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

April 27, 2022

Abstract

Extraction of key information from financial documents is a tedious and error-prone process. This process is critical for compliance reasons, so it must be performed in a competent manner, often at great expense.

This project automates the parsing of expenditure documents (invoices etc.) using a machine learning pipeline. The pipeline contains advanced open-source models that have been fine-tuned specifically for this task. Extracted data is then fed into a novel system to allow for the viewing of both expenditure and revenue in near real-time.

The system is an event-driven, microservice architecture utilizing both industry standard and bleeding-edge technologies. This allows for a highly scalable and fault-tolerant system, thanks to the use of Kubernetes, Apache Kafka and Debezium. Debezium is an open-source CDC technology that tracks changes in a database. These changes are propagated to the next steps in the data flow, Kafka topics. Other microservices can consume these changes and action upon the changed data. This allows for the further decoupling of the system as consumers/subscribers to the change stream can be added or removed entirely without any downtime of the system.

Contents

List of Figures iv

Listings v

Acknowledgements vi

Link to Demo Video vii

1 Introduction and Project Goals 1
- 1.1 Introduction . . . 1
- 1.2 Motivation . . . 1
- 1.3 Scope . . . 3
- 1.4 Side Benefits . . . 6
- 1.5 Planning and Strategy . . . 7

2 Architecture and Technologies 8
- 2.1 Architecture and Data Flow . . . 8
  - 2.1.1 The Pivot, Explained . . . 9
- 2.2 Technologies Considered for the Inference Server - AI / ML Pipeline . . . 11
  - 2.2.1 Three-Step Process . . . 11
  - 2.2.2 Visually-rich Document Understanding Competition - SROIE . . . 12
  - 2.2.3 SROIE Models . . . 15
- 2.3 Transformers and the path to LayoutLMv2 . . . 16
  - 2.3.1 Encoder and Decoder High Level Overview . . . 19
  - 2.3.2 Transformers Architecture . . . 20
  - 2.3.3 Context . . . 23
  - 2.3.4 Tokenisation . . . 24
  - 2.3.5 Multi Modal Token Embeddings . . . 26
  - 2.3.6 LayoutLMv2 Architecture . . . 27
  - 2.3.7 Attention, Is All You Need . . . 28
    - 2.3.7.1 Encoder . . . 32
    - 2.3.7.2 Query, Key & Value . . . 32
    - 2.3.7.3 Attention Score . . . 33
  - 2.3.8 LayoutLMv2 Pre-Training Tasks . . . 36
    - 2.3.8.1 Masked Visual-Language Modeling (MVLM) . . . 36
    - 2.3.8.2 Text-Image Alignment (TIA) . . . 37
    - 2.3.8.3 Text-Image Matching (TIM) . . . 37
  - 2.3.9 Fine-Tuning . . . 37
  - 2.3.10 Transfer Learning . . . 38
- 2.4 Hugging Face . . . 41

3 Implementation 43
- 3.1 Overview . . . 43
- 3.2 Dataset Preparation . . . 44
- 3.3 Training the Model . . . 48
  - 3.3.1 Model Evaluation . . . 53
- 3.4 Inference . . . 55
  - 3.4.1 Post-Processing . . . 58

3.5 Exposing the Pipeline as a REST API 58
3.6 Gmail Scraper 59
3.7 ML Pipeline Performance 59

4 Conclusion **61**
4.1 Technical 61
4.2 Personal 62

A Appendix **63**
A.1 Self Attention in the Decoder 63
A.1.1 Attention in the Encoder-Decoder 64

Bibliography **65**

List of Figures

1.1 System Architecture 7

2.1 Semester 2 System Architecture. 8
2.2 Bounding Box Example 11
2.3 SROIE Dataset Example (source [118]). 13
2.4 SROIE Results (KIE) 14
2.5 The LayoutLMv2 ancestry. 17
2.6 Basic Transformer Sequence to Sequence example. 18
2.7 Transformer Stacked Encoder and Decoder example (source [4]). 19
2.8 Transformer Encoder architecture example (source [4]). 19
2.9 Transformer Decoder architecture example (source [4]). 20
2.10 Original Transformers Architecture [145] 21
2.11 Classic Feed Forward Neural network layer architecture [63], The left side depicts a single perceptron whilst the right depicts a Multi-Layer Perceptron network. 22
2.12 Invoice depicting inconsistencies 23
2.13 LayoutLMv2 Architecture (source [158]). 28
2.14 Attention example for the total figure in an invoice. 30
2.15 Attention Usage 31
2.16 Encoder Self-Attention 32
2.17 Key, Value and Query creation via linear layers (source [45]). 33
2.18 Attention Score Formula visualized in the context of matrices (source [4]). 34
2.19 The input and output of the BERT token classification layer (source [35]). 38

3.1 Training and Inference Labels created for the system. 45
3.2 Annotating the dataset using UBIAI Annotation Tool. 46
3.3 Annotated example of a document. 47
3.4 UBIAI Export Options. 48
3.5 Training Dataframe. 49
3.6 Sanity Check of the Length of the Training Input, can also see the words as input. . . 51
3.7 Calling Training Function, output of training depicting training loss, validation loss, epoch number and F_1 score. 53
3.8 Formulas for scoring 54
3.9 Output of Textract API call, the boxes are the bounding boxes of the words in the invoice. 55
3.10 Example 1 of the inference, bounding boxes around the words with the corresponding labels inferred by the model. 57
3.11 Example 2 output of the inference, the boxes are the bounding boxes of the words in the invoice, with the corresponding labels inferred by the model. 57
3.12 Timing of the pipeline run time. 59

A.1 Self Attention Mechanism for Decoder (source [42]), depicting the resulting attention score. 63
A.2 Attention Mechanism for Encoder-Decoder in the decoder (source [42]). 64
A.3 Attention Mechanism for Encoder-Decoder (source [42]). Same input and target as Figure A.1. 64

Listings

2.1 GPT-3 Python code for human language translation as per [104]. 18

3.1 Bounding Box Normalization. 49
3.2 Data Config. 50
3.3 Dataset Preperation. 51
3.4 Model Creation. 52
3.5 Training Function. 52
3.6 Perperation for inference. 56
3.7 Inference. 56
3.8 Inference Output Post-processing. 56
3.9 Initializing the Server. 58
3.10 Inference Endpoint Defined. 58
3.11 Starting the Server. 59

Acknowledgements

I would like to thank my supervisor, Dr Kieran Murphy, for your fantastic help and excellent guidance throughout this process. It's been a pleasure working with you. I've learned an awful lot from you. This formatting is for you.
I would also like to thank Walid from UBIAI, without the generous student discount for the annotation software I would still be manually annotating documents now, and this would be a very short report.

Most of all I would like to thank my wife and kids for their support, inspiration and for putting up with me throughout this entire process. This report would be nonexistent without ye. This one's for you Becca.

Link to Demo Video

1 Introduction and Project Goals

1.1 Introduction

This report's structure will follow this style:

- 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.
- Implementation of the various system components from the start to the working prototype.
- Outline the challenges faced in the implementation of the system design.
- A conclusion is given including both technical and personal reflection.

1.2 Motivation

The motivation for this project comes from a number of major pain points of mine, from my side / previous career as a bar owner and manager.
This project focuses on building a system for use by bars / 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 vs 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 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 automated.
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.
This project will be used 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 potential pitfalls 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 semester one 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

[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 semester one report.

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 mocked. This is done through the main transactional 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[126] and Clover PoS [10] 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 would not be very difficult to implement to allow for nice visualisations of the data.

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 (Section 4.2).

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. 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 which in turn would reduce waste and cut costs.

1.5 Planning and Strategy

ClickUp [21] is used as the project management solution. 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:

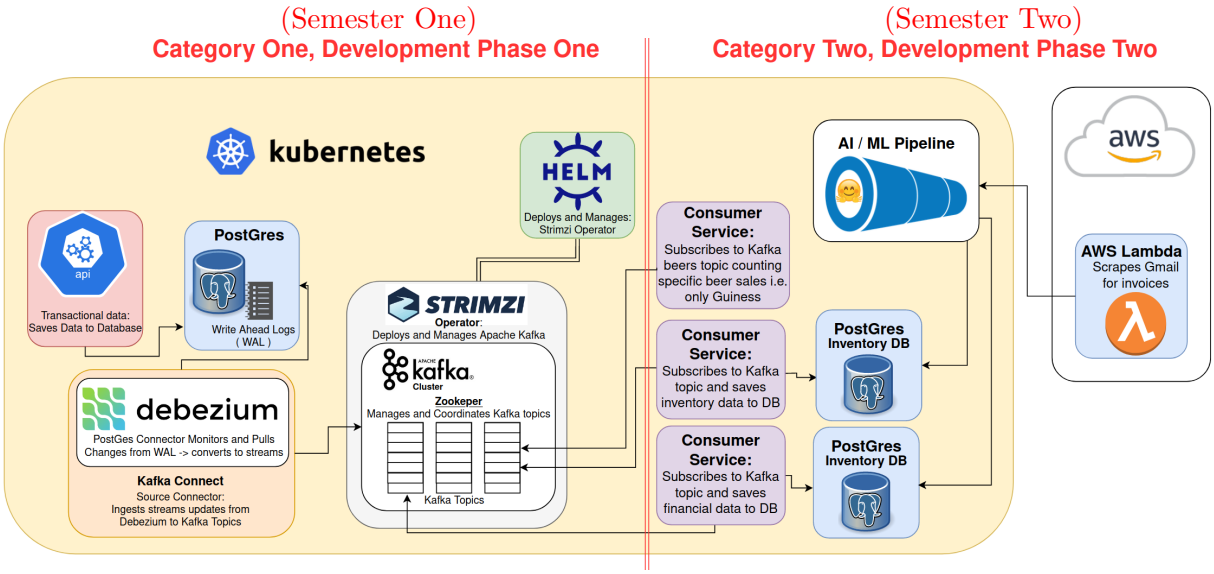

Figure 1.1: System Architecture.

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 (Section 2.1.1) 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:

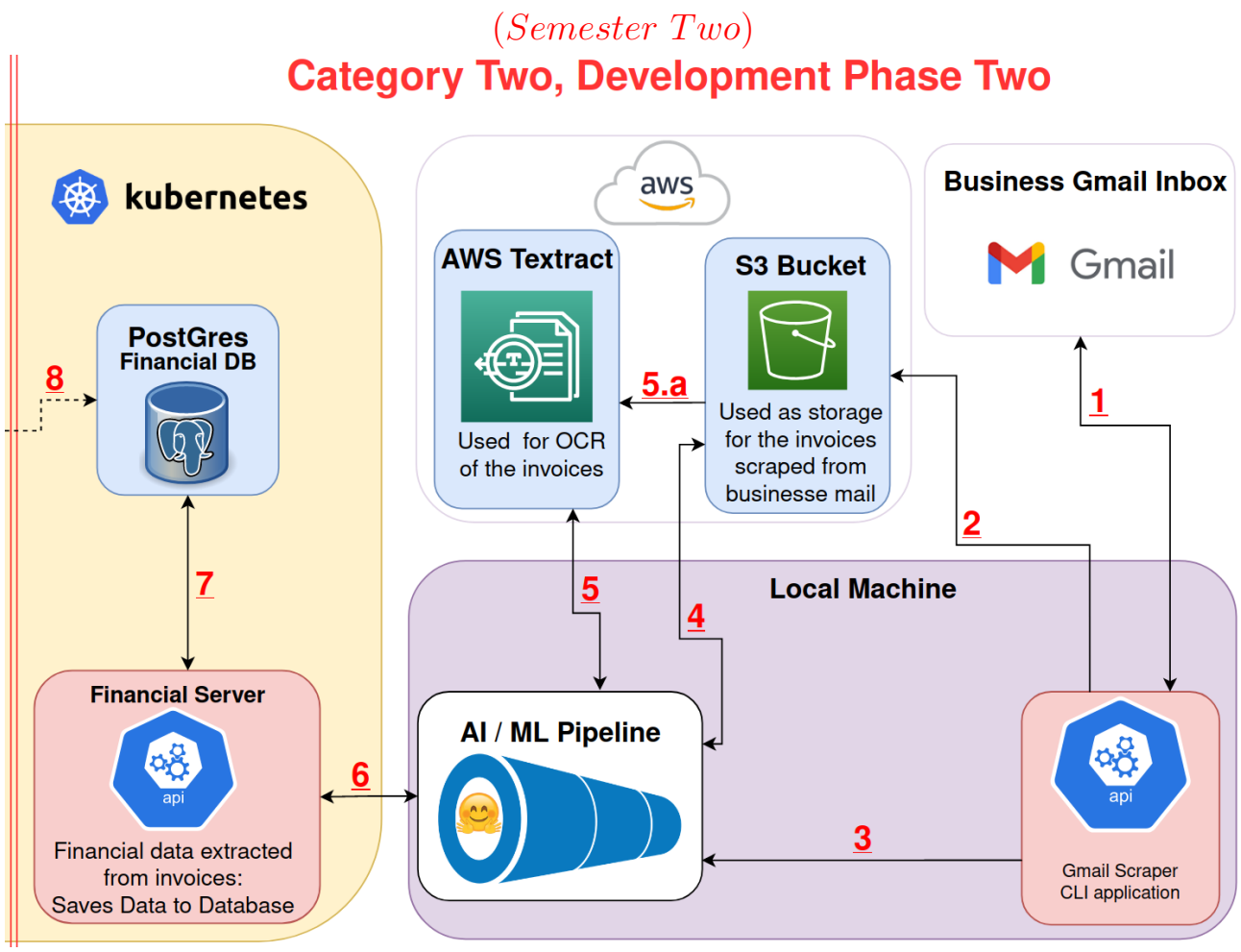

Figure 2.1: Semester 2 System Architecture.

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 `inbox` [120] 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 [125] *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 *Kubernetes-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 a Ngrok pod in the cluster to expose the Inference Server's endpoint to the internet.
- 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.

