Waterford Institute of Technology

Automation of Public House Back-end Business Processes

Dimitri Saridakis

Work submitted
within the
BSc in Applied Computing
Cloud & Networks

Supervisor: Dr Kieran Murphy
Second Reader: Clodagh Power

April 20, 2022

Contents

List of Figures

Listings

1 Introduction and Project Goals

1.1 Introduction

This report's structure will follow this style:

- The report will outline the motivation behind the project.
- The project's initial goals and goals achieved.
- The system's initial proposed architecture. Along with the necessary pivots that occurred during development and the justification for those pivots.
- The technologies considered, the technologies chosen and the plan which was followed.
- It will detail the implementation of the varies system components from the start to the working prototype.
- The report will outline the challenges faced in the implementation of the system design.
- A conclusion will be given including both technical and personal reflection.
- Detailed analysis on the system in its current implementation along with documenting some proposed work to in order to bring the system to its full potential for real world, production-ready, use cases.

1.2 Motivation

The motivation for this project comes from a number of major pain points of mine, from my previous / side career as a bar owner and manager.
This project focuses on building a system for use by bar / public houses (pubs) and as such a single bar entity will be referred to as a *user* of the system.
There is also scope for this system to be altered slightly and to be used with any business which operates with a similar back-end structure. In fact, as the system is currently implemented, it is possible to use components of the system, for any business which receives invoices (more on this later).

Some of the most time-consuming and least valuable, from a time - reward perspective, are the back-end processes of running the business. Reward is defined here as the actions which result in potential business growth. Time spent scanning invoices from suppliers, filling out income and expenditure spreadsheets and calculating gross and net figures (which will be referred to as '*group A*' activities). Whilst these processes are critical to a business' operation and regulatory compliance they do not do much for business growth.
On the other hand, time spent on sourcing new products / inventory, finding new / novel forms of entertainment, business promotion and customer engagement (which will be referred to as '*group B*' activities) are the catalysts which drive sales and business growth.
The ultimate aim of this project is to provide more time for group B growth activities and processes by automating the group A processes.

I hypothesize that this should lead to a healthier and more innovative industry by virtue of the extra amount of time spent on group B activities. With implementation of this system, barriers to entry should be broken down which should only help to increase innovation. This comes from the new entrants into the industry who may excel in group B processes but do not have the knowledge, cannot afford to pay accountants or have the confidence in their ability to perform the group A processes at a satisfactory standard. If these processes are automated then there should be fewer barriers of entry coupled with a reduction in accounting costs.
Furthermore, the implementation of this system should increase the quality of life of business owners who no longer have to carry out menial, manual data entry and monotonous, simple and repetitive data manipulation.

For this to be accomplished there is a list of core and essential processes that need to be tackled. These core processes fall into two broad categories, defined as:

1. **Category One**: comes from the data collected from a sale of a user's product, i.e., a pub selling a beverage (Income).
2. **Category Two**: stems from the data collected from a user's purchases in relation to inventory and other purchases needed for the running of the business, i.e., a pub buying a crate of beer to be resold to a consumer or rent for premises etc. (Expenditure).

Category One core processes include:

- Keeping a record of all sale transactions that enter the system, sorted by user, which will allow for the processing of sales.
- The subsequent saving and updating of the transactional sales figures, i.e., gross, net and tax figures.
- The updating of the inventory levels of products per sale.

Category Two core processes include:

- The scanning of supplier invoices and key information extraction from the invoices. This key information will be used to:
  - Update user's inventory levels as stock is invoiced / delivered.
  - Updating of cash flow levels to reflect the current available funds.
  - Updating of tax collected and tax due figures.

This is quite a lengthy and complex list of processes to automate.

There exists further motivation for this project. It is a purely personal reason, which is a massive interest in developing knowledge about the entire process of developing a deep learning system. From the dataset creation phase to the training, evaluation and inference phases. In essence the entire ML stack. Furthermore, this area of Data Science is the major point of focus for work completed in semester two, the second half of this project.
I will use this project as a vehicle to explore and test both industry standard and brand-new technologies with an emphasis on open-source tech. This will stress test my knowledge of the technologies and both allow me to see what I can implement along with being a showcase of my skills in the field to potential employers. This system will be built with the primary target of having each component implemented in such a way as to ensure maximum efficiency is the front and center focus.

1.3 Scope

The scope of this project was highly ambitious. The project focuses on tackling both the Category One and Two processes, with an eye on weeding out every potential pitfall as if this project is the prototype of a real world product.

Note: The Category One processes have been implemented, are functional and are reported on in the first report. These processes are mentioned briefly here, for the reader to get an understanding of the system's goals end-to-end. This report will focus on the development carried out in the second phase of development (semester two) namely the Category Two processes, which have a heavy Data Science focus.[1]

Given the time constraints and the complexity of the system in development there are some non-core components that have been omitted or mocked. There are also some pivots that needed to be made in order to get a functional demo by the deadline. These are documented here to allow the reader to know that these processes have been thought about thoroughly, before the decision was made to continue as detailed.
These include:

1. For the Category One processes, the users' sales transactions and details are faked / generated. This is done through the main entry API which has a `.../transactions/fake/create` endpoint. There is no Point of Sale (PoS) till software created.

   To obtain transactional data into the system from real data a new API server would need to be implemented to fetch data from popular PoS systems. This has not been implemented, although preliminary research reveals that two leading PoS systems APIs, namely SquareUp[97] and Clover PoS [7] both have the necessary APIs available to allow for the capturing of such data and the integration of it into the system is not considered difficult.

2. There is no GUI to interact with or view data from a user standpoint. This would involve creating a web app which would be used to view data from the system about a user. This would include sales figures, inventory levels, cash flow levels and tax figures.

   Upon stating that, it is possible to obtain data insights from the system via SQL calls to the database, including total expenditure, tax expenditure (to be offset against future tax payments) and net goods amounts.
   Furthermore, the implementation of a Grafana (an open-source visualisation tool) dashboard

3. There is one area to note here is that there is a possible discrepancy between newly ordered inventory invoiced and the product actually delivered by a supplier to a business.
   Sometimes products which have been ordered are not delivered, i.e., out of stock with supplier or damaged in transit.

   For a solid fix for this situation the system should utilize a way of comparing delivery dockets with invoices. This would ensure only as accurate data as possible enter the system. However, these situations happen infrequently and for this project the potential discrepancy will be ignored with the invoice taken at face value of goods delivered.

4. The initial design was to include and automate all the Category Two processes in the system. The final implementation of the system does include the inventory, but due to the time-consuming nature of the post-processing of the inventory data returned by the model, the decision was made to concentrate on saving the financial aspects of the invoices (totals, tax, metadata etc.). This decision was made to allow the system to be used for the processing of invoice totals for real world usage. The system has now been used to ready documents for a VAT return, with surprisingly accurate results. (this is reported on in greater detail later)

5. Whilst trying to balance the goal of using open-source components with creating a system that performs as effectively as possible, some design decisions were made to prioritize the accuracy of the system's performance at the expense of using only open source components.

[1] For a more in depth look at how the Category One processes have been implemented please refer to section 2.1.2 *Current Architecture* of the first report.

Upon that being said, the system can be operated using only open source components.
The proprietary components and the open-source alternatives used in the deliverable are:

- **AWS Textract**: This is a combined Text Localisation and OCR engine that delivers very accurate results. This project utilizes the AWS Textract API to for both the Text Localisation and OCR

  The system is capable of using other OCR engines and is already configured to use PyTesseract. This is an open-source Text Localisation and OCR engine, but the output results are not as accurate as AWS Textract. To allow for the best overall results, the system leverages AWS Textract.
  AWS provides a very reasonable free tier for use of Textract with 10,000 free requests per month.

- **AWS S3 Bucket Storage**: S3 Bucket Storage is a service that allows for the storage of files in a bucket. This is used to store the invoices.

  This can easily be replaced with local storage or any other storage service.

1.4 Side Benefits

There are many useful features which become available as a result of having all of this information available in one system. These insights come in the form of individual data per business but also trends and such from the data aggregated from all users of the system. A brief example of some of these include:

- The ability to query the financial and inventory figures. With a simple GUI the user can be served up current sales vs other time periods and many other powerful ways to gain insight into the business with information already in the system.

- The ability to do some exploratory data analysis (EDA) and other data analytical activities which can provide the business owners with some new data driven insights about their business. These insights would usually only be available to larger businesses with IT teams or businesses with owners who are data science savvy. Businesses with these characteristics in the drinks' industry make up only a tiny fraction of the population based on my decade plus of experience within the industry in Ireland and around numerous European cities.

- An example of another insight that can be derived from the information in the system which is of value to external entities, given decent levels of adoption in the industry, are live sales per product. Having a multitude of different users in the system, the sale quantities of specific items can be accessed and / or extrapolated in real-time. This can provide some invaluable data to brewers, of sales which could be utilized to precisely schedule production times and production quantities.

1.5 Planning and Strategy

ClickUp [14] is used as the project management solution (more details in **??**: Conclusion).
As the project is split in two development cycles, the following is an architectural diagram of the system in its entirety, Category One and Category Two processes inclusive.
The components implemented for each phase are clearly distinguished from each other by the vertical, double-red lines:

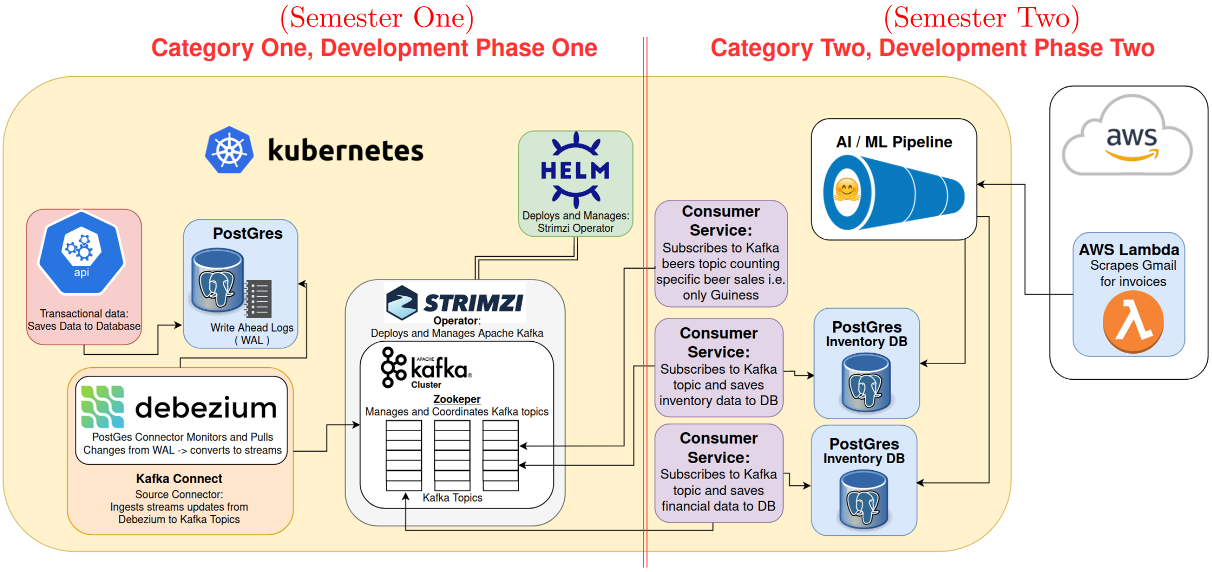

Figure 1.1: System Architecture

Note: The actual delivered architecture varies slightly from the one depicted in Figure 1.1, only with regard to the Category Two section. Details to follow in the Architecture chapter (Chapter 2). The reason this diagram is included is that this is the architecture that would be used in the final product, and the optimal architecture. Owing to some difficulties with the development tools (**??**) the delivered architecture had to be slightly altered, as will be explained.

