERT, making it suitable for NER tasks in the medical domain, where model efficiency and speed are important.
- **Scalability:** The lighter architecture of ALBERT allows for easier scaling of the model size, enabling the development of large-scale models capable of handling complex medical NER tasks without significant computational overhead.
- **Domain adaptation:** Like BERT, ALBERT can also be fine-tuned on specific medical datasets or adapted into domain-specific models for improved NER performance in the medical field, leveraging its efficient architecture to provide accurate entity recognition.

IMPLEMENTATION ->PRE-PROCESSING ->CONVERSION

```
#reading the train, valid, test dataset
#converting it into csv
data_dir = '/content/drive/MyDrive/NLP_project/data/output/'

def createFeature(data):

  # Split the text into a list at the empty lines
  sentences = re.split('\n\s*\n', data)

  #getting word: from index 0 and tag: from index 3
  words, tags = [], []
  for sent in sentences:
      #getting list of lines
      lines = sent.split("\n")
      w = []
      t = []
      for line in lines:
        #if the line not empty
        if line.split():
          #print(triData.split())
          w.append(line.split()[0])
          t.append(line.split()[3])
      words.append(w)
      tags.append(t)
```

```
  return {'word':words, 'tag':tags}

def dataReader(dataPath):
  train_dict = createFeature(open(dataPath+'train.txt').read().strip())
  test_dict = createFeature(open(dataPath+'test.txt').read().strip())
  valid_dict = createFeature(open(dataPath+'valid.txt').read().strip())

  #creating dataframe
  train_df = pd.DataFrame(train_dict)
  test_df = pd.DataFrame(test_dict)
  valid_df = pd.DataFrame(valid_dict)

  #print(len(valid_dict['word']))
  return train_df, test_df, valid_df

train_df, test_df, valid_df = dataReader(data_dir)

#saving the data to csv
train_df.to_csv(data_dir + "train_conll.csv")
test_df.to_csv(data_dir + "test_conll.csv")
valid_df.to_csv(data_dir + "valid_conll.csv")
```

Reading *conll format txt and converting to word-tag column CSV

IMPLEMENTATION ->PRE-PROCESSING ->TOKENIZING

```
def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(examples["word"], truncation=True, is_split_into_words=True, padding="max_length")
    labels = []
    for i, label in enumerate(examples["tag"]):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        prev_word_idx = None
        label_ids = []
        for word_idx in word_ids:
            if word_idx is None:
                label_ids.append(-100)
            elif word_idx != prev_word_idx:
                label_ids.append(label2id[label[word_idx]])
            else:
                label_ids.append(-100)
            prev_word_idx = word_idx
        labels.append(label_ids)
    tokenized_inputs["labels"] = labels
    return tokenized_inputs
```

Function to Tokenize and Align the labels

IMPLEMENTATION ->PERFORMANCE METRICS

```python
def compute_metrics(p):
    predictions, labels = p
    predictions = np.argmax(predictions, axis=2)

    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    results = metric.compute(predictions=true_predictions, references=true_labels)
    overall_result =  {
        "overall_precision": results["overall_precision"],
        "overall_recall": results["overall_recall"],
        "overall_f1": results["overall_f1"],
        "overall_accuracy": results["overall_accuracy"]
    }

    #label-wise f1 score
    for key in results.keys():
      if key not in overall_result.keys():
        overall_result[key+"_f1"] = results[key]["f1"]

    return overall_result
```

Implementation of Performance Metrics: Precision, Recall, Accuracy & F1

EXPERIMENTS & RESULTS ->BERT FINE-TUNING - >TRAIN/VALIDATION METRICS

Epoch	Training Loss	Validation Loss	Overall Precision	Overall Recall	Overall F1	Overall Accuracy	Person F1	Problem F1	Pronoun F1	Test F1	Treatment F1
1	0.368000	0.213285	0.793032	0.828129	0.810201	0.936638	0.870690	0.757895	0.962766	0.771976	0.798906
2	0.150200	0.209606	0.823885	0.852294	0.837849	0.943098	0.890398	0.792636	0.954424	0.814315	0.822208
3	0.108400	0.214214	0.822263	0.850394	0.836092	0.945893	0.882865	0.787819	0.962963	0.816353	0.825289
4	0.044800	0.246469	0.840572	0.846049	0.843302	0.948504	0.888780	0.795888	0.957447	0.828323	0.831821
5	0.033200	0.271435	0.845966	0.848493	0.847228	0.948413	0.884407	0.807139	0.958115	0.831430	0.837912
6	0.021800	0.284084	0.845614	0.850665	0.848132	0.949054	0.886709	0.804144	0.952632	0.833557	0.842975

Training & Validation set metrics: Precision, Accuracy, F1 :: 6 Epochs

EXPERIMENTS & RESULTS ->BERT FINE-TUNING ->TEST METRICS

```
Test results:
eval_loss: 0.2851
eval_overall_precision: 0.8710
eval_overall_recall: 0.8742
eval_overall_f1: 0.8726
eval_overall_accuracy: 0.9557
eval_person_f1: 0.9056
eval_problem_f1: 0.8755
eval_pronoun_f1: 0.9655
eval_test_f1: 0.8688
eval_treatment_f1: 0.8217
eval_runtime: 10.1726
eval_samples_per_second: 104.8900
eval_steps_per_second: 13.1730
epoch: 6.0000
```

