PROJECT REPORT

SUBMITTED TO

SVKM’S NMIMS (Deemed to be) UNIVERSITY

IN PARTIAL FULFILLMENT FOR THE DEGREE OF

**MASTER OF SCIENCE IN**

**STATISTICS & DATA SCIENCE**

BY

**RAJ NARSINHA MALKAR SHIVANGI GAURAV TANEJA SHRUTI SUDHIR PATHAK**

**A black text on a white background  Description automatically generated**

NMIMS NILKAMAL SCHOOL OF MATHEMATICS, APPLIED STATISTICS & ANALYTICS

V. L. Mehta Road, Vile- Parle (West) Mumbai – 400056

APRIL, 2024

PROJECT REPORT SUBMITTED TO

SVKM’S NMIMS (Deemed to be) UNIVERSITY

IN PARTIAL FULFILLMENT FOR THE DEGREE OF

**MASTER OF SCIENCE IN**

**STATISTICS & DATA SCIENCE**

**BY**

**RAJ NARSINHA MALKAR SHIVANGI GAURAV TANEJA SHRUTI SUDHIR PATHAK**

**Program Director (Statistics) Dean NSOMASA**

**(Prof. Sunil S. Shirvaiker) (Dr. Sushil Kulkarni)**

**A black text on a white background  Description automatically generated**

NMIMS NILKAMAL SCHOOL OF MATHEMATICS, APPLIED STATISTICS & ANALYTICS

V. L. Mehta Road, Vile- Parle (West)

Mumbai – 400056

**CERTIFICATE**

This is to certify that work described in this thesis entitled “Automated Document Summarizer using Transformers” has been carried out by Raj Malkar, Shivangi Taneja and Shruti Pathak under my supervision. I certify that this is his/her bonafide work. The work described is original and has not been submitted for any degree to this or any other University.

**Date:**

**Place:**

|  |  |  |
| --- | --- | --- |
| **Internal Mentor** | **Company Mentor** |  |
| **(**  **Date:** | **) (**  **Date:** | **)** |

****

8th Floor, Bhive Office, Summit A, Brigade Metropolis, Mahadevpura, Bangalore, 560048.

**ACKNOWLEDGEMENTS**

Many people have contributed in a variety of ways to the successful completion of this term project. We would gladly like to acknowledge our indebtedness for the ideas and information they have provided.

We would like to express our sincere gratitude to our internal mentor **Dr. Pradnya Khandeparkar** for pushing us to achieve the best of our capabilities with every facet of this project. The completion of this project would not have been possible without her guidance and advice. We are extremely grateful to the faculty for sharing her knowledge and experience which played a vital role in this project.

We would like to thank Auronova Consulting for providing us with this wonderful opportunity to work, learn and explore more.

We owe a wholehearted appreciation to our external mentors **Mr. Umesh Patharkar, Mr. Shubham Jain, Ms Akanksha Singh and Mr. Kedar Pinge** for their continual mentorship and support throughout the journey of our internship despite their busy schedule.

Their guidance, encouragement, and suggestions provided us with the necessary insight into the research problem and paved the way for the meaningful progress of this project. For that, we owe them our deepest gratitude.

We are immensely grateful to **Mr. Pintu Kumar, Ms. Rohini Mukherjee** and **Mr. Milanshu Khandelwal**, and the entire staff of Auronova consulting for providing us the resources and assistance during this project.

We also would like to thank NMIMS NSoMASA and all the Faculty Members for enabling us to become an inquisitive learner.

TABLE OF CONTENTS

1. [ABSTRACT 8](#_bookmark0)
2. [INTRODUCTION 9](#_bookmark1)
3. [RATIONALE 10](#_bookmark2)
4. [AIMS AND OBJECTIVES 12](#_bookmark3)
   1. [Aims 12](#_bookmark4)
   2. [Objectives 12](#_bookmark5)
5. [TEXT EXTRACTION 13](#_bookmark6)
   1. [Object detection approach 14](#_bookmark7)
   2. [Yolo (you only look once) 15](#_bookmark8)
      1. [How does yolo object detection work? 15](#_bookmark9)
      2. [Training the yolov5 model 18](#_bookmark10)
   3. [Cropping the images 21](#_bookmark11)
   4. [Scaling the bboxes 25](#_bookmark12)
   5. [Storing the text chapterwise 26](#_bookmark13)
6. [LSTM – Long Short Term Memory 29](#_bookmark14)
   1. [Introduction 29](#_bookmark15)
   2. [Embeddings 30](#_bookmark16)
   3. [Training process 30](#_bookmark17)
   4. [Challenges 31](#_bookmark18)
7. [TRANSFORMERS 32](#_bookmark19)
   1. [Transfer learning 32](#_bookmark20)
   2. [Transformer models 32](#_bookmark21)
      1. [General architecture of transformer models 33](#_bookmark22)
8. [SUMMARIZATION TECHNIQUES 34](#_bookmark23)
9. [ABSTRACTIVE SUMMARIZATION MODEL SELECTION 35](#_bookmark24)
   1. [Longformer encoder decoder(LED) 35](#_bookmark25)
      1. [How LED processes large texts 36](#_bookmark26)
      2. [LED pretrained on booksum 37](#_bookmark27)
   2. [T5 – Text to Text Transfer Transformer 37](#_bookmark28)
      1. [General architecture of the T5 model 38](#_bookmark29)
      2. [T5 – small model 38](#_bookmark30)
   3. [Pegasus 39](#_bookmark31)
      1. [Pegasus and cnn/dailymail 41](#_bookmark32)
10. [EXTRACTIVE SUMMARIZATION MODEL SELECTION 42](#_bookmark33)
    1. [BERT – Bidirectional Encoder Representations from Transformers 42](#_bookmark34)
       1. [Rationale for using BERT with a pre-trained pipeline for extractive](#_bookmark35) [summarization. 43](#_bookmark35)
    2. [BART - Bidirectional and Auto-Regressive Transformers 44](#_bookmark36)
       1. [BART on CNN/DailyMail 45](#_bookmark37)
    3. [T5 - Base 45](#_bookmark38)
11. [DATA ACQUISITION AND PREPARATION FOR MODEL FINE TUNING 48](#_bookmark39)
12. [MODEL FINETUNING 49](#_bookmark40)
    1. [Pegasus model for abstractive summarization 49](#_bookmark41)
    2. [T5 – Base Model for extractive summarization 51](#_bookmark42)
13. [SUMMARY GENERATION 52](#_bookmark43)
14. [MODEL EVALUATION 54](#_bookmark44)
    1. [Rouge 54](#_bookmark45)
    2. [BERTSscore 54](#_bookmark46)
    3. [Comparitive Results 56](#_bookmark47)
15. [HOSTING THE APP USING STREAMLIT 58](#_bookmark48)
16. [CONCLUSION 61](#_bookmark49)
17. [GLOSSARY 62](#_bookmark50)

# ABSTRACT

Auronova Consulting is a specialized Risk Management Consulting firm providing cutting edge financial consulting to BFSI segments inclusive of central banks, banks, fintech, investment banks, PE Firms, insurance firms, brokerage, fintech, non-banking finance companies and other financial institutions. Auronova covers the entire Risk Management spectrum from Credit risk, Market risk and Operational risk along with regulatory consulting like Basel II/III, IFRS 9, FRTB, ICAAP, Stress Testing etc. The company's unique value proposition is to provide deep risk management-related insights to their clients (majorly banks) and to enable them to use risk management as a strategic tool for value creation through a structured risk management approach.

In today's dynamic financial landscape, the effective management of risks is paramount for sustained success in the Banking, Financial Services, and Insurance (BFSI) segment.

The proliferation of financial rules and regulations underscores the complexity of the BFSI environment, necessitating a comprehensive understanding and adherence to an array of legal frameworks. In this context, document summarization emerges as a crucial tool for distilling vast volumes of regulatory content into concise and actionable insights. Our internship project delves into the realm of document summarization, aiming to develop efficient methodologies for extracting pertinent information from a myriad of financial regulations.

This report encapsulates our endeavour to revolutionize BFSI sector using document summarization, elucidating the significance of this approach in augmenting operational efficiency, mitigating compliance risks, and fostering strategic decision-making. Through an exploration of our methodologies, findings, and implications, we aim to provide a comprehensive understanding of the transformative potential of document summarization in reshaping risk management paradigms within the BFSI landscape.

This report gives a detailed explanation of the analysis, methodologies & process adopted, and interpretation of results thereof. The assumptions made during the analysis and their limitations have also been highlighted.

# INTRODUCTION

In the rapidly evolving landscape of Banking, Financial Services, and Insurance (BFSI), effective risk management stands as a cornerstone for sustainable growth and resilience. Central to our approach is the utilization of advanced technologies and methodologies to distil vast volumes of regulatory content into actionable insights, thereby empowering our clients to navigate the intricate web of financial regulations with confidence and agility.

Summarization is closely related to data compression and information understanding, both of which are key to information science and retrieval. The technology of text summarization can improve information extraction systems and allows readers to quickly view a large number of documents for important information. Indeed, automatic summarization has been recently recognized as one of the most important natural language processing (NLP) tasks, yet one of the least solved one.

In the literature, there are two main approaches to text summarization – extractive summarization and abstractive summarization. While extractive methods are arguably well suited for identifying the most relevant information, such techniques may lack the fluency and coherency of human-generated summaries. Abstractive text summarization is the task of generating a summary consisting of a few sentences that capture the salient ideas of the input text document. The adjective ‘abstractive’ is used to denote a summary that is not a mere selection of a few existing passages or sentences extracted from the source, but a compressed paraphrasing of the main contents of the document, potentially using vocabulary unseen in the source document. Abstractive summarization has shown the most promise towards addressing issues in extracting important information from the text documents, but abstractive summary generation may produce sentences not seen in the original input document.

We discuss both abstractive and extractive text summarization in this report.

# RATIONALE

Document summarization is a critical task in natural language processing (NLP), aiming to distill the essential information from a document while preserving its meaning and context. With the exponential growth of digital content, there is an increasing need for automated solutions to efficiently process and extract insights from vast amounts of textual data. Our document summarization app addresses this need by leveraging state-of-the-art transformer- based models to generate concise and informative summaries.

1. **Efficiency and Scalability:**

Traditional methods of document summarization often rely on handcrafted features or statistical algorithms, which may struggle to capture the nuances and complexities of natural language. In contrast, transformer-based models have demonstrated superior performance across various NLP tasks by learning hierarchical representations of text data. By harnessing the power of transformers, our app offers scalable and efficient summarization capabilities, capable of handling large volumes of documents with minimal computational overhead.

1. **Contextual Understanding:**

One of the key advantages of transformer-based models is their ability to capture contextual information from input sequences. Unlike simpler methods that treat each word or sentence in isolation, transformers can consider the entire document context when generating summaries. This enables our app to produce summaries that are not only concise but also contextually relevant, ensuring that important information is retained while irrelevant details are filtered out.

1. **Adaptability and Customization:**

Another benefit of using transformers is their flexibility and adaptability to different domains and languages. Pre-trained transformer models, such as BERT, are often fine-tuned on domain-specific corpora to further enhance their performance on specialized tasks. Our app provides users with the flexibility to fine-tune the summarization model on custom datasets, allowing them to tailor the summarization process to their specific requirements and domain expertise.

1. **Evaluation and Quality Assurance:**

Ensuring the quality and coherence of generated summaries is paramount for any document summarization tool. Our app incorporates robust evaluation metrics, such as ROUGE (Recall-Oriented Understudy for Gisting Evaluation), to assess the similarity between generated summaries and human-written references. By continually refining the summarization model based on user feedback and evaluation results, we strive to maintain high standards of summarization quality and accuracy.

1. **User-Friendly Interface:**

In addition to powerful backend algorithms, our app features an intuitive and user-friendly interface designed to streamline the document summarization process. Users can easily upload their documents, customize summarization settings, and review generated summaries within a unified platform. By prioritizing usability and accessibility, we aim to empower users of all skill levels to leverage the benefits of automated document summarization in their workflow.

In conclusion, our document summarization app represents a cutting-edge solution for efficiently distilling key insights from textual documents using transformer-based models. By combining the latest advancements in NLP with a user-centric design philosophy, we strive to democratize access to advanced summarization capabilities and unlock new possibilities for knowledge extraction and information retrieval across diverse domains.

# AIMS AND OBJECTIVES

### Aims

Develop an innovative document summarization app leveraging transformer-based models to efficiently distil essential information from textual documents.

