**READ THIS DOCUMENT BEFORE RUNNING ANY SCRIPTS**

**Note: Due to GPU constraints as well as VRAM constraints, the complete data was not used. Instead, a subset of 30% of the training and testing data was used and Google Colab was used for training the Inference Model.**

# Intelligent Document Extraction

The entire pipeline can be divided into 3 distinct phases. In order to reproduce results, it is recommended to run the scripts—provided in each phase—in the given order.

## Phase 1: Preprocessing

The data was not used in its given raw format. The file Preprocessing.py prepares the data to be used further. First, it loads the data from all 3 given folders (images, bounding boxes and ocr text, labels). It used fuzzy matching to align text lines with correct labels, handles overlapping matches and maintains confidence scores for traceability. Third, it prevents duplicate labeling for similar entities in the same document. It then splits ocr text lines into words, adjusting bounding boxes accordingly to create token level data required by layout aware models like the one used here. It then combines text, bounding boxes, labels and image metadata into a unified dataset (30% of the total), normalizes coordinates to a 1000x1000 grid and writes 3 files: {split}.txt (text + label), {split}_box.txt (text + bounding boxes), {split}_img.txt (text + bounding boxes + image metadata(filename))

Files used: Preprocessing.py (modules), process_data.py (scripts), data (dataset folder)

Output: preprocessed_data (folder)

## Phase 2: Training LayoutLMv2 Model for Text Extraction

The data preprocessed earlier is loaded and compiled into a PyTorch Dataset. The

Dataset groups data by document, with each document containing a list of words, corresponding labels and bounding boxes. It organizes all elements (words, labels, bboxes and images) into lists, one per document. A layout aware processor (LayoutLMv2Processor: Microsoft/layoutlmv2-base-uncased) to tokenize text, process images and normalize bounding boxes, and uses label2id mapping to convert labels into numerical form. Also ensures uniform sequence length using padding and truncation.

Due to class im-balancing, enhanced due to earlier preprocessing in phase 1, a weighted trainer is used—extending from the Trainer class from transformers—which applies weighted cross entropy and assigns higher importance to underrepresented classes. A pre-trained model (LayoutLMv2ForTokenClassifications) is imported and fine-tuned on the training dataset using the Trainer Class and the model is saved for inference.

File used: TrainInferenceModel.ipynb (colab scripts)

Output: model folder (modules)

## Phase 3: Final Integration

Inference is performed using samples from the test dataset. A wrapper class is used to load the saved model and initialize it for inference. The inference method takes a sample from the dataset and runs model inference to obtain token-level predictions, which are then mapped back into text and used to reconstruct entity fields, which are returned. A simple function is created for ocr text extraction from images using easyocr. The sample for the inference model and for ocr text extraction share the same index, and the ocr text extractor is mainly used for potential future samples whose ocr text are not given. Both the output from the inference model and ocr text extractor are passed to the LLM wrapper. The llm wrapper makes use of Geminis Api key and the "gemeni-flash-2.0" model for reasoning. The prompt checks the json output received from the inference model for missing fields or inconsistent information, and uses the ocr text as context to fill in or revise the contents of the json, as well as includes comments on observations, changes made and conclusions drawn.

Bonus: A Watermark detector class is added that provides a simple method to determine whether an input image is visually obscured. The basis of the method is the example provided in the assessment file (dark watermark obscuring company name). It converts images to greyscale and computes the ratio of dark pixels across the image. It also performs analysis using opencv to detect large contiguous dark regions (possible watermarks/logos). It flags an image as obscured if the dark_ratio_threshold is exceeded or large dark regions are detected.

File used: LLMIntegration.ipynb (colab scripts)

## **Final Notes:**

Other than the files mentioned above, the other files .py files were created since I planned to run the script on local machine instead of colab, using colab only to train the inference model. However, due to issues in downloading detectron2 (my laptop does not have a GPU) I was unable to do so. So the other files are mainly just baselines for my final code. Same reason I did not opt for an Api approach, instead leaving my code as a script.

***THANK YOU***