LOW LEVEL DESIGN AND IMPLEMENTATION DOCUMENT

VISUAL QUESTION ANSWERING ON STATISTICAL PLOTS

UE18CS390B – Capstone Project Phase – 2

Submitted by:

Sneha Jayaraman	**PES1201802825**
Sooryanath I T	**PES1201802827**
Himanshu Jain	**PES1201802828**

Under the guidance of

Prof. Mamatha H.R.
Professor
PES University

August - December 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

TABLE OF CONTENTS

1. Introduction

The Section deals with the lowest level dissection of the high level design developed in Phase 1. Here we delve into the modular and unit components that constitute the development of this tool.

1.1. Overview

The below diagram Summarises the high level design that we intended to do.

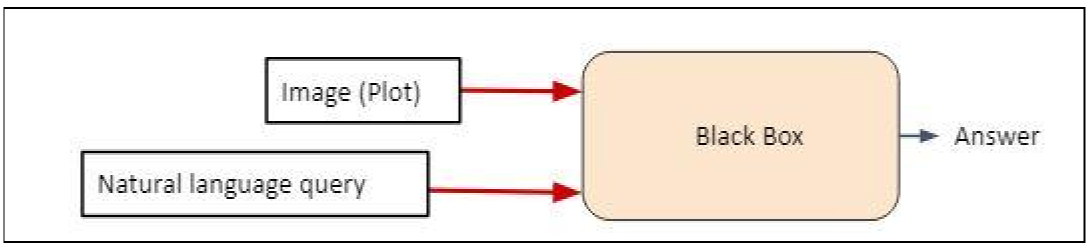


Fig 1. 1 : Proposed High level View of The VQA system

In the figure we have two inputs being fed into the Black Box which are an input image [the graphical plot in scope] and a Natural language query related to the graphical image. whereas in the lower level design the black box is further expanded and cut open. the Black Box Constitutes the core of the visual question answering system.

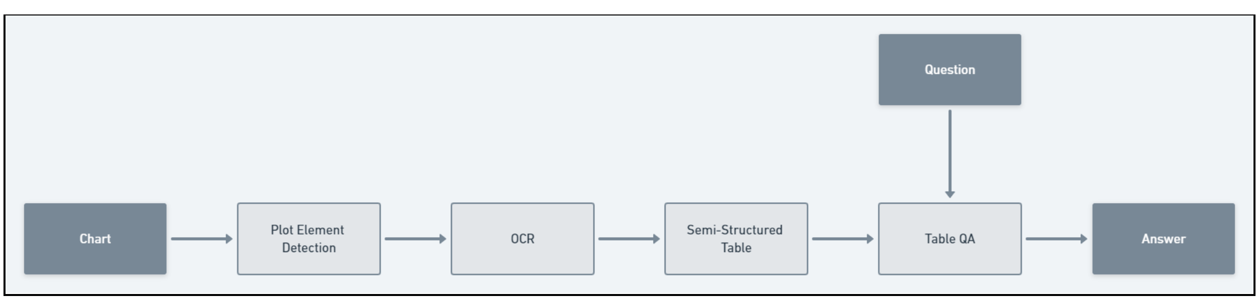


Fig 1.2 The cut open view of the black box and deep delve into the modules

In figure 1.2 , There are 4 stages within the Black Box .

- **The plot element detection stage**
- **The optical character recognition stage**
- **The Semi-structured table generation stage**
- **Table Question Answering Stage**

The inputs to black-box/ VQA system

- A graphical plot [bar {vertical/horizontal/grouped} , dot , line]
- A question posed [in vocab / out of vocab] related to the plot image

The phase wise dissection will be done in the following sections

1.2. Purpose

The purpose of the low level design document is to provide a detailed description about the working of each and every unit and how they all work in union to achieve the desired result. It is used by designers, operation teams, implementers/dev and dev-test members . This will serve as a documentation for developing stubs/drivers .

Here we provide a description about the four stages within the Black Box and how they inter communicate with each other and interface with each other. Moreover this is the phase of a project in which the application logic is designed and ready to be implemented. The exit criteria / input criteria - data to every phase needs to be dissected and presented in detail.