Address the increasing demand for automated solutions to process and extract insights from vast amounts of digital content, catering to diverse user needs and domains.

Democratize access to advanced document summarization capabilities by providing a user- friendly platform that enhances productivity and knowledge extraction.

Fine-tune the summarization models on data from the BFSI (Banking, Financial Services, and Insurance) sector to generate summaries of regulatory documents, facilitating better navigation of complex regulations.

### Objectives

1. Implement transformer-based models, within the app framework to enable accurate and contextually relevant document summarization.
2. Design and develop a user-friendly interface that allows seamless document upload, customization of summarization settings, and easy review of generated summaries.
3. Incorporate robust evaluation metrics, including ROUGE, to ensure the quality and coherence of generated summaries, continually refining the summarization model based on user feedback and evaluation results.
4. Provide flexibility and adaptability by allowing users to fine-tune the summarization model on custom datasets, tailoring it to their specific domain requirements, particularly in the BFSI sector.
5. Conduct extensive testing and validation to verify the performance, scalability, and usability of the app across diverse document types and user scenarios.
6. Collaborate with domain experts and stakeholders to gather feedback and iterate on app features, enhancing its effectiveness and relevance for real-world applications, especially in regulatory compliance within the BFSI sector.
7. Disseminate the app to a wide audience through appropriate channels, including online platforms, conferences, and workshops, to maximize its impact and reach within the NLP community and beyond, thereby enhancing efficiency, compliance, and decision-making processes.

# TEXT EXTRACTION

In a text summarizer project, the first step is to extract text from the input source, which could be a Word document or a text file, as well as more complex formats such as PDF. This process is known as text extraction and serves as the foundation for further processing, including summarization.

When it comes to PDFs, text extraction can be more challenging compared to other formats like plain text or Word documents. Extracting text from a PDF can be challenging for two primary reasons:

* Complex Layouts: PDFs often contain complex layouts with columns, tables, images, and different font styles that make text extraction non-trivial. Text might be positioned in non-linear ways across different parts of a page. (e.g. Research paper text which is stored in two columns.)
* Embedded Content: PDFs may contain embedded content such as images or scanned documents. In such cases, the text may not be directly accessible, and additional techniques like optical character recognition (OCR) may be required.

The documents that will be summarized in this project are from the banking domain and often contain a variety of complex elements, such as formulas, tables, graphs, captions, and images. These types of content can introduce additional challenges for the text summarizer, as they may be considered noise in the summarization process. Additionally, elements like headers and footers may contain information such as page numbers and side notes that are not directly relevant to the summarization task.

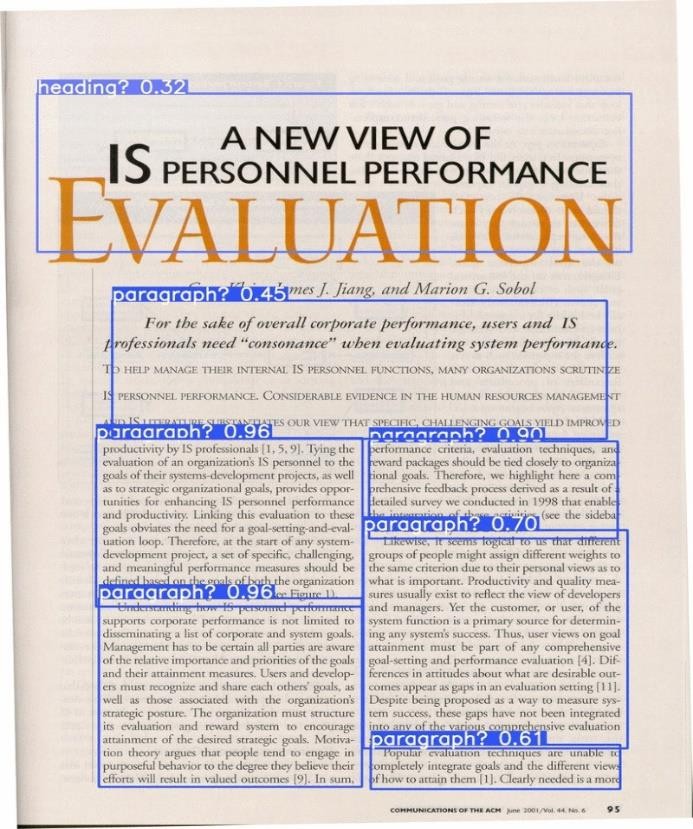
While the goal of the text summarizer is to generate concise and meaningful summaries from the input documents, the presence of these extraneous elements can lead to less accurate or less relevant summaries. This is because summarizer models typically perform best when the input text is clean and focused on the key information. The objective hence is to extract the main text body that contains the information required for the summary leaving out texts from the tables, captions, and formulas which will contaminate the summary.

However, the libraries commonly used for text extraction from PDFs, such as PyPDF2, PDFminer, Pdfplumber, pyMuPDF tend to extract all content as is from the input PDF, without distinguishing between essential and non-essential elements. This can result in the

inclusion of noise such as headers, footers, and formatting artefacts in the text that is fed to the summarizer.

### Object detection approach

To address the issue of unwanted noise in the extracted text from PDF documents, we used a comprehensive object detection approach. This method allows us to classify and differentiate between various elements in the PDF and retain only the essential ones for summarization.

Here's how the approach works:

* Convert PDF to Images: We begin by converting the input PDF document to a series of images, one for each page, so that we have a PNG format image of every page in the PDF document using the library fitz.
* Create Dataset with Annotations: Next, we create a dataset from the images, annotating the different elements present in the document. This includes labeling headers, footers, text blocks, tables, and formulas using bounding boxes. This annotated dataset serves as the training data for the object detection model.

Figure 1 Document layout using object detection

* Train the Object Detection Model: The annotated dataset is used to train an object detection model that can accurately identify and classify the different elements in the PDF document images. The model learns to recognize each type of content based on its visual characteristics.
* Classify Essential Elements: Once the model is trained, we use it to classify the different elements in a new PDF document's images. The model identifies headers, footers, text blocks, tables, and formulas, distinguishing between essential and non-essential content.
* Extract Text from Essential Elements: After classifying the elements, we focus on the essential content, such as text blocks. We extract the text from these elements while disregarding noise such as headers and footers.
* This approach significantly improves the quality of text extraction from PDF documents by eliminating noise and retaining only the most important elements. As a result, the summarizer model produces better, more useful summaries for the end user.

### YOLO (You Only Look Once)

YOLO (You Only Look Once) is a popular object detection model known for its speed and accuracy. It is a deep learning-based approach that revolutionized the field of object detection by offering a unified framework for both detecting and classifying objects in images. In the context of the text summarizer project, YOLO can be used to identify and classify various elements in PDF document images, such as headers, footers, text blocks, tables, and formulas.

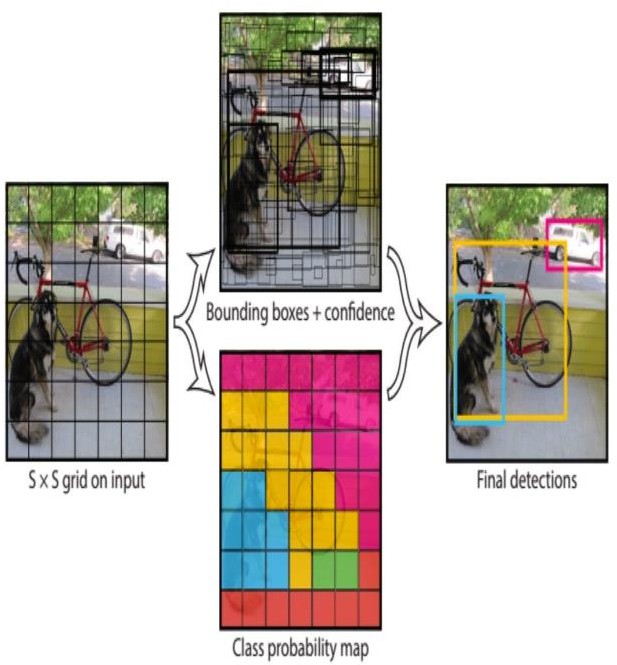


Figure 2 : Object detection using YOLO

#### HOW DOES YOLO OBJECT DETECTION WORK?

This first step starts by dividing the original image into NxN grid cells of equal shape. Each cell in the grid is responsible for localizing and predicting the class of the object that it covers, along with the probability/confidence value (indicating how sure the model is about the presence of an object).

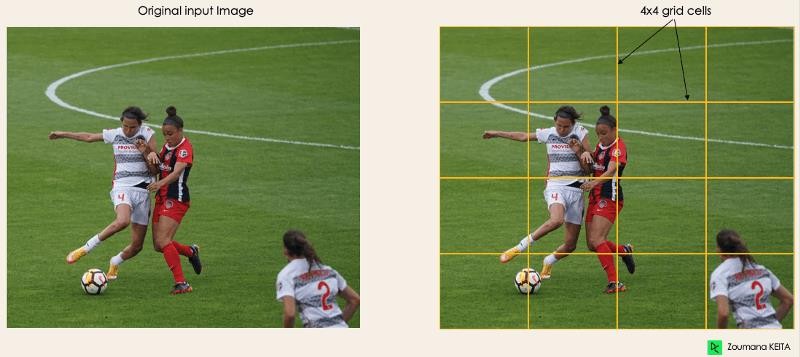


Figure 3 Grid cells used by YOLO Model

For each cell, YOLO predicts bounding boxes. Each bounding box is represented by four parameters: the center coordinates (x, y), the width w, and the height h.

During training, YOLO learns to adjust the bounding box predictions so that they match the ground truth boxes as closely as possible. This process is called regression because the model is refining its predictions based on the actual object locations.

YOLO determines the attributes of these bounding boxes using a single regression module in the following format, where Y is the final vector representation for each bounding box.

Y = [pc, bx, by, bh, bw, c1, c2] where,

pc: probability of the grid containing an object.

bx, by: x and y coordinates of the centre of the bounding box with respect to the enveloping grid cell.

bh, bw: height and width of the bounding box with respect to the enveloping grid cell.

c1,c2: Correspond to the two classes(Player and Ball in the image).There can be as many classes as the use case requires.

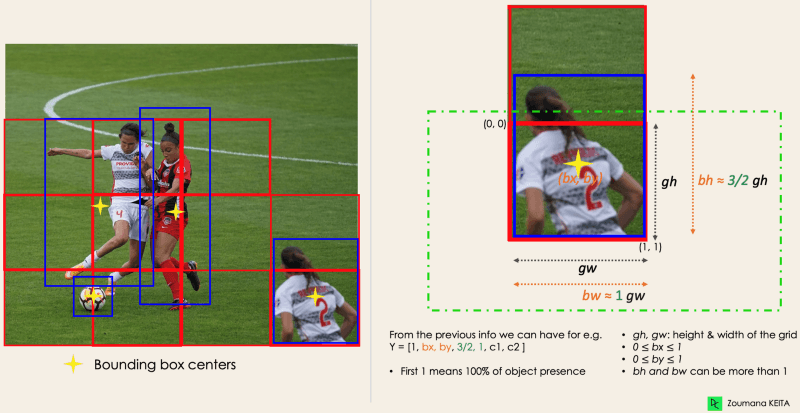


Figure 4 Bounding boxes for object prediction

Most of the time, a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The goal of the IOU (a value between 0 and 1 and measures the overlap between two boxes) is to discard such grid boxes to only keep those that are relevant. Here is the logic behind it:

* The user defines its IOU (Intersection Over Union) selection threshold, which can be, for instance, 0.5.
* Then YOLO computes the IOU of each grid cell which is the Intersection area divided by the Union Area.
* Finally, it ignores the prediction of the grid cells having an IOU ≤ threshold and considers those with an IOU > threshold.



Figure 5 Selection of bounding box based on IOU

If multiple boxes have IOU beyond the threshold, then the boxes with the highest probability score of detection are retained.

#### TRAINING THE YOLOV5 MODEL

We train the YOLOv5 model for object detection on our custom dataset using documents from the banking domain. Even though YOLOv8 is the latest version of the YOLO models we choose YOLOv5 because of the computational constraints.

Creating Custom Dataset

To train the YOLOv5 model effectively for the text summarizer project, it is essential to have a diverse and comprehensive dataset of annotated images. Initially, we used the “LabelImg” tool to create a dataset of 50 images, labelling different elements in the PDF document images such as headers, footers, text blocks, tables, and formulas using bounding boxes.

LabelImg is a user-friendly annotation tool that allows for efficient creation of datasets with labeled images.

