# **To be discussed with Sergio**

Should we use benchmark sets?

Test out MTT <https://github.com/microsoft/table-transformer>

# **Tasks**

## Caption retrieval model

Note: Log all the articles used to generate both training and test datasets in a DOI list

Generate a caption annotation set (3000 annotated images with captions + 500 images with no captions) \*currently 906 images

1. Download pdfs of **open-access** articles (different topics different publishers and journals)
   1. Chem Archive
   2. Wiley – can use API to download using wiley-pdf-batch-download.py
   3. RSC (manual dl)
   4. ACS (manual dl)
   5. Springer Nature (manual dl)
   6. Cell Press (manual dl)
   7. Elsevier (manual dl)
2. Run pdfs through **TF-ID-chem-large model** to extract images
3. Use **COCO-annotator** to annotate images
4. Convert COCO dataset to Florence dataset using prepare-florence-training-set.py
5. Augment Florence dataset with non-annotated images using prepare-florence-training-set.py
6. Split into train.jsonl and test.jsonl using prepare-florence-training-set.py

Train a caption retrieval and segmentation model using

* Modify code to save segmented caption as <base\_img\_name>\_caption.png

Prepare 200 images as test set

* Obtained from other **open-access** articles using **TF-ID-chem-large model** (MUST NOT be in train set)
* Run trained caption retrieval and segmentation model on these images using
* Manually collate **evaluation statistics** (csv)
  + True positive – caption present in image and properly segmented
  + True negative – caption absent in image and not segmented
  + False positive – caption absent in image, but image was segmented
  + False negative – caption present in image but image was not segmented
  + Precision, recall, F1, specificity, accuracy

## Caption annotation module

Note: Make it modular to facilitate integration into final pipeline)

Lit search to identify **3 open-source optical character recognition (OCR)** libraries

* Requirements in order of importance: open-source, good performance, easy to install (with few dependencies issues)
* Some libraries to consider: Florence-2’s OCR, easyOCR, Tesseract, Google Cloud Vision

Test on a **small number of caption images (maybe 8)** to see which one gives better performance (save results either in a Jupyter notebook or export out to .json or .txt files)

* Requirement: does not need to capture word-for-word but should capture gist of caption. Okay to have typos if overall readability/ understandability is minimally compromised

Compile **OCR results of selected OCR library on 30 caption images** (Jupyter notebook/export)

## Comparison with other toolkits

Note: separate statistics for figures and tables because not all toolkits have both capabilities

To be decided: How many articles to evaluate? What are the evaluation metrices? How to present results?

Figure-caption extraction capability

* Precision, recall, F1, accuracy, specificity at retrieving figures + captions

|  |  |
| --- | --- |
| Lit search for any latest packages/papers |  |
| PDF-FigCap-X | <https://github.com/pengyuanli/PDFigCapX> |
| Layout parser | <https://github.com/Layout-Parser/layout-parser>  <https://layout-parser.github.io/> |

Table extraction capability

* Precision, recall, F1, accuracy, specificity at retrieving tables + headers
* If it’s an object detection model 🡪 compare how well it identifies
* If it’s a table extraction model 🡪 compare with final results

|  |  |
| --- | --- |
| Lit search for any latest packages/papers |  |
| Extractable (built on top of Microsoft’s TATR) | <https://github.com/SuleyNL/Extractable>  <https://github.com/microsoft/table-transformer> |
| Nougat | <https://github.com/facebookresearch/nougat> |
| Deepdoctection | <https://deepdoctection.readthedocs.io/en/latest/>  <https://github.com/deepdoctection/deepdoctection> |
| Layout Parser | <https://github.com/Layout-Parser/layout-parser>  <https://layout-parser.github.io/> |
| Table Net | <https://github.com/AmanSavaria1402/TableNet> |
| (KIV) base TF-ID-model |  |

# **Manuals**

## Create COCO dataset using COCO-annotator [images of pdf pages with annotated schemes –captions included]

(common objects in context) COCO format

* Dictionary with 5 keys: info, licenses, images, annotations, categories
* Info (contains metadata of the dataset): a dictionary with the following keys –description, URL, version, year, contributor, date created
* Licenses (contains information about the licenses associated with the dataset): a list where each element is a dictionary with the following keys – URL, id, name
* Images (contains information about the images in the dataset): a list where each element is a dictionary with the following keys – id, license, coco URL, flickr URL, width, height, file name, date captured
* Annotations (contains annotation data about the objects within the images): a list where each element is a dictionary with the following keys – id, category id, iscrowd flag, segmentation (list of lists), image id, area, bbox (list of four floats)
* Categories (contains information about the object categories): a list of dictionaries with the following keys – supercategory, id, name

<https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/md-coco-overview.html>

Link: <https://github.com/jsbroks/coco-annotator/>

1. Convert individual pdf pages to high-resolution .png (using pyMuPDF or pdf2image)
2. Install docker desktop <https://docs.docker.com/desktop/install/windows-install/> and create an account
3. Perform the following in Powershell:
   1. confirm docker is successfully installed