2 Architecture and Technologies

2.1 Architecture and Data Flow

The following is the delivered architecture design of the system. The red numbers denote the flow of data and are explained below:

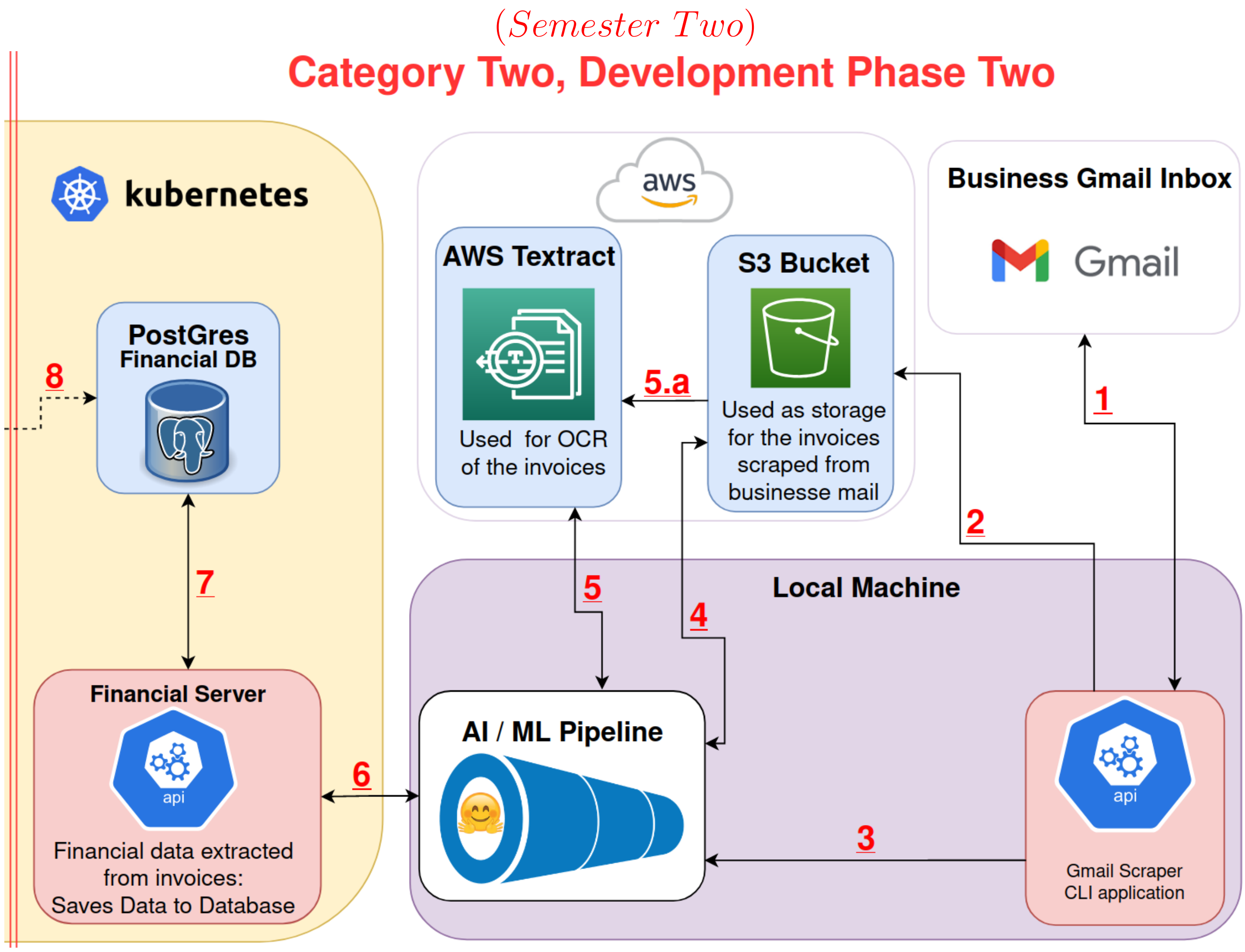

Figure 2.1: System Architecture

1. The Gmail Scraper CLI application scrapes the business' Gmail inbox for invoices. It is currently configured to accept a scrape start date (until present time) and an integer value for the number of invoices to scrape (this is for testing / demo purposes).
   The CLI application is written in Python and uses the `imbox` [92] package to scrape the Gmail inbox.

   In a production environment, this would be altered slightly and deployed as a Lambda function to periodically scrape the business' Gmail inbox.

2. The Gmail Scraper saves invoices, which match the input criteria, to a secure S3 Bucket.

3. The scraper then sends the invoice file name, location and Bucket name to the Machine Learning Pipeline (ML Pipeline), which is deployed behind a Flask server, from here on it will be referred to as the ***Inference Server***.

4. The Inference Server pulls the desired invoice locally.

5. The Inference Server then requests Optical Character Recognition (OCR) data for the desired invoice via an AWS Textract API call. This call tells Textract the location of the invoice in the S3 Bucket and the desired region.

   a) AWS Textract obtains the invoice from the S3 Bucket and performs OCR on the invoice.

   When it finishes, the OCR data is sent to the Inference Server.

6. The Inference Server then prepares the OCR data for inference in a pre-process step, once this step is complete the model performs the inference.
   The results from the inference are returned, and the data then goes through a final post-process step. Once the inference and post-processing are complete, and the data is in the required format the Inference Server sends the data to the Financial Server.

7. The Financial Server is another Flask server written in Python. The server is a running service located in the Kubernetes cluster. The Financial Server utilizes the SQLAlchemey [96] *Object Relational Mapper (ORM)* as a *translational* layer to communicate with the Postgresql database, also deployed in the Kubernetes cluster. The Financial Server saves the data to the Financials DB.

8. The dotted line depicts the interaction between the Kafka consumer, obtaining and saving transactional data (not operational) to the Financials database.

2.1.1 The Pivot, Explained

As can be seen by comparing the proposed architecture, Figure 1.1, and the delivered architecture, Figure 2.1, the system architecture has been altered. The shift may look significant, but the components are fundamentally the same. As the deployment of a full Kubernetes environment was prohibitively expensive, the system was deployed in a Minikube cluster. This actually increased the complexity as components to link services running locally to services running in the Minikube cluster needed to be created.
The change in architecture is due to the following reasons:

- As mentioned, Minikube is used as the development version of Kubernetes. In essence, it is a single node Kubernetes cluster[1]. The initial architecture, as per Figure 1.1, is designed to incorporate the Inference Server into the Kubernetes cluster. Whilst this is still possible, as the Inference Server is containerised and *Kuberentes-ready*, Minikube does not allow external calls from inside the Kubernetes environment. This seems like a drastic limitation and was not known before the choice of Minikube as the development Kubernetes tool. Minikube will allow endpoints exposed in the cluster to be accessed from outside the cluster but only from the localhost system upon which Minikube is installed.

- Numerous, unsuccessful attempts were made to try and circumvent this limitation of Minikube including:
  - Configuration of a Kubernetes Ingress resource in the cluster.
  - The use of Ngrok on the local machine to expose the Inference Server's endpoint to the internet.
  - The deployment of an Ngrok pod in the cluster to expose the Inference Server's endpoint to the internet.

[1]For more information see section 2.2 *Technologies Used* of the first report

The technical implementations of the above are further detailed in the Section 2.4.2 section.

- The deployment of the Gmail Scraper application locally was primarily done to facilitate the demo and to aid in development. The deployment of the Gmail Scraper to AWS Lambda can be achieved with a minor refactor.

As one can now visualise the data flow throughout the system components, the next step is a deeper dive into the technologies considered for use in the system along with explanation of the chosen technologies and their implementation.

2.2 Technologies Considered

2.2.1 Inference Server - AI / ML Pipeline Structure

The Inference Server consists of the Artificial Intelligence (AI) / Machine Learning (ML) pipeline, which is deployed behind a Flask server. Whilst the implementation of a Flask server is trivial, the AI / ML pipeline was the most challenging component of the entire system. But also the most interesting.
Other sections of this project had a large quantity of 'known unknowns', this section has had a huge amount of 'unknown unknowns'. To extract desired key information from an invoice, the document must first go through a series of steps where each step's input is dependent on the previous step's output. This is why the term used is in industry is 'pipeline'. The approach to tackling this problem must first be outlined:

2.2.1.1 Three-Step Process

To solve the KIE from an invoice problem, this is the three-stage process that will be used:

1. **Text Localisation**: For this step a model is used to identify the location of text in the invoice. The text is wrapped in bounding boxes. As per Figure 2.2:

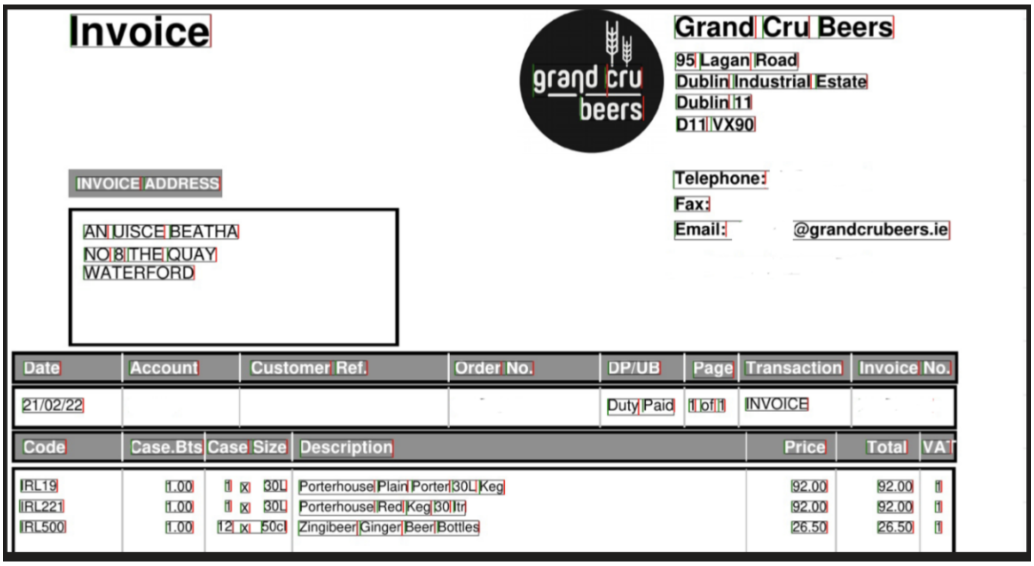

Figure 2.2: An example of the bounding boxes. The locations of each word / text are detected, and a bounding box is created around each piece of text. For clarity, this example has the bounding boxes drawn on. The start of each word starts with a green line and finishes with red.
Note: Some of the text has been removed as these are real documents which contain sensitive data.

This step is not the most difficult and there exist many open-source models that can achieve this with relatively good performance metrics.

2. **Optical Character Recognition (OCR)**: For this step the bounding boxes obtained from the initial step are used by a model to extract the text from the image. The text is returned in the form of a key value pair, where the key is the text and the value is the bounding box or vice versa.
   This step is also not the most difficult and models exist such as Tesseract and OpenCV that can achieve this, also with relatively decent performance metrics. As previously alluded to, the problem lies with the pipeline effect.