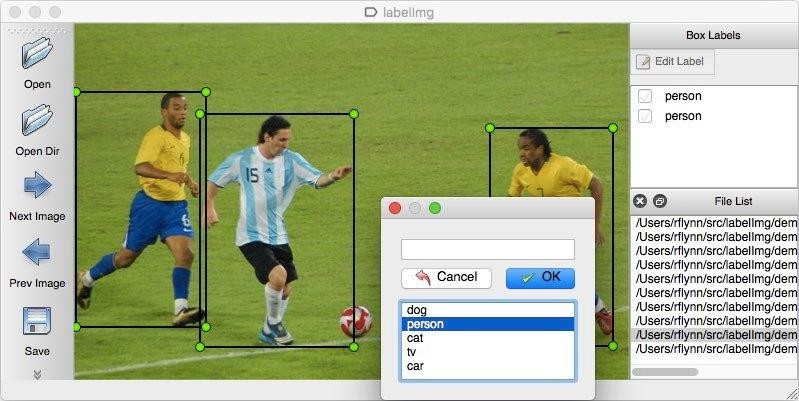


Figure 6 LabelImg for annotating image data

The dataset of 50 images was used to train the dataset. However, with a dataset of just 50 images, we were facing challenges in achieving optimum accuracy during training. The dataset was too small and lacked the diversity necessary for the model to generalize well across various types of PDF documents.

To overcome this limitation, we sought out a larger, more comprehensive dataset and found the [DocLayNet](https://github.com/DS4SD/DocLayNet) dataset online. [DocLayNet](https://github.com/DS4SD/DocLayNet) is a human-annotated document layout segmentation dataset containing 80,863 pages from a broad variety of document sources.

It provides several unique features such as:

* + - 1. Human Annotation: DocLayNet is hand-annotated by well-trained experts, providing a gold-standard in layout segmentation through human recognition and interpretation of each page layout.
      2. Large layout variability: DocLayNet includes diverse and complex layouts from a large variety of public sources in

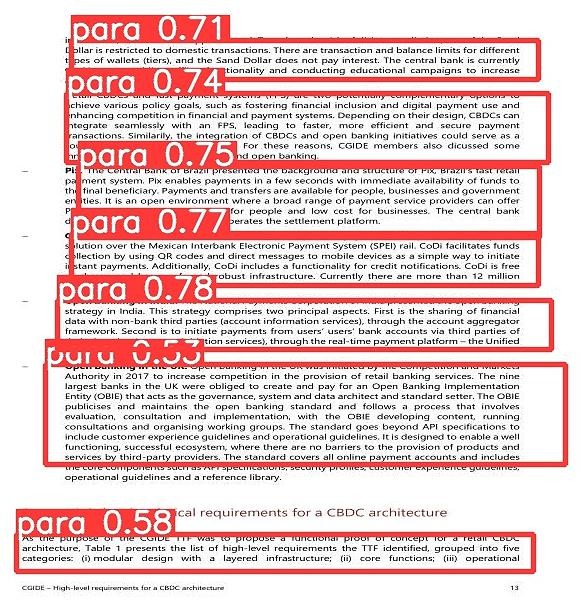


Figure 7 Prediction from YOLOv5 model

Finance, Science, Patents, Tenders, Law texts and Manuals.

* + - 1. Detailed label set: DocLayNet defines 11 class labels to distinguish layout features in high detail. The 11 Class labels include Caption, Footnote, Formula, List-item, Page- footer, Page-header, Picture, Section-header, Table, Text, Title.

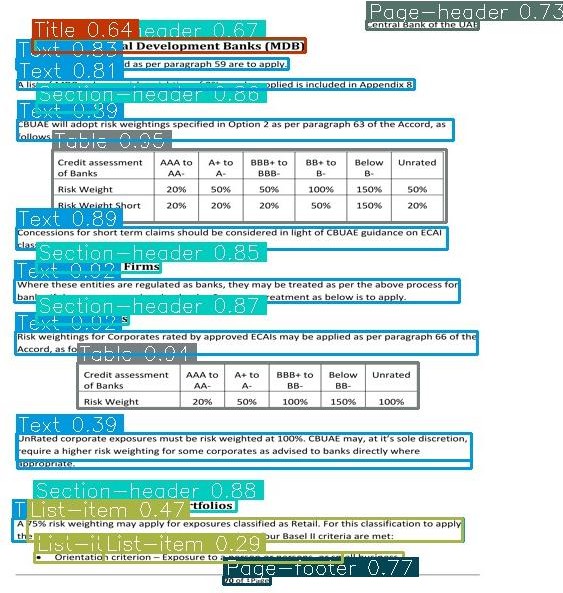


Figure 8 Prediction from YOLOv8 (trained on DocLayNet dataset)

We found an already trained YOLOv8 model (latest version of YOLO) on the DocLayNet dataset on GitHub by [LynnHaDo.](https://github.com/LynnHaDo/Document-Layout-Analysis) We tested the model on our dataset (images from BIS document PDFs) and it provided excellent results with high detection accuracy thus we integrated the model into our project.

Once the PDF is converted into a set of images, before feeding them to the model the images are resized to a size of 640x640. This size is required due to the architecture of the YOLO model also keeping input images at a consistent size helps standardize the input data, making it easier for the model to learn and generalize across different images. If the input dimensions of the image are not 640x640 it affects the model’s performance. The caveat of resizing the image is that there is a significant loss of image quality which may affect in extraction of text this problem is explained ahead in the OCR section.

### Cropping the images

The next step after detecting the text elements is to crop the images according to the bounding box coordinates.

cls: tensor([9.])

conf: tensor([0.9675])

data: tensor([[ 81.6268, 471.8428, 581.4836, 575.7029, 0.9675,

9.0000]])

id: None is\_track: False

orig\_shape: (640, 640)

shape: torch.Size([1, 6])

xywh: tensor([[331.5552, 523.7728, 499.8568, 103.8601]])

xywhn: tensor([[0.5181, 0.8184, 0.7810, 0.1623]])

xyxy: tensor([[ 81.6268, 471.8428, 581.4836, 575.7029]])

xyxyn: tensor([[0.1275, 0.7373, 0.9086, 0.8995]])

Figure 9 Output from YOLO model.

Shown above is a snippet of the output of a detection from YOLOv8.

Let's break down the different components of the output and explain their significance:

1. cls: This is the class label of the detected element. It represents the category of the object that has been detected. In the output, tensor([9.]) indicates that the detected object belongs to class 9.
2. These class labels are then mapped to the actual class names like Caption, Header, Table…etc.
3. conf: This is the confidence score of the detection. It measures the model's confidence in its detection, with a value between 0 and 1. In the output, tensor([0.9675]) indicates a high confidence score (96.75%) for the detection.
4. data: This is a tensor that contains the bounding box coordinates, confidence score, and class label of the detected element. In the output, the tensor contains the coordinates [[ 81.6268, 471.8428, 581.4836, 575.7029, 0.9675, 9.0000]], which

represent the bounding box coordinates (in terms of pixel positions) and confidence score, followed by the class label.

1. orig\_shape: This is the original shape of the image. In the output, (640, 640) shows that the original image was resized to 640x640 pixels for processing.
2. shape: This indicates the shape of the detection output. In the output, torch.Size([1, 6]) suggests that there is one detection (1 row) with six pieces of information (columns). xywh: This represents the bounding box coordinates in the form of center coordinates (x, y), width (w), and height (h). In the output, the tensor [[331.5552, 523.7728, 499.8568, 103.8601]] shows the center coordinates, width, and height of the bounding box.
3. xywhn: This represents the normalized bounding box coordinates in the same format as xywh, but the values are normalized relative to the image dimensions. In the output, the tensor [[0.5181, 0.8184, 0.7810, 0.1623]] shows the normalized coordinates.
4. xyxy: This represents the bounding box coordinates in the form of the top-left (xmin, ymin) and bottom-right (xmax, ymax) corners. In the output, the tensor [[ 81.6268, 471.8428, 581.4836, 575.7029]] shows these coordinates.
5. xyxyn: This represents the normalized bounding box coordinates in the same format as xyxy, but the values are normalized relative to the image dimensions. In the output, the tensor [[0.1275, 0.7373, 0.9086, 0.8995]] shows the normalized coordinates.

Once the classes and the bounding box coordinates are obtained, the values corresponding to class Headers, Footers, Formulas, Captions, Tables etc. are filtered out and only the class Title and Text are retained.

The coordinates for these blocks are stored in a list but before that they need to be sorted, out of the different notations provided by the YOLO output we use the xyxy notation i.e. (xmin,ymin and xmax,ymax notation).

Sorting the blocks depends on how the text is stored in the input document, primarily text is stored linearly in a single column or it is stored in two columns (like in a research paper).

If the text is stored in a single column the bboxes are sorted based on the ymin coordinate (ascending order) and stored in the list.



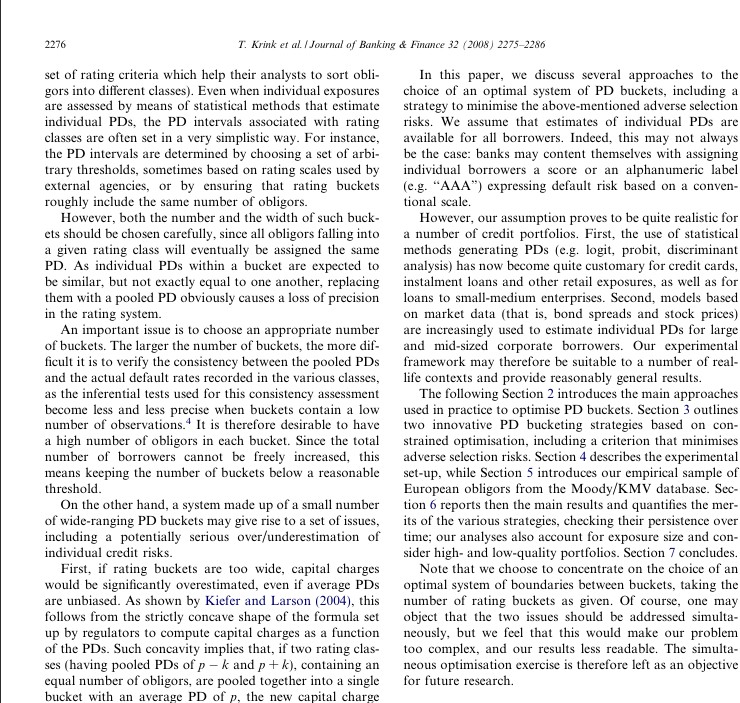
xmin,ymin

xmax,ymax

Figure 10 bbox coordinate representation

When the original document has text stored in two columns, the logical order of reading the text is to first read the text in the left column and then move to the column of text on the right-hand side. Accordingly, we use the image width and divide it by two to get the halfway

point of the x-axis and then every block with xmin on the left-hand side of the halfway point is sorted first based on the ymin and stored in the list and similarly, the text blocks with xmin on right side of the halfway point are sorted and appended to the list.



**Half-way line**

Figure 11 Process text in two column layout

Now the list contains the titles and the text bbox coordinates in the correct order as present in the input image, the next step is to access each element one by one and cropping the image according to the bbox coordinates such that the resulting cropped image only has that particular text element and them extracting the text from the cropped image using OCR.

OCR (Optical Character Recognition) is a technology that converts visual text in images, such as scanned documents or photographs, into machine-readable text. It allows for the extraction of text from images, enabling editing, searching, and processing of the content as text data. The OCR engine segments the image into smaller units such as lines, words, and individual characters. This helps the system focus on specific areas of text for more accurate recognition.

Once the image is segmented, the OCR engine uses pattern recognition algorithms to identify and recognize each character within the segmented text areas.

We use pytesseract for OCR in the project. pytesseract is a Python library that provides an interface for working with Tesseract, an open-source OCR engine. Pytesseract allows you to perform OCR on images, converting visual text into string format. It supports multiple languages and can extract text from various image formats, making it suitable for processing images from PDF documents.

Show below is an example list corresponding to the image,

List = [Section-header bbox 1, Text bbox 1, Text bbox 2, Text bbox 3, Section-header bbox 2, Text bbox 4, Text bbox 5, Section-header bbox 3, Text bbox 6, Text bbox 7]



Figure 12 Filtered detection

First, the Section-header 1 coordinates are accessed, and the image is cropped according to the access using OpenCV to obtain the image shown below.



Figure 13 Cropped title image.

Then we use pytesseract to extract the text from the image. The extracted text is then stored and then we move to the second element Text bbox 1 and again crop the image to obtain the below image.