1.3. Scope

The scope of this document is to address the flow of information through the pipeline and stage wise requirement which is needed to accomplish the goals of corresponding stages. This implementation of VQA on statistical plots takes into consideration **plots of type = {Dot , Line , Bar[Hbar , Vbar , Grouped]}** and **Questions = {Open-ended, In-vocabulary} .**

Overview of the tools that have significant contribution in every stage is provided. Alongside constraints , dependencies and assumptions existing between the modules/stages is also discussed. An overview regarding the novel practices and new ideas infused within the system is also discussed along with examples and variants of those.

2. Design Constraints, Assumptions, and Dependencies

I. The Environment , hardware software dependencies needed to run this pipeline

- The training environment specification is as follows
    - **Platform** : Google Colab Pro
    - **RAM** : 26GB
    - **Disk** : 110GB
    - **GPU** : 16GB , Tesla - T4
    - **Training Data Size** : 6.x GB
    - **Test Data Size** : 1.5xGB

II. Assumptions of the model/VQA system

- The VQA is limited to only certain class of graphs and questions
- The input image to the model should be one among **{Dot , Line , Bar[Hbar , Vbar , Grouped]}**
- The questions posed should be of type **{Open-ended, In-vocabulary}.**

- The type of questions can be based on arithmetic mean , median , difference , sum , data retrieval . comparison or boolean.
- Structural questions are not handled within the scope of this implementation.

III. Dependencies between the stages and the Input/Exit data+Criteria

Stage	Input Criteria	Exit Criteria	Output
Plot element detection	Input Plot Image belonging to certain class of graphs	Bounding box annotation around the plot elements.	A text tabular formatted equivalent of a json file , holding the coordinates of the plot elements[topleft_x , topleft_y,bottom_x , bottom_y] and the confidence that the element belongs to a class
OCR	text tabular file from previous stage + Image	use OCR module like tesseract to read the character within the bounding coordinates in the image	extracted texts from the bounding box specific region according to the category of the plot element

Semi-structured table generation	textual data extracted from OCR	format the data into a semi-structured table on which queries can be executed	A CSV file which is a tabular format of the graph input image
Table Question Answering	The CSV file/tabular format of the graph + Question	classify the queries into boolean vs data-retrieval Execute the queries on the table and accumulate answer	The final answer to the input question

We can clearly see the **linear dependency that exists across the modules** , failure of any one of the intermediate modules will tend to have a cascading effect on the follow up stages.

There also exists few dependencies on the modules/of the shelf components that are being made use of in every stage.

IV. Dependencies on the Modules/libraries and packages used

- Detectron-2
- Pytorch - 1.9.0
- Tesseract , conda environments
- cv2 , TAPAS , TABFACT & SEMPRE
- All other utility modules like OS , JSON , NUMPY , CSV , Scipy etc.

V. Constraints regarding the questions posed

- Data retrieval , sum , average , columnar max- min questions handling capability was already built into TAPAS whereas we have added custom methods/operators handling methods to accept a variety of questions based on range , quartile , difference etc , these are based on break up over a particular keyword.

3. Design Description

3.1. Plot Element Detection Stage

- Input : Image{graphical plot}
- Output : formatted text file derived from JSON
- the plot elements of an input image are extracted by training an object detection model over a large collection of samples .
- **Detectron-2** is faster , flexible and vast in terms of configuration,models and implementation due to its API availability when compared with its parent , hence we use it as the object detection and bounding box generation tool.
- Few samples are shown below

