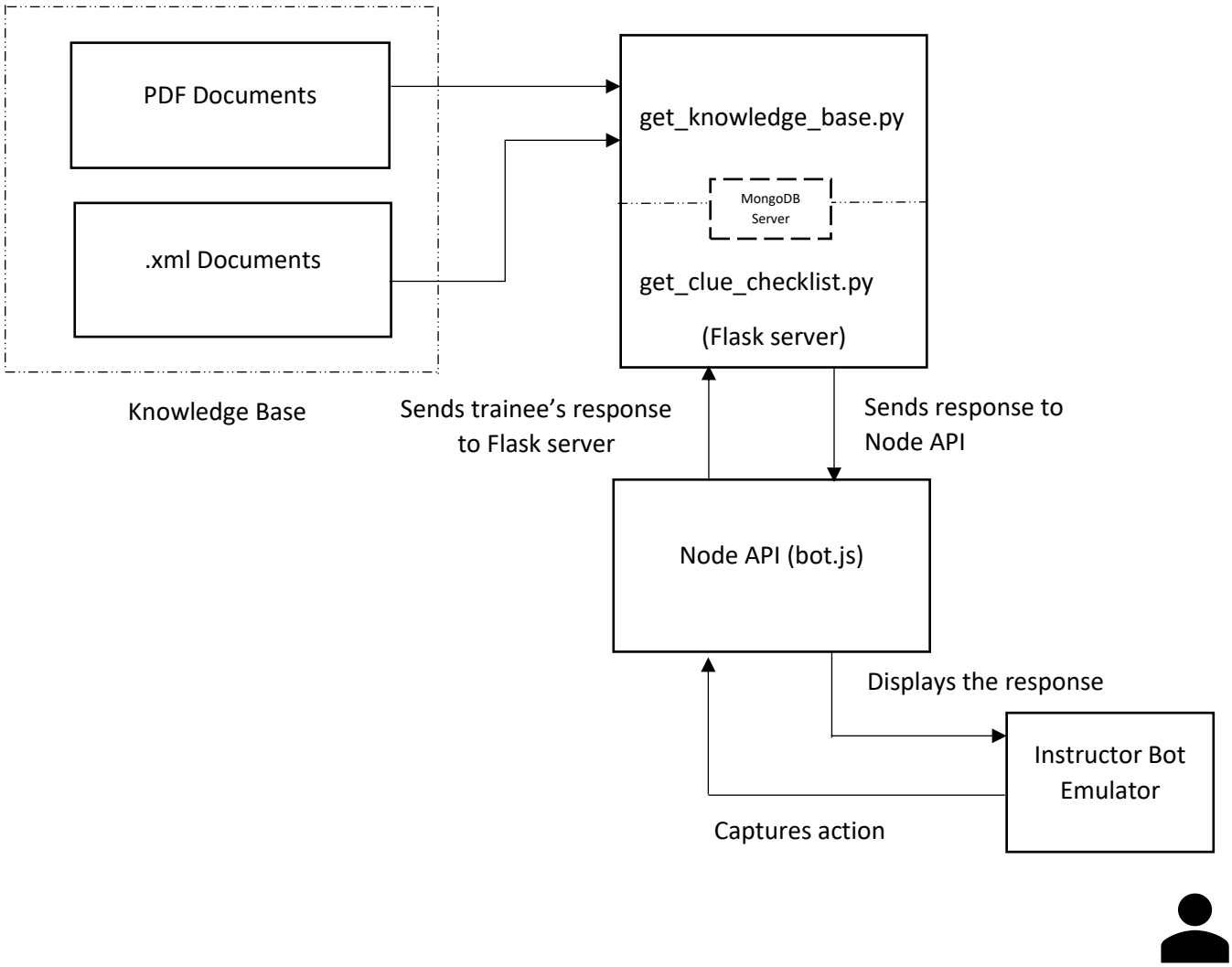
Intern: Ruchi Shukla

INSTRUCTOR BOT

Use Case: Provide smart responses to the Trainee based on the tag info extracted from the model documents

Architecture Diagram:



Part 1: Extraction of tag related info from the given set of pdf documents

Initial Attempts:

----Libraries used for extraction

1) **py2pdf**
   Limitation: Ignores whitespaces between the words
2) **pdfplumber**
   Advantage: Gives a lot of metadata about objects present in pdf (likes coordinates of table etc.)
   Limitation: Separated the whole word with random whitespaces.
3) **win32com**
   Advantage: Can understand structure of word document very well
   Limitation: - Only works for windows OS
   - Very few documentations available online

(Can be used when we want to extend the current implementation to word documents as well)

Currently in use:

Model Documents:

1) Set of pdf documents containing desired information
   Source: Trainer
2) .xml documents
   a) Procedure Checklists (PCs)
   b) Key Process Variables (KPVs)