A close up of text  Description automatically generated

Figure 14 Cropped text image.

This is then again followed by using pytesseract to extract the text, this process is repeated for all the elements in the page and all the pages in the documents so that finally all the relevant text is extracted and stored in a text file.

### Scaling the bboxes

The YOLO model requires input images to be resized to a dimension of 640x640 pixels to maintain consistency and efficiency during processing. However, this resizing can lead to a loss of quality in the images. When these resized images are cropped according to the bounding box coordinates, the loss of quality is further compounded.

This degradation in image quality negatively impacted the accuracy of OCR results when extracting text from the cropped images. To resolve this issue, we decided to work with the original, non-resized high-quality images.

To utilize the bounding box coordinates obtained from YOLO for the 640x640 pixel images, we scaled these coordinates back to match the dimensions of the original high-quality images. This scaling was done based on the ratio of the original image dimensions to the resized image dimensions.

𝑛𝑒𝑤 𝑥𝑚𝑖𝑛 = ⌊𝑥𝑚𝑖𝑛 ∗ 𝑥𝑠𝑐𝑎𝑙𝑒 ⌋

𝑛𝑒𝑤 𝑥𝑚𝑎𝑥 = ⌊𝑥𝑚𝑎𝑥 ∗ 𝑥𝑠𝑐𝑎𝑙𝑒 ⌋

𝑛𝑒𝑤 𝑦𝑚𝑖𝑛 = ⌊𝑦𝑚𝑖𝑛 ∗ 𝑦𝑠𝑐𝑎𝑙𝑒⌋

𝑛𝑒𝑤 𝑦𝑚𝑎𝑥 = ⌊𝑦𝑚𝑎𝑥 ∗ 𝑦𝑠𝑐𝑎𝑙𝑒⌋

(𝑥scale =

new\_width

)

old\_width

(𝑦scale =

new\_height

)

old\_height

Where 𝑛𝑒𝑤\_𝑤𝑖𝑑𝑡ℎ ,𝑛𝑒𝑤\_ℎ𝑒𝑖𝑔ℎ𝑡 are the width and height of the high quality input image.

𝑜𝑙𝑑\_𝑤𝑖𝑑𝑡ℎ , 𝑜𝑙𝑑\_ℎ𝑒𝑖𝑔ℎ𝑡 are the width and height of the resized 640x640 dimensioned images.

The obtained new xyxy coordinates represent the bbox for the same elements but scaled to the dimension of the high quality non-resized input images.

By feeding the high-quality images to OCR and using the scaled bounding box coordinates to accurately crop the essential elements, we were able to preserve the image quality and achieve better OCR results.

### Storing the text chapter-wise

After extracting the desired text from the PDF documents, we wanted to store the text chapter-wise. This approach was necessary because summarization models often have a limit on how much text they can process at a time, and it needs to be chunked into smaller manageable pieces of text as per the model input limit. If the entire document is fed into the

model without separating it by chapter, the model might split the text arbitrarily. For example, one chunk could contain content from the end of the first chapter and the beginning of the second chapter. This disjointed processing can lead to fragmented summaries that lack context and coherence.

To address this issue, we decided to generate summaries for each chapter individually. By maintaining the chapter structure and summarizing each chapter separately, we can ensure that the context and flow of information are preserved within each summary. This approach results in richer and more coherent summaries, as the model can focus on the specific content and context of each chapter.

To identify chapter titles in the text, we leveraged common patterns found in chapter titles. These patterns typically include numerical or Roman numeral prefixes, as well as keywords that signify a chapter start. Here are some examples of patterns in chapter titles that we used:

1. Numeric Prefixes: Titles often begin with numbers followed by periods or parentheses, such as "1.", "2.", "3)", and so on.
2. Roman Numerals: Titles may also use Roman numerals at the beginning, followed by periods or other punctuation, such as "I.", "II.", "III:", and so on.
3. Section Keywords: Titles may include keywords like "Section," "Chapter," or "Part" along with numbers, for example, "Section 1", "Chapter 2", or "Part III."
4. Combination of Prefixes: Sometimes titles combine numeric or Roman numeral prefixes with keywords such as "1. Introduction" or "II. Methodology."

We used regular expressions (regex) to find these patterns within the text. Regex provides a powerful and flexible way to search for specific text patterns. By crafting regex patterns that match the different ways chapter titles might be formatted (e.g., "1.", "I.", "Section 1"), the chapter titles are located.

Given that we are using OCR (Optical Character Recognition) technology, which is not 100% accurate, there are sometimes variations in the extracted text. For instance, the OCR might convert uppercase letters to lowercase, or it may struggle to recognize Roman numerals accurately. These discrepancies made it challenging to precisely identify chapter headings using the OCR-extracted text alone.

To address this issue we used Tika library, which is a standard library for extracting text from PDFs as is, it maintains the exact structure as input documents and includes everything present in the input document from header to footer (including images) in the extracted output. This means that the titles as well are extracted as is accurately. We used the regex patterns to search for title headings in the Tika extracted text. It gave us better accuracy in finding a match and fetching the titles.

Once the titles were fetched and stored in a list, we wanted to find these titles in the text from OCR (text without headers, footers, tables etc.) so that it can be structured in a chapter-wise format. We used the fuzzywuzzy library to match these titles within the OCR-extracted text.

Fuzzywuzzy is a library that performs fuzzy string matching, allowing us to compare text even if it is not an exact match. The library provides a similarity score for each comparison,

indicating how closely the two strings match. We set the similarity score threshold to 90% to ensure a high level of accuracy in matching chapter titles.

Once we found a match of the chapter title in the OCR-extracted text, we obtained the index of the title in the text. This index allowed us to segment the text and store it chapter-wise.

Finally, we have a list where each element in the list is a string which contains the text from the chapter.

chapter\_wise\_list = [chap1 text, chap 2 text, chap3 text….]

Summaries are generated finally for each of these elements and then clubbed together to get a coherent document summary.

# LSTM – LONG SHORT TERM MEMORY

### Introduction

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network architecture, have shown significant promise for text summarization tasks. LSTMs are particularly well- suited for processing sequential data, such as text, and can effectively capture long-range dependencies within the input sequence.

In the context of text summarization, LSTMs are often employed in an encoder-decoder framework. The encoder LSTM processes the input text sequentially, encoding it into a fixed- length vector representation (the context vector). This context vector is then passed to the decoder LSTM, which generates the summary one word at a time, conditioned on the context vector and the previously generated words.

The key advantage of LSTMs over traditional RNNs lies in their ability to selectively remember and forget information from previous time steps. This is achieved through the use of memory cells and a set of gates (input, output, and forget gates) that control the flow of information into and out of the memory cell. This gating mechanism allows LSTMs to better capture long-range dependencies, which is crucial for summarizing longer texts effectively.

To further enhance the summarization performance, attention mechanisms are often incorporated into LSTM-based models. The attention mechanism enables the decoder to selectively focus on different parts of the input text when generating each word in the summary, rather than relying solely on the fixed-length context vector. This allows the model to better capture the most relevant information from the input text and generate more accurate and coherent summaries.

Training an LSTM-based text summarizer involves optimizing the model's parameters using backpropagation and techniques like teacher forcing, where the ground truth summary is fed as input during training to help the decoder learn to generate accurate summaries. The model is trained on a large dataset of input texts and their corresponding summaries.

During inference, the trained LSTM model takes the input text, encodes it using the LSTM encoder, and then uses the decoder to generate the summary word by word, based on the encoded context vector and the previously generated words.

Effective text summarization using LSTMs often requires careful data preprocessing, hyperparameter tuning, and techniques like regularization and beam search to improve the quality of the generated summaries. Additionally, incorporating domain-specific knowledge or leveraging pre-trained language models can further enhance the performance of LSTM- based summarization models.

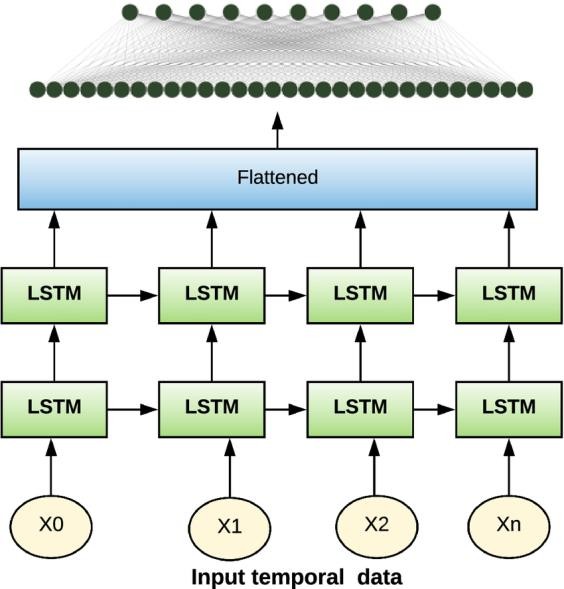


Figure 15 LSTM Architecture

### Embeddings

Word Embeddings: The model utilizes pre-trained word embeddings (e.g., GloVe or FastText) to convert words into dense vectors. These embeddings capture semantic meanings and relationships between words, providing a robust foundation for the model to understand and generate text.

### Training process

The training of the LSTM model was meticulously planned to optimize performance:

1. Parameters: The model was trained using an Adam optimizer, known for its efficiency in handling sparse gradients on noisy problems. A sparse categorical crossentropy loss function was used to compute the loss between the predicted probabilities and the actual output.
2. Batch Size and Epochs: The model was trained with a batch size of 64 for up to 100 epochs, with early stopping implemented to prevent overfitting. The training process

was monitored using a validation split of 20% to assess the model's performance on unseen data.

1. Callbacks: Callbacks such as Model Checkpoint were used to save the best model based on validation loss, and Early Stopping was used to halt training when the validation loss ceased to decrease for two consecutive epochs.

### Challenges

Throughout the project, several challenges were encountered:

1. Handling Long Sequences: LSTM models can struggle with very long sequences due to the vanishing gradient problem. To address this, sequences were truncated to a manageable length, and attention mechanisms were employed to help the model focus on relevant parts of the input.
2. Vocabulary Size: A large vocabulary can make the model slow and inefficient. To mitigate this, words that appeared infrequently in the dataset were removed, and tokenization techniques were optimized.
3. Resource Intensiveness: Training LSTMs on large datasets requires considerable computational resources. This is because the model needs to update parameters across many layers and over long sequences, which is computationally expensive. Employing more efficient hardware or optimizing model architecture can help manage resource usage.
4. Complexity of Understanding Context: LSTM models, while good at capturing sequences, often face difficulty in understanding broader context or the global significance of a text. This can lead to summaries that might miss nuanced or contextual meanings essential for a coherent summary. Techniques like hierarchical attention have been tested to improve context understanding.
5. Long Training Times: Due to the recurrent nature of LSTMs, training times can be lengthy especially for very large datasets. This can slow down iterations and improvements in model development. Using accelerated computing resources or simplifying the model are common strategies to reduce training times.

Thus, due to these major challenges faced by us this approach was dropped and Hugging Face transformer models were considered.

# TRANSFORMERS

HuggingFace Transformers provides APIs and tools to easily download and train state-of-the- art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch. These models support common tasks in different modalities, such as:

1. Natural Language Processing: text classification, named entity recognition, question answering, language modelling, summarization, translation, multiple choice, and text generation.
2. Computer Vision: image classification, object detection, and segmentation.
3. Audio: automatic speech recognition and audio classification.
4. Multimodal: table question answering, optical character recognition, information extraction from scanned documents, video classification, and visual question answering.

### Transfer learning

Pretraining is the act of training a model from scratch: the weights are randomly initialized, and the training starts without any prior knowledge. This pretraining is usually done on very large amounts of data. Therefore, it requires a very large corpus of data, and training can take up to several weeks. Fine-tuning, on the other hand, is the training done after a model has been pretrained. To perform fine-tuning, you first acquire a pretrained language model, then perform additional training with a dataset specific to your task. The fine-tuning will only require a limited amount of data: the knowledge the pretrained model has acquired is “transferred,” hence the term transfer learning.

### Transformer models

Transformers are language models. All the Transformer models mentioned above (GPT, BERT, BART, T5, etc.) have been trained as language models. This means they have been trained on large amounts of raw text in a self-supervised fashion. Self-supervised learning is a type of training in which the objective is automatically computed from the inputs of the model. That means that humans are not needed to label the data!