$ docker --version

* 1. download COCO Annotator

$ git clone <https://github.com/jsbroks/coco-annotator>

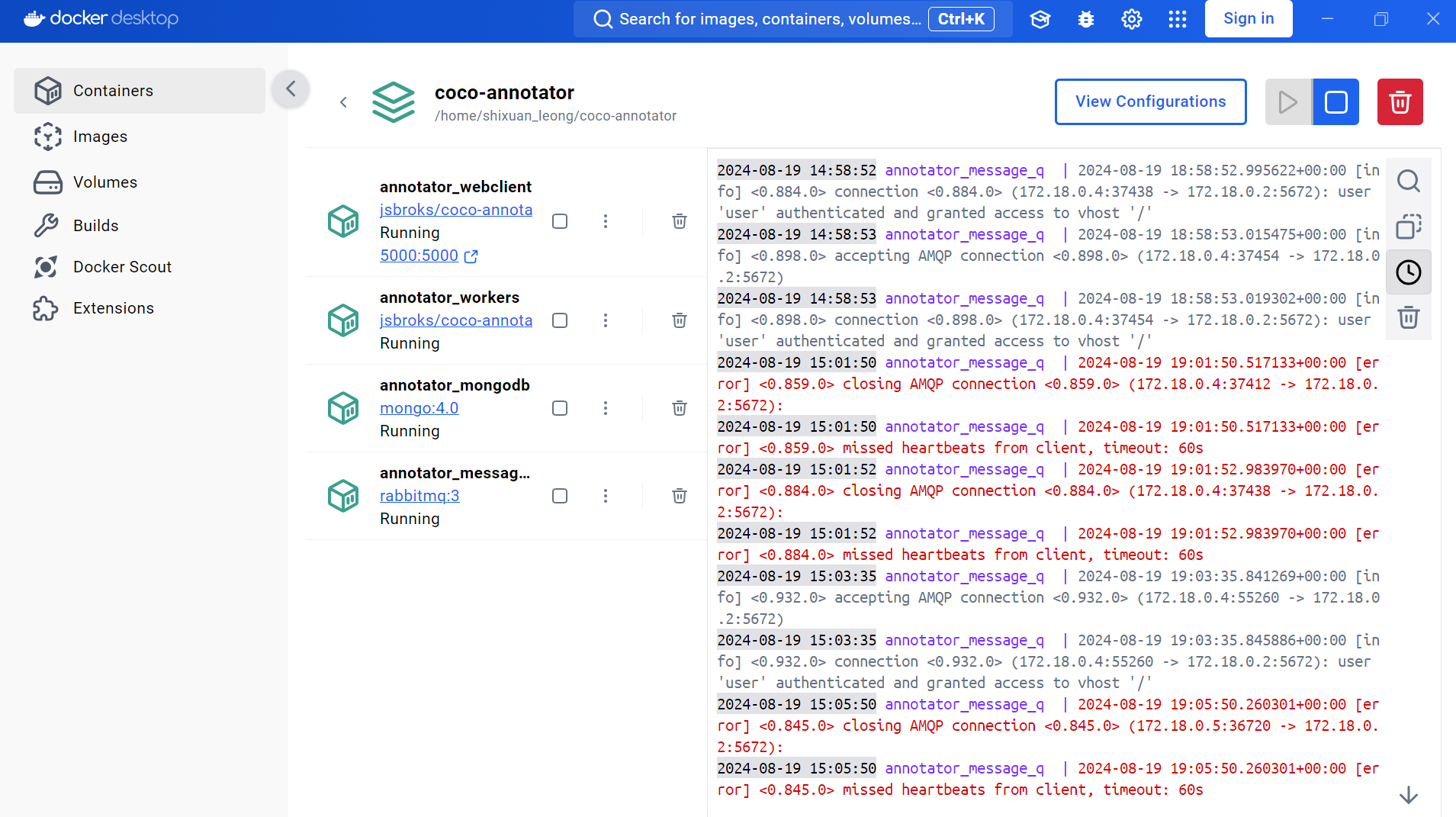
$ cd coco-annotator

* 1. install and run production build

$ docker-compose up

* 1. once docker container is loaded, open in docker desktop for easy navigation (press ‘v’ key)

1. In Docker Desktop:
   1. Navigate to Containers folder (found on the left panel) < annotator\_webclient < launch COCO annotator web application via local host machine (orange arrow)



1. The COCO Annotator web application looks like this:

A screenshot of a computer

Description automatically generated

1. Create new dataset and categories by selecting ‘Create’ and updating ‘Dataset name’ and ‘Default categories’ – e.g., ‘figure’, ‘table’, ‘scheme’.
2. Add new images to created dataset:
   1. Go to the root directory of your COCO Annotator project using File Explorer
   2. Copy your image files into the correct subfolder corresponding to the datasets you have created.
   3. The COCO annotator application should automatically detect the new images and update the dataset.
   4. Otherwise, use the ‘scan’ button in the web interface to force it to refresh or just click the Dataset button again.
   5. Go to the root directory of your COCO Annotator project. You should see a datasets folder.
3. To annotate the images within the dataset:
   1. Double click on the dataset
   2. Double click on the image to open it in the annotator
   3. Click on the + icon beside each category (in the right panel) to create an annotation in the corresponding category.
   4. Use the BBox tool (in the left panel) to draw the bounding box.
   5. Keyboard shortcuts:

*New annotation SPACE*

*Delete current annotation BACKSPACE*

*Undo last action CTRL + Z*

*BBox tool R*

*Remove current BBox ESCAPE*

*Save CTRL + S*

1. When all of the images are annotated, download the annotated dataset in COCO format using the DOWNLOAD COCO button. The dataset will be saved in the same folder as the dataset but in a .output subfolder
2. When done, close the Docker container using CTRL + C in POWERSHELL