   Source: PTS Application

Install Libraries:

```
pip install pdf2image
pip install pytesseract
pip install nltk
pip install pdfplumber
pip install pymongo
pip install pandas
pip install spacy
pip install xml
```

Additional software to use these libraries:

For pdf2image, install poppler:
http://poppler%20for%20windows%20users.%20https//github.com/oschwartz10612/poppler-windows/releases/

For pytesseract, install Tesseract OCR Engine:

https://tesseract-ocr.github.io/tessdoc/Downloads.html

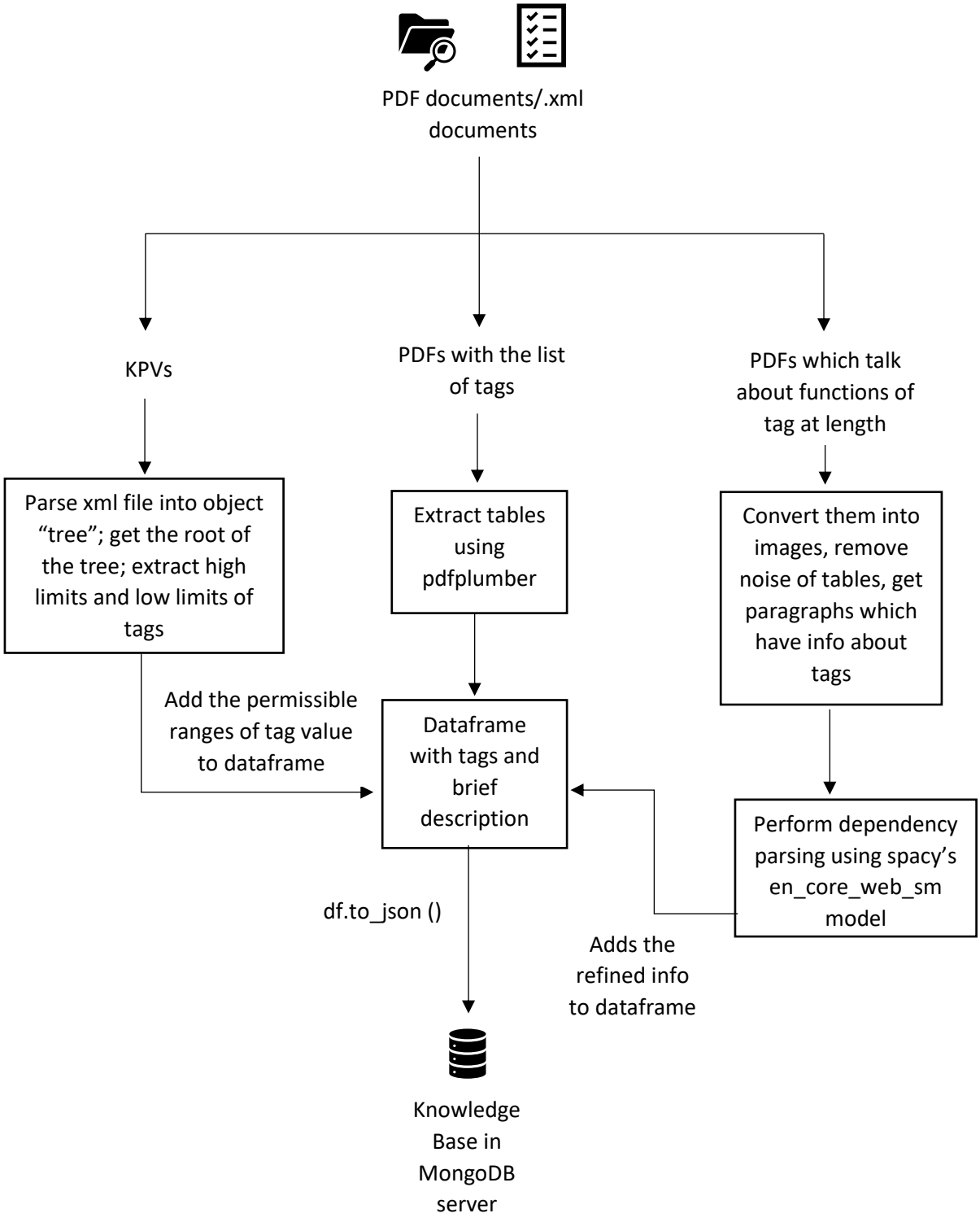
After you download and install the software, you can add their executable paths into Environment Variables. Alternatively, you can directly include their paths in the program.

```
poppler_path=r'C:\Users\H501271\Downloads\Release-22.04.0-0\poppler-22.04.0\Library\bin'
pytesseract.pytesseract.tesseract_cmd = r"C:\Users\H501271\AppData\Local\Programs\Tesseract-OCR\tesseract.exe"
```