This type of model develops a statistical understanding of the language it has been trained on, but it’s not very useful for specific practical tasks. Because of this, the general pretrained

model then goes through a process called transfer learning. During this process, the model is fine-tuned in a supervised way — that is, using human-annotated labels — on a given task.

#### GENERAL ARCHITECTURE OF TRANSFORMER MODELS

The general architecture of transformer models involves an encoder-decoder structure that does not rely on recurrence and convolutions. Transformers are neural networks designed for sequence transduction tasks, transforming input sequences into output sequences. The encoder maps input sequences to continuous representations, which are then passed to the decoder for generating output sequences. The key components of the transformer architecture include:

1. Encoder: The encoder processes input sequences and generates matrix representations of the input. It consists of multiple layers, each with the same structure, and uses self- attention mechanisms to compute representations of the input sequence.
2. Decoder: The decoder takes the encoded representations from the encoder and iteratively generates output sequences. Like the encoder, the decoder also comprises multiple layers with the same structure, and it uses self-attention and positional encodings to generate output sequences.
3. Self-Attention: Transformers rely heavily on self-attention mechanisms to understand the relationships between different elements in a sequence, allowing them to comprehend context and meaning without recurrent connections.
4. Positional Encoding: Positional encodings are added to the input embeddings of the decoder to introduce positional information into the input sequences.
5. Fully Connected Layers: Transformers include fully connected feed-forward networks in both the encoder and decoder to process the representations and generate predictions for the next elements in the sequence.
6. Bidirectional Processing: The encoder in a transformer is bidirectional, attending to all words in the input sequence, while the decoder works sequentially, attending only to the words it has already translated.

This architecture allows transformers to excel in tasks like machine translation, document summarization, named entity recognition, and various other natural language processing applications. Transformers have also shown success in computer vision and protein folding applications, demonstrating their versatility and potential for real-world applications.

# SUMMARIZATION TECHNIQUES

This section delves into the modelling techniques employed in document summarization, highlighting two prominent methodologies: abstractive models and extractive models.

1. Abstractive Models: In the realm of abstractive summarization, models generate summaries that may contain rephrased or even novel sentences not present in the source document. These models leverage advanced natural language understanding capabilities, often based on transformer architectures, to comprehend the context of the document and generate human-like summaries.
2. Extractive Models: Conversely, extractive summarization techniques focus on identifying and selecting the most salient sentences or passages directly from the source document. These models do not generate new text but rather extract and reassemble existing content, prioritizing coherence and relevance to produce a condensed version of the document.

In the following sections, we explore the intricacies of both abstractive and extractive approaches, examining their methodologies, strengths, limitations, and applications in document summarization tasks.

# ABSTRACTIVE SUMMARIZATION MODEL SELECTION

This section will delve into the abstractive summarization techniques in which we explore three models namely, Longformer Encoder Decoder, T5 – small and Pegasus/CNNDailyMail. Our objective was to determine which model would generate the most coherent and high quality summaries for the given Bank Regulatory documents.

### Longformer Encoder Decoder(LED)

Most summarization models have a token size limit of either 512 or 1024 tokens. This means that when dealing with long documents, we must break them down into smaller chunks to fit within these constraints. Splitting a document into chunks can lead to a loss of context, as each chunk is processed independently, without necessarily retaining connections between them.

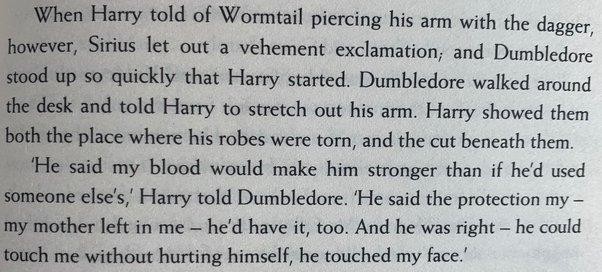


Figure 16 Processing with low token size.

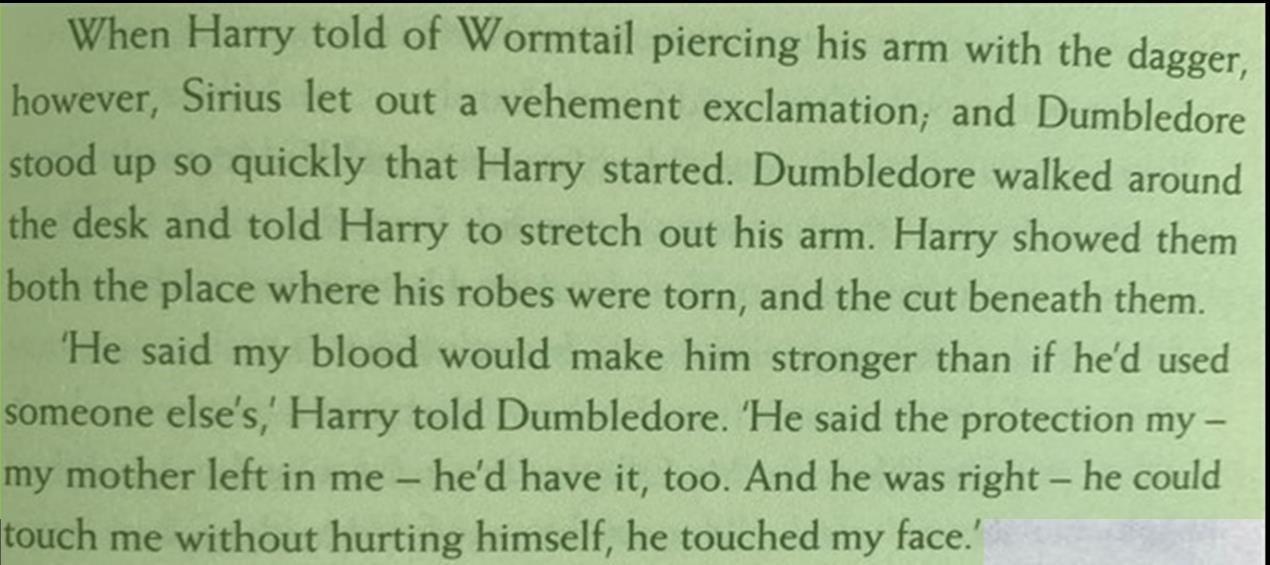


Figure 17 Processing with high token size.

Consider the example above the paragraph is broken down into smaller chunks, as both the chunks will be processed separately if a work in yellow chunk is referenced in the red chunk as the model cannot read the yellow chunk while processing the red chunk important context is lost while summarization.

This loss of context can be especially problematic in complex, structured documents such as those from the banking domain, where important information may be spread across multiple sections. When context is lost, the summarization may become disjointed and miss critical relationships between different parts of the document.

To address this issue, we chose the Longformer Encoder-Decoder (LED) model for summarization. This model has a larger input token size limit of 16,384 tokes, allowing it to process larger portions of text at once. By handling more of the document in a single pass, LED maintains context and continuity, resulting in more cohesive and accurate summaries that better reflect the original content and structure of the source material.

##### How LED processes large texts

As mentioned in [Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-](https://arxiv.org/pdf/2004.05150.pdf) [document transformer. arXiv preprint arXiv:2004.05150.](https://arxiv.org/pdf/2004.05150.pdf) The Longformer Encoder-Decoder (LED) model is able to process large amounts of tokens in a document effectively because it uses an attention mechanism optimized for handling long sequences of text.

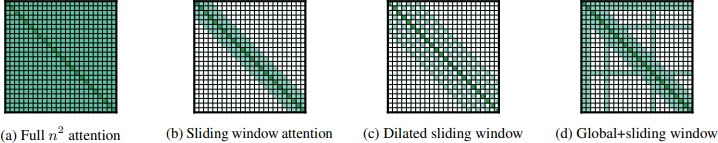


Figure 18 Attention mechanism in LED

This attention mechanism includes the following key components:

* + - 1. Sliding Window Attention: In traditional models, every token is compared with every other token, which can become slow and inefficient for longer texts. LED uses a sliding window approach instead, where each token attends only to its neighbors within a specific range. This means the model focuses on and understands the relationships between nearby words, saving processing time and resources.
      2. Dilated Sliding Window Attention: To improve the model's ability to capture longer- range dependencies in the text, LED introduces dilated attention, which allows the model to attend to words at intervals (e.g., every second, third, or fourth word) within the sliding window. This helps the model gather information from a broader context without increasing the computation too much.
      3. Global Attention: In addition to the sliding window and dilated sliding window attention, LED uses global attention on certain tokens, such as the start of a document, the end of a document, and the titles of sections. This global attention mechanism allows the model to attend to important or pivotal points throughout the document, ensuring that key information from different sections can be connected and understood as a whole.

##### LED pretrained on Booksum

Specifically we are using the [pszemraj/led-base-book-summary](https://huggingface.co/pszemraj/led-base-book-summary) from Hugging Face which is a LED model([led-base 16384 by Allenai](https://huggingface.co/allenai/led-base-16384)) pretrained on the BookSum Dataset.

BookSum is a collection of datasets for long-form narrative summarization. This dataset covers source documents from the literature domain, such as novels, plays and stories, and includes highly abstractive, human written summaries on three levels of granularity of increasing difficulty: paragraph-, chapter-, and book-level. The domain and structure of this dataset poses a unique set of challenges for summarization systems, which include processing very long documents, non-trivial causal and temporal dependencies, and rich discourse structures.

Model trained on BookSum has experience working with long texts (such as books). This familiarity with extended passages and their structure is advantageous when dealing with similarly lengthy banking documents. BookSum-trained models are skilled in capturing context and maintaining flow across long texts, they can help ensure that the summary preserves the overall meaning and context of lengthy banking documents.

### T5 – Text to Text Transfer Transformer

The T5 (Text-To-Text Transfer Transformer) model is a versatile transformer-based architecture developed by researchers at Google. Unlike many other transformer models that are designed for specific tasks such as translation, text generation, or question answering, T5

is formulated as a text-to-text framework. This means that all tasks, including summarization, are cast into a unified text-to-text format, where both inputs and outputs are textual strings.

##### General architecture of the T5 model

The architecture of T5 is similar to other transformer models such as BERT and GPT, consisting of stacked encoder-decoder layers. However, what sets T5 apart is its simplicity and consistency in design. T5 uses the same transformer architecture for both encoder and decoder, making it easier to train and fine-tune. It employs a bidirectional architecture, allowing it to effectively capture contextual information from both past and future tokens.

T5's architecture consists of several key components:

* + - 1. Tokenization: T5 uses a Byte Pair Encoding (BPE) tokenizer to tokenize input text into subword units. This tokenizer helps handle out-of-vocabulary words and improves the model's ability to generalize.
      2. Encoder-Decoder Layers: T5 employs a stack of transformer encoder-decoder layers. Each layer consists of self-attention mechanisms and feed-forward neural networks. The encoder processes the input text, while the decoder generates the output text.
      3. Pre-training and Fine-tuning: T5 is pre-trained on large-scale text corpora named C4 (Colossal Clean Crawled Corpus) dataset using unsupervised learning objectives such as denoising autoencoding or masked language modelling. After pre-training, the model can be fine-tuned on downstream tasks like text summarization using task- specific datasets.

#### T5 – SMALL MODEL

The T5 Small model is a scaled-down version of the larger T5 models, designed to offer a balance between performance and computational efficiency. Here are the key architectural components and specifications of the T5 Small model:

**Overall Model Size**: The T5 Small model has approximately 60 million trainable parameters.

**Encoder**:

* + - 1. The encoder consists of 6 layers.
      2. Each encoder layer has 8 attention heads.
      3. The dimension of the input and output embeddings in the encoder is 512.
      4. The feed-forward layer in each encoder layer has an inner dimension of 2048.

**Decoder**:

1. The decoder also consists of 6 layers.
2. Similar to the encoder, each decoder layer has 8 attention heads.
3. The dimension of the input and output embeddings in the decoder is 512.
4. The feed-forward layer in each decoder layer has an inner dimension of 2048.

**Attention Mechanism**:

1. Both the encoder and decoder utilize multi-head self-attention mechanisms to capture long-range dependencies in the input and output sequences, respectively.
2. In addition, the decoder employs a cross-attention mechanism to attend to the encoder's output representations.

**Positional Encodings**:

1. Since the Transformer architecture lacks recurrence or convolutions, positional encodings are added to the input embeddings to incorporate positional information.
2. The T5 Small model uses learnable positional embeddings.

**Feed-Forward Networks**:

1. Each encoder and decoder layer includes a feed-forward neural network with a ReLU activation function and a dropout layer.
2. The feed-forward layers in the T5 Small model have an inner dimension of 2048.