Test Set metrics: Precision, Accuracy, F1 :: 6 Epochs

EXPERIMENTS & RESULTS ->ALBERT FINE-TUNING - >TRAIN/VALIDATION METRICS

Epoch	Training Loss	Validation Loss	Overall Precision	Overall Recall	Overall F1	Overall Accuracy	Person F1	Problem F1	Pronoun F1	Test F1	Treatment F1
1	0.374600	0.234656	0.800605	0.794921	0.797753	0.931789	0.869691	0.730175	0.932249	0.782123	0.769014
2	0.203200	0.231905	0.812020	0.837520	0.824573	0.935972	0.883789	0.757660	0.960630	0.826305	0.803020
3	0.154700	0.231322	0.821649	0.835336	0.828436	0.938178	0.894587	0.778596	0.958115	0.803793	0.792478
4	0.080400	0.237391	0.842291	0.847351	0.844814	0.944613	0.896117	0.813354	0.963158	0.823446	0.806345
5	0.059400	0.261234	0.839628	0.837794	0.838710	0.944843	0.891732	0.795421	0.957447	0.830162	0.802703
6	0.040200	0.278624	0.844256	0.840797	0.842523	0.946084	0.891389	0.800799	0.954907	0.831351	0.813187

Training & Validation set metrics: Precision, Accuracy, F1 :: 6 Epochs

EXPERIMENTS & RESULTS ->ALBERT FINE-TUNING ->TEST METRICS

```
Test results:
eval_loss: 0.2891
eval_overall_precision: 0.8708
eval_overall_recall: 0.8627
eval_overall_f1: 0.8667
eval_overall_accuracy: 0.9518
eval_person_f1: 0.9106
eval_problem_f1: 0.8611
eval_pronoun_f1: 0.9691
eval_test_f1: 0.8644
eval_treatment_f1: 0.8123
eval_runtime: 14.7946
eval_samples_per_second: 72.1210
eval_steps_per_second: 9.0570
epoch: 6.0000
```

Test set metrics: Precision, Accuracy, F1 :: 6 Epochs

GENERAL OBSERVATIONS & ANALYSIS

- **General Performance**: State-of-the-art transformer models, BERT and ALBERT, are selected and fine-tuning on medical data is done, both models appear to be viable options for medical NER.
- **Overfitting:** From the provided training and validation loss values for both BERT and ALBERT, it appears that overfitting is not a significant issue. In both models, the training loss decreases consistently over epochs, while the validation loss initially decreases before stabilizing or increasing slightly.

GENERAL OBSERVATIONS & ANALYSIS

- **Pronoun Performance:** The 'pronoun' tag exhibits better performance compared to other tags, as indicated by higher F1 scores. This can be attributed to several factors:
  - **Frequency**: Pronouns are generally more common and less diverse than other entity types, such as medical problems or treatments. This makes it easier for the model to recognize and predict them accurately.
  - **Context**: Pronouns have a more consistent usage and syntactic structure in sentences. This consistency helps the model to better learn their context and recognize them with higher precision and recall.
  - **Pre-trained knowledge**: Since pronouns are common in general language, the pre-trained BERT and ALBERT models likely have a strong foundational understanding of pronouns, which can be fine-tuned effectively for the medical domain.

COMPARITIVE ANALYSIS – BERT VS. ALBERT

- **Overall F1**: BERT's overall F1 score on the testing dataset is 0.8726, while ALBERT's is 0.8667. Both models have comparable performance, with BERT having a slightly higher F1 score.
- **Overall Accuracy**: BERT has a higher overall accuracy (0.9557) on the testing dataset compared to ALBERT (0.9518).
- **Entity-wise F1 scores:** Comparing the F1 scores for different entity types, BERT performs better on 'Problem' and 'Test' entity types, while ALBERT performs better on 'Person', 'Pronoun', and 'Treatment' entity types.
- Both BERT and ALBERT models show competitive performance on the medical NER task, with BERT having a slightly better overall F1 score and accuracy on the testing dataset. The choice between the two models depends on the specific requirements of the task and which entity types are more important for the application.

FUTURE DEVELOPMENTS

- Test the trained models on other datasets (e.g., MIMIC-III and ShARe/CLEF eHealth datasets) to evaluate performance on unseen data
- Create parser models to convert other data formats to the i2b2/n2c2 model format for easy model training with other datasets
- Deploy the fine-tuned NER model for real-time usage
- Perform experiments and analyze results to validate the hypothesis and gain insights into the model's performance on different datasets
- Develop a website that allows users to:
    - Upload a text file
    - Extract Named Entities from the uploaded text file using the optimized model

REFERENCES

- BioELECTRA: Pretrained Biomedical text Encoder using Discriminators
- A clinical trials corpus annotated with UMLS entities to enhance the access to Evidence-Based Medicine Leonardo Campillos-Llanos, Ana Valverde-Mateos, Adrián Capllonch-Carrión, Antonio Moreno-Sandoval - BMC Medical Informatics and Decision Making 21, 69 (2021)
- BioBERT: a pre-trained biomedical language representation model for biomedical text mining
- SciBERT: A Pretrained Language Model for Scientific Text

THE UNIVERSITY OF TEXAS AT DALLAS

QUESTIONS?