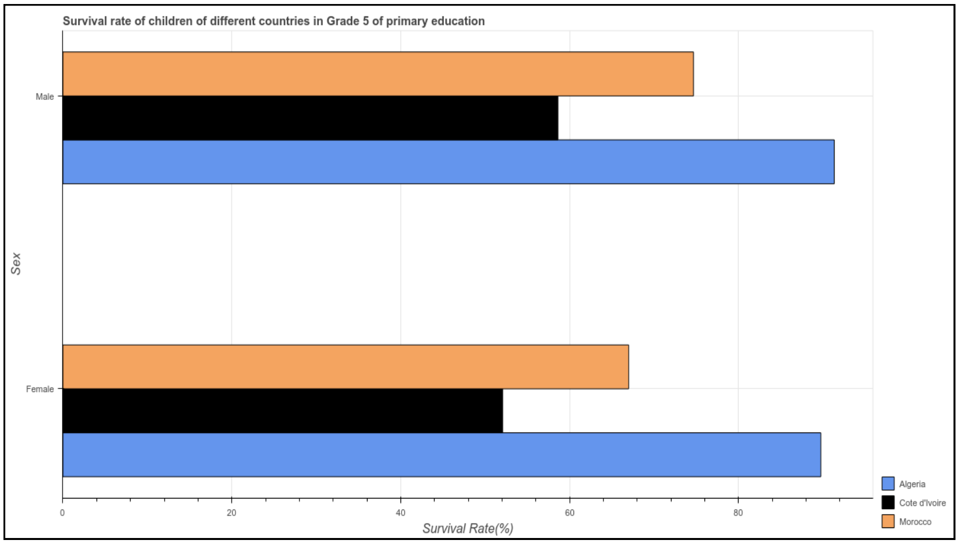


Fig 2.1 An input bar graph {horizontally grouped}

```
cat    score              left_x                top_y               right_x               bottom_y
bar 0.9999642372131348 74.69889831542969 109.78387451171875 726.3924407958984 163.60403442382812
bar 0.9999309778213501 76.31050872802734 164.42819213867188 1101.3407821655273 217.30908203125
bar 0.9999561309814453 79.21051025390625 521.163330078125 1080.1463623046875 574.4030151367188
bar 0.9999512434005737 75.50065612792969 467.32183837890625 653.8601531982422 521.538818359375
bar 0.9999502897262573 79.16533660888672 56.312198638916016 898.4965744018555 110.08055877685547
bar 0.9999496936798096 76.8082275390625 414.31732177734375 815.82861328125 467.9610595703125
legend_label 0.9999446868896484 1174.101806640625 596.1893920898438 1235.420166015625 616.2551879882812
legend_label 0.9999395608901978 1173.954833984375 572.9148559570312 1208.17236328125 592.91943359375
preview 0.9999374151229858 1148.673583984375 572.9439697265625 1168.8115234375 593.0646362304688
preview 0.999933123588562 1148.6539306640625 619.1229858398438 1168.7286376953125 638.9912109375
xticklabel 0.9999247789382935 512.8422241210938 611.9534912109375 524.9588012695312 625.9398803710938
xticklabel 0.9999061822891235 290.6319580078125 612.0994262695312 303.05084228515625 626.0181274414062
xlabel 0.9998942613601685 546.7985229492188 629.4925537109375 666.7786865234375 647.783447265625
xticklabel 0.9998867511749268 72.15039825439453 612.00048828125 78.21342468261719 625.9788818359375
xticklabel 0.9998866319656372 956.0477905273438 612.1361083984375 968.4012451171875 626.0216064453125
xticklabel 0.9999575614929199 735.3030395507812 612.0874633789062 747.5630493164062 625.9378662109375
preview 0.9998660087585449 1149.1019287109375 595.9469604492188 1169.0247802734375 616.4177856445312
title 0.9996670484542847 76.6021499633789 9.004288673400879 769.3832778930664 27.014885902404785
ylabel 0.9995629191398621 6.173298358917236 301.8087463378906 24.040911197662354 330.2987976074219
yticklabel 0.9999945163726807 27.25185203552246 459.97637939453125 63.65839195251465 474.07421875
yticklabel 0.9999923706054688 40.192203521728516 102.92390441894531 64.24410247802734 117.1244888305664
legend_label 0.9999701976776123 1173.8287353515625 619.1683349609375 1215.5662841796875 639.175048828125
```

Fig 2.2 : Text formatted JSON file containing coordinates of the bounding boxes

- As seen above , all the bounding boxes corresponding to the plot elements have been extracted according to their classes .
- We chose **Faster-RCNN** as our object detection model which is a **descendant from the R-CNNs series.**

- The heuristic behind doing so is due to the presence of the RPN (Regional proposal network is known for locating feature targets accurately) and the inference time.
- We also need to decide on the **feature extractor model** for serving as the **backbone/feature-extractor** for our faster-RCNN Setup.
- We Chose **Resnet-101** to be our feature extractor due to **skip connections** and **residual blocks** which can reduce the problem of vanishing gradients in a very deep network like Resnet-101.

3.2. OCR detection

- The textual format of the json file is passed into this stage .
- the images directory is made accessible to this module
- With the help of bounding box coordinates extracted , we can locally capture the text info within the bounding boxes rather than passing an entire image into the OCR module .
- The captured text is then read using OCR and classified into its category.