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

2.2 Technologies Considered for the Inference Server - AI / ML Pipeline

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 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:

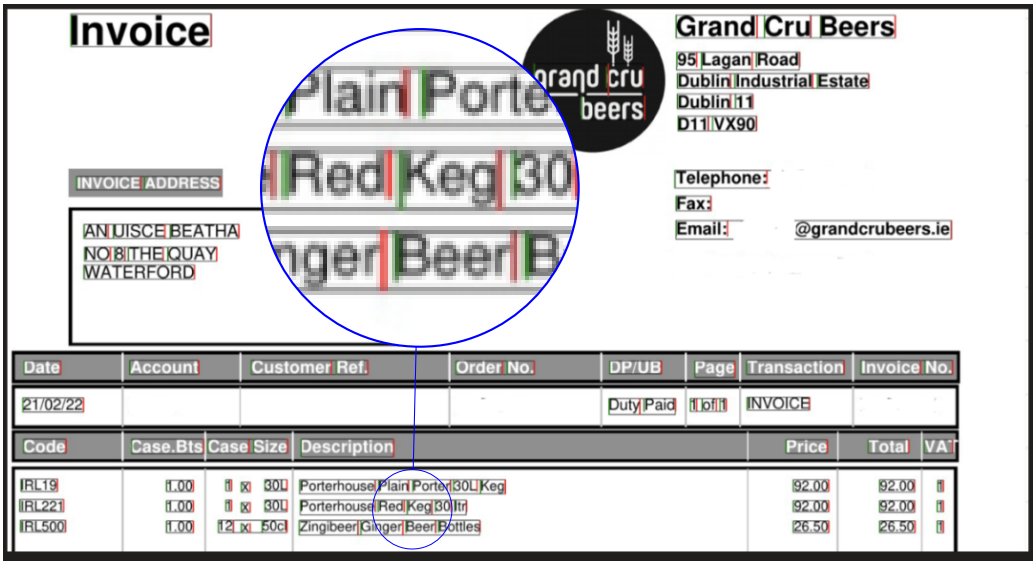

Figure 2.2: An example of the bounding boxes. The locations of each word / text are detected, and a bounding box is created around each piece of text. For clarity, this example has the bounding boxes drawn on. The start of each word starts with a green line and finishes with red.
Note: Some of the text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

2.2.2 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* [106]. This is driven by the Computer Vision Center [24], 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:

Figure 2.3: SROIE Dataset Example (source [118]).

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 [95]. 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 [118] 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. The dominance of the transformers based architecture in the leaderboard, suggests that these models are among the most accurate.

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.

2.2.3 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

[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.

implement a solution from scratch. It was during this research that the **LayoutLMv2** [158] model was discovered. This is a newly open sourced model, released toward the tail end of 2021 by Microsoft Azure AI [11], and it showed some great promise both in terms of performance and in terms of model size, due to the model utilizing *transfer learning* (Section 2.3.10). 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 [14] and other variations of BERT like LamBERT [87] and RoBERTa [119]. These models all share something in common, they are all built on the same *Transformers* architecture.

2.3 Transformers and the path to LayoutLMv2

There are not very many resources available describing how LayoutLMv2 works.[3] Being less than a year old and dwelling in a somewhat niche space may be some of the reasons for this. To understand the components which constitute the LayoutLMv2 model, research of other models (LayoutLMv2 ancestors) is carried out with information weaved together to ascertain a knowledge of the architecture and components.

LayoutLMv2 is a new model which has its ancestry in the original transformers model, but it is actually quite a hybrid of different models that have come along since the original transformers model was released [145]. It is not possible for me to document the entire history of the model and its ancestry in great detail, as there is simply too much information to aggregate. Therefore, I will mention the lineage here, Figure 2.5, and pick out the important features which give LayoutLMv2 its extraordinary power.

[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 [43]. Some other great articles on the topic, here [101], here [29] and here [4].

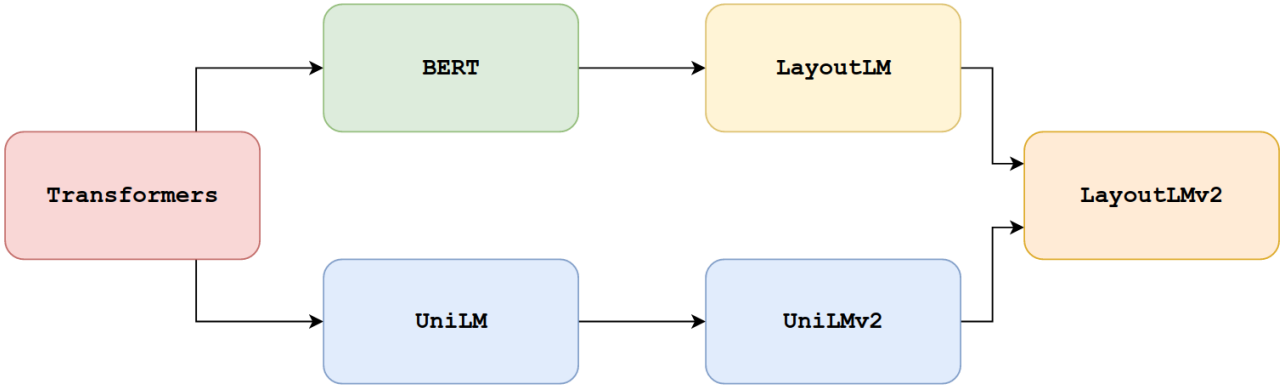

Figure 2.5: The LayoutLMv2 ancestry.

Note: The models depicted in Figure 2.5 and the order are taken from each paper which introduces each model.[4] The authors usually say that they have based the model in question on the architecture of another. Following this line of statements, Figure 2.5 has been produced. There are many other models which inspired components of each model, but the main ones upon which the architecture is based are the details that are captured.

The transformers architecture underpins all the models depicted in Figure 2.5 and introduces the important *Self-Attention* mechanism which powers all of the models above. As such, this report will outline the main concepts underpinning the transformers architecture with a particular focus on the differences and additions that make up the LayoutLMv2 model.

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* [145] 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 [14] and OpenAI's GPT series of models, the latest of which is the GPT-3 model [61]. 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 [61].

[4] The Transformer model is introduced by this paper [145], BERT by [35], UniLM by [40], UniLMv2 by [13], LayoutLM by [159] and LayoutLMv2 by [158].