   If the Text Localisation stage is not successful or optimal then there is no way any subsequent step can return the desired information. For example, if the Text Localisation step is 90% accurate, The best result that can be returned from the OCR step and subsequent steps is, theoretically, 90%.
   Although just 'theoretically' as in practice no ML step is ever 100% accurate, therefore, each subsequent step will bring with them their *'price'*, a reduction in performance.
   This is why it is crucial that all steps are as accurate as possible as the third and final step is, by an order of magnitude, more difficult than the previous two.

3. **Key Information Extraction (KIE)**: This is the fascinating step. There are no real open source models, like Tesseract for OCR, of any real merit for KIE. This may be because of a lack of research in general along with the variance in source data. The lack of any kind of standard or structure for receipts, but in particular for invoices makes this task all the more difficult. The variance in data makes it very difficult to obtain a model that is generalized (can work on all / different forms of data).
   A number of different approaches / model architectures can be used to try and accomplish this step.

2.2.1.2 Visually-rich Document Understanding Competition - SROIE

From the three stage process as outlined above, the Text Localisation and the OCR steps have both open-source and very good proprietary models. Not to say that they are trivial, as they most certainly are not, but the main area of interest is the KIE step.
In general, the area of visually rich document / semi-structured document understanding is not considered a solved problem in the discipline of computer science. To the extent that organizations exist which run competitions to try and further this field. The largest of which is a competition that was started in 2019 by a collaboration of universities from across the globe known as the *Scanned Receipts OCR and Information Extraction (SROIE)* as part of the larger set of challenges in the area of computer vision, the *Robust Reading Competition* [79]. This is driven by the Computer Vision Center [16], a specialised research campus in the Universitat Autonoma de Barcelona (The Autonomous University of Barcelona). Along with a host of other universities from Shanghai to Aston to Nanyang, amongst others.

The organisers for this competition created one of the first publically available and largest datasets (of receipts) for use in this competition, known as the SROIE dataset. The competition is still ongoing, there is a leader board and there are still entries being added periodically. The following is an example of the SROIE dataset:

(a) (b) (c) (d) (e)

Figure 2.3: SROIE Dataset Example

The SROIE competition was, initially, the main focus of research for this project and was an invaluable source for gaining a look into the cutting edge research carried out on visually rich document understanding [72]. The papers also reveal the different approaches taken by the participating teams.

The SROIE website contains links to some open-source code repos for the entries. It was the perfect place to start research and to get a better understanding of the problem space.

Ranking Table

Description Paper Source Code

Date	Method	Recall	Precision	Hmean
2021-11-24	StrucTexT	98.70%	98.70%	98.70%
2022-03-18	GraphDoc	98.13%	98.77%	98.45%
2022-01-21	Textmind + ERNIE-Layout	97.26%	99.48%	98.36%
2021-04-19	IE	97.05%	99.56%	98.29%
2021-07-20	Linklogis_BeeAI	97.05%	99.34%	98.18%
2021-01-02	Applica.ai Lambert 2.0 + Excluding OCR Errors + Fixing total entity	96.83%	99.56%	98.17%
2021-06-02	Multimodal Transformer for Information Extraction	96.76%	99.56%	98.14%
2021-02-16	Applica.ai TILT + Excluding OCR Errors + Fixing total entity	96.83%	99.41%	98.10%
2020-12-24	LayoutLM 2.0 (single model)	96.61%	99.04%	97.81%
2021-01-01	Applica.ai Lambert 2.0 + Excluding OCR Mismatch	96.40%	99.11%	97.74%

Figure 2.4: The current results of the SROIE competition in the KIE task.
Note: An interesting observation is that the overwhelming majority of the top end of the leaderboard are all using some variation of a model based on the *transformer* architecture.

The methods used by different teams vary greatly as can be seen in the ranking graph [90] by the large variation in both models used and scores achieved.
Some very large and innovative tech companies have entries in the competition including Baidu, Microsoft, Tencent and Samsung to name but a few.

It must be noted that the dataset differs substantially from the use case for this project. No publically available dataset (of invoices) was available for this project, so one was created from the authors personal business.

A further point of interest is that the SROIE competition requires only four fields, to be extracted. As such most projects limited their tags (the tagged field i.e. `total_amount` for receipt total) to four fields - company, date, address, and total. For comparison, this project ended up with over 20 fields in order to extract the desired information.

Whilst format of receipts differs, the variance is not that great. Most receipts have a similar structure. The same can not be said about invoices. For invoices, the structure is much more varied as to are the borders / boxes / white space which separate the values.

2.2.1.3 SROIE Models

Whilst these differences posed challenges to completing this project, it was none-the-less decided to start trying to implement some of the open-source models from the competition. The initial attempts proved to be extremely time-consuming and joyless. The text localisation models were attempted first. From 5 models attempted, only one was successful in deployment.

The attempts at running the OCR models proved a little more successful with two of four being successfully deployed. No KIE models could be successfully deployed from the competition. There were many factors which added to the many unsuccessful attempts:

- The models used varied greatly in the dependencies needed to run and the versions of the different packages used. There is a considerable difference in running a model on PyTorch and Tensorflow / Keras.
- An initial lack of implementation / deployment experience or initial working knowledge of Python and its dependencies structure increased the difficulty level.
- Another obstacle was that most of the repos contain comments and explanations of the code in Mandarin. This was an interesting observation. The vast majority of entries were from China[2].
- Once the initial obstacles and challenges were cleared. The biggest limiting factor in the reproduction of the model deployment became apparent. The models used by teams were trained with machines with more than the GPU memory on the development machine for this report. At 4gb of GPU memory, the hardware limitations were proving to be a problem. Even with pretrained models and weights available from one or two of the repos.

Only a single Text Localisation model could be successfully run on the development machine and the other successful attempts came from running models on AWS ec2 instances optimised for GPU memory. Although this too came with limitations as the instances with GPU access are expensive and there are no free tier options for the hardware needed. At this point a different approach was needed.

Instead of merely trying to implement the open source models as per the repo, it was decided to look at some of the top performing models and try to implement a solution from scratch. It was during this research that the LayoutLMv2 [124] model was discovered. This is a newly open sourced model, released toward the tail end of 2021 by Microsoft Azure AI [8], and it showed some great promise both in terms of performance and in terms of model size, due to the model utilizing *transfer learning* (more details to follow). This model designed especially for visually-rich documents. Implementations of the original LayoutLM model were consistently near the top of the leaderboard for the KIE in the SROIE competition. As to were other models like BERT ber[10] and other variations of BERT like LamBERT [67] and RoBERTa [91]. These models all share something in common, they are all built on the same *Transformers* architecture.

[2] Considering the driving force is a European University and part of the funding for the competition came from the EU, the overwhelming majority of the entries being from China was a surprise. That said, most of the entries in the top 10s in all three tasks were Chinese. It is clear the country focuses its universities in this area.

2.3 Transformers and the LayoutLMv2 Architecture

As LayoutLMv2 is a *Transformers* based model, this section will outline the main concepts underpinning the transformers architecture with a particular focus on the differences and additions that make up the LayoutLMv2 model[3].

The Transformers architecture has revolutionized the area of Natural Language Processing (NLP) since its architecture was proposed in the excellent paper *Attention Is All You Need* [112] developed by Vaswani et al. at Google in 2017.

This architecture is used as the backbone and therefore has given rise to a number of very famous and powerful models such as the aforementioned BERT [10] and OpenAI's GPT series of models, the latest of which is the GPT-3 model [45]. The GPT-3 model has a massive variety of use cases such as English to other language translation (French, Spanish and Japanese are some of the languages supported), Python code to Natural Language, as per Listing 2.1 and many others. A more comprehensive list can be found here [45]:

Listing 2.1: GPT-3 Python code for human language translation as per [77]

```python
def remove_common_prefix(x, prefix, ws_prefix):
    x["completion"] = x["completion"].str[len(prefix) :]
    if ws_prefix:
        # keep the single whitespace as prefix
        x["completion"] = " " + x["completion"]
return x

# Output generated by GPT-3:
# The code above is a function that takes a dataframe and a prefix as input and returns a
    dataframe with the prefix removed from the completion column.
```

Note: not all of the output is as coherent and accurate as the chosen example, although a sub-product of GPT-3 can be found powering tools such as GitHub Co-Pilot which is a tool to aid developers by making suggestions for code completion based on the context of the code already written or by the comments written in the code which Co-Pilot can then use to suggest code completion and / or code generation with very decent results in real time.
This is just a single example of the type of applications for these type of models.

Before Vaswani's paper, the state-of-the-art models for NLP were Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) which implemented an *encoder* and a *decoder*. These types are known as sequence to sequence models (seq2seq) as a sequence is used as the input and output.

For example, Machine Translation (MT) is a seq2seq model where the input is a string of words and the output is a translation in another language for the input string. This was the initial use case for the Transformers architecture, although English to French and French to German were the chosen languages[4].

[3] This section is by necessity quite technical, but there are a number of great resources to introduce this topic in more detail than what is summarized here and can be found in this excellent series of articles [31]. Some other great articles on the topic, here [74], here [20] and here [2].

[4] The Greek translation is pronounced 'Yasu Cosme'

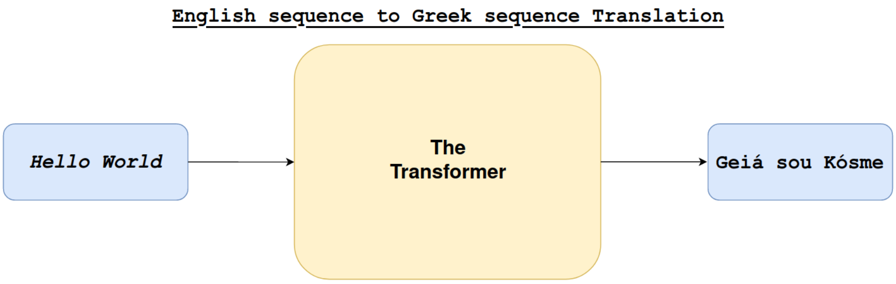

Figure 2.5: Basic Transformer Sequence To Sequence example

2.3.1 Encoder and Decoder High Level Overview

Peeking under the hood, we can see that the encoder and decoder are responsible for the translation. The original paper proposed a stack of six encoders and similarly a matching set stack of six decoders, although these numbers can be altered.

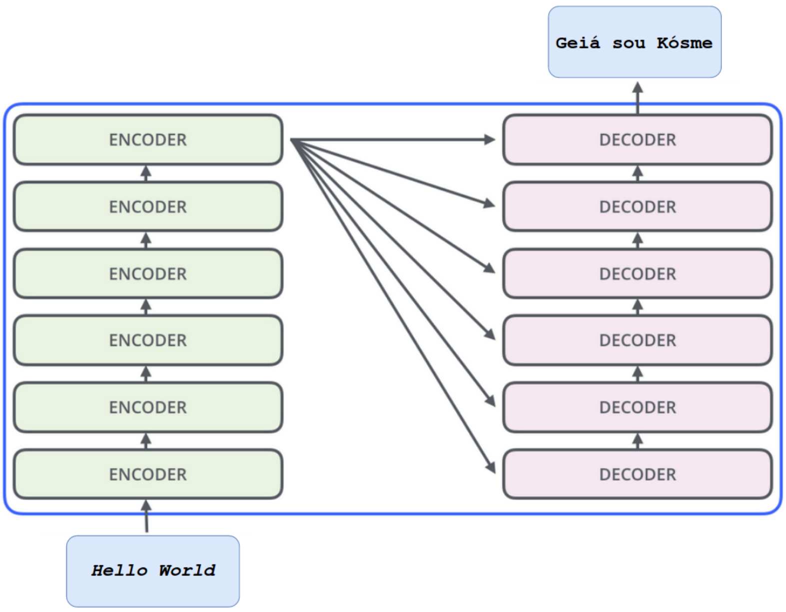

Figure 2.6: Transformer Stacked Encoder and Decoder example [2]

The encoder itself are identical in structure, but they have their own weights associated to them. At first these are set at random, but they are altered through the back-propagation of the training phase by the use of a softmax function.