**Normalization Layers**:

1. Layer normalization is applied before the attention and feed-forward layers in both the encoder and decoder.

**Activation Functions**:

1. The T5 Small model primarily uses the Gaussian Error Linear Unit (GELU) activation function in the feed-forward layers.

### Pegasus

Pegasus, which stands for "Pre-training with Extracted Gap-sentences for Abstractive Summarization Sequence-to-sequence," is a state-of-the-art text summarization model developed by Google Research. This model is specifically designed for the task of abstractive

text summarization, which involves generating concise and coherent summaries from longer text documents in a way that mimics human-like understanding and phrasing.

The mechanism of gap generation in the PEGASUS model is a key component of its pre- training approach, designed to improve the model's performance in text summarization tasks. This technique, unique to PEGASUS, simulates the summarization process during the model's training phase in a way that's distinct from traditional language model pre-training methods.

The mechanism followed is as follows:

**Selection of Gap Sentences**

The first step involves identifying and selecting key informative sentences from the input text document or article. These sentences are deemed critical in capturing the essence and main points of the content. The selection process aims to pinpoint sentences that, if summarized effectively, would provide a concise yet comprehensive overview of the text.

**Removal and Masking**

Once the key sentences are identified, they are removed from the original text, creating gaps at the positions where these sentences originally appeared. The removed sentences are essentially masked or hidden from the remaining text, leaving only the surrounding context.

**Gap Sentence Generation**

With the gaps created by the removal of key sentences, the model is trained to predict or generate the missing sentences based solely on the contextual information present in the remaining text. The objective is for the model to learn to reconstruct the removed sentences by understanding the context and extracting the most relevant information.

This training process encourages the model to develop a deep understanding of the text's content and to identify the most important information necessary for generating concise summaries.

By following this mechanism, the model is effectively trained to produce concise summaries that capture the essence of the original text. The selection of gap sentences ensures that the model focuses on the most critical information, while the removal and masking of these sentences forces the model to rely on contextual cues and understanding to reconstruct the missing content.

The generation of gap sentences requires the model to distil the most salient information from the surrounding context, promoting the development of summarization capabilities that prioritize conciseness and clarity. This approach helps the model learn to identify and synthesize the key points while avoiding unnecessary verbosity or redundancy.

##### Pegasus and CNN/DailyMail

The CNN/DailyMail dataset is commonly used for training and benchmarking summarization models. It consists of news articles from the CNN and DailyMail websites, paired with multi- sentence summaries. These summaries are professionally written, often appearing as bullet points below the article headlines, making them ideal for training models on a summarization task.

When applied to the CNN/DailyMail dataset, Pegasus has demonstrated remarkable performance, outperforming many other models in generating accurate and coherent summaries. The model leverages the inherent structure of news articles and summaries in this dataset, which helps in learning how to extract and condense the most critical information from journalistic texts.

# EXTRACTIVE SUMMARIZATION MODEL SELECTION

In this section, we discuss 3 models, BERT, BART and T5-base.

### BERT – Bidirectional Encoder Representations from Transformers

The BERT-Extractive-Summarizer takes advantage of the robust capabilities of BERT (Bidirectional Encoder Representations from Transformers) to provide efficient extractive summarization of texts. This method is distinct from abstractive summarization as it directly selects the most informative sentences from the original text. This ensures that the original tone and factual content are preserved, which is particularly important in contexts where accuracy is critical.

The process begins with the initial breakdown of the input text into manageable segments or sentences. This segmentation is crucial as it prepares the text for processing by fitting it within the operational constraints of BERT. Each sentence is then processed individually to obtain contextual embeddings. These embeddings are detailed vector representations that capture not only the content but also the contextual significance of each sentence within the larger text.

After processing, each sentence is scored based on its relevance and informativeness. The scoring mechanism can be customized and enhanced by fine-tuning BERT on specific domains or integrating additional parameters such as sentence position, length, and key term frequency. The highest-scoring sentences are then selected for inclusion in the summary. The flexibility in the number of sentences chosen allows the summary length and detail level to be tailored to specific needs.

Finally, the selected sentences are compiled in their original order to maintain logical coherence in the summary. This method produces a concise and accurate representation of the original text, making the BERT-Extractive-Summarizer particularly useful in legal, academic, and scientific domains where the precise replication of original information is crucial. By relying on direct text extraction, this tool ensures quick summarization without the introduction of the biases or inaccuracies that can sometimes arise with sentence reformulation in abstractive methods.

The pipeline followed for BERT is slightly different from other transformer models as it is pre-trained to predict masked words within a sentence, thus learning deep bidirectional representations. The pipeline is as follow:

1. **Text Preprocessing**: This initial phase involves cleaning and preparing the raw text data for further processing. It may include steps such as removing special characters, lowercasing the text, correcting typos. The goal is to normalize the text and reduce noise that could detract from the model's performance.
2. **Sentence Tokenization**: In this stage, the preprocessed text is broken down into individual sentences. Tokenization is the process of converting the continuous text into discrete units (tokens), which in this case are sentences. Sentence tokenization is essential for extractive summarization because the model will consider each sentence as a potential candidate for inclusion in the summary.
3. **Sentence Embeddings**: Here, the tokenized sentences are passed through the BERT model, which generates a numerical representation for each one, known as embeddings. These embeddings capture the contextual meaning of sentences, including the nuances influenced by the surrounding text. The high-dimensional vectors resulting from this process are used to understand and compare the semantic similarity between sentences.
4. **Clustering**: With sentence embeddings computed, clustering algorithms can be applied to group together sentences that convey similar meanings or discuss similar topics. This step helps in identifying which sentences cover the key points or themes in the text. Sentences that are central to their respective clusters often contain the core ideas of the text and are strong candidates for the summary.
5. **Summary Generation**: The final step is to generate the summary. The model selects sentences from the text, typically those that are most central or representative of their clusters. These sentences are then compiled to form a summary. The aim is to choose sentences that, when combined, provide a coherent and concise version of the original text, covering all major points without redundancies.

#### RATIONALE FOR USING BERT WITH A PRE-TRAINED PIPELINE FOR EXTRACTIVE SUMMARIZATION.

For the BERT extractive summarization model, we utilized a pre-existing library developed by third-party researchers, which precluded the ability to finetune the model directly.

**Precision in Extractive Summarization**

BERT's ability to precisely identify key sentences ensures summaries are 100% extractive, maintaining the original text's integrity.

**Efficiency in Processing Time**

The BERT-based pipeline offers rapid summarization by directly extracting sentences, significantly cutting down processing time.

### BART - Bidirectional and Auto-Regressive Transformers

BART (Bidirectional and Auto-Regressive Transformers) is a versatile and powerful language model developed by Facebook AI. BART is trained by (1) corrupting text with an arbitrary noising function, and (2) learning a model to reconstruct the original text. It achieves new state of-the-art results on a range of abstractive dialogue, question answering, and summarization tasks.

BART is trained as a denoising autoencoder, meaning it takes in corrupted or masked text and learns to reconstruct the original, uncorrupted text. This training strategy helps the model understand contextual information and improve its performance on various tasks, including summarization.

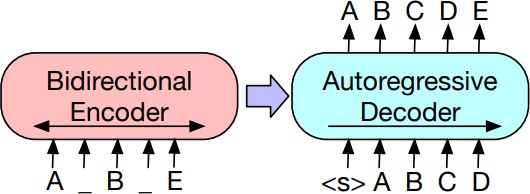


Figure 19 Denoising Objective used to train BART

Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder.

This training strategy helps the model understand contextual information and improve its performance on various tasks, including summarization.

It is pre-trained on a large corpus of diverse datasets:

1. SQuAD (Rajpurkar et al., 2016)a an extractive question answering task on Wikipedia paragraphs.
2. MNLI (Williams et al., 2017), a bitext classification task to predict whether one sentence entails another.
3. ELI5 (Fan et al., 2019), a long-form abstractive question answering dataset.
4. XSum (Narayan et al., 2018), a news summarization dataset with highly abstractive summaries.
5. ConvAI2 (Dinan et al., 2019), a dialogue response generation task, conditioned on context and a persona.

Training on such diverse datasets help the model enhance understanding of the language and also improves generalization making it more versatile and capable of handling variety of content.

##### BART on CNN/DailyMail

We use the facebook/bart-large-cnn which is the BART large fine-tuned on the CNN, Daily Mail dataset.

CNN/Daily Mail is a dataset for text summarization. Human generated abstractive summary bullets were generated from news stories in CNN and Daily Mail websites as questions (with one of the entities hidden), and stories as the corresponding passages from which the system is expected to answer the fill-in the-blank question.

CNN/Daily Mail dataset provides the model with exposure to a variety of topics and styles of writing, enhancing its ability to handle a wider range of summarization tasks.

Summaries on the CNN/Daily Mail are typically closely related to source sentences which enables the model to generate extractive like summaries.

### T5 - Base

The T5 Base model is a larger variant of the T5 architecture, offering improved performance compared to the T5 Small model while still being computationally feasible for many applications. Here are the key architectural components and specifications:

**Overall Model Size**

The T5 Base model has approximately 220 million trainable parameters.

**Encoder**:

1. The encoder consists of 12 layers.
2. Each encoder layer has 12 attention heads.
3. The dimension of the input and output embeddings in the encoder is 768.
4. The feed-forward layer in each encoder layer has an inner dimension of 3072.

**Decoder**:

1. The decoder also consists of 12 layers.
2. Each decoder layer has 12 attention heads.
3. The dimension of the input and output embeddings in the decoder is 768.
4. The feed-forward layer in each decoder layer has an inner dimension of 3072.

**Attention Mechanism**:

1. Similar to the T5 Small model, both the encoder and decoder utilize multi-head self- attention mechanisms and cross-attention mechanisms.

**Positional Encodings**:

1. Like the T5 Small model, the T5 Base model uses learnable positional embeddings.

**Feed-Forward Networks**:

1. Each encoder and decoder layer includes a feed-forward neural network with a ReLU activation function and a dropout layer.
2. The feed-forward layers in the T5 Base model have an inner dimension of 3072.

**Normalization Layers**:

1. Layer normalization is applied before the attention and feed-forward layers in both the encoder and decoder.

**Activation Functions**:

* 1. The T5 Base model primarily uses the Gaussian Error Linear Unit (GELU) activation function in the feed-forward layers.

The key differences between the T5 Base model and the T5 Small model lie in the increased number of layers (12 vs. 6), attention heads (12 vs. 8), and the dimensionality of the embeddings and feed-forward layers (768 vs. 512 and 3072 vs. 2048, respectively).

These architectural changes result in a significantly larger model size (220 million parameters vs. 60 million parameters) and increased computational requirements. However, the larger model size and increased capacity of the T5 Base model led to better performance on text summarization task compared to the T5 Small model.

# DATA ACQUISITION AND PREPARATION FOR MODEL FINE TUNING

In preparation for the finetuning phase aimed at enhancing our model's text summarization capabilities, a comprehensive corpus of paired instances comprising documents and their corresponding summaries was essential. These summaries, also referred to as true or reference summaries, serve as pivotal benchmarks for model evaluation.

To procure this dataset, we systematically evaluated various online tools including popAI, scholarcy, claude AI, perplexity AI, and smmry.com. Following meticulous manual analysis of the summaries generated by each tool and thorough deliberation on their respective strengths and limitations, we made informed selections.

For the generation of abstractive summaries, we opted for the advanced AI tool, perplexity AI, which provided accurate chapter wise abstractive summaries. Complementarily, for extractive summaries, we leveraged the capabilities of smmry.com, a reliable online tool renowned for its proficiency in generating precise chapter wise extractive summaries.

The selection of documents was strategically centred on the financial domain, drawing from reputable sources such as the BASEL website, RBI guidelines, CBUAE guidelines, and pertinent regulatory frameworks. Consequently, we assembled a corpus comprising 100 meticulously curated documents along with their corresponding chapter wise summaries.

This meticulously curated corpus serves as the cornerstone for the subsequent finetuning phase of each model, ensuring optimal performance and relevance to our target domain.