Listing 2.1: GPT-3 Python code for human language translation as per [104].

```
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 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[5].

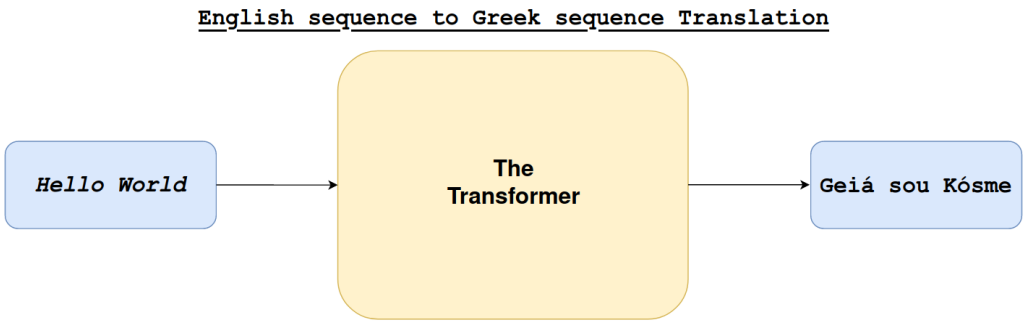

Figure 2.6: Basic Transformer Sequence to Sequence example.

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

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.

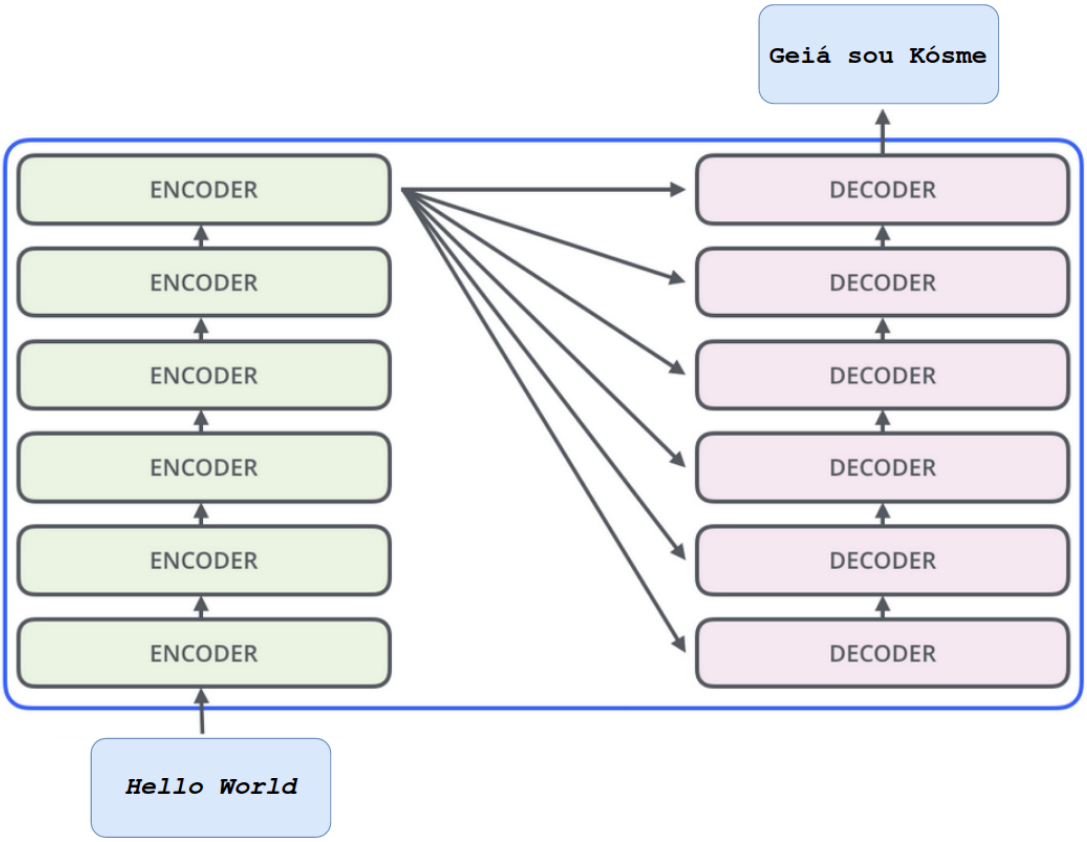

Figure 2.7: Transformer Stacked Encoder and Decoder example (source [4]).

All the encoders are identical in structure, but they have their own weights associated to them. At first these weights are set at random, but they are altered through the back-propagation of the training phase.

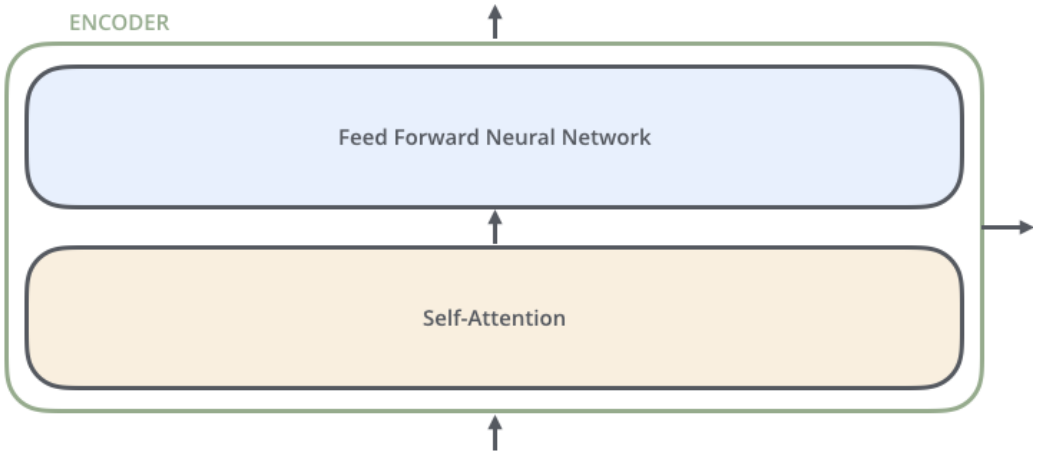

Figure 2.8: Transformer Encoder architecture example (source [4]).

The Self-Attention layer will be described in more detail shortly (Section 2.3.7) but for now lets look at the decoder architecture:

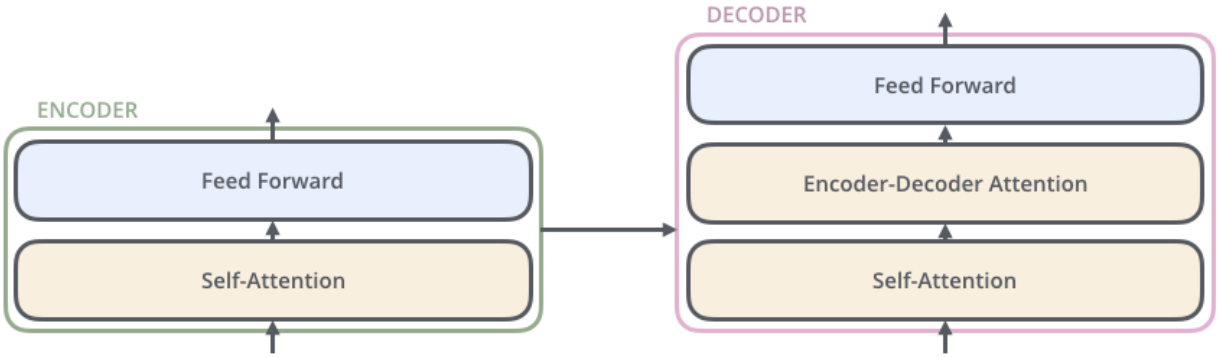

Figure 2.9: Transformer Decoder architecture example (source [4]).

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. It does this by masking the positions in the sequence that are not in current focus.

2.3.2 Transformers Architecture

The level of detail is about to increase, as to is the complexity, but I've tried to keep the detail to a minimum whilst covering the concepts needed to understand how transformers work.
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 Transformers model. To start Figure 2.10 shows the inner workings of the encoder on the left, and then the decoder on the right.

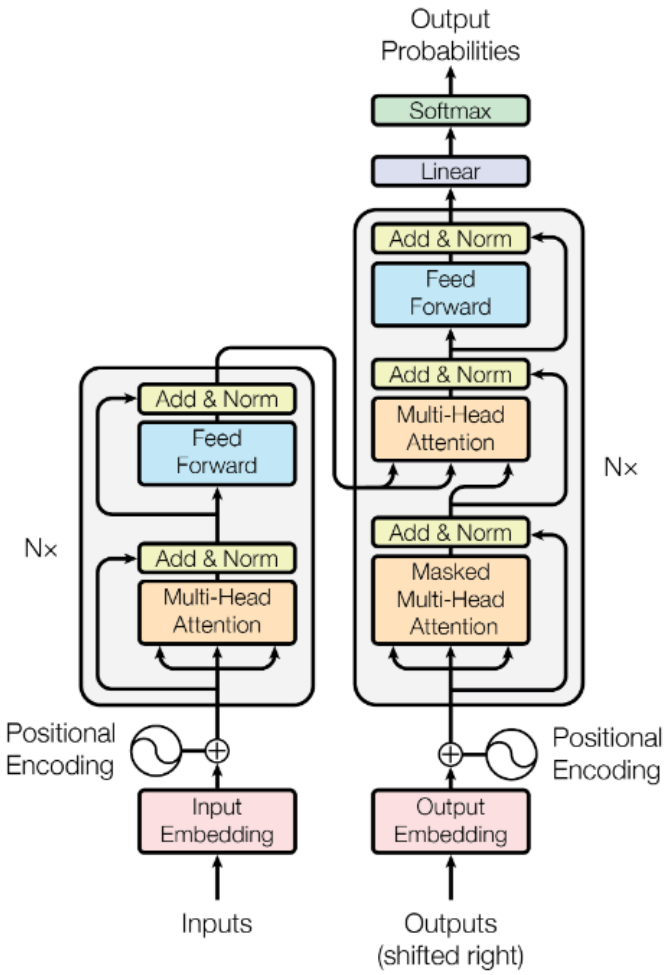

Figure 2.10: Original Transformers architecture [145], akin to the previous example the encoder stack is situated on the left whilst the decoder is situated on the right.

- **Black Arrows**: In Figure 2.10, the black arrows depict the dataflow.
- **Input Embeddings**: As can be seen at the bottom of the Figure 2.10, 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' [147] to

normalise.

- **Multi-Head Attention**: The `Multi-Head Attention` is the heart of the transformer. It is essentially numerous self attention layers stacked together (Section 2.3.7).
- **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 only forwards. The goal of the feed forward network is to approximate some function of the input.

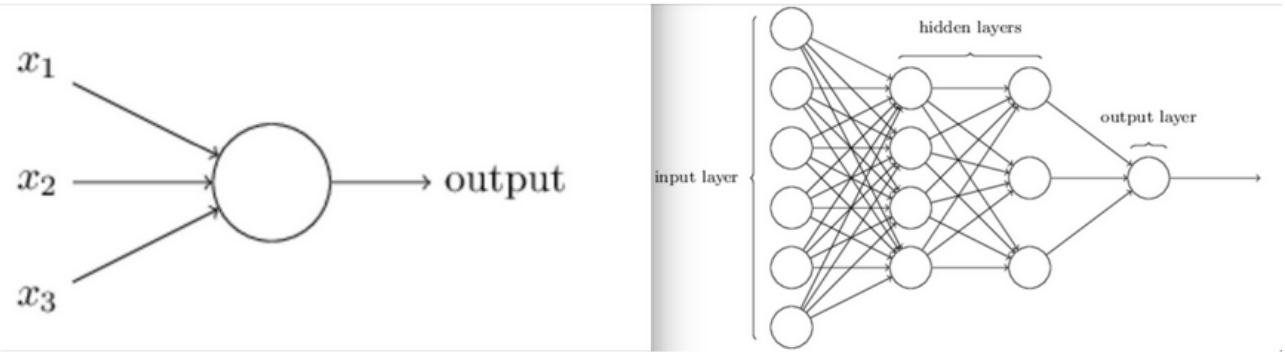

Figure 2.11: Classic Feed Forward Neural network layer architecture [63], The left side depicts a single perceptron whilst the right depicts a Multi-Layer Perceptron network.

They are also known as a *Multi-Layer Perceptron (MLP)*. One of the first and most popular deep learning models [55].

- **Softmax**: The Softmax function is used to compress the outputs to form a number in the range 0 - 1, whose values all add up to 1. These values can now be thought of as probabilities, these are the `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 is prepended to the 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.

This idea of context in a sentence as above can be thought of as the different geographical location of text blocks on a document, in particular an invoice. To be able to extract the relevant meaning, the context is vital. As aforementioned, invoices have no set standard and can vary significantly. During the creation of the dataset for this project an invoice was discovered with two varying styles of expression for two vary similar products. It is one of three to four test invoices used during system development. A brief look through other documents corroborated that this type of finding is quite common across a multitude of different suppliers invoices. This is an unfortunate trend.

Both highlighted rows are identical in every field, except for the description. The beers are even made by the same company, yet the description of the quantity was, presumably, left to be filled in by two different humans with no structure or naming standards. With such inconsistencies in the data, the task of learning becomes more difficult and the concept of context becomes extremely important.

Description	Price	Total	VAT
Porterhouse Plain Porter 30L Keg	92.00	92.00	1
Porterhouse Red Keg 30 ltr	92.00	92.00	1
Zingibeer Ginger Beer Bottles	26.50	26.50	1

Figure 2.12: This is a real invoice depicting inconsistencies in the description, highlighted in green.

For optimal performance, a model would need to have a way of associating different parts of an invoice with similar parts of other invoices, throughout

different layouts to establish a pattern. The model would need to pay particular attention to the context of the data.

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 main text model 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 [154].

- **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 proximity to and 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 words that occurr less frequently 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 [129]:

    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 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-modality* 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 original input token.

In the case of LayoutLMv2, the multi-modality is the combination of the token and the position the initial training phase. This is known as the *Layout Embedding* and refers to the geometrical position of the token in the document.[6] 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, in LayoutLMv2 segment embeddings are present. These have been picked up from BERT and are not present in the original Transformers model. They are used to distinguish different text segments. These segments can be thought of as a block of text t long. Each token that is tokenized from the OCR input is assigned to a segment.

[6] The positional encoding is done in a two step process using a *sin* for the first step and a *cos* for the second stop. More details on this can be found here [44].

After this, some special tokens are inserted, along with the text tokens. Each sequence is prepended with a `[CLS]`, token whilst the end of each segment is denoted with a `[SEP]`. If the sequence is shorter than the max sequence length, then a number of `[PAD]` tokens are inserted to bring the minimum sequence length. We will see the use cases for these tokens later.

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 [88]. 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* [54]. LayoutMLv2 (the Hugging Face version) uses Detectron2 [53] 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

These state-of-the-art models must not be treated as back boxes. We have already learned that our own biases get baked into models, when these models are deployed in situations where their consequences has proven to be huge[7], every care should be taken to ensure that the models output can be questioned and the factors driving the decision should be available. The models and the people who use them should be accountable for their actions. It is for this reason that the architecture will be explained in as fine a detail as can be.

Having gained a greater understanding of the input for the LayoutLMv2 model we observe the architecture[8].

[7] As is the case with the *Correctional Offender Management Profiling for Alternative Sanctions (COMPAS)* model [116], which was / is used in multiple states in the USA. This model has been shown to have an inherit racial bias and recommend African Americans have a higher chance of re-offending than whites and therefore should get stronger sentences. This model does not take into account generational racism along with the difficulty an African American from a disadvantaged area has in life.

[8] The difference between the original LayoutLM and LayoutLMv2 is that the visual and positional embedding occurs only during the fine-tuning phase for the original LayoutLM, whereas, LayoutLMv2 includes this data during the initial training phase too.

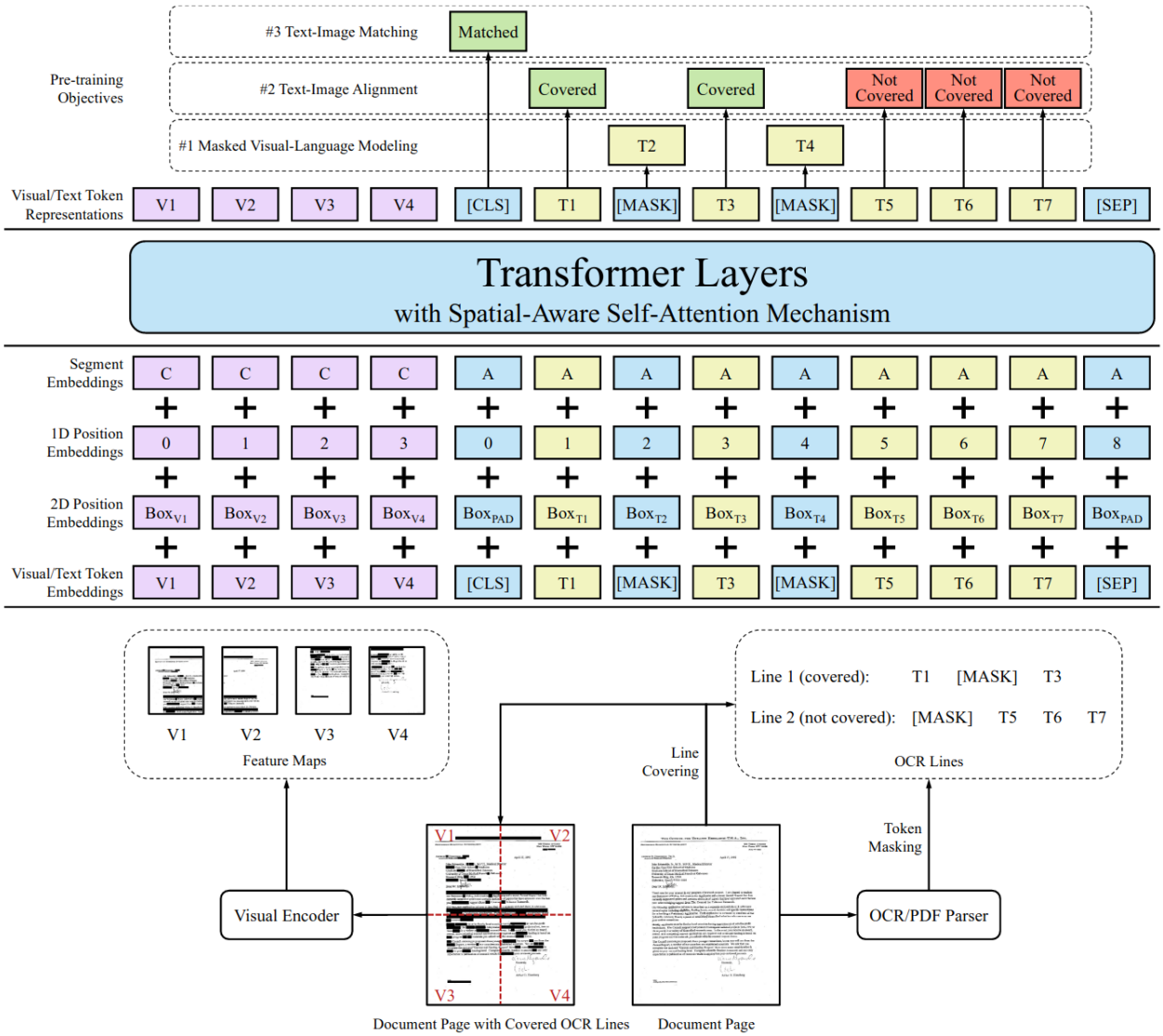

Figure 2.13: LayoutLMv2 Architecture (source [158]).

Observing Figure 2.13, there are still many components which we have not dived into. We have defined the layers which make up the embedding input and now, having covered the basic transformer layers architecture, our attention can turn to a deep dive into the *Transformers Layers*, from Figure 2.13, and in particular the self attention mechanism.

2.3.7 Attention, Is All You Need

As humans, we are subjected to an enormous amount of information every second we are conscious. Smells, sounds, visuals, interactions with objects, environmental factors like heat or rain. We obviously have a limited capacity to store this information, so how does the brain decide which bits of information should be stored as a memory and which bits can be discarded?

Attention, appears to be a key factor of what constitutes criteria for memory

creation. We can all remember important life events, celebrations and sadness. As we engage in the moment and the moment draws our full attention, we have a higher chance of remembering the moment. Most people can remember details from the far past of events like weddings, births, etc., yet we collectively struggle to find our sets of keys or phones.

Previous state-of-the-art models have had limited *memory*, how far back in time (or how far back a sequence of tokens). RNNs and CNNs have a limited capacity to remember relationships between tokens further back in a sequence. Long-Short-Term-Memory models improve upon RNNs and CNNs, but they too suffer from the same problem.

Just as attention is a key factor of memory for humans, can the same be said for ML models?

Attention is the key component of the transformer architecture[9]. 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.

We will try to get an intuition for attention via the lens of our use case, trying to predict the labels for text in an invoice, we'll focus on what happens per individual item of text and try to develop an intuition as to how attention is used in transformers based models.
As an example we will use the *total figure* value, which is always on an invoice. When the model tries to classify the total figure, let's call this the Query, with the correct label, the model looks at every other text block in the invoice and tries to determine if there is any similarity or closeness between every *i-th* word in the invoice and the Query. This method of paying attention to every word, *i*, with respect to itself (the Query) is why it is called ***self**-attention*.
The way that the similarity between words is determined, is by way of a score that every *i'th* word has in relation to the Query.
It tries to use the context of the surrounding text to determine which label should be predicted.

[9] Attention is also one of the more difficult concepts to grasp. These resources played a pivotal role in acquiring an understanding [42], here [45], here [28]. and here [4].

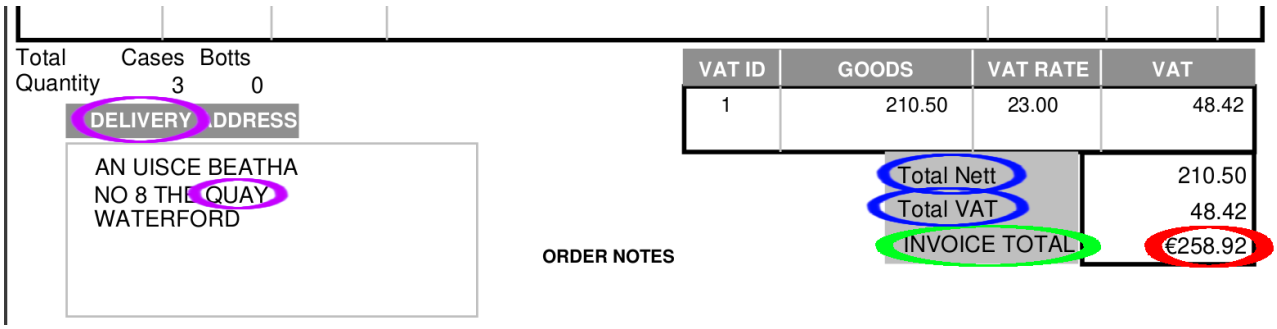

Figure 2.14: Attention example for the total figure in an invoice.

From Figure 2.14, the Query or the desired *total figure* is circled in red.

- The words 'INVOICE TOTAL' are circled in green, these words would be given a very high score as this combination of words appears quite frequently and in a similar geometric position relational to the Query. If one was looking for the 'total value' then looking for 'INVOICE TOTAL' is a good place to start.
- The words circled in blue, would be given high scores too, but not as high as the words in green. That makes sense, as words like these, 'total', 'VAT' and 'Nett', can often be found in close to proximity to the Query.
- The purple words are examples of words that will have a low score towards the Query. They are further away geometrically and the words are not likely to be repeated in a similar position in other invoices (from a different supplier) that the model has seen for training.

In training the model has access to the entire annotated invoice.
At the start the training, model will make a pretty random prediction as it hasn't learned anything yet. When the time comes to evaluate its prediction, if the prediction is wrong the model may determine that it has paid too much attention to the purple encircled words, when those words brought little value to making an accurate prediction, the scores for these words, in relation to the Query, will be reduced. The text enclosed in the blue and green circles will see their scores increase.
Therefore, in the next training round or for inference the model will have a better understanding, by virtue of these scores, of the text to pay attention to when predicting the label / classifying the token. The multiple epochs of training allows these scores to be compounded and the model (should) get a little better each time.

The concept of *Attention* is used in the Transformers model in three places as per Figure 2.15:

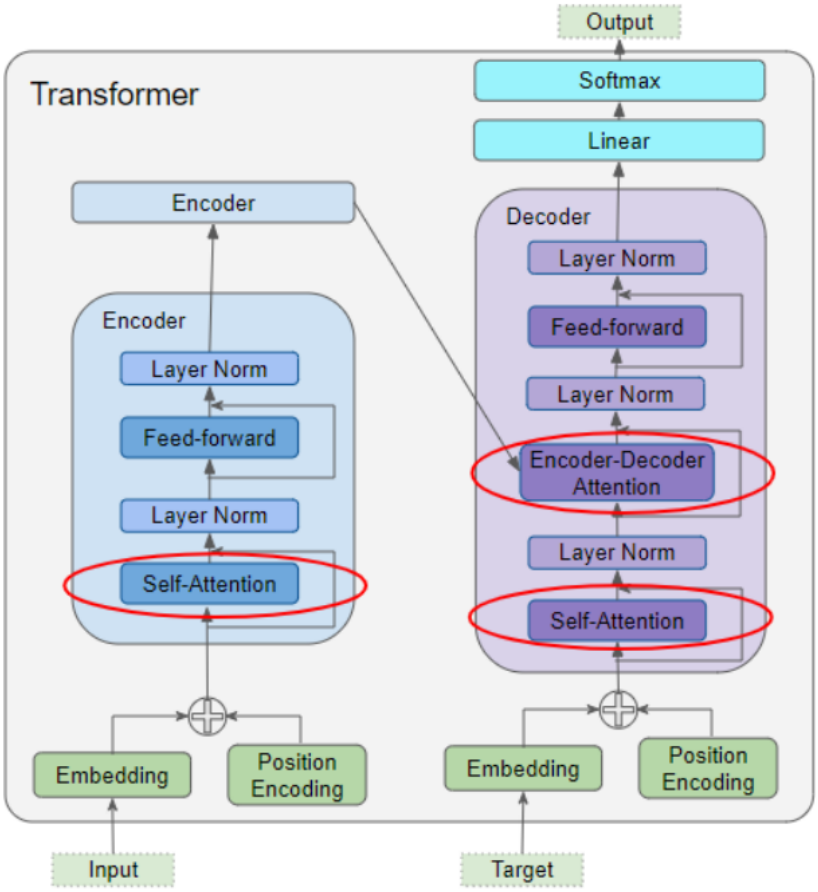

Figure 2.15: Attention is used in three places from here [45].

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

But how are the scores calculated? Well these *Attention Scores* are calculated in the *Attention Layers* of the model. 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. The concept of the Q, K and V can be thought of as a retrieval process, explained excellently here [50].

As an example, when one types a **query** to search for a recipe for cheesecake on Google search, the search engine maps the query against a set of **keys** — article title, description, words in articles (cheese, cake), ratings, etc., associated with candidate websites. The search engine then returns a list of websites (**value**) which match the initial query.

We will look at the self-attention layer in the encoder.

2.3.7.1 Encoder

The Encoder is key to LayoutLMv2. Another feature inherited from BERT, is that there are no Decoder layers present in the architecture. There are only stacked encoders. We will see shortly how this operates.

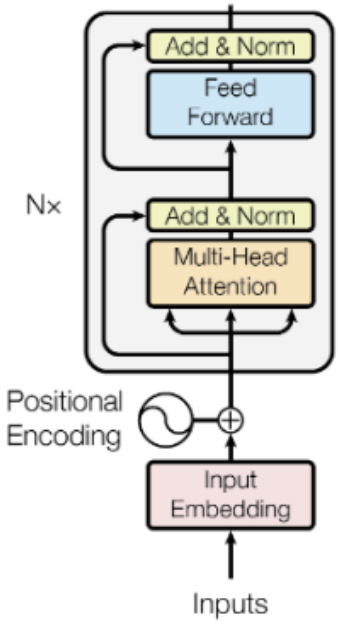

Figure 2.16: Encoder Self-Attention (source [145]).

What does the Encoder do? What is its role? At a high level the Encoder takes in an input sequence and maps it to an abstract continuous representation that contains all the learned information for that sequence. It does this in a series of steps.

2.3.7.2 Query, Key & Value

The first step is to create a *Query*, *Key* and a *Value* matrix for each word in the input sequence. This is done by feeding the input sequence into three distinct fully connected linear layers. The Q, K and V matrices are multiplied by their respective weights matrices.

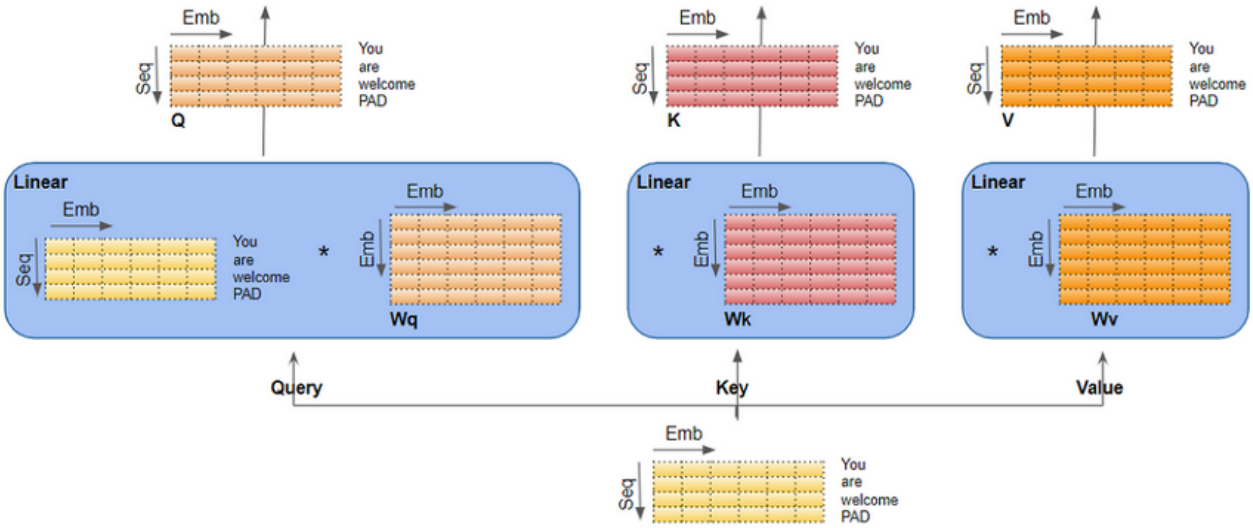

Figure 2.17: Key, Value and Query creation via linear layers (source [45]).

2.3.7.3 Attention Score

The next stop is to calculate the attention scores. This is the formula used:

$$Z = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- Q is the query matrix.
- K is the key matrix.
- V is the value matrix.
- QK^T is the dot product of the query matrix and the transpose of the key matrix.
- d_k is the dimensionality of the key matrix.

A nice way to visualise this can be seen in the figure fig. 2.18.

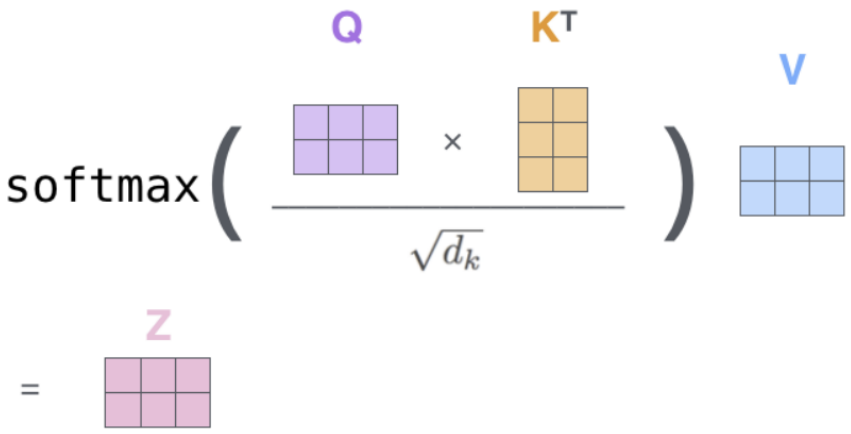

Figure 2.18: Attention Score Formula visualized in the context of matrices (source [4]).

We will now walk through the formula step by step:

1. The Query and the Key undergo a dot product, and the result is a score matrix. This matrix determines the amount of focus each word should put on another word. The higher the score, the higher the focus.

   From our invoice example, how much focus should the model place on the 'INVOICE TOTAL' words in relation to determining the label for the total invoice figure.

2. As these numbers may be quite high, they need to be scaled down. This is done by dividing by QT^T by the square root of the dimension of the key matrix. This step also increases efficiency.

3. The resulting matrix goes through a softmax function. This transforms arbitrary input values so that the output is in the range $0 \ldots 1$. Effectively turning the values to probabilities.
   Using a softmax function has two advantages:
   - It is computationally safer, and it leads to more robust algorithm, to compute.
   - It also has a conceptual advantage since we can treat the output as probabilities.

   These are known as the attention weights.

4. The attention weights are then multiplied by the value matrix. The resulting matrix is the output. The higher softmax scores from the previous step will keep the important words and the lower softmax scores will drown out the less relevant words.

5. The output layer, Z, then goes through a linear layer to process. Z is our attention score matrix.

To recap attention, at a very high level attention acts as a windowing processes facilitating the ability of the model to focus on a subset of the given data. The attention score is a metric of relevance / proximity between two words.

As this is a multi-headed attention module the input, Q, K and V are split up before they go through the self-attention module, and the attention weights are calculated for each head[10]. Within the attention layer multiple attention scores are calculated in parallel for performance reasons. They are logical partitioned. i.e., they do not share data and run separately and run on separate CPU /GPU cores.
There is a further reason. In theory, each head should learn something different. As each set of weights starts from a different randomized matrix, the use of the loss function[11] should each explore a different space. The outputs of each head are then concatenated together to form the final output, which goes through the last linear layer. This gives the model more representation power.

As we now have our output from the attention layer, the next step is a Residual Connection. This can be seen in the Figure 2.16, as the arrow which starts before the attention layer, skips it and goes straight to the add norm. This process sees the original input added to the output of the attention layer. The resulting matrix is then 'layer normalized'. Layer normalization normalizes each feature to a mean of zero and unit variance, this helps the model network produce better, smoother gradients in the next weight update [147] which also help to reduce the training time. This is the **Add & Norm** component.