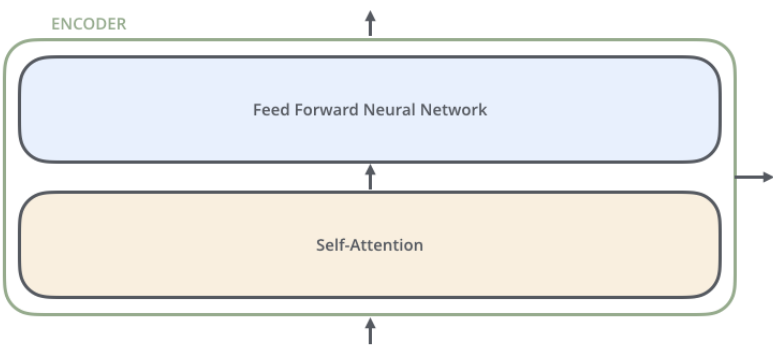

Figure 2.7: Transformer Encoder architecture example [2]

The Self-Attention layer will be described in more detail shortly but for now lets look at the decoder architecture:

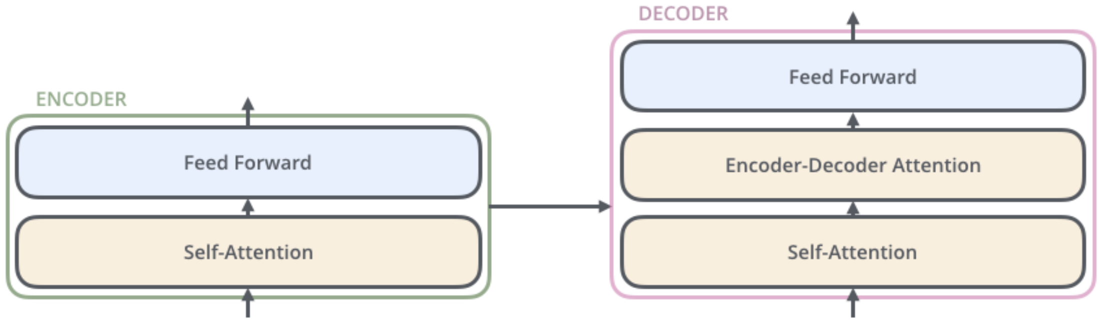

Figure 2.8: Transformer Decoder architecture example [2]

As is observed the decoder is almost identical to the encoder save for an added `Encoder-Decoder` layer. This layer *helps* the decoder to 'focus' on the relevant part of the input sequence. Now that we have a very high level idea of the main components of the Transformers architecture, we can look at the actual architecture of the original Transformers model.

2.3.2 Original Transformers Architecture

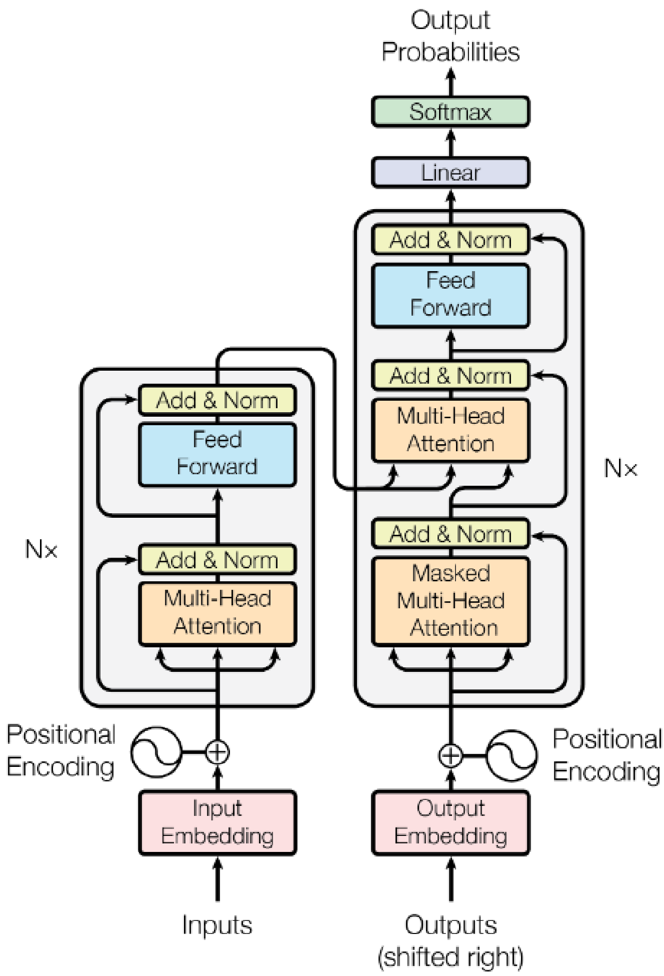

Figure 2.9: Original Transformers architecture [112], akin to the previous example the encoder stack is situated on the left whilst the decoder is situated on the right.

- **Black Arrows**: In Figure 2.9, the black arrows depict the dataflow.
- **Input Embeddings**: As can be seen at the bottom of the Figure 2.9, the `Inputs` are fed in to the encoder side and creates the `Input Embedding`. `Outputs` flow into the decoder side and create the `Output Embedding`. This is only during inference. In training the model works differently as the output is known and as such that known sequence is fed in to the decoder side. Part of the output embedding is masked during training so the model doesn't 'peek ahead'.

  In the original transformer architecture the input embeddings are a vector of a fixed size (this usually varies from model to model). The input vectors combine the input sequence, actually a 'tokenized' sequence (see Section 2.3.4) combined with the positional data (1-D) of that particular token in the input sequence.
- **Add & Norm**: The `Add & Norm` refer to the addition of weights and a normalisation function, which uses 'layer normalization' [114] to normalise.
- **Multi-Head Attention**: The `Multi-Head Attention` is the heart of the transformer. It is essentially numerous self attention layers stacked together (more detail to follow).
- **Linear**: There are two linear translations in the `Linear` component which directly proceed a softmax function.
- **Feed Forward**: The feed forward neural network is a stack of layers. An input layer, some hidden layers and an output layer. The data never flows backwards (back propagation) only forwards. The goal of the feed forward network is to approximate some function of the input.

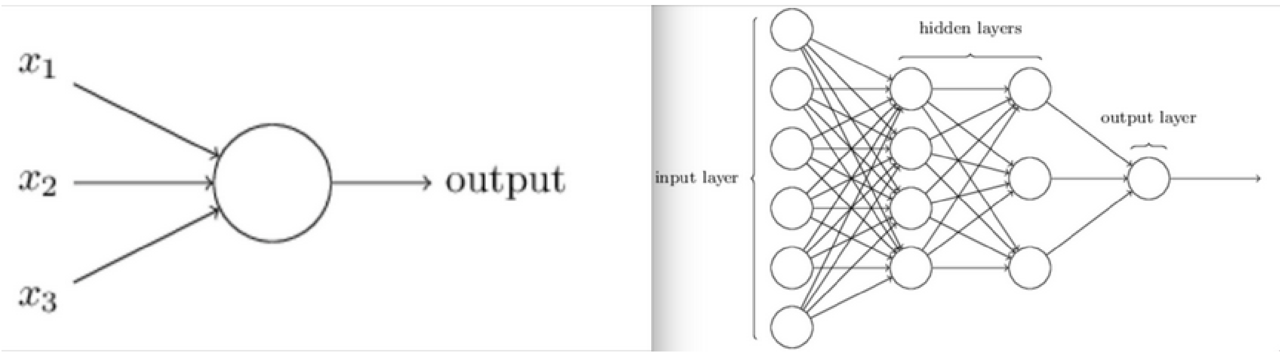

Figure 2.10: Classic Feed Forward Neural network layer architecture [47], The left side depicts a single perceptron whilst the right depicts a multilayer network.

  They are also known as a *Multi-Layer Perceptron (MLP)*. One of the first and most popular deep learning models [41].

- **Softmax**: The Softmax function is used to compress the outputs to form a number in the range 0 - 1 `Output Probabilities`.
- Output Probabilities determine the token for that position. The token sequence is then sent back around to the start of the decoder stack.
- They are shifted right by one position as a special kind of token to indicate the start of a sequence.
- The process is repeated until the end of the sequence is reached - for inference, or until the epochs are completed - for training.

2.3.3 Context

Human languages are a beautiful construct. The ability to express complex ideas and meanings to each other is fundamental to our species evolution, both technical and cultural. But they are also incredibly complex to learn. There are many different syntaxes, rules and of course, rule breakers. A word can take on a range of different meanings depending on the context (a homonym).
For example:

1. *The sound of a dog bark startled the cat.*
2. *The cat scampered up the tree bark.*

The word *bark* has two different meanings depending on the context. Humans are quite good at being able to tell which meaning should be derived from the context, but trying to teach this to a machine is a much more complex task.

To overcome the complexities of the human language, some pre-processing must first be performed on the text data to convert it to numbers which the model can use to manipulate and ultimately learn from. The way that transformers based models remember the distances between words, in sentences or what their *closeness* / association is to other words by using the attention mechanism.

To understand what that is, it is helpful to understand the data which flows into the model as the input. We have already briefly touched on the `embedding` procedure for the original transformer model, but now we will look at it in more detail as we compare it to LayoutLMv2. Possessing the knowledge of what data is in the input embeddings (and output embeddings) will allow us to understand what the attention mechanism is doing.

The input string is not a sequence of words, but a sequence of *tokens*.

2.3.4 Tokenisation

To improve performance a body of text is first tokenized, or split into smaller chunks.
Tokenization usually comes in three different forms; there are word, subword and character-based tokenization methods [120].

- **Word Tokenization**: A word is defined as a sequence of characters which are separated by a delimiter, usually, separated by a space. This method has some drawbacks, like when the model encounters *Out of Vocabulary words (OOV)*, these are words that the model has not encountered in training and as such do not appear in the vocabulary.
  There are some ways to deal with OOV words, but they are not very performative.

  A further issue is the size of the vocabulary. Pre-trained models, such as the transformers based models, are usually trained on a massive corpus of data. As each unique encountered word is stored, the model size quickly explodes.

- **Character-Based Tokenisation**: A character-based tokenization is a method of tokenizing a text document by splitting the text into individual characters.
  The downside of character-based tokenization is that the model is not able to learn the meaning of the words as there are far more combinations of individual characters than that of words. So detecting a pattern between these characters is extremely difficult. The vocabulary size is only 26.

- **Subword Tokenization**: A subword tokenization is a method of tokenizing a text document by splitting the text into subwords. For example, the word *smartest* is split into *smart* and *est*, whilst the word *largest* is split into *large* and *st*. The subword tokenization method is very performative, as it is able to learn the meaning of the words. If the model encounters an OOV

word, it will break it down and may learn meaning from the subwords and the subwords *distance* from other words.
It is important to note that this method will not split every word into subwords. Frequently occurring words are kept as is, whilst less frequently occurring words are split. This is highly dependent on the corpus, but as an example, the word *annoy* will not be split, if the word *annoying* occurs frequently in the corpus then *annoying* will also be kept. If however, it is rare in the corpus then it will be split into *annoy* and *ing*. If they model encounters an OOV word such as *annoyingly*. It can make sense of the word by splitting it into *annoy*, *ing* and *ly* then it will find subwords which match.

There are many subword tokenization methods, but two shall be explained for relevance.