Steps for running current implementation:

1) Build the knowledge base by first running get_knowledge_base.py, which will store the database in MongoDB server
2) Run flask server; get_clue_checklist.py
3) Run the Node API
4) Launch the bot emulator:
   Endpoint: http://localhost:3978/api/messages
5) Start taking exercises

get_knowledge_base.py



Part 2: Generate clues out of the information extracted and get the steps to perform the exercise from procedure checklist xml files and accept requests from Node API

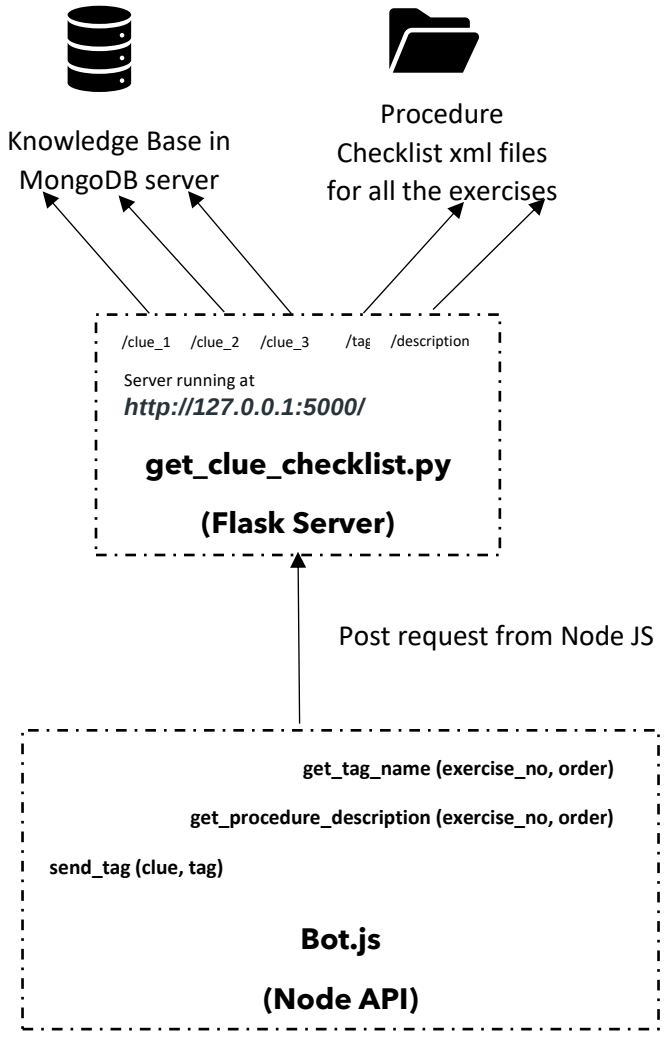
Our bot is working in node js but we want to process the data in python. So, we are using a **Flask** microframework to set up a server in python and post requests from Node JS

Install flask module for python:

```
pip install flask
```

Install request module for Node JS

```
npm install request-promise
```



Note: Currently, the event info being sent by PTS has all the events, some might not even be of relevance to us, so filter the event info and then one can try to map actions using get_procedure_description () or get_tag_name () function

Part 3: Cause-effect learning model

init.py

Distillation.xml

processing using
xml.etree.ElementTree

Causes	11LC16	11TI21	11LC14	11FC15	11TI24	11PI14	11TI26	11TI22	11FC19	11PC15	11FC17	11PC16	11TI20	11TI25	11FC20	11AC12
11AT12: Overhead C5 composition analyser fails high	-2	0	1	1	0	0	-1	0	2	0	1	1	0	-2	-2	2
11HS14: Overhead condenser fans fail off	-2	-1	1	1	-1	1	1	0	-1	2	2	2	0	-2	-2	-2
11PV15: Overhead pressure control valve fails closed	-2	0	2	2	-1	2	0	0	1	0	-2	0	0	0	0	0
11FV15: Rebolier condensate flow control valve fails closed	-2	-1	2	-2	-1	-1	0	-1	0	-1	-2	-1	-1	-1	-1	-2
11PV16A: Reflux drum pressure control valve fails	0	0	0	0	0	0	-1	0	0	0	0	-2	0	0	0	-1

cause-effect dataframe

One Hot Encoding

	-2	-1	0	1	2
0	1.0	NaN	0.0	0.0	0.0
1	1.0	NaN	0.0	0.0	0.0
2	1.0	NaN	0.0	0.0	0.0
3	1.0	NaN	0.0	0.0	0.0
4	0.0	NaN	1.0	0.0	0.0
5	1.0	NaN	0.0	0.0	0.0
6	0.0	NaN	0.0	0.0	1.0
7	0.0	NaN	0.0	0.0	1.0
8	0.0	NaN	0.0	1.0	0.0