The normalized output is then fed forward into the **Feed Forward** component, made up of two linear translation layers. Residual connections follow again as the output from the Feed Forward is added and normalized to the normalized output of the attention layer.
These residual connections help the network train by allowing gradients to flow through the network.

[10] The Transformers architecture has 8 attention heads whilst BERT has variations with 12 or 24 attention heads.
[11] A loss function is used in the process of the model changing weights, to get a more accurate outcome, during training. The model makes predictions and the error for those predictions is calculated. It is this *loss* that we are trying to minimize. It can be seen as a search / optimization problem. We need to find values for weights which reduce the loss, therefore, giving more accurate prediction.

Cross-entropy loss is used in the transformers models. With entropy being defined from the field of information Theory [18], Cross entropy loss multiplies the true value by the log of its predicted values, it tries to penalise a classifier more when the classifier misclassifies with a high probability. That is if one prediction is 100 and the other is 10, if the true answer is 2 then the classifier which predicted the 100 will be penalised more than the classifier which predicted 10.

2.3.8 LayoutLMv2 Pre-Training Tasks

Note: As mentioned the LayoutLMv2 architecture does not have any decoders. It uses the *pre-training tasks* to both train and predict.

The pre-training tasks used in the LayoutLMv2 architecture are heavily influenced from BERT [35]. With some additional modifications to allow for the positional encodings and visual information to be included. There are three pre-training tasks, of which we are mostly concerned with the first two. First 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**]: Is used to initialize the sequence and is used as the token or label to predict, it is always prepended to the input sequence. It stands for 'classification'.
  This is the output that will be put through a softmax function, the probabilities for each word in vocabulary are calculated. The highest probability becomes the prediction.

  In our use case for inference. The `[CLS]` will be the label that the model predicts. For each input sequence the model looks 'around' the invoice. It notices the relationship between every word in the sequence and the invoice. The model then uses the highest attention score as the output.
- [**MASK**]: To mask tokens for pre-training and inference.
- [**SEP**]: Separates questions and answers [35].

2.3.8.1 Masked Visual-Language Modeling (MVLM)

During training an input sequence is fed into the model. Some tokens in the input sequence are masked. The model tries to predict the masked tokens and fill in the sequence using the context of every token with every other token. We calculate the Cross-Entropy loss, adjust the weights and the cycle continues until epochs are complete.
If we applied no masking then the model would predict the entire sequence with 100% accuracy as it has its entire input sequence.

This is achieved by masking 15% of the input sequence. Of this 15% [35]:

1. 80% of the tokens are masked with the `[MASK]` token.
2. A further 10% are given a new token at random from the entire model vocabulary. This allows the model to deal with words that it hasn't seen, in the input sequence. It also allows the model to learn the context of the tokens with respect to all tokens.
3. 10% are left back as is.

It is important to note that the layout information (positional encodings) remain unchanged. This step is crucial as it allows the model to learn from the positional elements.
The image regions are masked before they are fed to the visual encoder to avoid visual leakage.

2.3.8.2 Text-Image Alignment (TIA)

This pre-training task is a novel task, that the writers proposed in the paper [158]. This task aids the model to learn the spatial location context between the image and the bounding boxes.
In this task some of the images are randomly selected, and their image regions are covered on the document image. During pre-training, a classification layer is built above the encoder outputs. This layer predicts a label for each text token depending on whether it is covered, i.e., [Covered] or [Not Covered][12], and computes the binary cross-entropy loss.

2.3.8.3 Text-Image Matching (TIM)

Text-Image Matching is applied to help the model learn relationships between the text and the document image. The output representation from the `[CLS]` token is fed into a classifier. This prediction is used to determine if the text and the image come from the same document.

The authors of the LayoutLMv2 paper tested differing versions of the model using all combinations of pre-training tasks. The combination of combining these three tasks has given the best results.

2.3.9 Fine-Tuning

In Section 2.3.10, *Transfer Learning* is discussed where the time-consuming and expensive training of large models is broken in two stages: (expensive)

[12]The paper authors use the `[Covered]` or `[Not Covered]` to avoid confusion with the MLVM `[MASK]` tokens.

pre-training on large dataset to gain generalised language capabilities and (much cheaper) fine-tuning.
The pre-training tasks are used to train the model to gain a generalized understanding of language and context. That is, the model has more capacity to learn specifics needed for a task.

Then for the fine-tuning tasks, a few task-specific layers are added. In our use case, this is token classification. The authors of the LayoutLMv2 paper do not document the process for token classification, using BERT as the example we can get a rough visualisation of the inputs and outputs during this task.

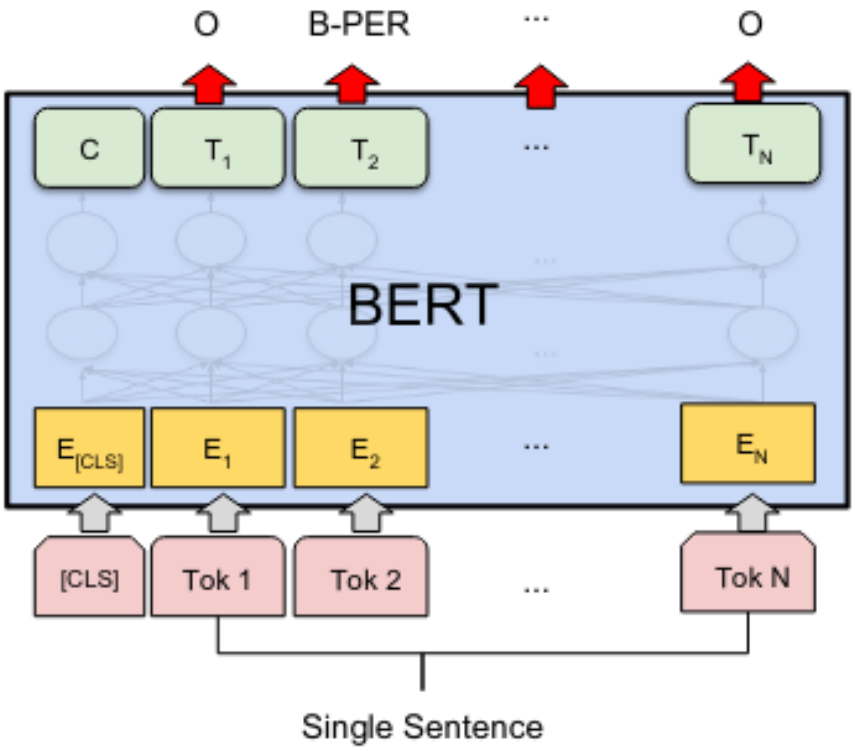

Figure 2.19: The input and output of the BERT token classification layer (source [35]).

The `[CLS]` token is inserted to denote the start of the sequence, the labels and their positions are also passed into the model. The model learns which labels should be predicted for each token, through the back propagation of the weights. The model is given a few new layers that sit on top of the hidden layer text which specifies for which kind of fine-tune task the model is required to learn.

2.3.10 Transfer Learning

The advancement in machine learning in recent years has been nothing short of remarkable. We have some fantastic models with near human level performance at some tasks, but these models take a huge amount of resources to train. From time training to the huge amount of compute resources and by

virtue, energy. If every one that uses a model has to start from scratch then that can only be viewed as a devastating waste of resources.

Transfer learning is a technique which allows a model to be trained on a dataset or numerous datasets to be used as its base. The initial training of these models still requires a huge amount of data and compute power along with expertise in the domain, but once the model has been *pre-trained* it can be used as a starting point for model on a new task. The pre-training phase usually includes similar but not identical data sets to allow for a model that is more generalized, that is more readily applicable to a variety of different tasks.

In essence, transfer learning is a more human way of dealing with knowledge. It allows for knowledge that has been gained from a particular set of tasks to be stored, distributed and applied to other tasks. When a child learns the physics of real world interactions by walking, playing, observing how objects move through space, etc., it uses this base knowledge to shortcut the learning process when learning new but related things.

The following are some examples are intended to give the reader an appreciation of the complexities and costs associated with training modern large models:

- The *BigScience Large Language Model* [16] 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 [17]. The data set contains 341.6 billion tokens. which is approximately 1.5 TB of data.
  That is an astonishing amount of compute, data and engineering expertise.
- The GPT3 model costs between 10 and 20 million dollars to train [2].
- Google's DeepMind model that was used to learn the Chinese board game Go, is estimated to have cost a staggering 35 million dollars to train [3].
- Microsoft and Nvidia's Megatron-Turing language model, one of the largest of its kind, is estimated to cost in the millions of dollars to training [96].

These examples are some of the more extreme, but these 'more extreme' examples are becoming more and more common place.
These examples show the scale of what the initial base training can entail.

In the case of LayoutLMv2, the base model has been trained on six different datasets, from the visually-rich document understanding area including the

aforementioned, SROIE dataset and others including the FUNSD dataset [57] and the CORD dataset [27].

These models are then *fine-tuned* on custom datasets for the specific task at hand. For example, LayoutLMv2 in this instance is fine-tuned on the custom invoice dataset which is optimized for invoices, but the same base model would also be used, in a specialised fine-tuned version for receipts and other documents which see a drastic shift in document layout.
A model optimized for invoices may still work for some receipts, but to obtain state-of-the-art results, the model should be fine-tuned on the dataset for which it will be used to infer.
A downstream task[13] to first classify what structure the document is and then use a trained model that is optimized for that document structure could be implemented to further increase the accuracy of the system.

The fine-tuning phase takes a lot less resources. It can be done on most mid-range machines. The addition of a GPU greatly increases the efficacy of the training, if utilized.

This idea of transfer learning cuts out the large training time and allows most anyone to use and fine-tune a model to their specific needs. It allows for the implementation of models which would traditionally have been cost-prohibitive to implement. Transfer Learning should lead to surge in innovation as more people are exposed to more powerful models. The budget for training LayoutLMv2 has not been disclosed, but preliminary research indicates that the cost of training LayoutLMv2 runs into the thousands, maybe more.

Some libraries now exist which contain not only the pretrained models, but datasets of varying sizes and forms, along with other components to aid in the pre-processing of the data to be used in the fine-tune training model phase. The *transformers library* by Hugging Face [71] is one such library.

[13] A downstream task can be thought of as a models capabilities or different implementations, for example, LayoutLMv2 can be used for token classification (like this project) but it can also be used for *Sequence Classification* and *Question Answering* tasks [89].

2.4 Hugging Face

Hugging Face [71] 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 [130]. Here is what they say about themselves:

> *Hugging Face Transformers provides APIs to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you time from training a model from scratch. The models can be used across different modalities such as: text, audio, images, video, and multimodal.*
>
> — *Hugging Face Team*

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 base models available in over 100 languages [73]. The models are mostly of based on the transformers architecture.
Hugging Face also provide *The Model Hub* [98], this works akin to a code repository for models, once a model is uploaded to the hub, it can be made public or private. Hugging Face also provide a large variety of open-source datasets [72] which can be used to train models.
The team have a massive amount of tutorials and guides on model architecture available along with a large number of articles, blogs and open-sourced code for model evaluation, training and inference.

They make money via their Inference API where there are a plethora of models ready behind endpoints for a variety of different use cases. They also charge for support and for deployment and training of models on their servers.

The Transformers library is a very powerful library, it looks to cut away at the *boiler-plate* code required to implement a model.
The library supports the PyTorch, TensorFlow and JAX frameworks. It even supports cross framework deployment so a model may be trained using PyTorch and deployed using TensorFlow or JAX for inference. All cross framework integrations appear to be supported [138].
The library also contains some great functions for data pre-processing:

- **Tokenizer**: The pre-processing step to convert text into a sequence of tokens.

- **FeatureExtractor**: This function resizes images for use in models and can even be used with the OCR engine enabled (PyTesseract). Although this method is quite basic and is not suitable for more complex tasks such as the token classification task in this project.
- **Processor**: This is a model specific function which wraps the tokenizer and the feature extractor in to a single function for ease of use.

The tokenizer is utilized in this project. The transformers library along with the available resources on the Hugging Face website were used extensively during the research and development phases of the project.

3 Implementation

3.1 Overview

There are a number of steps that are required to be implemented in order for the system to be able to return the desired output. These are the steps of development.

- **Dataset Preparation**: The dataset is prepared for the system to work with.
- **LayoutLMv2 Model Training**: The model is trained on the dataset. The model is then saved for use with inference.
- **Data Pre-Processing**: The dataset is pre-processed to be ready for the system to work with. This step includes, Text Localisation and OCR (these are now API calls to AWS) and subsequent data manipulation to ready the data for the inference model.
- **LayoutLMv2 Model Inference**: The model is used for inference. This means the model is used to process invoices, and classify which labels (if any) should each text block in the invoice be labelled with.
- **Data Post-Processing**: The data is post-processed to be ready to be saved into the database.
- **Deployment**: This step involves deploying a model behind a REST API (Flask Server).
- **Financial DB Server**: This step involves creating another REST API (Flask Server) to receive pipeline output and save to the Postgresql financial database utilizing SQLAlchemey as the ORM.

The inference model is the heart of this project. We start with a brief reminder of what the inference model attempts to do. At a very basic level the inference 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 and other input embeddings. 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.

3.2 Dataset Preparation

To fine-tune 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 to ensure the model, when trained and operational, is actually useful. If the annotations in the training set are not accurate the model will never be able to infer the correct labels.

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 [148], VGG Image Annotator [46] and Label Studio [86], 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 [92] and UBIAI [47], 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 [48], [49] and [5], 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.

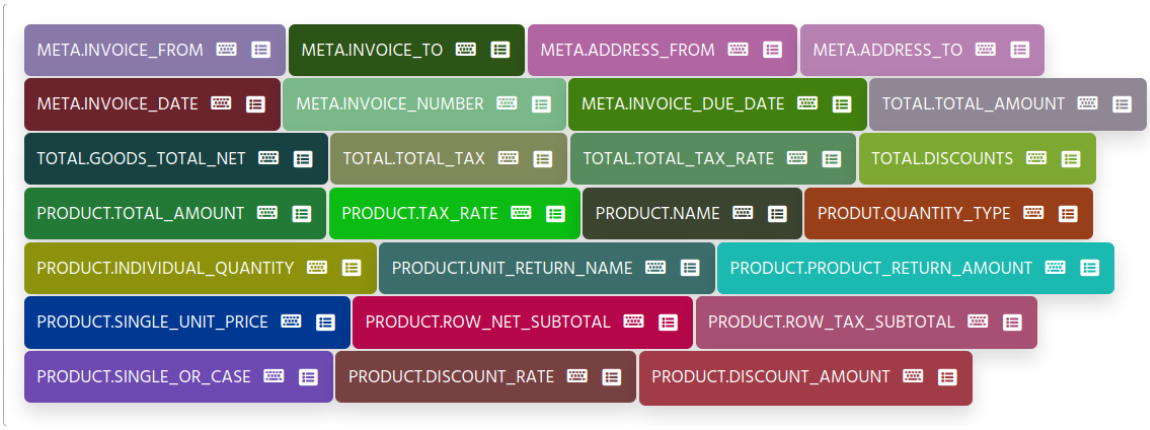

Figure 3.1: Training and Inference Labels created for the system.

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.

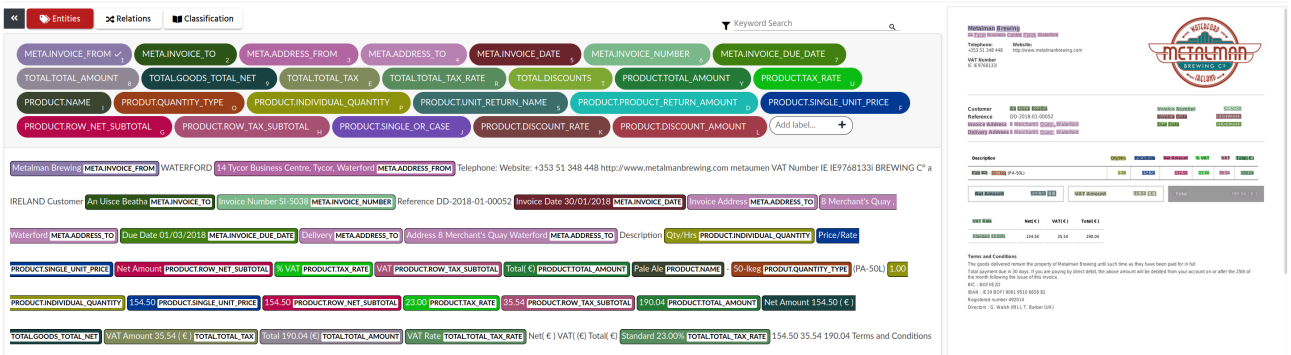

Figure 3.2: Annotating the dataset using UBIAI Annotation Tool.

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 approximation. 50 documents LayoutLMv2 was able to return decent results. This finding was corroborated by Walid.

A further example of an annotated document is shown below.

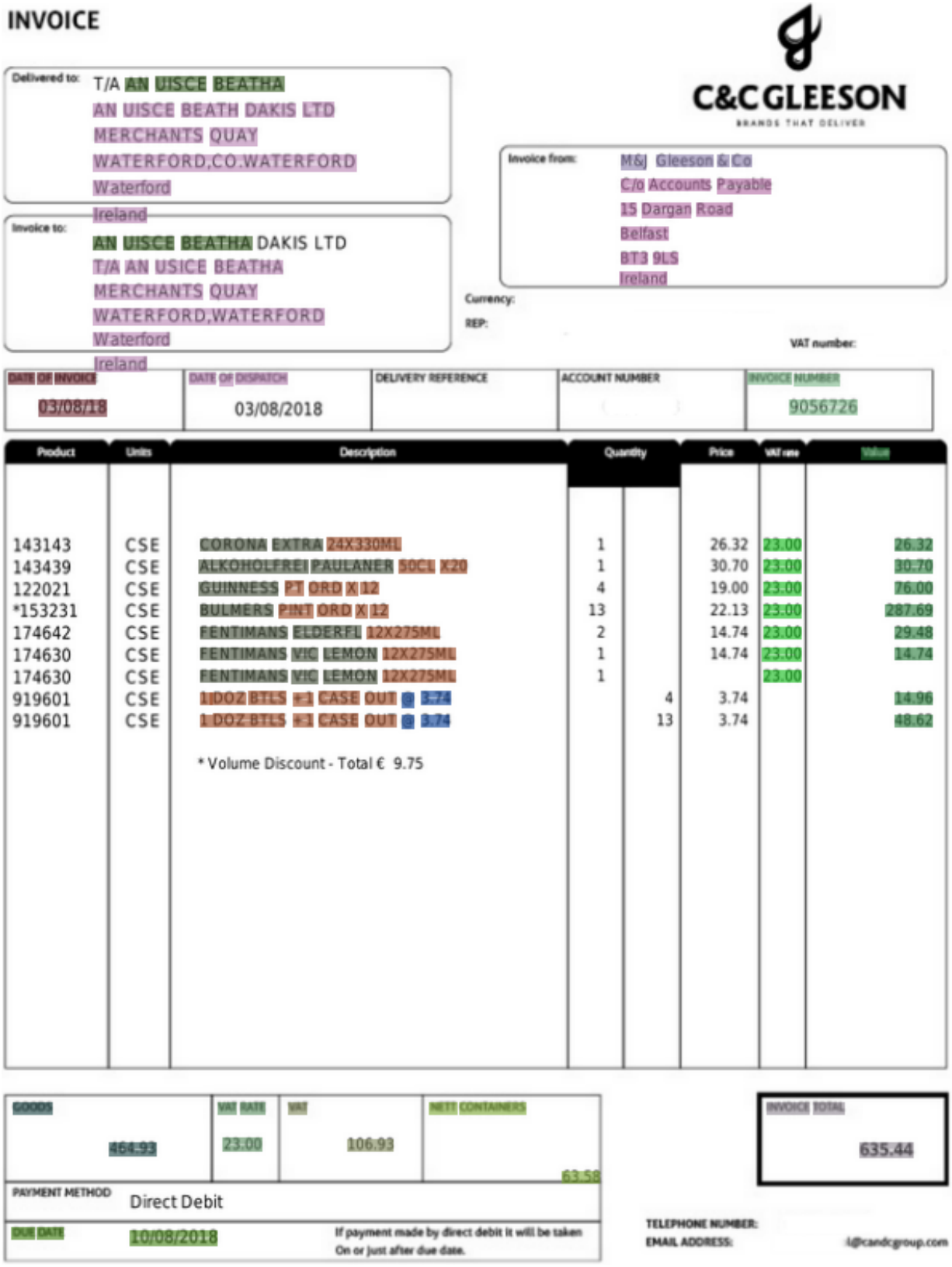

Figure 3.3: Annotated example of a document.

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:

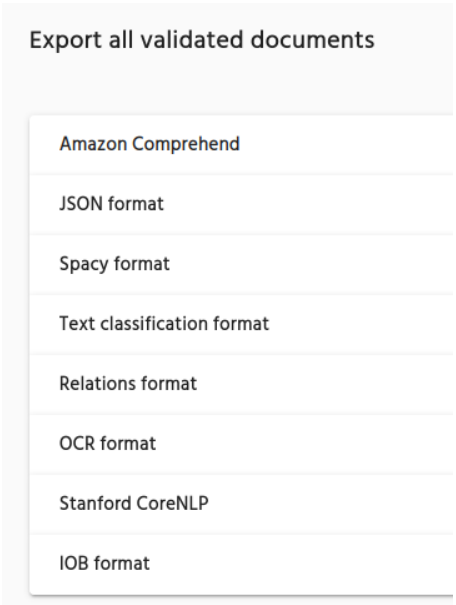

Figure 3.4: UBIAI Export Options.

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.

The data is saved to Google drive so that it can be used in Google Colab along with being able to run locally. The model was actually trained on Colab due to issues with the GPU memory on the development machine.

3.3 Training the Model

Hugging Face provide a number of excellent tutorials and guides on how to train models here [56], there are also some excellent resources on the topic here [6] and here [5][1]. The training script is written in Python, and it utilizes a number of libraries. Including the following:

- **PyTorch**: the main framework for the model.

[1]Neither of the first two articles focus on LayoutLMv2, the former article is a generic guide whilst the second article focuses on the original LayoutLM model. The third article does focus on LayoutLMv2 but was only released after this project finished implementing training. I know of it and mention it here as the writer of the blog is the same Walid as I had interacted with about the annotation software. We had talked about the training and compared evaluation, scoring and results, etc., LayoutLMv2 is about to go into production for UBIAI. This is some further proof that the architecture chosen for this project is production ready.

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

With the dataset annotated and imported into the training script, the next step is to process the data into the exact form needed to train the model. The data is manipulated and a Pandas dataframe is created as per Figure 3.5:

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, 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, 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, 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 3.5: 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, these words will be tokenized in the tokenizer and form part of the input embedding.
- **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 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 prefix of `I_` [136]
- **S**: This denotes a single token entity.
- **E**: The token is the end of an entity.
- **O**: Doesn't correspond to any entity.

These labels are then saved in the data_config class which will be passed to the model during training and reused for inference.

Listing 3.2: Data Config.