- **Byte-Pair Encoding (BPE)**: is a popular and performative method that initializes the vocabulary to include every character present in the corpus, and each set of characters' (words) frequency is determined. BPE then counts the frequency of each possible symbol pair that occurs most frequently and merges them together. This merge strategy is repeated until stopped by the user. It is a tunable hyperparameter.
  This is easier explained by looking at an example, as per the Hugging Face documentation [100]:

  This is the corpus and frequency:
  `("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)`
  Which gives us this base vocabulary:
  `["b", "g", "h", "n", "p", "s", "u"]`
  The first symbol pair chosen is:
  `("u" + "g")`
  As this combination appears most $10 + 5 + 5 = 20$.
  The symbol pair is added to the vocabulary:
  `["b", "g", "h", "n", "p", "s", "u", "ug"]`

  The merging process is repeated until the user defined limit of the vocabulary size or number of merges is reached.
- **WordPiece Tokenization**: is a subword tokenization method which is used in the LayoutLMv2 model. It is based on and is very similar to the BPE method. The difference is that the BPE method chooses the most frequent symbol pair, whilst the WordPiece method uses probabilities.
  As per our example:
  `"u"` followed by `"g"` would only have merged if the probability of `"ug"` divided by `"u"`, `"g"` was greater than any other pair.

Most of the Transformers architecture based models use some BPE or some variation of it.

2.3.5 Multi Modal Token Embeddings

These tokens are given numerical IDs which are kept in a look-up table.
The tokens are not the only data fed into the model. The *multi-moldality* refers to a combination of other data types. With the original transformers architecture the 1-D position or position in a sequence is combined with the In the case of LayoutLMv2, the multi-modality is the combination of the token and the position. This is known as the *Layout Embedding* and refers to the geometrical position of the token in the document. As this is a requirement and a large part of the reason that this model is so performative, a list of bounding boxes must be provided to the model upon calling it for training or inference.
The tokenizer process determines the bounding box (from the initial OCR'd input bounding boxes for words) per token in the tokenization stage.

Furthermore, the segment embeddings are used to distinguish different text segments.
A further addition to the LayoutLMv2 model which again differs from the original transformers model is further input into the final embedding which will traverse the system. This input is a visual representation AKA an *image embedding* of the token [68]. As the document in JPG or PNG format is also passed in as a parameter when calling the model, the model is able to slice out the token locations from the document. This is achieved in parallel to the token and position encoding via a *Faster R-CNN* [40]. LayoutMLv2 (the Hugging Face version) uses Detectron2 [39] as the Faster R-CNN which essentially acts as the model's visual backbone and object detection algorithm.

The combination of tokens, positional data in both the 1-D sequence and 2-D the entire document along with the visual aspect provided by the Detectron2 and the segment embeddings are the final embedded input matrices. As detailed, these models are known as multi-modal as there are a variety of input mediums that constitute the embeddings.
It is important to note that embedding only occurs once, in the initial input stage of the model.

2.3.6 LayoutLMv2 Architecture

Know having a greater understanding of the input for the LayoutLMv2 model we observe the architecture.

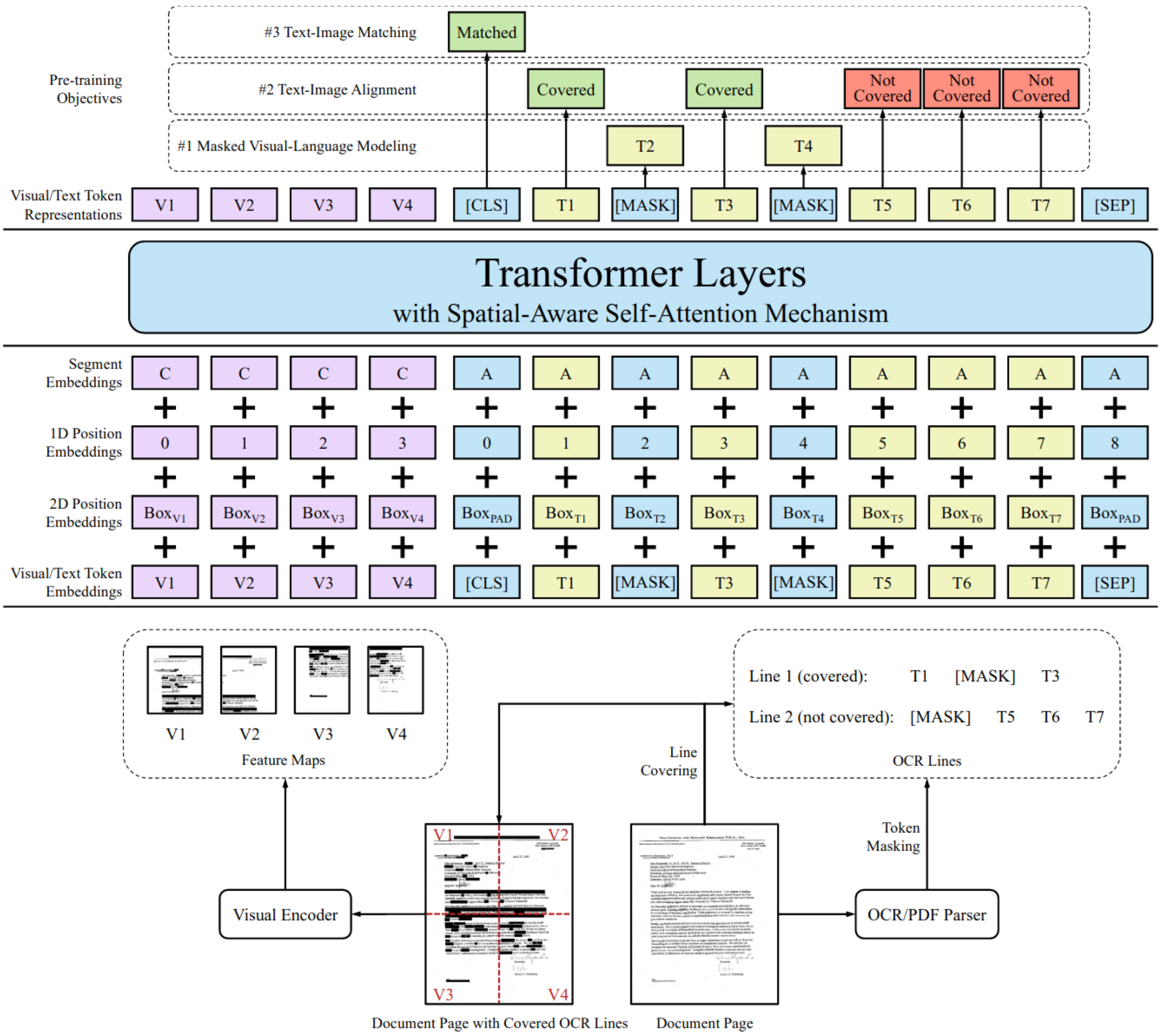

Figure 2.11: LayoutLMv2 Architecture as per [124]

Should I swap the architecture image and the multi modal token embeddings section? it essentially explains the layers, I kind of wanted to let the reader build an image in their head by reading it and then seeing the architecture. I may not have done a good enough job in my descriptions though or maybe its just easer to map a diagram to a bullet list? oh yea I should restructure that as a bullet list ## Observing Figure 2.11, it is useful to define some special case tokens which have not been covered thus far:

- [**BOS**]: Beginning of sentence.
- [**EOS**]: End of sentence.
- [**UNK**]: Out-of-vocabulary tokens
- [**PAD**]: When a sequence size is smaller than expected constant sequence size padding is added. The model knows to ignore the padding during processing due to its special type.
- [**CLS**]: to initialize the sequence.
- [**MASK**]: to mask tokens (for pre-training purposes).

Having already covered the basic transformer layers architecture, our attention can turn to the self attention mechanism.

2.3.7 Multi-head Self Attention, Is All You Need

Attention is they key component of the transformer architecture[5]. Remember the difficulties surrounding human language context? (Section 2.3.3) This is the layer that attempts to capture and learn the complexities of that context. Attention is used in the model in three places as per Figure 2.12:

[5] Attention is also one of the more difficult concepts to grasp. These resources played a pivotal role in acquiring an understanding [30], here [32], here [19]. and here [2].

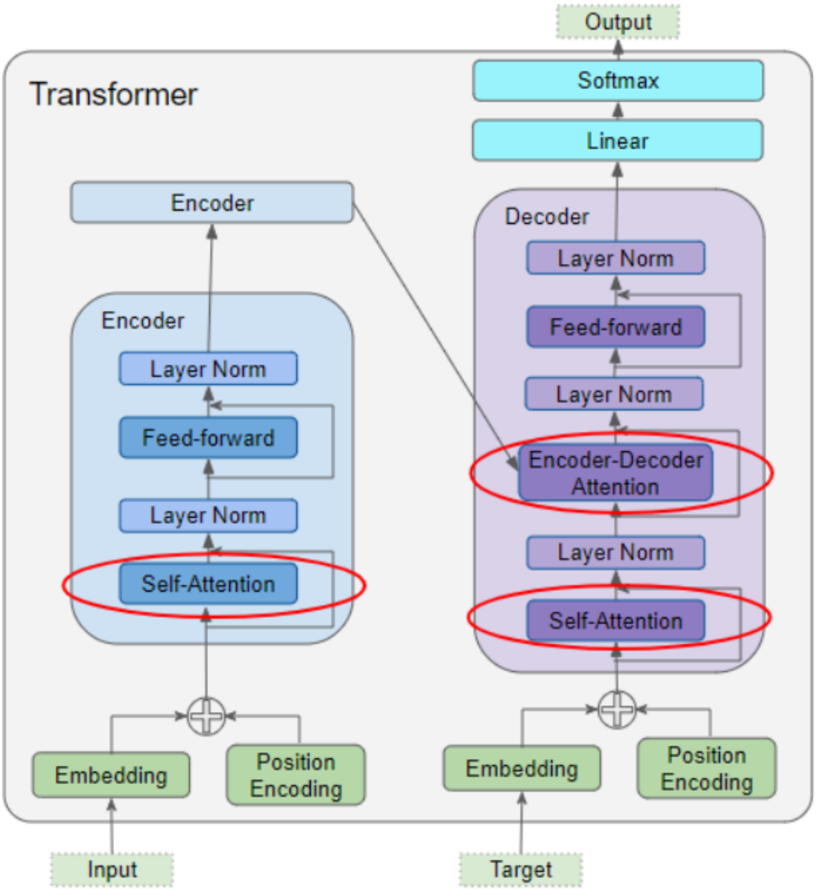

Figure 2.12: Attention is used in three places from here [32]

- Self-Attention in the Encoder.
- Self-Attention in the Decoder.
- Self-Attention in the Encoder-Decoder, also in the Decoder.

The attention layer has three input parameters namely, the *Query (Q), Key (K), Value (V)*. All three of these parameters are matrices of the same size.
From the input sequence every single word (actually embedding, but it is easier to think of it as a word and will be referred to as such in this section) in the sequence is represented as a vector. In practice, this would be just like a single self-attention layer. As a Multi-head attention layer is used in this model architecture this would be analogous to concatenating other word vectors to create a matrix. This is the difference between a regular self attention layer and a multi-head attention layer, the increase in dimensionality. This increase in dimensionality brings with it many performance improvements, the model can train and infer quicker. Along with more accurate results, higher dimensionalities allow for more details to relationships to be recorded.
For the sake of clarity this report will initially reduce the dimensionality and describe a single word vector.
This gives a matrix of size (Sequence Length, Embedding Size, ~~Sample Size~~) - as we have reduced dimensionality.