tag_matrix

calculate probability

```
[[0.2 0 0.0 0.0 0.0]
 [0.2 0 0.0 0.0 0.0]
 [0.2 0 0.0 0.0 0.0]
 [0.2 0 0.0 0.0 0.0]
 [0.0 0 1.0 0.0 0.0]
 [0.2 0 0.0 0.0 0.0]
 [0.0 0 0.0 0.0 0.5]
 [0.0 0 0.0 0.0 0.5]
 [0.0 0 0.0 1.0 0.0]]
```

probability_tag_matrix

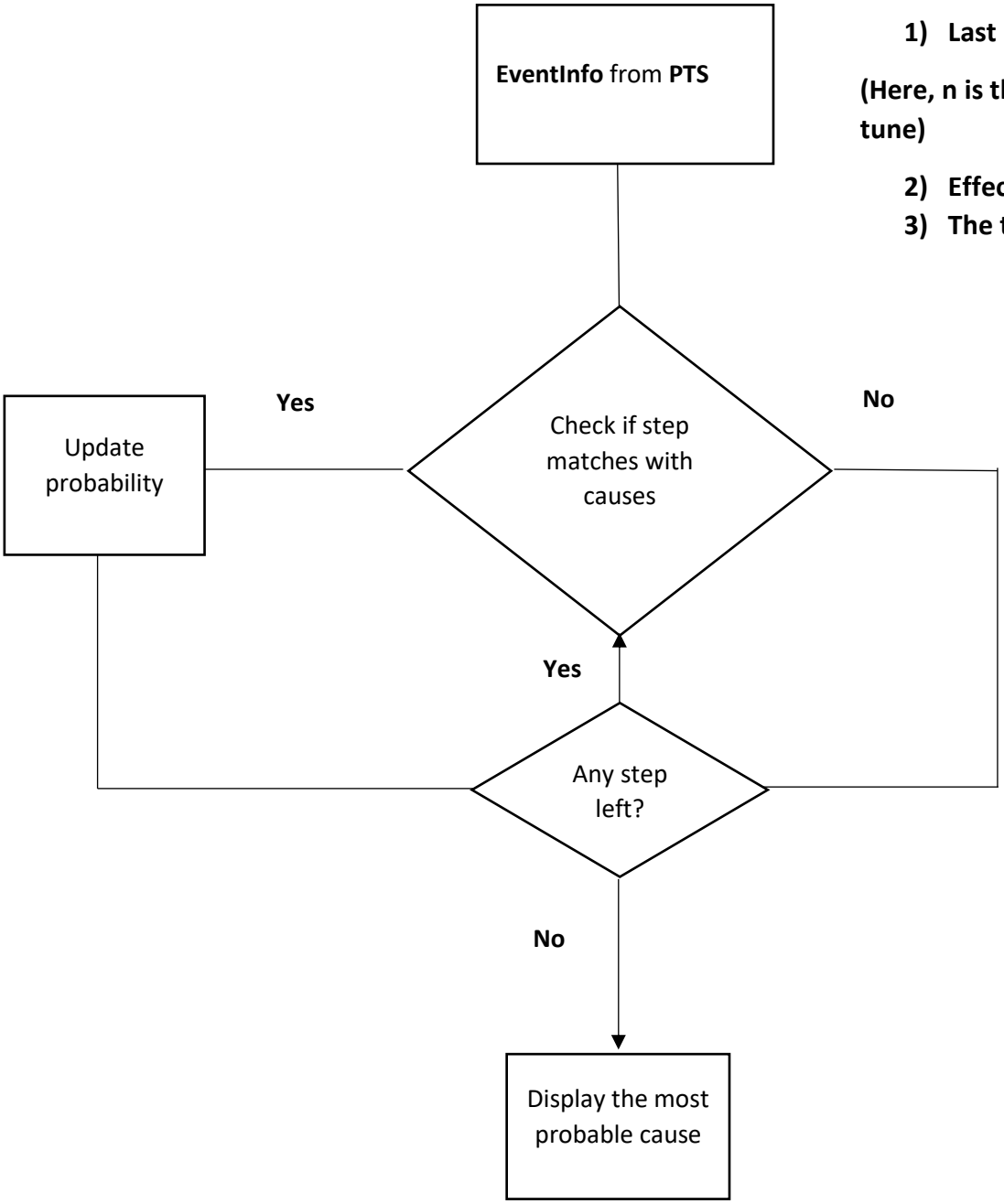
update.py

Keep record of the following:

1) **Last n steps**

(Here, n is the hyperparameter to tune)

2) **Effect on tag**
3) **The tag**



updated probability = $\frac{Number\ of\ times\ the\ cause\ was\ present\ in\ last\ n\ steps}{Total\ number\ of\ times\ effect\ on\ tag\ observed}$