# MODEL FINETUNING

For this project utilizing transformers, we evaluated a total of six transformer-based models – three for abstractive summarization and two for extractive summarization. Owing to the utilization of a third-party library for the BERT extractive summarization model, the option to finetune the model itself was unavailable. Although the process of finetuning was carried out on all five models, we have chosen to detail the finetuning and generation parameter selection processes specifically for Pegasus (the finalized model for abstractive summarization) and T5 (the finalized model for extractive summarization) in this report. This decision was made because Pegasus and T5 not only produced the best summaries in terms of quality and coherence but also achieved the highest scores across various evaluation metrics, as discussed in the [model evaluation](#_bookmark47) section. By concentrating on these top-performing models, we aim to provide a comprehensive understanding of the most successful techniques employed, while acknowledging that similar finetuning and parameter tuning processes were undertaken for the other models explored during the project.

### Pegasus model for abstractive summarization

In the initial phase of our model training, we adopted a methodical approach to parameter selection. This process began with the application of five PDF documents to closely monitor validation loss, laying the groundwork for understanding how different parameters influenced performance on a manageable scale. The selection of parameters was a deliberate process of trial and error, allowing us to pinpoint effective settings before scaling up. We engaged in an iterative fine-tuning process, leveraging systematic grid analysis to assess the impact of various parameter combinations on loss metrics. This meticulous procedure was pivotal for optimizing memory usage, a critical consideration when training more extensive, resource- intensive models.

For the training approach, we capitalized on the benefits of mixed precision training (fp16), which substantially accelerated the training duration and reduced memory demands without sacrificing the model's precision. Concurrently, we implemented gradient accumulation, a technique that amasses gradients over multiple steps, thus facilitating the use of larger batch sizes that hardware limitations would typically preclude. To ensure the training progressed smoothly, we utilized the Hugging Face Trainer API, which streamlined the process with built-in efficiencies. The API's robust error handling and checkpointing capabilities were instrumental, allowing for uninterrupted training and the ability to resume from saved

checkpoints. These strategic decisions were integral in achieving a stable and effective training regime for the large Pegasus model, demonstrating the success of combining thoughtful parameter selection with an optimized training methodology.

|  |  |
| --- | --- |
| **Parameters** | **Significance** |
| Epochs | represents one complete cycle through the entire training dataset, used to fine-tune model parameters. |
| Bach size | number of training examples utilized in one iteration to update the model's parameters. |
| Gradient Accumulation Steps | Number of steps over which gradients are accumulated before updating weights |
| Weight Decay | Regularization technique that reduces the weights slightly during training to prevent overfitting |
| Warmup Steps | Initial phase of training where learning rate gradually increases to its target value |

Upon hyperparameter tuning, the best parameters for finetuning are:

|  |  |
| --- | --- |
| **Parameters** | **Values** |
| Epochs | 3 |
| Bach size | 16 |
| Gradient Accumulation Steps | 8 |
| Weight Decay | 0.01 |
| Warmup Steps | 500 |

### T5 – Base Model for extractive summarization

We used the python library Optuna to find the best hyperparameters to train the T5 model. The objective was to minimize validation loss. The hyperparameters considered for training the model are :

|  |  |
| --- | --- |
| **Parameters** | **Significance** |
| Learning Rate | controls the step size or the rate at which the model parameters are updated during training. |
| Batch size | number of training examples utilized in one iteration to update the model's parameters. |
| Epochs | one complete pass through the entire dataset during the training phase. |
| Max Sequence Length | maximum number of tokens allowed in an input sequence. |
| Weight Decay | regularization technique commonly used during the training of machine learning models, particularly neural networks. |

Upon hyperparameter tuning, the best parameters for finetuning are :

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Learning Rate | 0.0003421311045043332 |
| Batch Size | 16 |
| Epochs | 11 |
| Max Sequence Length | 512 |
| Weight Decay | 1.020803789978182e-05 |
| No of Warmup Steps | 994 |
| Gradient Accumulation Steps | 1 |

# SUMMARY GENERATION

For selecting the optimal generation parameters, we adopted a manual trial-and-error approach. Rather than solely relying on maximizing ROUGE scores, which may not necessarily produce the most desirable summaries, we experimented with different parameter values and carefully examined the generated summaries. This iterative process involved reading through the summaries, assessing their quality, and adjusting the parameters accordingly until we obtained summaries that aligned with our ideal requirements.

We recognized that while ROUGE scores provide a quantitative measure of summary quality, they may not capture all aspects of what constitutes a high-quality summary. Therefore, we prioritized a balanced approach that combined quantitative metrics with qualitative evaluations. By meticulously reading and evaluating the generated summaries, we aimed to strike a harmonious balance between achieving favourable ROUGE scores and producing summaries that met our desired criteria for coherence, relevance, and overall quality.

This manual approach allowed us to fine-tune the generation parameters based on our specific needs and preferences, rather than solely relying on automated metric optimization. We believe that this iterative and hands-on process was crucial in obtaining summaries that not only performed well on quantitative measures but also met our qualitative expectations for the summarization task.

We examine the key generation parameters employed to tailor the summarization model for producing concise and clear summaries. The goal is to strike a balance between capturing essential details while avoiding excessive verbosity or repetition.

Generation Parameters:

1. Max Length
   * This parameter sets an upper limit of tokens for the generated summaries.
   * Allowing for longer output ensures that the summaries can be detailed and comprehensive, capturing a wide range of important information while allowing for shorter output ensures there is no repetition in the summaries.
2. Min Length
   * A minimum length of tokens is enforced for the summaries.
   * This parameter guarantees that the summaries are sufficiently substantive, avoiding overly brief or superficial representations that may miss crucial points.
3. Length Penalty
   * A length penalty is applied during the generation process.
   * This penalty encourages the model to generate more concise summaries by slightly favouring shorter outputs over longer ones.
   * It helps strike a balance between detail and brevity, preventing excessive wordiness while still allowing for adequate coverage.
4. No Repeat N-Gram
   * This parameter prevents the model from generating repeated sequences of the given value or more consecutive tokens.
   * It promotes unique and diverse phrasing within the summaries, reducing the likelihood of repetitive or redundant content.
   * This feature enhances the clarity and coherence of the generated text.

The combination of these generation parameters aims to achieve concise yet comprehensive summaries that effectively balance detail with brevity. By setting appropriate length constraints, applying a length penalty, and preventing repetitive phrasing, the tailored summarization model can generate summaries that capture essential information in a clear and concise manner.

The maximum and minimum length parameters ensure that the summaries are neither excessively long nor too brief, while the length penalty and the restriction on repeated n- grams contribute to conciseness and clarity by discouraging verbosity and promoting diverse phrasing.

Overall, these parameters play a significant role in enabling the summarization models to produce beneficial summaries that meet the desired criteria of conciseness and clarity, making the generated content more accessible and easier to comprehend for the intended audience.

**SUMMARY GENERATION PARAMETERS**

|  |  |  |
| --- | --- | --- |
| Parameter | Pegasus Model | T5 Model |
| Max Length | 3000 | 512 |
| Min Length | 300 | 512 |
| Length Penalty | 3 | 2.0 |
| No Repeat N-Gram | 4 | 5 |

# MODEL EVALUATION

We used the following metric for model evaluation.

## ROUGE

ROUGE is a set of metrics used for evaluating automatic summarization systems. It was proposed by Chin-Yew Lin in 2004 and has become one of the most widely used evaluation metrics for summarization tasks. ROUGE measures the quality of a summary by comparing it to one or more reference (human-generated) summaries.

The main idea behind ROUGE is to measure the overlap of n-grams (sequences of n words) between the system-generated summary and the reference summaries. The more overlap there is, the higher the ROUGE score, indicating better summary quality.

There are several variants of ROUGE, but the most commonly used ones are:

ROUGE-N: This measures the n-gram overlap between the system summary and reference summaries. Common values of N are 1 (unigram) and 2 (bigram).

ROUGE-L: This measures the longest common subsequence (LCS) between the system summary and reference summaries. It focuses on capturing sentence-level structure similarity.

ROUGE = No of n−grams found in model generated summary and reference summary

No.of n−grams found in reference summary

### BERTSscore

BERTScore is a metric used to evaluate the quality of natural language generation tasks, particularly in the context of comparing generated text against reference text. It utilizes the Bidirectional Encoder Representations from Transformers (BERT) model, a powerful pre- trained neural network developed by Google.

Here's a detailed breakdown of how BERTScore works:

Tokenization: Both the generated text and the reference text are tokenized. Tokenization involves breaking down the text into smaller units, such as words or subwords, for further analysis.

Embeddings: Each token (word or subword) is converted into a high-dimensional vector representation using the BERT model. BERT embeddings capture the contextual meaning of each token based on its surrounding words in the sentence.

Scoring: BERTScore calculates the similarity between the token embeddings of the generated text and the reference text. It computes a similarity score for each token pair between the generated and reference sequences. The similarity score is determined based on the cosine similarity between the token embeddings.

Aggregation: The individual token similarity scores are aggregated to compute an overall similarity score for the entire generated sequence. Various aggregation methods can be used, such as taking the average or weighted average of the token scores.

Normalization: To ensure that the similarity scores are comparable across different text lengths and contexts, BERTScore normalizes the aggregated score by dividing it by a normalization factor. This normalization factor accounts for differences in token length between the generated and reference sequences.

Final Score: The final BERTScore is obtained by subtracting the normalized score from 1, resulting in a score between 0 and 1. A higher BERTScore indicates greater similarity between the generated text and the reference text.

|𝐺|

|𝑟|

∑ 𝑚𝑎𝑥𝑗=1𝑆(𝑔𝑖,𝑟𝑗)

BERTScore = 1 - 𝑖=1

𝑁

where,

1. ∣G∣ is the number of tokens in the generated sequence G.
2. ∣R∣ is the number of tokens in the reference sequence R.
3. S(𝑔𝑖, 𝑟𝑗) is the cosine similarity between the embedding of token 𝑔𝑖 from the generated sequence and the embedding of token 𝑟𝑗 from the reference sequence.
4. N is the normalization factor, typically the maximum of ∣G∣ and ∣R∣.
   1. **Comparative Results**

**Abstractive Summarization**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MODEL | Rouge 1 | | | Rouge 2 | | | Rouge L | | | BERTScore |
| T5 – small | 42.38 | 30.7 | 35.6 | 17.31 | 10.95 | 13.42 | 40.2 | 29.12 | 33.78 | 82.61 |
| LED  Booksum | 29.02 | 41.49 | 34.16 | 10.17 | 17.63 | 12.82 | 27.72 | 39.63 | 32.62 | 84.82 |
| Pegasus  Daily Mail | 50.5 | 34.31 | 40.86 | 19.88 | 12.28 | 15.18 | 46.87 | 31.85 | 37.93 | 85.04 |

Table 1 Comparative Score of abstractive models

**Extractive Summarization**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MODEL | Rouge 1 | | | Rouge 2 | | | Rouge L | | | BERTScore |
| BERT | 60.47 | 38.74 | 47.22 | 39.08 | 27.13 | 32.03 | 58.44 | 37.44 | 45.64 | 82.48 |
| BART | 50.68 | 43.60 | 46.87 | 27.41 | 27 | 27.20 | 47.13 | 40.55 | 43.59 | 94.79 |
| T5 - Base | 56.25 | 47.77 | 51.66 | 31.76 | 26.93 | 29.15 | 54.72 | 46.48 | 50.27 | 96.27 |

Table 2 Comparative scores of extractive models

**Snippet of summary generated using finetuned Pegasus model**

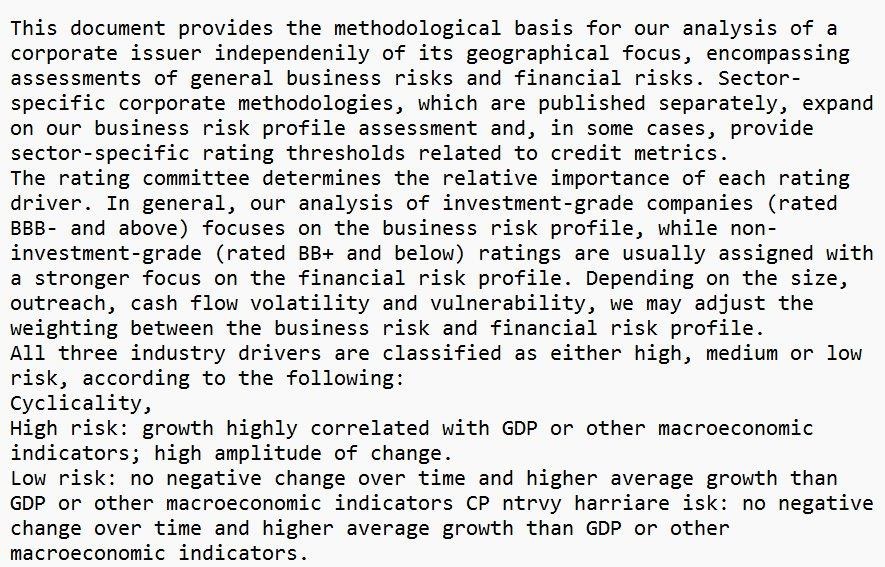


Figure 20 Snippet of summary using abstractive model.