The first step is to create a Query, Key and a Value matrix for each word in the input sequence. They are updated via the three separate linear layers as per Figure 2.13. Each linear layer has its own weights [2].

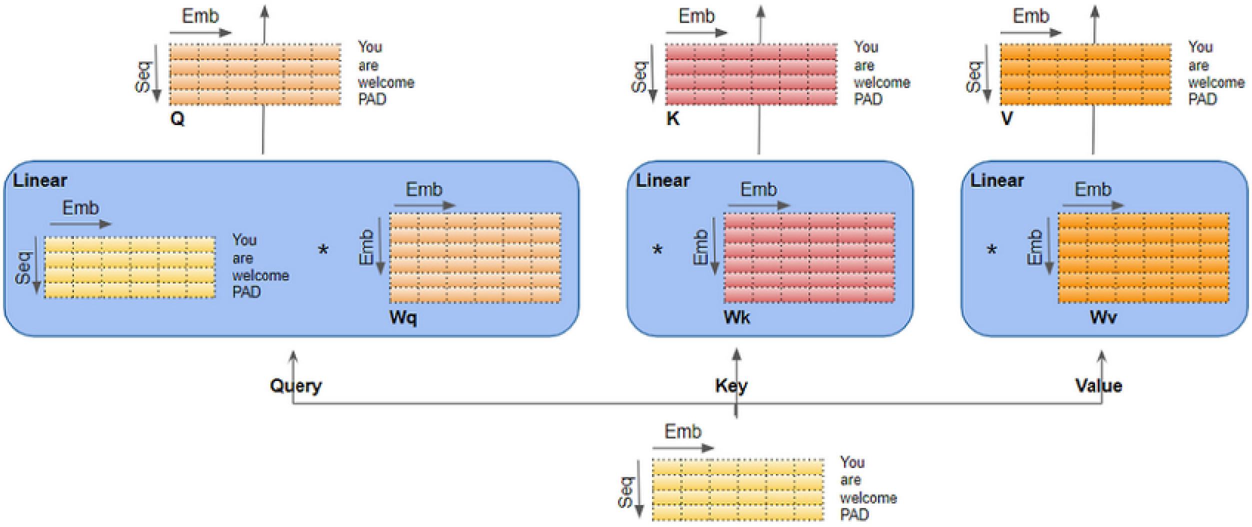

Figure 2.13: Key, Value and Query linear layers from here [32].

In training the values for the linear layers are initially set randomly and are updated during training to return the final weights[6].
For the multi-head attention layer the matrices are **logically** split up and distributed across the heads to allow for computation in parallel. Choosing a *query size* parameter determines the size of the logical partitioning.

The important part to note here is that each input word has gone through a series of transformations. Position encoding, Embedding and the Linear Layers.
Each of these translations is tunable, and it is how the model learns. When a predicted output is wrong, the weights are altered to reflect the wrong decision. Similarly, for correct output in the training phase, the weights are altered to reflect the correct decision.

2.3.7.1 Attention Score

2.4 Hugging Face LayoutLMv2

Hugging Face [53] is a French company which initially developed a chat app that has since pivoted into becoming one of the most popular deep learning model platforms [101]. They specialise in Natural Language Processing (NLP) and they have a large number of models available for public use. The company provide the `Transformers` library which contains over thirty pre-trained models available in over 100 languages [54]. The models are all of based on the transformers architecture as proposed in the excellent paper *Attention Is All You Need* [112] developed by Vaswani et al. at Google in 2017. The state-of-the-art models for NLP were Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) which implemented an encoder and a decoder [13]. ## I want to stick in a proper description and explanation of the encoder and decoder architecture along with the transformer architecture here. is this too technical? I feel like it may be? ## The paper, along with subsequent varied implementations of it, are responsible for a shift away from these models toward the transformer architecture.

2.4.0.1 Transfer Learning

Transfer learning is a technique which allows a model to be trained on a dataset or numerous datasets to be used as its base corpus. The initial training of these models take a huge amount of

[6]This depends on the training, pre-training or fine-tuning, these concepts will be covered in detail later.

data and compute power.

As an example, The *BigScience Large Language Model* [11] is a model that is currently in training. The training of BigScience's main model started on March 11, 2022 11:42am PST and will continue for 3-4 months on 384 A100 80GB GPUs of the Jean Zay public supercomputer [12]. The data set contains 341.6 billion tokens. which is approx. 1.5 TB of data.
That is an astonishing amount of compute, data and engineering expertise.
The transformers library is a very powerful library and it looks to cut away at the *boiler plate* code required to implement a model.

2.4.1

use.

2.4.2 The Challenges

A kubernetes Ingress exposes HTTP and HTTPS traffic from outside the cluster to services inside the cluster. Rules are defined in the Ingress resource and these rules control the traffic

3 Implementation

3.1 Overview

There are a number of steps that are required in order for the system to be able to return the desired output. We must first gain an understanding of how the model works. At a very basic level the model works as follows:

The model will aim to process a previously unseen sample (an invoice not in the training set), look through each piece of text / word in every bounding box, determine the words relationship with other words via its position. Then based on the annotated examples it has ‘learned from’, determine which labels (if any) should apply to each word in the unseen document. The words and their label should be returned by the model.
This section may not be needed dependent on the detail in the previous section.

3.2 Dataset Preparation

To fine-tune ## fine-tune needs to be explained in the context of transfer learning ##
the model on the custom invoice dataset, the dataset needs to be prepared. The dataset needs to be annotated with labels. These labels are the labels the model will predict for fields on an invoice during inference.

The annotation process is both extremely tedious and time-consuming but has to be done with great care and precision for optimal results from the model. If the annotations in the training set are not accurate the model will never be able to infer the correct labels. If there is too many mistakes in the training set then the model may fail to converge.

Initial attempts at using PDF viewers and annotating the invoices with them proved extremely slow and problematic. Most readers are simply not designed for the level of annotation that is required for model training. This was only found out after testing numerous applications, including Master PDF Editor, Acrobat Reader and Zathura PDF. All in all about eight such applications were tested before the change of course to specialised annotation software.

There are some open-source annotation software that can be used to annotate documents for ML purposes. This process was also very time-consuming. A lot of the software out there was difficult to install / had to be installed from scratch. Having tried 5 or 6 different applications, including LabelMe [115], VGG Image AnnotatorãutociteduttaAnnotationSoftwareImages2019 and Label Studio [66], it is clear that the vast majority are designed for image classification and not NLP. After reading countless articles about how to use the tools, none of them allowed for the swift annotation of invoices. Some also have steep learning curves. More often than not the applications had clunky UIs and would be extremely buggy. At this point a decision was taken to try and find some student rates for professional annotation software.

There are a number of professional annotation tools and having researched them thoroughly, Light-Tag [70] and UBIAI [34], were the options attempted. Light-Tag proved extremely expensive. But UBIAI was the most affordable option, and what an option. The software was incredibly straightforward to use and the support team (the founder) was very helpful. I reached out to the support team for help with the software as I was looking for a student discount. The founder, Walid, was the same person who had written a number of articles about the use and training of the original

LayoutLM model for invoice training [35] [36] and [3], which were very helpful in the writing of the training script.
Talking with Walid, he was very interested in this project. Walid was doing something similar in the area of semi-structured document understanding and was also frustrated by the annotation tools available. This is how UBIAI was born. With a very generous student discount secured for the project and some great conversations about the structure of the model etc. the next step was to use the software to get the annotation done.

The steps are as follows:

1. **Label Creation**: involves planning and creating the desired labels. The labels denote the key pieces of information that are of interest. So the appropriate labels must be created for the relevant fields.
   For this project these are the labels which are relevant for the desired output and functionality of this part of the system.

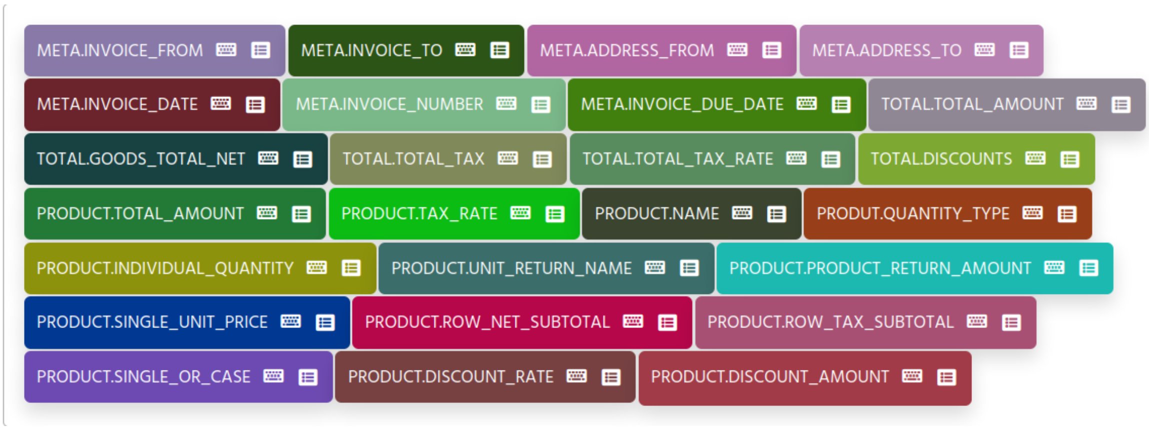

Figure 3.1: Training and Inference Labels created for the system

2. **Annotation**: of all invoices in the dataset by applying the labels to the words. The annotation process is very straightforward, the labels are color-coded and have customizable keyboard shortcuts. The text from the document is OCR'd and displayed sequentially in the bottom half of the screen. The PDF document is displayed to the side. One can simply highlight an area of text on the document and the current enabled label is applied to the text. As text on the document is labeled the corresponding text is highlighted in the OCR'd output section.

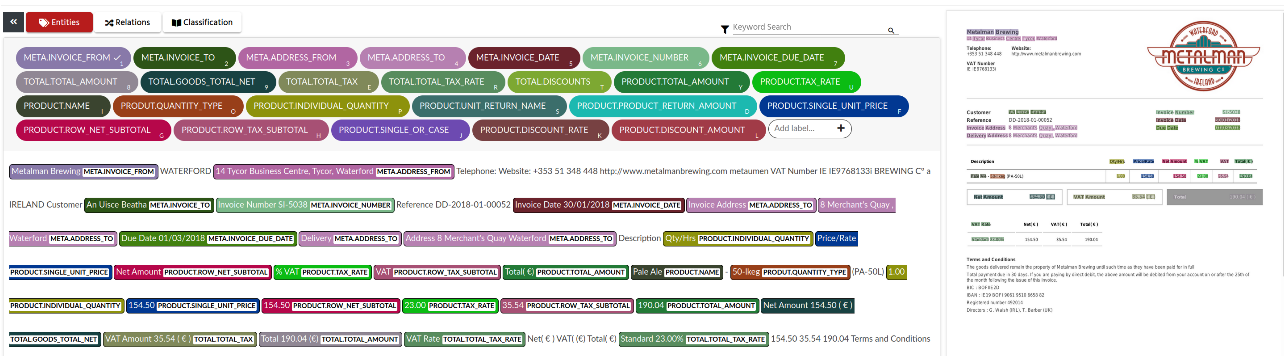

Figure 3.2: Annotating the dataset using UBIAI Annotation Tool

Even with this enhanced workflow, the annotation process was still very time-consuming. In total 86 documents were annotated. Whilst this isn't an ideal amount, the larger the training

set the better the model can learn, research showed that from approx. 50 documents LayoutLMv2 was able to return decent results.
This finding was corroborated by Walid.