3.3. Semi Structured table generation

- This phase is the crucial phase of converting the graphical data to its tabular format based on the OCR readings done in accordance with classes.

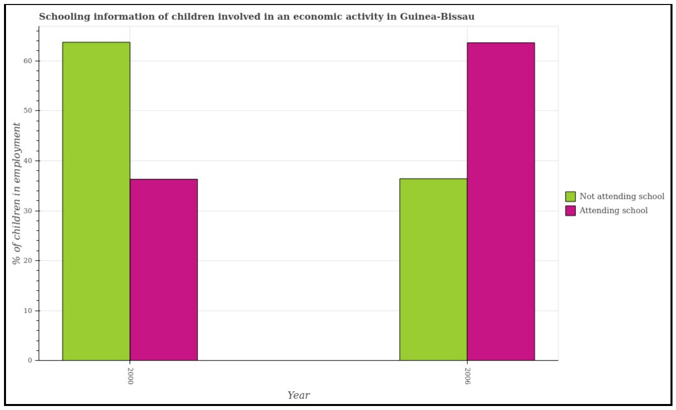


Fig 2.3 : A simple Vertical Bar graph

```
Category   Score             left_x             top_y             right_x            bottom_y
yticklabel 0.9999904632568359 27.946645736694336 246.9253387451172 41.919342041015625 260.8976135253906
yticklabel 0.9999895095825195 27.873197555541992 328.9063720703125 41.86872100830078 343.05303955078125
bar 0.9999873638153076 640.291259765625 284.0574645996094 749.989990234375 580.9351806640625
bar 0.9999823570251465 201.92819213867188 281.8130798339844 312.2918701171875 578.4205322265625
legend_label 0.9999779462814331 932.5986328125 303.1197509765625 1071.5126953125 322.6226806640625
yticklabel 0.9999773502349854 27.868160247802734 409.8987121582031 41.989967346191406 423.9773254394531
bar 0.9999696016311646 750.8607177734375 62.46775817871094 861.982421875 581.0493621826172
xticklabel 0.9999669790267944 194.113037109375 590.9034423828125 208.31503295898438 618.8867797851562
yticklabel 0.9999643564224243 34.982704162597656 572.9815063476562 41.99324035644531 587.0731201171875
bar 0.9999592304229736 91.97103881835938 59.920265197753906 202.5250701904297 581.5347671508789
preview 0.9999498128890991 908.1177368164062 303.0318908691406 928.0110473632812 322.7835998535156
yticklabel 0.9999486207962036 28.012062072753906 164.76251220703125 42.00925827026367 178.90281677246094
yticklabel 0.9999397993087769 28.047203063964844 492.2265625 42.02458572387695 505.870849609375
xlabel 0.9999388456344604 457.864501953125 627.0880737304688 493.36260986328125 646.0440673828125
xticklabel 0.999916672706604 743.106201171875 590.95849609375 756.9241943359375 619.1182861328125
legend_label 0.9998894929885864 932.1422119140625 325.8930969238281 1043.439697265625 345.8262023925781
preview 0.9998648166656494 908.1642456054688 326.23406982421875 928.19921875 346.0467834472656
yticklabel 0.9998058676719666 28.096742630004883 84.25582885742188 42.00232696533203 97.83050537109375
ylabel 0.9998049139976501 6.735439777374268 190.0906524658203 26.02379274368286 425.5273742675781
title 0.9994369149208069 49.137367248535156 9.029414176940918 796.2882461547852 27.020319938659668
```

Fig 2.4 : Text format of the bbox JSON output

	Year	Attending school	Not attending school
0	2000	37.06615560158244	64.78018619662012
1	2006	64.48456969920525	37.0474010349917

Fig 2.5 : Tabular Format of the Graph

3.4. Table Question Answering Stage

- We have made use of **Google's TAPAS** to answer questions from tabular data
- Tapas selects a subset of table cells and applies aggregation/retrieval operations on top of them
- It extends BERT architecture with additional embeddings that capture tabular structure, and with two classification layers for selecting cells and predicting a corresponding aggregation operator.
- We have added our custom operations and methods to suit the desired output.
- It is trained on Wikipedia Tables and provides a pre-trained model for end tasks.