```
class data_config:
  labels = np.unique([tag for doc_tag in ner_tags for tag in doc_tag]).tolist()
  num_labels = len(labels)
  id2label = {v: k for v, k in enumerate(labels)}
  label2id = {k: v for v, k in enumerate(labels)}
```

- **image_path**: the path to the image file. The tokenizer needs a copy of the image.

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

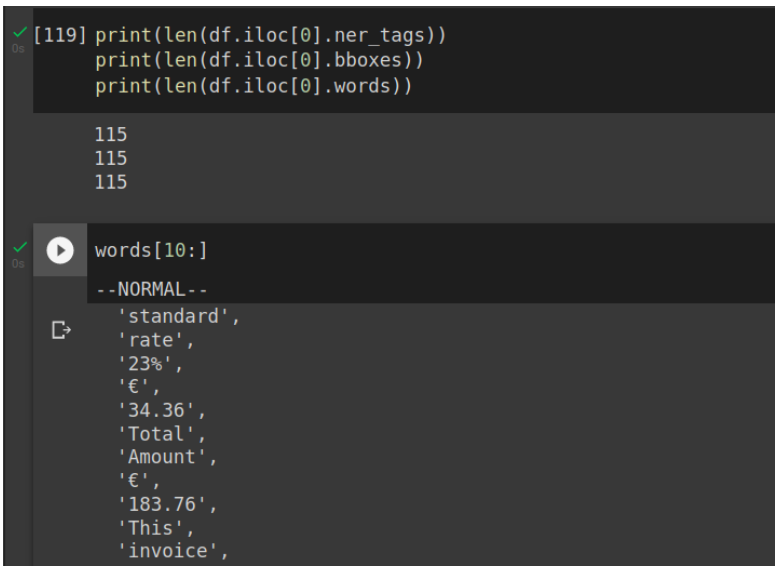

Figure 3.6: Sanity Check of the Length of the Training Input, can also see the words as input.

- The processor is downloaded from Hugging Face and the necessary inputs are passed in as parameters. Only the tokenizer part of the processor is used.
- The datasets are then created using the Hugging Face Dataset function.
- The datasets are formatted.

Listing 3.3: Dataset Preperation.