**Snippet from the summary generated using finetuned T5 – Base model**

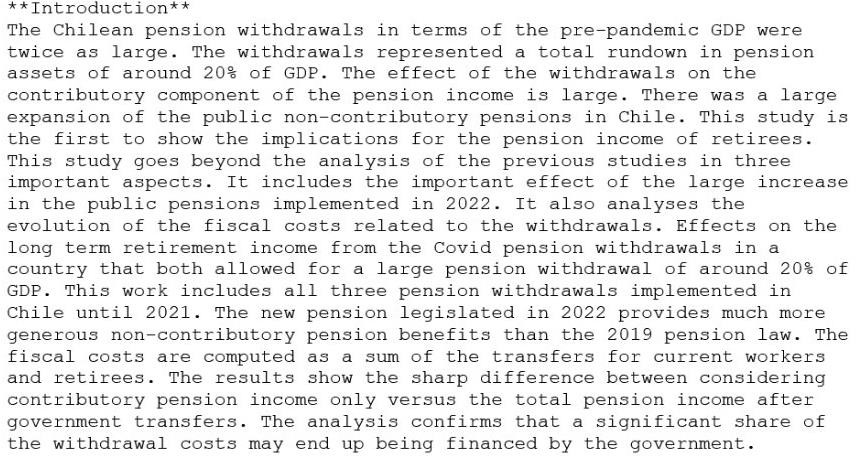


Figure 21 Snippet of summary using extractive model.

# HOSTING THE APP USING STREAMLIT

Streamlit is a Python library that allows developers to create interactive web applications quickly and easily. With its intuitive and straightforward API, Streamlit enables users to build data-driven applications without the need for extensive web development experience.

Developers can seamlessly integrate data visualizations, machine learning models, and other interactive elements into their applications using Streamlit's simple syntax and built-in components. Additionally, Streamlit provides real-time feedback, enabling developers to instantly see changes as they code, facilitating rapid iteration and experimentation. Overall, Streamlit empowers developers to create engaging and responsive web applications with minimal effort, making it an ideal choice for hosting our document summarization app.

Here’s a quick tour of the Document Summarizer App

In order to provide input document to the file, You can choose between a PDF file, a text file, or a word document.

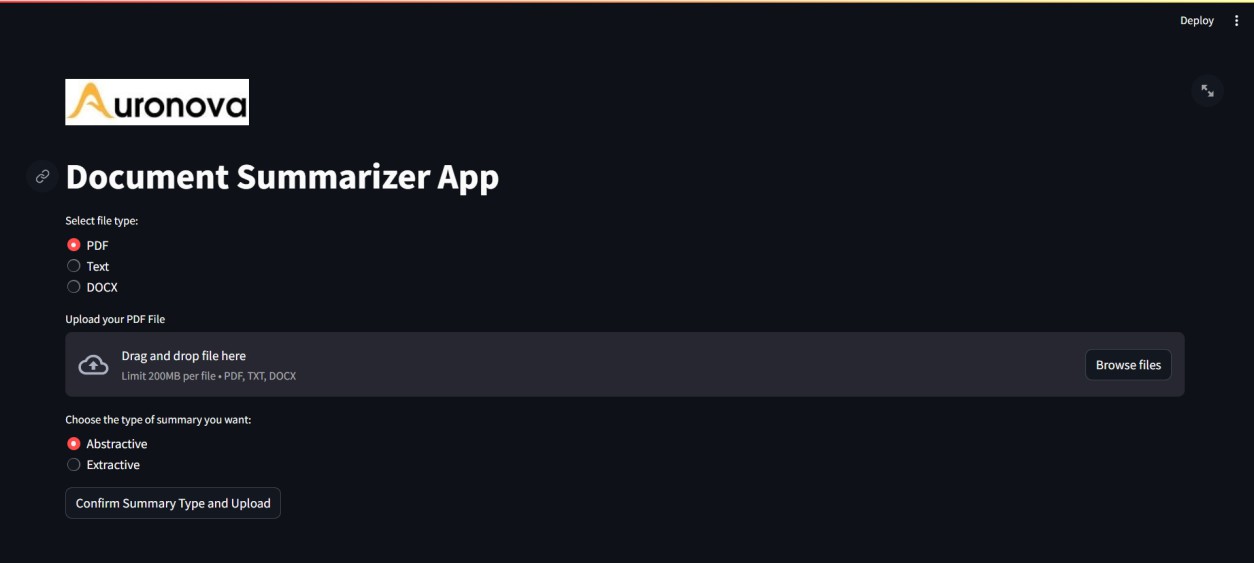


Figure 22 Document Summarization App

You can browse on your system (a) and choose the file that you want to summarize. You can verify that the file is uploaded to the app (b).

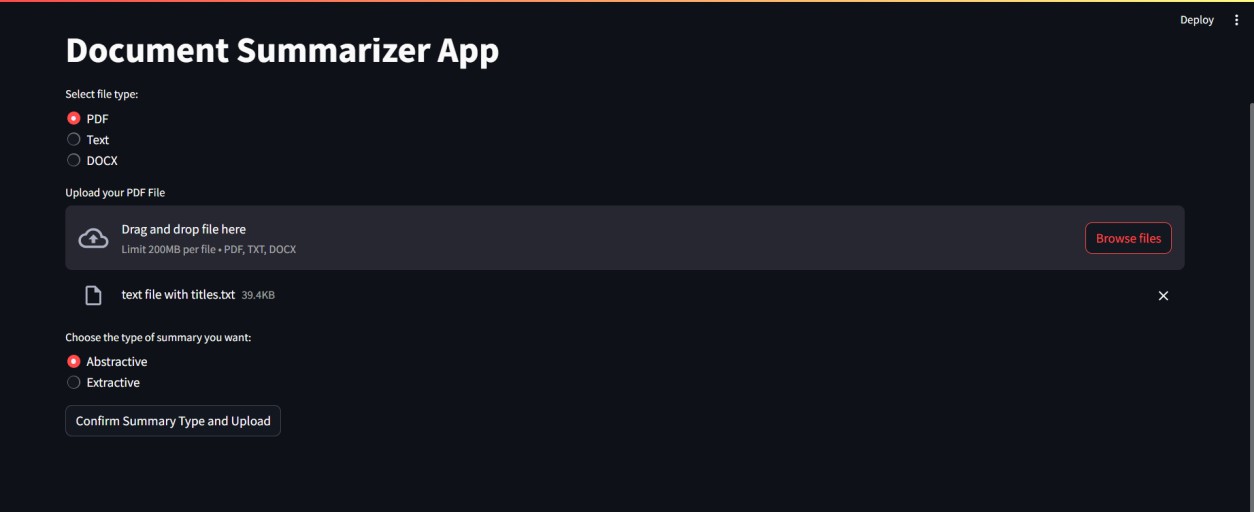


Figure 23 Document Summarization App

You can choose between abstractive or extractive summary (a) . One you click on “Confirm Summary Type and Upload” button, the app reads the contents of your file and tries to detect chapters/ titles from the content. You can verify whether the detected titles are correct or not

(b) . Suppose you want to make changes to the detected titles, you can click on “No” and manually enter comma-separated titles (c) . If the detected titles are correct, confirm yes and click on “Generate summary with detected titles” to generate summary (d) .



Figure 24 Document Summarization App

You can track the progress of summary generation using the progress bar. Once the summary is generated, you can read the summary from the display box. You can download the summary in a text file (a) and can check the time required to generate the summary (b). Once you download the summary, the app resets and goes back to the home page.

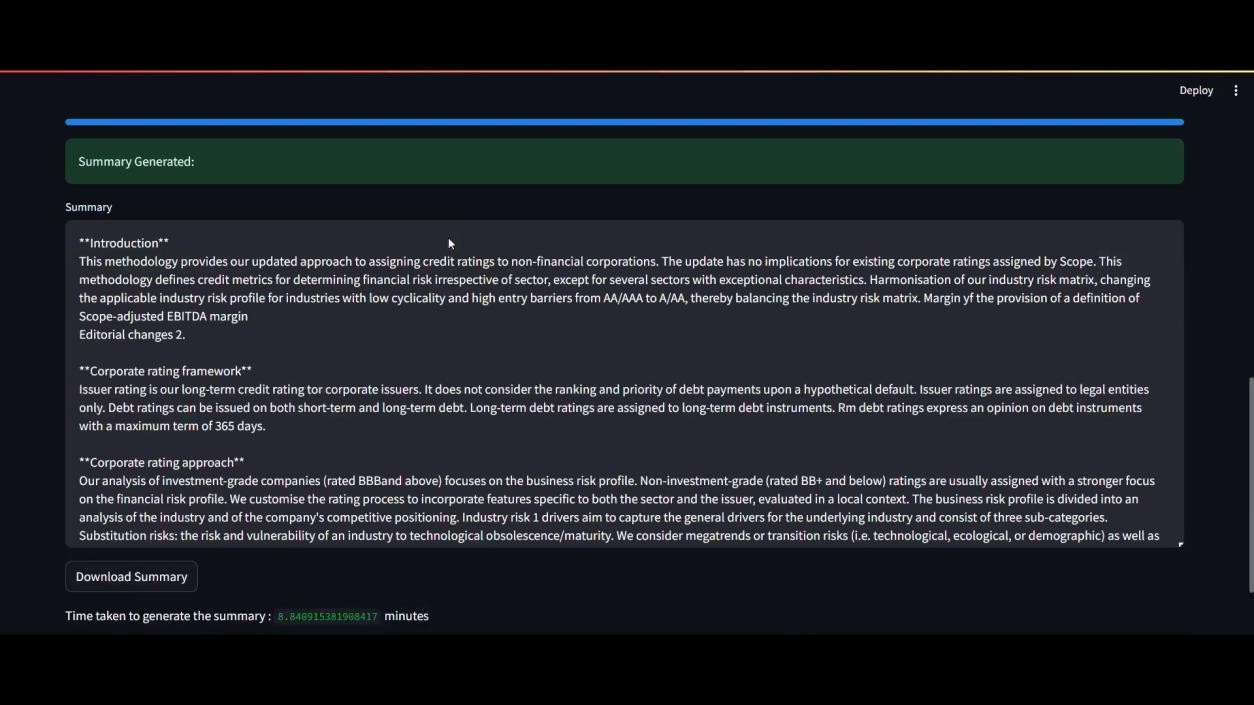


Figure 25 Document Summarization App

# CONCLUSION

In conclusion, our document summarization app represents a significant advancement in the field of natural language processing, offering transformative capabilities for efficiently distilling key insights from textual documents. By leveraging state-of-the-art transformer- based models and incorporating user-centric design principles, we have created a powerful yet accessible tool for enhancing productivity, knowledge extraction, and decision-making across diverse domains. The integration of Streamlit for hosting the app ensures seamless deployment and accessibility, further democratizing access to advanced summarization capabilities. With a focus on efficiency, scalability, and adaptability, our app is poised to make a meaningful impact in various sectors, including regulatory compliance within the BFSI industry. Moving forward, we remain committed to continual improvement and innovation, guided by user feedback and emerging advancements in NLP, to further enhance the effectiveness and relevance of our document summarization app in addressing real-world challenges and opportunities.

# GLOSSARY

1. Fine-tuning: The process of taking a pre-trained model and continuing the training to specialize on a narrower task, which in your case would be specific to summarization.
2. Tokenization: The process of converting text into smaller pieces, called tokens, which are easier for models to process.
3. Attention Mechanism: A component of neural networks that helps the model dynamically focus on different parts of the input sequence, critical for understanding context in summarization.
4. Sequence-to-Sequence Models (Seq2Seq): A type of model that transforms a sequence of text in one form to a sequence in another form, used extensively in tasks like translation and summarization.
5. Bounding Box: A rectangular box that is used to define the position and scale of an object in an image, video, or other visual representations. It is typically specified by the coordinates of its top-left corner and its width and height.
6. Ground Truth: The term used to describe the real-world data or annotation in the context of machine learning and image processing. In the context of bounding boxes, it refers to the manually annotated boxes that accurately define the location of objects.
7. Annotation: The process of marking up images or other data, often with bounding boxes, to indicate the location of objects. This is commonly done in the training of machine learning models for tasks like object detection.