3. **Exporting the Data**: the data is exported from the UBIAI software in an optimized format for the transformers based architecture model. There appears to be a wide range of different options for data. With UBIAI supporting these methods:

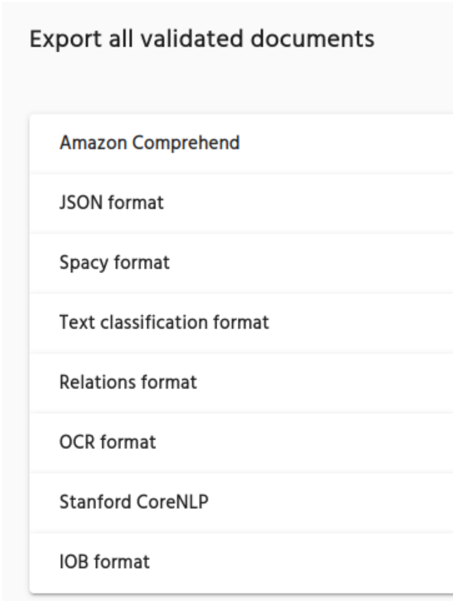

Figure 3.3: UBIAI Export Options

If the data is not exported in the correct format it can take a large amount of time to transform the data into a suitable format for the model.
The documents are exported along with some metadata files which contain the labels and the bounding boxes for each document.

3.3 Training the Model

With the data in the correct format and the labels created, the next step is to train the model. The training script is written in Python, and it utilizes a number of libraries. Including the following:

- **PyTorch**: the main framework for the model.
- **torchvision**: the library for image processing.
- **transformers**: the library for the model and processor.
- **Detectron2**: the library for object detection.
-

There is still quite a bit of processing to do before the model can be trained. The data is manipulated to arrive at a pandas dataframe of as per Figure 3.4:

df

	id	words	bboxes	ner_tags	image_path
0	0	[Tullys, Wholesale, Arles,, Ballickmoyler,, Ca...	[[407, 36, 487, 58], [496, 36, 636, 55], [337,...	[4, 25, 0, 42, 21, 57, 57, 57, 57, 57, 57,...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...
1	1	[Tullys, Wholesale, Arles,, Ballickmoyler,, Ca...	[[407, 36, 487, 58], [496, 36, 636, 55], [337,...	[4, 25, 58, 58, 58, 57, 57, 57, 57, 57, 57...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...
2	2	[Tullys, Wholesale, Arles,, Ballickmoyler,, Ca...	[[408, 37, 488, 58], [496, 37, 637, 55], [337,...	[62, 62, 58, 58, 58, 57, 57, 57, 57, 57, 5...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...
3	3	[Tullys, Wholesale, Arles,, Ballickmoyler,, Ca...	[[408, 37, 488, 58], [496, 37, 637, 55], [337,...	[4, 25, 0, 42, 21, 57, 57, 57, 57, 57, 57,...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...
4	4	[Tullys, Wholesale, Arles,, Ballickmoyler,, Ca...	[[408, 37, 488, 58], [496, 37, 637, 55], [337,...	[4, 25, 58, 58, 58, 57, 57, 57, 57, 57, 57...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...
...	...	...	...	...	...
81	81	[Carlow, Brewing, Company, Muine, Bheag, Busin...	[[96, 50, 145, 59], [149, 50, 207, 61], [210, ...	[4, 46, 25, 0, 42, 42, 21, 57, 57, 57, 57, 0, ...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...
82	82	[Carlow, Brewing, Company, Muine, Bheag, Busin...	[[96, 50, 145, 59], [149, 50, 207, 61], [210, ...	[4, 46, 25, 0, 42, 42, 21, 57, 57, 57, 57, 0, ...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...
83	83	[Carlow, Brewing, Company, Muine, Bheag, Busin...	[[96, 50, 145, 59], [149, 50, 207, 61], [210, ...	[4, 46, 25, 0, 42, 42, 21, 57, 57, 57, 57, 0, ...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...
84	84	[Carlow, Brewing, Company, Muine, Bheag, Busin...	[[96, 50, 145, 59], [149, 50, 207, 61], [210, ...	[4, 46, 25, 0, 42, 42, 21, 57, 57, 57, 57, 0, ...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...
85	85	[INVOICE, Delivered, to:, T/A, AN, UISCE, BEAT...	[[23, 20, 109, 34], [28, 65, 72, 72], [73, 66,...	[57, 57, 57, 57, 6, 48, 27, 57, 1, 43, 43, 43,...	/content/drive/MyDrive/FYP/dataset/fyp_2_bKLum...

86 rows × 5 columns

Figure 3.4: Training Dataframe

It is important to note that each row in the dataframe represents a single document (invoice). For this training run, there were 86 documents annotated, therefore, there are 86 rows in the dataframe. The columns are as follows:

- **id**: the id of the document, incrementing from 0 to 85.
- **words**: this is a list of every OCR'd word in the document.
- **bboxes**: this is a list of bounding boxes (the four coordinates of the box) for each word in the document. the bounding boxes need to be normalized. This is done by dividing the bounding box coordinates by the width and height:

Listing 3.1: Bounding Box Normalization

```
def normalize_bbox(bbox, width, height):
 return [
     int(1000 * (bbox[0] / width)),
     int(1000 * (bbox[1] / height)),
     int(1000 * (bbox[2] / width)),
     int(1000 * (bbox[3] / height)),
 ]
```

- **ner_tags**: the Named Entity Relation (NER) tags for each word in the document. These are the labels that were applied to the words. There is some minor processing done, to get to this stage. The labels are encoded as integers and any word that doesn't have a label is assigned the label '0'.

  There are now more labels than were present initially. This is due to the giving the model some extra information, where each prefix letter is has a meaning. This meaning corresponds to the token position of the token in an entity.
  The token prefixes are as follows:
  - **B** - Beginning of a new entity.
  - **I** - Inside an entity. For example, the 'State' token is a part of an entity like 'Empire State Building'. this means the ner_tag would have a a prefix of `I_` [106]
  - **S** - This denotes a single token entity.
  - **E** - The token is the end of an entity.
  - **O** - Doesn't correspond to any entity.

The amount of elements in in the word, bboxes and ner_tags columns should be equal.

- **image_path**: the path to the image file.

The LayoutLMv2 model in this project is used in *TokenClassification* mode.
explain Tokenisation and the WordPiece Algo used by LayoutLMv2 ?

3.3.1 Data for Model Training

- As we prep the model needs to be trained. For the training of the model the dataset needs to be prepared.
  - annotation and correct export of data.
- The model needs to be saved
- The

Bibliography

[1] *A Gentle Introduction to Batch Normalization for Deep Neural Networks.* https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.

[2] Jay Alammar. *The Illustrated Transformer.* https://jalammar.github.io/illustrated-transformer/.

[3] Walid Amamou. *Fine-Tuning LayoutLM v2 For Invoice Recognition.* https://towardsdatascience.com/fine-tuning-layoutlm-v2-for-invoice-recognition-91bf2546b19e. Apr. 2022.

[4] *Apache Kafka.* https://kafka.apache.org/documentation/.

[5] *Apache Kafka on Kubernetes with Strimzi - Part 3: Monitoring Our Strimzi Kafka Cluster with Prometheus and Grafana.* Oct. 2020.

[6] *API — Flask Documentation (2.1.x).* https://flask.palletsprojects.com/en/2.1.x/api/#flask.Request.json.

[7] *API Reference Overview.* https://docs.clover.com/reference/api-reference-overview#.

[8] *Artificial Intelligence Research at Microsoft Aims to Enrich Our Experiences.*

[9] Bhargav Bachina. *How to Use Own Local Docker Images With Minikube.* Jan. 2020.

[10] *BERT.* https://huggingface.co/docs/transformers/model_doc/bert.

[11] *BigScience Research Workshop.* https://bigscience.huggingface.co/.

[12] *Bigscience/Tr11-176B-ml-logs · Hugging Face.* https://huggingface.co/bigscience/tr11-176B-ml-logs.

[13] Jason Brownlee. *Encoder-Decoder Recurrent Neural Network Models for Neural Machine Translation.* Dec. 2017.

[14] *ClickUp™ — One App to Replace Them All.* https://clickup.com.

[15] CloudFactory. *Data Annotation Tools for Machine Learning: An Evolving Guide.* https://www.cloudfactory.com/data-annotation-tool-guide.

[16] *Computer Vision Center — The Computer Vision Centre (CVC) Is a Not for Profit Institute, Leader in Research and Development in the Field of Computer Vision.*

[17] *Concepts.* https://kubernetes.io/docs/concepts/.

[18] *Connectors :: Debezium Documentation.* https://debezium.io/documentation/reference/stable/connectors/index.html.

[19] Stefania Cristina. *The Transformer Attention Mechanism.* Oct. 2021.

[20] Stefania Cristina. *The Transformer Model.* Nov. 2021.

[21] *DD5D5V08elb.* https://link.medium.com/DD5D5V08elb.

[22] *Debezium Architecture :: Debezium Documentation.* https://debezium.io/documentation/reference/architecture.html.

[23] Debezium Community. *Debezium.* https://debezium.io/.

[24] Loris Degioanni. *3 Phases of Prometheus Adoption.* https://www.infoworld.com/article/3275887/3-phases-of-prometheus-adoption.html. May 2018.

[25] *Deploying a Containerized API on Kubernetes — by Victor Steven — Level Up Coding.* https://levelup.gitconnected.com/deploying-dockerized-golang-api-on-kubernetes-with-postgresql-mysql-d190e27ac09f.

[26] Dharti Dhami. *Understanding BERT — Word Embeddings.* July 2020.

[27] *Docker-Compose and Create Db in Postgres on Init.* https://stackoverflow.com/questions/59715622/docker-compose-and-create-db-in-postgres-on-init.

[28] *Docker-Images/Postgres/13 at Main · Debezium/Docker-Images.* https://github.com/debezium/docker-images.

[29] *Docker-Images/Postgres/9.6 at Main · Debezium/Docker-Images.* https://github.com/debezium/docker-images.

[30] Ketan Doshi. *Transformers Explained Visually — Not Just How, but Why They Work so Well.* https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3. June 2021.

[31] Ketan Doshi. *Transformers Explained Visually (Part 1): Overview of Functionality.* https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452. June 2021.

[32] Ketan Doshi. *Transformers Explained Visually (Part 3): Multi-head Attention, Deep Dive.* https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853. June 2021.

[33] Abhishek Dutta and Andrew Zisserman. "The VIA Annotation Software for Images, Audio and Video". In: *Proceedings of the 27th ACM International Conference on Multimedia.* Nice France: ACM, Oct. 2019, pp. 2276–2279. ISBN: 978-1-4503-6889-6. DOI: 10.1145/3343031.3350535.

[34] *Easy to Use Text Annotation Tool — Upload Documents, Start Annotating, and Create Advanced NLP Model in a Few Hours.* https://ubiai.tools/.

[35] *Easy to Use Text Annotation Tool — Upload Documents, Start Annotating, and Create Advanced NLP Model in a Few Hours.* https://ubiai.tools/blog/article/how-to-annotate-pdfs-and-scanned-images-for-nlp-applications.

[36] *Easy to Use Text Annotation Tool — Upload Documents, Start Annotating, and Create Advanced NLP Model in a Few Hours.* https://ubiai.tools/blog/article/fine-tuning-transformer-model.

[37] *Engine Configuration — SQLAlchemy 1.4 Documentation.* https://docs.sqlalchemy.org/en/14/core/engines.html.

[38] *Event-Driven Architecture and Microservices Combination Concerns.* July 2020.