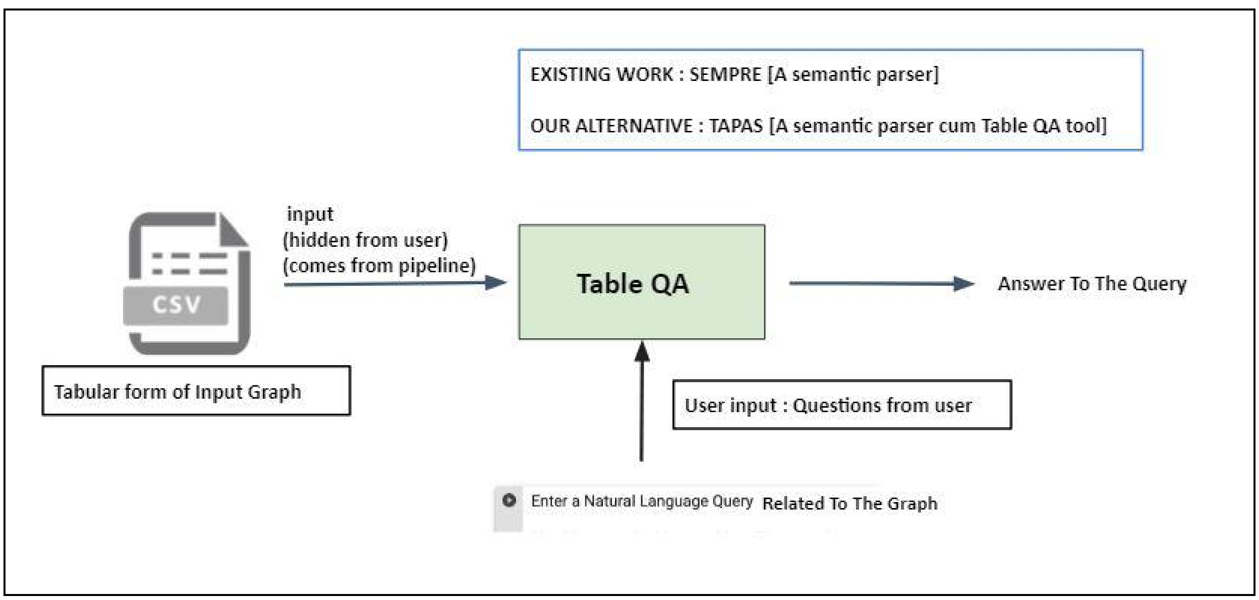


Fig 2.6 : The workflow in the TABLE QA Stage

4. Proposed Methodology / Approach

4.1 Algorithm and Pseudocode

- **Plot Element detection Stage (TRAIN + TEST)**

```
#TRAINING

Load (train_images , Annotation_file)
model = Trainer()
dataset : Split(train_images , Annotation_files , Splits = 3)
model.data : Correlate_dataset_with_annotations(dataset)
sample(data) #check If Image And Annotation Are Corresponding
choose Object_detection_model And Backbone_network
model.object_detection_model : Faster-rcnn
model.backbone_model : Resnet_101
iters : 21 ; Lr : 0.0025 ; Gamma : 0.1
model.train(data , Iters , Lr , Gamma Object_detection_model , Backbone)

#TESTING

Load(test_images)
model.test_data :Register_for_testing(test_images)
json_result : model.test(generate_json_result = True)
txt_format : convert_json_o_text(json_result)
```

- **OCR**

```
---------------------------------------------------------------------------------


ocr : OCR()
ocr.load_utilities()
ocr_extractions : ocr.load(txt_format  , input_image)



---------------------------------------------------------------------------------
```

- **Semi structured table generation**

```
CSV_maker : CSV()
CSV_maker.ocr_readings : ocr_extractions()
Tabular_format : CSV_maker.generate_table_csv()
```

- **Table QA**

```
table_qa : TAPAS()
csv_table : preprocess(Tabular_format)
table_qa.table : csv_table
table_qa.queries : List(Read_from_users())
table_qa.queries.classify()
table_qa.answers()
```