```
# processor download
processor = LayoutLMv2Processor.from_pretrained("microsoft/layoutlmv2-base-uncased",
    revision="no_ocr")

# inputs that will become embeddings
encoded_inputs = processor(images, words, boxes=boxes, word_labels=word_labels,
                           padding="max_length", truncation=True)

# dataset creation
train_dataset = train_dataset.map(preprocess_data, batched=True,
    remove_columns=train_dataset.column_names,
                                  features=features)
valid_dataset = valid_dataset.map(preprocess_data, batched=True,
    remove_columns=valid_dataset.column_names,
                                  features=features)

# GPU settings for training
train_dataset.set_format(type="torch", device="cuda")
valid_dataset.set_format(type="torch", device="cuda")
```

The LayoutLMv2 model in this project is used in *TokenClassification* mode. It will try to classify each token in the invoice as belonging to a label category. The model must be downloaded and fed the configurations.

Listing 3.4: Model Creation.

```
model_path = 'microsoft/layoutlmv2-base-uncased'

# config creation
config = LayoutLMv2Config.from_pretrained(model_path, num_labels=data_config.num_labels,
    id2label=data_config.id2label, label2id=data_config.label2id)

# model instantiation
model = LayoutLMv2ForTokenClassification.from_pretrained(model_path, config=config)

# model to CUDA, if available
model.to(device)
```

Now that the data is ready, and the model is created and configured the training function must be defined. It takes in the training data, the model and the optimizer, in this case AdamW, as parameters:

Listing 3.5: Training Function.

```
def train_fn(train_dataloader, model, optimizer):
    total = len(train_dataloader)
    tk0 = tqdm(train_dataloader, total=total)
    train_loss = 0.0
    for bi, batch in enumerate(tk0):
        # embedding components
        input_ids = batch['input_ids'].to(device)
        bbox = batch['bbox'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        token_type_ids = batch['token_type_ids'].to(device)
        labels = batch['labels'].to(device)
        resized_images = batch['image'].to(device)
        # forward pass
        outputs = model(image=resized_images, input_ids=input_ids, bbox=bbox,
            attention_mask=attention_mask, token_type_ids=token_type_ids, labels=labels)
        # loss and optimize
        loss = outputs.loss
        train_loss += loss.item()
        loss.backward()
        optimizer.step()
        # zero the parameter gradients for next iteration
        optimizer.zero_grad()
    return train_loss / total
```

This training function is then called in a loop. To match this with traditional learning the number of calls to this function would correspond to the number of epochs.

```
MODEL_PATH ="/content/model_weights/myModel.bin"
try:
  os.mkdir('/content/model_weights')
except FileExistsError:
  pass
epochs = 30
optimizer = AdamW(model.parameters(), lr=5e-5)
global_step = 0
best_f1_score = 0
best_score = None
for epoch in range(epochs):
    train_loss = train_fn(train_dataloader, model, optimizer)
    current_f1_score = eval_fn(valid_dataloader, model)
    if current_f1_score["f1"] > best_f1_score:
        best_score = current_f1_score
        torch.save(model.state_dict(), MODEL_PATH)
        best_f1_score = current_f1_score["f1"]
    print("epoch :", epoch,"===>",f"train_loss: {train_loss:2.3f}",f"valid_loss: {current_f1_score['loss']:2.3f} ",end="")
    print(f"best_f1_score : {best_f1_score:2.3f}")
```

```
--NORMAL--
epoch : 37 ===> train_loss: 0.189 valid_loss: 0.457 best_f1_score : 0.848
100% 37/37 [00:48<00:00, 1.15s/it]
100% 13/13 [00:03<00:00, 4.11it/s]
epoch : 38 ===> train_loss: 0.184 valid_loss: 0.467 best_f1_score : 0.848
100% 37/37 [00:48<00:00, 1.15s/it]
100% 13/13 [00:03<00:00, 4.05it/s]
epoch : 39 ===> train_loss: 0.173 valid_loss: 0.444 best_f1_score : 0.848
```

Figure 3.7: Calling Training Function, output of training depicting training loss, validation loss, epoch number and F_1 score.

3.3.1 Model Evaluation

The default metric in a classification problem is the accuracy, due to its simplicity, which is the fraction of correct predictions relative to the total number of predictions. However, there are many situations in which this metric is unsuitable, for example when a dataset is imbalanced or, as in this case, where our interest is in the positive cases only — within this document did we identify/tag the relevant regions. In this situation, a more appropriate metric is the F_1 score / metric. To explain the F_1 score [99], the *Confusion Matrix*[2] must be defined:

[2]The example depicted is a simple binary confusion matrix, the actual confusion matrix for this model will have an $n * n$ matrix, where n is the number of labels.

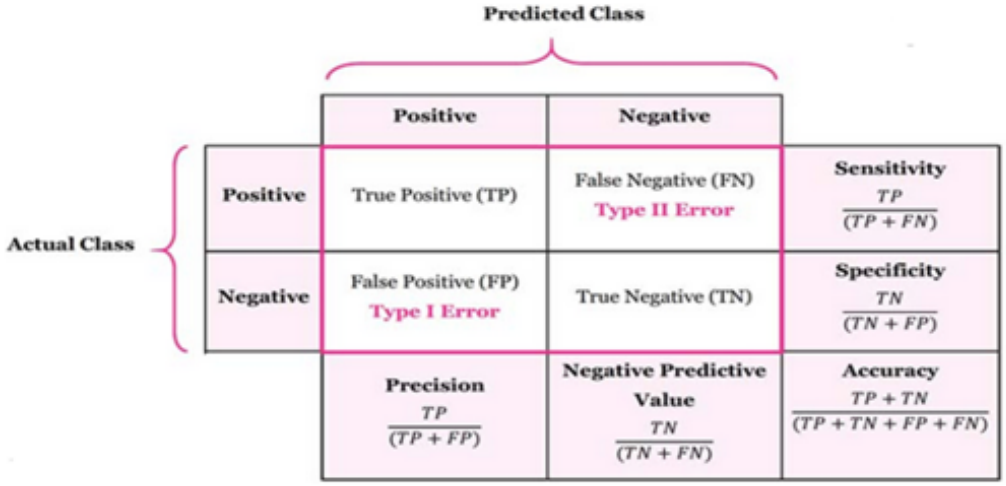

Figure 3.8: Formulas for scoring sourced from here [90].

As can be seen in Figure 3.8, the confusion matrix is essentially built by comparing the model's predictions (inference) and the actual values of the data. The formula for the F_1 score is::

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

As the formula depends on *Precision* and *Recall* scores, first they must be defined:

- **Precision**: Taking every prediction from the model, where that prediction is positive, precision counts the percentage that is correct [80]. Therefore,

  $$\text{Precision} = \frac{\text{Number of True Positives } (TP)}{\text{Number of True Positives } (TP) + \text{Number of False Positives } (FP)} = \frac{TP}{TP + FP}$$

  - A model with a low precision score may find many positives, but it also wrongly detects many *false positives* - positive predictions that are not actually positive.
  - A model with a high precision score may only find a few positives but from those that it predicts, it correctly identifies a high percentage of them.

- **Recall**: Taking every true positive value[3] how many of these values did the model succeed in capturing [80]. Therefore,

  $$\text{Recall} = \frac{\text{Number of True Positives } (TP)}{\text{Number of True Positives } (TP) + \text{Number of False Negatives } (FN)} = \frac{TP}{TP + FN}$$

  - A model with a low recall score may not find all, or a large proportion of the positive cases.

[3] In this case, the actual labels as annotated in the data preparation phase.

– A model with a high recall score will discover a high proportion of the positive cases in the dataset but may also wrongly identify some negative cases as positive cases.

Now that Precision and Recall are defined, the F_1 score is a *harmonic mean* [64] of the two. This returns the two scoring metrics as a single value, without having a bias toward one or the other, although this can be shifted if the model use case cares more for one or the other.

As can be observed from Figure 3.7, the F_1 score is coming in 84.8%. This figure needs to be put into context. What this figure actually means is that in terms of the documents, 84% of the documents have been classified correctly. This would mean that 84% of the documents need not be classified. There would be some overhead in manually validating the invoices. But the visual check of a document is a quicker process than the manual extraction.

3.4 Inference

The next step is to test the model on data that it has not seen before. For this step an invoice must be OCR'd. This is done in an API call to AWS Textract which returns OCR data in the following format:

```
Type: LINE
Detected: Grand Cru Beers
Confidence: 99.67%
Id: 9171cf3e-f2d0-4864-9dc3-9a095f38ce24
Relationships: [{'Type': 'CHILD', 'Ids': ['dcf15f6d-1a91-4884-a831-d1b07c8b9ae5', '19f470de-578e-4997-bc84-56136a8082bc',
'01e96e53-097f-42ec-89b6-b5973c7eee48']}]
Bounding Box: {'Width': 0.21471719443798065, 'Height': 0.0159408338367939, 'Left': 0.6572423577308655, 'Top':
0.022094136103987694}
Polygon: [{'X': 0.6572423577308655, 'Y': 0.022094136103987694}, {'X': 0.8719595074653625, 'Y': 0.022094136103987694},
{'X': 0.8719595074653625, 'Y': 0.03803497180342674}, {'X': 0.6572423577308655, 'Y': 0.03803497180342674}]
```

Figure 3.9: Output of Textract API call, the boxes are the bounding boxes of the words in the invoice.

Some pre-inference processing is required to prepare the data for the model. This step is similar to the data preparation for training step. The tokenizer needs:

- **words**: A list of all the words in the invoice.
- **bboxes**: The corresponding bounding boxes for each word.
- **image**: A copy of the image.

Listing 3.6: Perperation for inference.

```
# define the tokenizer
inference_processor =
    LayoutLMv2Processor.from_pretrained("microsoft/layoutlmv2-base-uncased",
    revision="no_ocr")

# initializing the tokenizer with the correct data
inf_encoding = inference_processor(test_im, words, boxes=nbox,return_tensors="pt",
    padding="max_length", truncation=True)

# map all tensors, inputs which will become embeddings, to cuda device
for k,v in inf_encoding.items():
  inf_encoding[k] = v.to(device)

# load the pre-trained model
model= torch.load(model_path,map_location=device)
```

Once these steps have been completed, the next step is inference.

Listing 3.7: Inference.

```
# put the model in inference mode
model.eval()

# make the inference
with torch.no_grad():
  inference_outputs = model(**inf_encoding)
  #Output Logits (just before applying softmax)

#sanity check to check for shape
inference_outputs.logits.shape
```

Some post-processing is essential to make sense of the models output:

Listing 3.8: Inference Output Post-processing.

```
# remove CLS,PAD,and SEP special tokens from input text
raw_input_ids = inf_encoding['input_ids'][0].tolist()
# these are the predictions
predictions = inference_outputs.logits.argmax(-1).squeeze().tolist()
token_boxes = inf_encoding.bbox.squeeze().tolist()
special_tokens = [inference_processor.tokenizer.cls_token_id,
    inference_processor.tokenizer.sep_token_id, inference_processor.tokenizer.pad_token_id]

# conver the labels from their numerical lookup table value to the actual word
input_ids = [id for id in raw_input_ids if id not in special_tokens]
predictions = [model.config.id2label[prediction] for i,prediction in enumerate(predictions)
    if not (raw_input_ids[i] in special_tokens)]
actual_boxes = [unnormalize_box(box, width, height) for i,box in enumerate(token_boxes) if
    not (raw_input_ids[i] in special_tokens )]
```

The predictions contains a list of labels for each word in the input. Words which do not fit any labels are denoted with an 'O'. These are stripped out in the final part of the post-processing. With the inference complete, the labels are drawn on to the image to visually show the results.

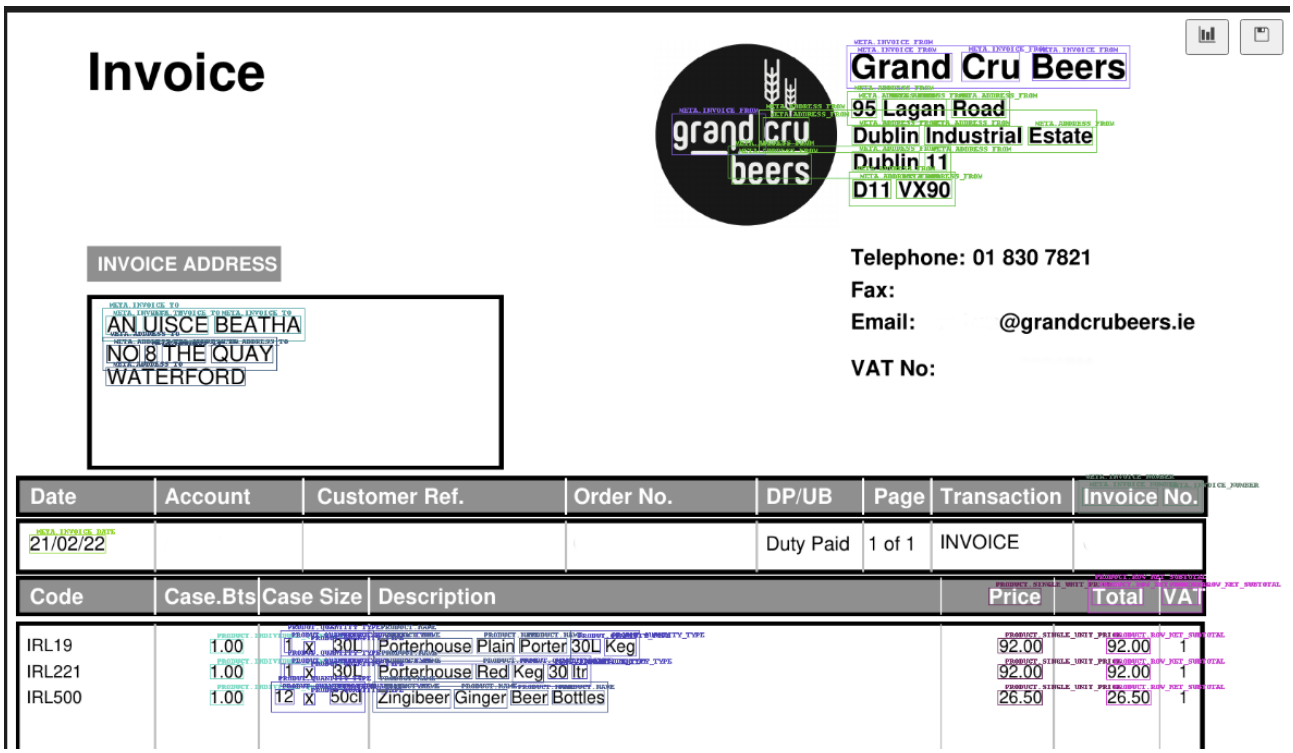

Figure 3.10: Example 1 of the inference, bounding boxes around the words with the corresponding labels inferred by the model.

Just based upon an informal visual inspection of the invoice, we can see the model is performing well. A further example of the output that is, currently, of most interest to the project:

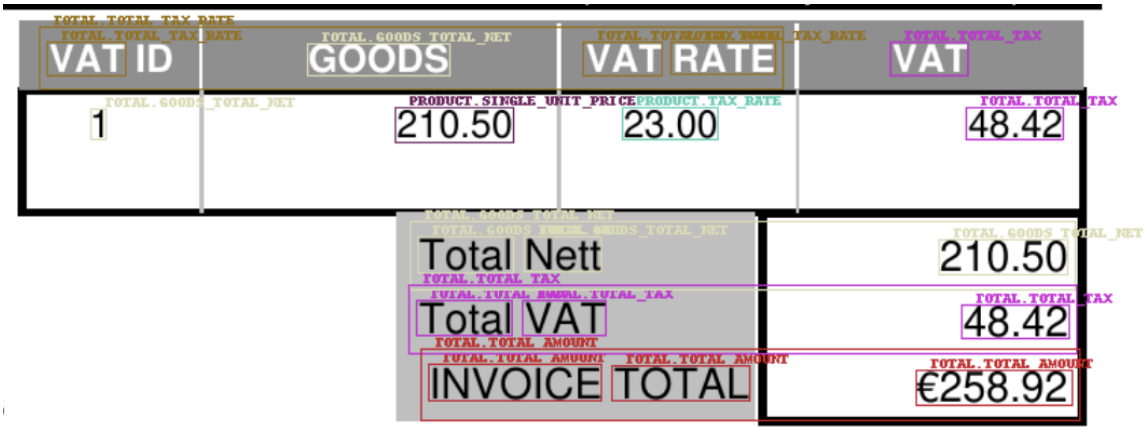

Figure 3.11: Example 2 output of the inference, the boxes are the bounding boxes of the words in the invoice, with the corresponding labels inferred by the model.

The model has correctly identified all the totals that are of interest. This leaves less post-processing to do to obtain the final results ready for the database.

3.4.1 Post-Processing

After this some validation checks and further manipulation of the output data is carried out. The checks include the following: The total amount of the invoice is labelled and is correct. This can done by checking the total amount figures are present. If there is no total amount, then the system will look to create it from the net and vat amounts. The validation added has meant that with every tested invoice thus far, the system has correctly identified all the figures that are of interest.

3.5 Exposing the Pipeline as a REST API

The pipeline consists of 20 functions which make up the entire pre-process, inference, post-process and saving to database in Kubernetes phases. To expose the pipeline as a REST API, a server must be initialized:

Listing 3.9: Initializing the Server.

```
# initialize the server
app = Flask(__name__)
# configurations for server
app.config.from_pyfile('config.cfg')

# setting the logger
log.getLogger('sqlalchemy.dialects.postgresql').setLevel(log.INFO)

# initialising metrices for prometheus monitoring
metrics = PrometheusMetrics(app)
metrics.info('app_info', 'Application info', version='0.0.1')

# custom metrics
REQUESTS = Counter("http_requests_total", "HTTP requests")
IN_PROGRESS = Gauge("inprogress_requests", "Inprogress HTTP requests",
       multiprocess_mode='livesum')
REQUEST_TIME = Summary('request_processing_seconds', 'Time spent processing request')
```

Note: Even though the server is running locally, there is some additional code such as the Prometheus metrics in Listing 3.9 which are specific to a Kubernetes deployment. As the initial development had been done to run the pipeline in the Kubernetes cluster, metrics and specialized logging are present in the code. The Inference Server is *Kubernetes-ready*.

Listing 3.10: Inference Endpoint Defined.

```
@app.route('/invoiceLocation', methods=['GET', 'POST'])
def invoiceLocation():
   # NUM_REQUESTS.inc()
   if request.method == 'POST':
      # get the document
      document = request.form['document']
      # get the bucket name
      bucket_name = request.form['bucket_name']

```

```
10        # start the pipeline
11        model = load_model(model_path)
12        inference_processor = load_inference_processor()
13        iServe = InferenceServer(model_path)
14        print('loaded model and inference processor')
15        print(f'bucket_name = {bucket_name}, document = {document}')
16        print ('starting pipeline')
17        # timing the pipline run
18        start_time = datetime.time()
19        # run the pipeline
20        filtered_words = InferenceServer.start_pipeline(document, bucket_name, model,
              inference_processor)
21        end_time = datetime.time()
22        print(f'Pipeline completed in: {end_time - start_time}')
23        # return the filtered words
24        return jsonify(filtered_words)
25    else:
26        return jsonify({'error': 'no document'})
```

The last step then is to start the server:

Listing 3.11: Starting the Server.

