LLMs for Patent Analysis

Task definition

The task consists in using an LLM to extract information on measurements and their values from patents. There are three approaches that can be used to achieve this, ordered by complexity:

- Option 1: Use prompt-engineering on a general pre-trained LLM model.
- Option 2: Fine-tune a general pre-trained LLM model on a custom dataset that includes examples of measurements and their corresponding values.
- Option 3: Train an LLM model from scratch on a custom, high-quality dataset that includes data scraped from the Internet, prioritizing examples of measurements and their values.

In the *design phase* of the challenge, we will focus on Option 2, as the scope is more appropriate for a Proof of Concept.

Prepare the training data

Build a labeled dataset that includes examples of measurements and their corresponding values.

1. Define the scope: Focus on a collection of measurements of interest. Examples may include temperature, weight, length, or volume.
2. Collect text data: The data should include examples of the measurements we want to extract. Examples may include patent texts, scientific articles, or technical documents.
3. Annotate the data: Manually annotate the data, identifying the measurement entities in the text and annotating the start and end positions of these entities.
    - We could use BIO encoding to label the tokens at the beginning (B), inside (I) and outside (O) of a measurement entity. Example: The seven tokens in “The temperature is 25 degrees Celsius.” would be labeled as “OBOBIIO”.
    - We need to handle the variation of formats in measurements. For example, "25 degrees Celsius," and "25°C," should be considered as the same measurement entity. This may be used to augment data and improve model generalization.
4. Tokenize and format the data: Tokenization breaks down the input text into a sequence of tokens, where each token corresponds to a word.
    - Tokenize the input text using the same tokenizer used by the pretrained LLM model that we will use for fine-tuning (more on next section).
    - As input texts are long, each of them needs to be splitted into smaller segments (e.g., paragraphs or sentences) that fit within the model's input token size. These segments will be processed separately during fine-tuning and inference.
    - We have to keep track of the mapping between the original text and the tokenized text to ensure proper alignment of annotations with tokenized sequences.
5. Correct class imbalance: This is probably required to make sure the dataset contains a similar number of positive (measurement entities) and negative (non-measurement entities) examples. This can be achieved by combining random undersampling and a collection of augmentations (for example, using variations of measurement units to expose the model to different forms of measurement representations).
6. Split the data and prepare for fine-tuning: Divide the annotated dataset into training, validation, and test sets. Furthermore, we convert tokens to input IDs, create attention masks, and convert the annotated labels of the BIO encoding into numerical format.

7. Create data batches: Group the input IDs, attention masks, and numerical labels into batches that can be fed into the model during training. Batch sizes are determined based on the available memory and computational resources.

Fine-tune a pretrained model

Load a pretrained model and update its weights on our specific task using the tokenized and formatted data we prepared in the previous section.

1. Select a pretrained model: The selected model should be appropriate for the task. In this case, we could use GPT-3.5 LLaMa, Claude or Falcon. Factors such as the tokenizer or the model size must be taken into account depending on the technical requirements.
2. Prepare the fine-tuning setup: Define parameters such as the learning rate, number of epochs, optimizer (e.g., Adam), and loss function (e.g., cross-entropy). In addition, set up any other fine-tuning details, such as whether to freeze certain layers of the model.
3. Fine-tune the model: The model is trained (fine-tuned) to predict a label for each token in the input sequence, indicating whether it is part of a measurement entity or not (B, I, O).
4. Model evaluation: After each epoch, we evaluate the model's performance on the validation set using metrics such as precision, recall, and F1-score to assess how well the model is extracting measurements and their values.
5. Save the fine-tuned model.

Inference with the fine-tuned model

Use the fine-tuned model to run inference on the test set.

1. Inference on the test set to extract measurements and their values.
2. Post-processing: The LLM would return a sequence of tokens along with their corresponding predicted labels, so post-processing is needed to extract relevant measurement information accurately from the model's output.
    - This process could require identifying contiguous tokens with the desired BIO labels, grouping tokens together to form complete measurement entities, extracting the measurement values from the grouped tokens, and returning the output in the desired format (e.g. JSON). Additionally, domain-specific rules or heuristics may be applied to handle edge-cases and to filter out irrelevant or incorrect predictions.
    - The effectiveness of the post-processing rules should be evaluated on a separate validation set before applying them to the test set. This can be helpful to fine-tune the rules and ensure they capture the relevant measurement information accurately.
3. Error analysis and iteration: We may need to iterate and fine-tune the model based on feedback and additional labeled data to improve accuracy.