4.2 Implementation and Results

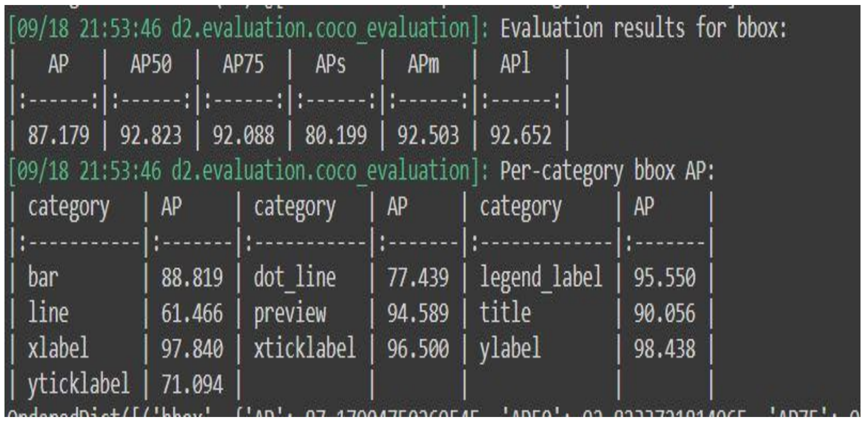
- Results of Plot Element detection stage

Trial - 1

```
[08/27 11:11:57 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 61.946 | 88.722 | 76.841 | 55.716 | 70.649 | 63.861 |
[08/27 11:11:57 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     | category     | AP     |
|:-----------|:-------|:-----------|:-------|:-------------|:-------|
| bar        | 73.387 | dot_line   | 57.155 | legend_label | 74.983 |
| line       | 27.329 | preview    | 50.912 | title        | 62.899 |
| xlabel     | 76.929 | xticklabel | 62.948 | ylabel       | 80.105 |
| yticklabel | 52.816 |            |        |              |        |
```

Training_data_size (in no.s)	50K
iterations	1K
Base_LR	0.001
AP	61.947

Trial - 2



```
[09/18 21:53:46 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 87.179 | 92.823 | 92.088 | 80.199 | 92.503 | 92.652 |
[09/18 21:53:46 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     | category     | AP     |
|:-----------|:-------|:-----------|:-------|:-------------|:-------|
| bar        | 88.819 | dot_line   | 77.439 | legend_label | 95.550 |
| line       | 61.466 | preview    | 94.589 | title        | 90.056 |
| xlabel     | 97.840 | xticklabel | 96.500 | ylabel       | 98.438 |
| yticklabel | 71.094 |            |        |              |        |
```

Training_data_size (in no.s)	150K
iterations	200K
Base_LR	0.004
AP	87.179

- Results of the Table Question Answering Stage

We have used accuracy as the evaluation metric. For textual answers, the answer would contribute to the accuracy only if an exact match was found between the expected and the predicted answers. However, in the case of numeric answers, we have allowed for an error window of 5\%. Answer values within this range will be considered correct.

Plot Type	Number of Images Tested	Total Number of Questions	Number of Correct Answers	Average Accuracy (in %)
Dot	2000	53970	25104	**46.965499**
Vertical	2000	47940	19898	**41.474200**
Horizontal	2000	49241	20128	**40.990114**
Line	2000	35353	14077	**36.669402**

4.2 Further Exploration Plans and Timelines (optional)

Fine tuning of the TAPAS Models.

Appendix A: Definitions, Acronyms and Abbreviations

ACRONYMS/TERMS	EXPANSIONS / DESCRIPTION
CPU	Central Processing Unit
GPU	Graphics Processing Unit
Detectron	An Object detection + BBOX generation trainer/visualizer
OCR	Optical Character Recognition
JSON	Javascript Object Notation
CSV	Comma separated Values
NLP	Natural Language Processing
Module	A Software Component
BBOX	Bounding Box

Appendix B: References

Title	Version Number	Date	Publishers	Reference
Plot QA	1.0	12/04/2020	Mitesh Khapra Nitesh Methani , Pritha Ganguly and Pratyush Kumar	[1]

Detectron 2	1.0	18/08/2021	Yuxin Wu and Alexander Kirillov and Francisco Massa and Wan-Yen Lo and Ross Girshick	[2]
TAPAS	1.0	08/09/2021	Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, Julian Eisenschlos	[3]

Appendix C: Record of Change History

#	Date	Document Version No.	Change Description	Reason for Change
1.	**26/09/2021**	**1.0**	Created the LLD document	**Deep delve into the modular components**