[39] *Facebookresearch/Detectron2.* Meta Research. Apr. 2022.

[40] *Faster R-CNN Explained for Object Detection Tasks.* https://blog.paperspace.com/faster-r-cnn-explained-object-detection/. Nov. 2020.

[41] *Feedforward Neural Network: Its Layers, Functions, and Importance.* Jan. 2022.

[42] René Ghosh. *Extending the Flask Tutorial App into a Fully-Fledged, Highly Available, Cloud-Based Application.* Mar. 2021.

[43] *GOCDC and Postgres.* https://dev.to/thiagosilvaf/gocdc-and-postgres-1m4m.

[44] *Gorilla/Mux.* Gorilla Web Toolkit. Oct. 2021.

[45] *GPT-3 Powers the Next Generation of Apps.* https://openai.com/blog/gpt-3-apps/. Mar. 2021.

[46] *Grafana: The Open Observability Platform.* https://grafana.com/.

[47] Tushar Gupta. *Deep Learning: Feedforward Neural Network.* https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7. Dec. 2018.

[48] *Helm.* https://helm.sh/.

[49] *How 8 Giant Companies Use Kubernetes & 60 Others That Use It.* https://www.containiq.com/post/companies-using-kubernetes.

[50] *How to Integrate Kafka with Istio on OpenShift.* https://labs.consol.de/development/2021/02/02/istio_and_kafka_on_openshift.html.

[51] *How to Plot Train and Validation Accuracy Graph?* https://discuss.pytorch.org/t/how-to-plot-train-and-validation-accuracy-graph/105524.

[52] *How to Use Change Data Capture (CDC) with Postgres.* https://dev.to/thiagosilvaf/how-to-use-change-database-capture-cdc-in-postgres-37b8.

[53] *Hugging Face – The AI Community Building the Future.* https://huggingface.co/.

[54] *Hugging Face Transformers Package - What Is It and How To Use It.*

[55] *Installation Guide - NGINX Ingress Controller.* https://kubernetes.github.io/ingress-nginx/deploy/.

[56] *Intelligent Document Processing with AI.* https://nanonets.com/blog/receipt-ocr/%23receipt-digitization-using-tesseract.

[57] Sergei Issaev. *Beginner's Guide to Loading Image Data with PyTorch.* https://towardsdatascience.com/beginners-guide-to-loading-image-data-with-pytorch-289c60b7afec. Dec. 2020.

[58] *Kafka Connect — Confluent Documentation.* https://docs.confluent.io/platform/current/connect/index.html.

[59] Soham Kamani. *Implementing a Kafka Producer and Consumer In Golang (With Full Examples) For Production.* https://www.sohamkamani.com/golang/working-with-kafka/.

[60] Chetna Khanna. *WordPiece: Subword-based Tokenization Algorithm.* https://towardsdatascience.com/wordpiece-subword-based-tokenization-algorithm-1fbd14394ed7. Aug. 2021.

[61] *Kubernetes Adoption Trends Report.* https://www.cockroachlabs.com/guides/kubernetes-trends/.

[62] *Kubernetes Operator — Stateful Kubernetes Application.* https://k21academy.com/docker-kubernetes/kubernetes-operator/. May 2021.

[63] *Kubernetes Security Best Practices: 10 Steps to Securing K8s.* https://www.aquasec.com/cloud-native-academy/kubernetes-in-production/kubernetes-security-best-practices-10-steps-to-securing-k8s/.

[64] Vinay Kudari. *SQLAlchemy — Python Tutorial.* https://towardsdatascience.com/sqlalchemy-python-tutorial-79a577141a91. Sept. 2020.

[65] John Kutay. *Change Data Capture (CDC): What It Is and How It Works.* May 2021.

[66] *Label Studio – Open Source Data Labeling.* https://labelstud.io/.

[67] *LAMBERT.* Applica. Apr. 2022.

[68] *LayoutLM Explained.* https://nanonets.com/blog/layoutlm-explained/. Mar. 2022.

[69] *Lessons Learned from Running Debezium with PostgreSQL on Amazon RDS.* https://debezium.io/blog/2020/02/25/lessons-learned-running-debezium-with-postgresql-on-rds/.

[70] *LightTag - The Text Annotation Tool For Teams*. https://www.lighttag.io/.

[71] *Logical Decoding Output Plug-in Installation for PostgreSQL :: Debezium Documentation*. https://debezium.io/documentation/reference/postgres-plugins.html.

[72] *Method: StrucTexT - Task 3 - Key Information Extraction - ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction - Robust Reading Competition*. https://rrc.cvc.uab.es/?ch=13&com=evaluation&view=method_info&task=3&m=90335.

[73] *Minikube Start*. https://minikube.sigs.k8s.io/docs/start/.

[74] Eduardo Muñoz. *Attention Is All You Need: Discovering the Transformer Paper*. https://towardsdatascience.com/attention-is-all-you-need-discovering-the-transformer-paper-73e5ff5e0634. Feb. 2021.

[75] Naincyjain. *Effect of Batch Size on Training Process and Results by Gradient Accumulation*. https://medium.com/analytics-vidhya/effect-of-batch-size-on-training-process-and-results-by-gradient-accumulation-e7252ee2cb3f.

[76] Pinku Deb Nath. *Graceful Shutdown of Golang Servers Using Context and OS Signals*. May 2019.

[77] *OpenAI API*. https://beta.openai.com.

[78] *Operator Pattern*. https://kubernetes.io/docs/concepts/extend-kubernetes/operator/.

[79] *Overview - ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction - Robust Reading Competition*. https://rrc.cvc.uab.es/?ch=13.

[80] *Part 5: Handling Migrations With Gorm in Go — Code Sahara*. https://codesahara.com/blog/making-migrations-with-gorm-in-go/.

[81] *PGO, the Postgres Operator from Crunchy Data*. https://access.crunchydata.com/documentation/postgres-operator/5.0.5/tutorial/connect-cluster/.

[82] *Pod — Kubernetes Engine Documentation — Google Cloud*. https://cloud.google.com/kubernetes-engine/docs/concepts/pod.

[83] *PostgreSQL: Linux Downloads (Ubuntu)*. https://www.postgresql.org/download/linux/ubuntu/.

[84] *Production-Grade Container Orchestration*. https://kubernetes.io/.

[85] Prometheus. *Prometheus - Monitoring System & Time Series Database*. https://prometheus.io/.

[86] Real Python. *Flask by Example – Setting up Postgres, SQLAlchemy, and Alembic – Real Python*. https://realpython.com/flask-by-example-part-2-postgres-sqlalchemy-and-alembic/.

[87] Real Python. *How to Build Command Line Interfaces in Python With Argparse – Real Python*. https://realpython.com/command-line-interfaces-python-argparse/.

[88] *Python HTTP Request Tutorial: Get & Post HTTP & JSON Requests*. https://www.datacamp.com/community/tutorials/making-http-requests-in-python. Sept. 2019.

[89] *Releases · Strimzi/Strimzi-Kafka-Operator*. https://github.com/strimzi/strimzi-kafka-operator/releases.

[90] *Results - ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction - Robust Reading Competition*. https://rrc.cvc.uab.es/?ch=13&com=evaluation&task=3.

[91] *RoBERTa*. https://huggingface.co/docs/transformers/model_doc/roberta.

[92] Martin Rusev. *Imbox: Python IMAP for Human Beings*.

[93] Sub says. *Kubernetes - Desired State and Control Loops*. Sept. 2019.

[94] *Secrets*. https://kubernetes.io/docs/concepts/configuration/secret/.

[95] Christos Sotiriou. *Fault Tolerance in Kubernetes Clusters*. Apr. 2020.

[96] *SQLAlchemy - The Database Toolkit for Python*. https://www.sqlalchemy.org/.

[97] *Square: Solutions & Tools to Grow Your Business*. https://squareup.com/ie/en.

[98] *Strimzi Documentation (0.16.2)*. https://strimzi.io/docs/0.16.2/.

[99] *Strimzi Quick Start Guide (0.26.0)*. https://strimzi.io/docs/operators/latest/quickstart.html.

[100] *Summary of the Tokenizers*. https://huggingface.co/docs/transformers/tokenizer_summary.

[101] Anuj Syal. *Hugging Face: A Step Towards Democratizing NLP*. https://towardsdatascience.com/hugging-face-a-step-towards-democratizing-nlp-2c79f258c951. Dec. 2020.

[102] *tEMxXnwlvmb*. https://link.medium.com/tEMxXnwlvmb.

[103] *The 7 Best CDC Tools (Change Data Capture) - Learn — Hevo*. https://hevodata.com/learn/7-best-cdc-tools/.

[104] *The Role That SMEs Will Play in Rebuilding the Irish Economy (Free Downloadable Executive Summary)*. https://aibf.ie/times/the-role-that-smes-will-play-in-rebuilding-the-irish-economy-free-downloadable-executive-summary/.

[105] *Tips & Tricks for Running Strimzi with Kubectl*. https://strimzi.io/blog/2020/07/22/tips-and-tricks-for-running-strimzi-with-kubectl/.

[106] *Token Classification*. https://huggingface.co/docs/transformers/main/en/tasks/token_classification.

[107] *Understanding Backpropagation in a Neural Network - 1*. https://www.linkedin.com/pulse/understanding-backpropagation-neural-network-1-srikanth-machiraju/.

[108] *Using Helm*. https://helm.sh/docs/intro/using_helm/.

[109] *Using SQLAlchemy with Flask and PostgreSQL*. https://stackabuse.com/using-sqlalchemy-with-flask-and-postgresql/. Jan. 2020.

[110] *Using SQLAlchemy with Flask to Connect to PostgreSQL*. https://vsupalov.com/flask-sqlalchemy-postgres/. Aug. 2017.

[111] *Using Strimzi*. https://strimzi.io/docs/operators/latest/full/using.html.

[112] Ashish Vaswani et al. "Attention Is All You Need". In: (), p. 11.

[113] Ivan Velichko. *How to Grasp Containers - Efficient Learning Path - Ivan Velichko*. https://iximiuz.com/en/posts/container-learning-path/.

[114] Nilesh Vijayrania. *Different Normalization Layers in Deep Learning*. https://towardsdatascience.com/different-normalization-layers-in-deep-learning-1a7214ff71d6. Mar. 2021.

[115] Kentaro Wada. *Labelme: Image Polygonal Annotation with Python*. Apr. 2022. DOI: 10.5281/zenodo.5711226.

[116] *What Is A Helm Chart? – A Beginner's Guide*. Aug. 2019.

[117] *What Is a Kubernetes Operator?* https://www.redhat.com/en/topics/containers/what-is-a-kubernetes-operator.

[118] *What Is a Makefile and How Does It Work?* https://opensource.com/article/18/8/what-how-makefile.

[119] *What Is Event-Driven Architecture?* https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture.

[120] *What Is Tokenization — Tokenization In NLP*. May 2020.

[121] *What Is Transfer Learning and Why Is It Needed? - Beginners*. https://discuss.huggingface.co/t/what-is-transfer-learning-and-why-is-it-needed/4431. Mar. 2021.

[122] *What Is Zookeeper and Why Is It Needed for Apache Kafka? - CloudKarafka, Apache Kafka Message Streaming as a Service*. https://www.cloudkarafka.com/blog/cloudkarafka-what-is-zookeeper.html.

[123] *Who's Using Debezium?* https://debezium.io/community/users/.

[124] Yang Xu et al. "LayoutLMv2: Multi-modal Pre-training for Visually-Rich Document Understanding". In: *arXiv:2012.14740 [cs]* (Jan. 2022). arXiv: `2012.14740 [cs]`.