```
1  if __name__ == '__main__':
2     load_dotenv()
3
4     logging.basicConfig(level=os.environ.get("LOGLEVEL", "INFO"))
5     # instantiate logger object
6     log = logging.getLogger(__name__)
7
8     # register_metrics(app, app_version="v0.1.2", app_config="stage")
9     dispatcher = DispatcherMiddleware(app.wsgi_app, {"/metrics": make_wsgi_app()})
10    run_simple(hostname="localhost", port=5000, application=dispatcher)
```

3.6 Gmail Scraper

The Gmail server is a Python CLI application which is used to scrape the Gmail inbox, searching for invoices. There is nothing earth-shattering here. If the reader wants to see the code it is present in the code repository.

3.7 ML Pipeline Performance

Pipeline run time:

```
func 20 save_image success
INFO:__main__:Pipeline completed in: 0:00:08.330912
INFO:werkzeug:127.0.0.1 - - [21/Apr/2022 16:45:31] "POST /invoiceLocation HTTP/1.1" 200 -
```

Figure 3.12: Timing of the pipeline run time.

As can be seen the entire pipeline takes a mere 8 seconds to run. This includes AWS interaction, pre- and post-processing, and saving to database.

This is a very satisfying result and is proof that efficiency was at front and center focus during the development of the project.

4 Conclusion

This section sees some reflection both technical and personal through a personal perspective.

4.1 Technical

How does the system match expectations?

The system performs to a very decent standard. Approximately 85% of invoices are correctly labeled. Considering the amount of labels that are in use and that this is the very first version of the model, I am happy with the result. Another factor to keep in mind is the low amount of data in the training set. With a better dataset — both more samples and more solid annotation[1], the model's performance should increase.
Furthermore, an interesting observation that will cut down the time to retrain a model is the fact that I can now use the model to predict all labels on a given invoice. These labels are exactly the same as used in annotation, therefore, I can now use the model to predict the labels on all new and non-annotated invoices. I can alter the annotation slightly (if needed) and build up a larger dataset much quicker by automating the annotation process. The more data (up to a point) should lead to more accurate results, resulting in an even easier task of correcting annotations for the next rounds of training.

It is important to note that we have not completely automated these processes. But we have replaced a very slow process with a good level of automation and a decrease in complexity (checking and correcting rather than full extraction).

How well did the chosen technologies fit together to realise the desired architecture?

The technologies fit together very well. Following a microservice architecture throughout the project, it becomes quite easy to upgrade components and even change the architecture whilst effecting minimal components directly.

[1]The annotating is something that I had never really thought about previous to this project. It is not the most straightforward of tasks when one is unfamiliar with how the model learns. I have other ideas on how to change the annotation standard and compare performance metrics.

A slight negative was the workaround which had to be used due to the Minikube issue, but with the troubleshooting, invariably, comes a greater understanding of the components which constitute the issue. Kubernetes handled this upgrade very smoothly.
A case in point that has not been mentioned, a new version of the Transactions API[2] was rolled out. This was actually quite a substantial change to the data types and the structure of the data in the system. The new API was upgraded in seconds, migrating and introducing changes to the database schema with no errors or issues.

4.2 Personal

On a personal note, this project has hit the objectives that I have set for it, and then some. Having a keen interest in the area of machine learning, gaining more knowledge about the area was one of the main goals of this project. From having no experience in the field a year ago, I have managed to implement an entire pipeline that may actually serve as a prototype to a real-world product.
I have also familiarised myself with the technologies used in industry. I have also gained a massive amount of understanding in the area of data manipulation and improved my Python skills considerably.

Elements of the project (the ML pipeline) have already been used in real life scenarios in getting documents ready for VAT returns. Working almost flawlessly[3], going forward I look forward to building on the implementation of the current solution. This will make my life easier and gives me back time that I otherwise would have had to spend on these manually completing these processes.

I remember fearing that I wouldn't be able to grasp the concepts that underpin machine learning, model architecture and how models work. With the transformers architecture family being regarded as the current state-of-the-art, not only have I gained a great deal of knowledge in the field of machine learning, but I now have a very good understanding of the architectures of models that are at the cutting edge of machine learning. Being able to have conversations with Walid, about scoring, training and evaluation methods was a big confidence boost. There is, obviously, a massive amount still to learn, but I look forward to doing so with a newly obtained confidence in my abilities.

[2] Entry point into the global system for transactional data from PoS systems.
[3] It may be noted that the 85% F_1 was calculated against the models ability to predict every label. For the VAT returns, only 5 labels were needed, and the model actually did a very good job for these labels —anecdotally.

Appendix A

Appendix

A.1 Self Attention in the Decoder

Whilst LayoutLMv2, BERT and most subsequent models do not have a decoder component as part of their architecture, for the sake of completeness and having described the encoder from the transformers model, the decoder will be briefly described here.
The self attention mechanism in the decoder works in a very similar fashion to the encoder, but the inputs are different as per Figure A.1.

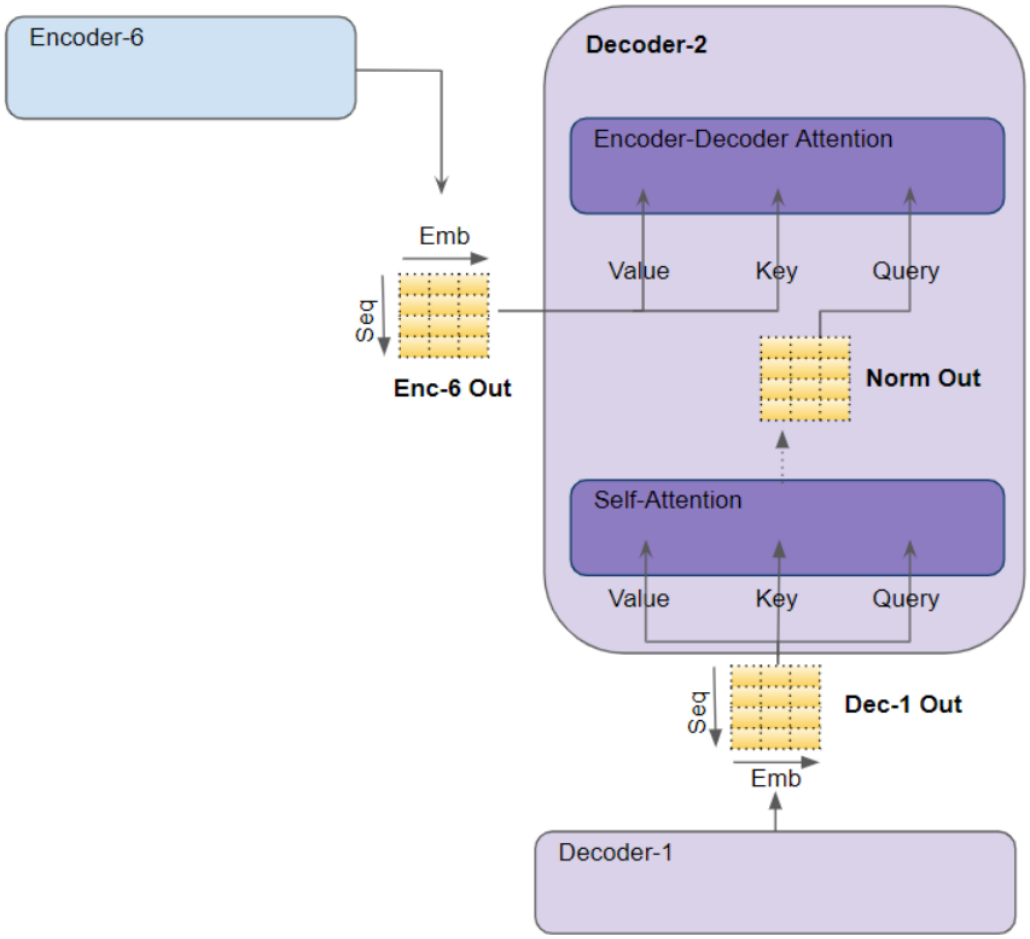

Figure A.1: Self Attention Mechanism for Decoder (source [42]), depicting the resulting attention score.

In the decoder the relevance of each word in the target sentence is computed

with respect to every other word in the target sentence as per Figure A.2.

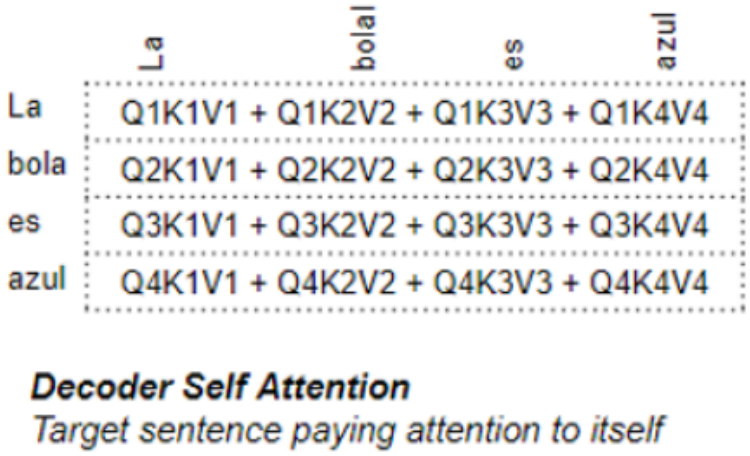

Figure A.2: Attention Mechanism for Encoder-Decoder in the decoder (source [42]).

A.1.1 Attention in the Encoder-Decoder

Again, very similar to the encoder attention but the Query is obtained from the target sentence and the Key and Value are obtained from the source sentence. The goal is to compute the relevance of each word in the target sentence to each word in the source sentence.

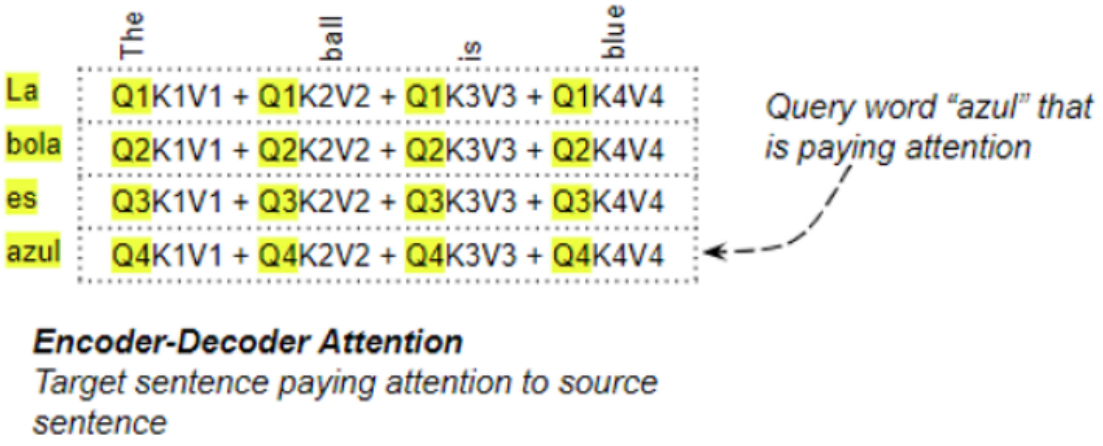

Figure A.3: Attention Mechanism for Encoder-Decoder (source [42]). Same input and target as Figure A.1.

Bibliography

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

[2] Last Week in AI. *GPT-3 Is No Longer the Only Game in Town*. https://lastweekin.ai/p/gpt-3-is-no-longer-the-only-game.

[3] *AI Weekly: AI Model Training Costs on the Rise, Highlighting Need for New Solutions*. Oct. 2021.

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

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

[6] Walid Amamou. *Fine-Tuning Transformer Model for Invoice Recognition*. https://towardsdatascience.com/fine-tuning-transformer-model-for-invoice-recognition-1e55869336d4. Aug. 2021.

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

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

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

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

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

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

[13] Hangbo Bao et al. "UniLMv2: Pseudo-Masked Language Models for Unified Language Model Pre-Training". In: *arXiv:2002.12804 [cs]* (Feb. 2020). arXiv: 2002.12804 [cs].

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

[15] *BERT for Natural Language Processing —All You Need to Know about BERT*. https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-bert/.

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

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

[18] Jason Brownlee. *A Gentle Introduction to Cross-Entropy for Machine Learning*. Oct. 2019.

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

[20] Miguel Romero Calvo. *Dissecting BERT Part 1: The Encoder*. May 2019.

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

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

[23] CodeEmporium. *BERT Neural Network - EXPLAINED!* May 2020.

[24] *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*.

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

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

[27] *CORD: A Consolidated Receipt Dataset for Post-OCR Parsing*. Clova AI Research. Mar. 2022.

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

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

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

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

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

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

[34] *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.

[35] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv:1810.04805 [cs]* (May 2019). arXiv: 1810.04805 [cs].

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

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

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

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

[40] Li Dong et al. "Unified Language Model Pre-training for Natural Language Understanding and Generation". In: *arXiv:1905.03197 [cs]* (Oct. 2019). arXiv: 1905.03197 [cs].

[41] dontloo. *Answer to "What Exactly Are Keys, Queries, and Values in Attention Mechanisms?"*. Aug. 2019.

[42] 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.

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

[44] Ketan Doshi. *Transformers Explained Visually (Part 2): How It Works, Step-by-Step*. https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34. June 2021.

[45] 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.

[46] 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.

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

[48] *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.

[49] *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.

[50] Emil. *Answer to "What Exactly Are Keys, Queries, and Values in Attention Mechanisms?"*. Jan. 2020.

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

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

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

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

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

[56] *Fine-Tune a Pretrained Model*. https://huggingface.co/docs/transformers/training.

[57] *FUNSD*. https://guillaumejaume.github.io/FUNSD/.

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

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

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

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

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

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

[64] "Harmonic Mean". In: *Wikipedia* (Apr. 2022).

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

[66] Rani Horev. *BERT Explained: State of the Art Language Model for NLP.* https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270. Nov. 2018.

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

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

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

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

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

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

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

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

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

[76] 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.

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

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

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

[80] Joos Korstanje. *The F1 Score.* https://towardsdatascience.com/the-f1-score-bec2bbc38aa6. Aug. 2021.

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

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

[83] *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/.

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

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

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

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

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

[89] *LayoutLMV2.* https://huggingface.co/docs/transformers/model_doc/layoutlmv2.

[90] Ron Lee. *Confusion Matrix (Precision, Recall, F1 Score).* Mar. 2021.

[91] *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/.

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

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

[94] Lei Mao. *Layer Normalization Explained*. https://leimao.github.io/blog/Layer-Normalization/. May 2019.

[95] *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.

[96] *Microsoft and Nvidia Team up to Train One of the World's Largest Language Models*. Oct. 2021.

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

[98] *Models - Hugging Face*. https://huggingface.co/models.

[99] Joydwip Mohajon. *Confusion Matrix for Your Multi-Class Machine Learning Model*. https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826. July 2021.

[100] James Montantes. *BERT Transformers — How Do They Work?* https://becominghuman.ai/bert-transformers-how-do-they-work-cd44e8e31359. Apr. 2021.

[101] 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.

[102] 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.

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

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

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

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

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

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

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

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

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

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

[113] 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/.

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

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

[116] Farhan Rahman. *COMPAS Case Study: Fairness of a Machine Learning Model*. https://towardsdatascience.com/compas-case-study-fairness-of-a-machine-learning-model-f0f804108751. Sept. 2020.

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

[118] *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.

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

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

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

[122] Sean. *What Exactly Are Keys, Queries, and Values in Attention Mechanisms?* Forum Post. Dec. 2020.

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

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

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

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

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

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

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

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

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

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

[133] The A.I. Hacker - Michael Phi. *Illustrated Guide to Transformers Neural Network: A Step by Step Explanation*. Apr. 2020.

[134] *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/.

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

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

[137] *Transformer Architecture, Self-Attention*. https://www.kaggle.com/residentmario/transformer-architecture-self-attention.

[138] *Transformers*. https://huggingface.co/docs/transformers/index.

[139] Sam Tseng. *Answer to "What Exactly Are Keys, Queries, and Values in Attention Mechanisms?"*. Apr. 2020.

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

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

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

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

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

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

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

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

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

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

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

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

[152] *What Is Attention?* https://machinelearningmastery.com/what-is-attention/.

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

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

[155] *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.

[156] *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.

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

[158] 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].

[159] Yiheng Xu et al. "LayoutLM: Pre-training of Text and Layout for Document Image Understanding". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Aug. 2020), pp. 1192–1200. DOI: 10.1145/3394486.3403172. arXiv: 1912.13318.