**Steps to create the tables:**

First it is needed to export Zotero data as csv file and then run the following code. The following also does topic modelling and creates the required dataframes.

1. import pandas as pd

2. import numpy as np

3. from tqdm import tqdm

4. from nltk.stem.porter import PorterStemmer

5. from nltk.corpus import stopwords, wordnet

6. from nltk import sent\_tokenize, word\_tokenize

7. import re

8. import nltk

9. nltk.download('punkt')

10. nltk.download('stopwords')

11.

12. papers\_df = pd.read\_csv("SCIPdb.csv")

13.

14. df\_abs = papers\_df['Abstract Note'].dropna().reset\_index()

15. df\_pub\_year = papers\_df ['Publication Year'][df\_abs['index']].reset\_index()

16. doc\_lst = list(df\_abs['Abstract Note'])

17.

18. def clean\_doc(text):

19. stemmer = PorterStemmer()

20. txt = re.sub(r'[^\w\s\.]', '', text.lower().strip())

21. words = word\_tokenize(txt)

22. words = [stemmer.stem(word) for word in words if word not in stopwords.words('english')]

23. data = " ".join(words)

24. return data

25.

26. doc\_lst = [clean\_doc(doc) for doc in tqdm(doc\_lst)]

27.

28. from bertopic import BERTopic

29. from sentence\_transformers import SentenceTransformer

30.

31. sentence\_model = SentenceTransformer("pritamdeka/S-PubMedBert-MS-MARCO")

32.

33. topic\_model0 = BERTopic(language="english", embedding\_model=sentence\_model, n\_gram\_range=(1,3), calculate\_probabilities=True, verbose=True)

34. topics0, probs0 = topic\_model0.fit\_transform(doc\_lst)

35.

36. freq = topic\_model0.get\_topic\_info()

37.

38. topics0 = topic\_model0.reduce\_outliers(doc\_lst, topics0)

39. from collections import Counter

40. item\_counts = Counter(topics0)

41. freq['Count'] = [item\_counts[id] for id in range(-1, max(item\_counts.keys())+1)]

42. freq\_table = freq

43.

44. df\_topic\_year = pd.concat([pd.DataFrame(topics0, columns=["topics"]), df\_pub\_year["Publication Year"]], axis=1)

45.

46. index\_topics = papers\_df['Abstract Note'].isna()[papers\_df['Abstract Note'].isna()==False].index

47.

48. papers\_df['Topics'] = np.nan

49. papers\_df.loc[index\_topics, 'Topics'] = df\_topic\_year['topics'].values

50.

Once you have the pdf files of the papers,

The following code will load the texts and their metadata.

1. import os

2. from langchain.document\_loaders import PyPDFLoader

3. from tqdm import tqdm

4.

5. files = papers\_df["File Attachments"] # change this line according to your dir of the pdfs

6. all\_data = []

7. errored\_files = []

8. i = 0

9. for file in tqdm(files):

10. try:

11. file = file.replace("\\", "/").replace(';', '')

12. loader = PyPDFLoader(file.strip())

13. data = loader.load()

14. for d in data:

15. d.metadata['Publication Year'] = int(papers\_df['Publication Year'][i])

16. d.metadata['Date Added'] = int(papers\_df['Date Added'][i][:4])

17. d.metadata['Author'] = str(papers\_df['Author'][i])

18. d.metadata['Title'] = str(papers\_df['Title'][i])

19. all\_data.append(data)

20. except Exception as e:

21. #print(e)

22. errored\_files.append(file)

23. finally:

24. i += 1

25.

Then the following code will chunk the texts.

1. from langchain.text\_splitter import RecursiveCharacterTextSplitter

2. text\_splitter = RecursiveCharacterTextSplitter(

3. chunk\_size = 1500,

4. chunk\_overlap = 50

5. )

6.

7. def func(doc):

8. doc.page\_content = doc.page\_content.encode().decode("utf-8", errors="replace").replace("\x00", "\uFFFD")

9. return doc

10.

11. flattened\_data = [func(item) for sublist in all\_data for item in sublist]

12.

13. splits = text\_splitter.split\_documents(flattened\_data)

14. len(splits)

15.

Then the following code will create “user\_table\_metadata” dataframe which is a table for user interactions and application sql tool interactions which includes only the metadata.

1. meta\_data = {}

2. for s in splits:

3. for key in s.metadata:

4. if key not in meta\_data.keys():

5. meta\_data[key] = [s.metadata[key]]

6. else:

7. meta\_data[key].append(s.metadata[key])

8.

9. user\_table\_metadata = pd.DataFrame(meta\_data).drop(columns=['source'])\

10. .drop\_duplicates(subset=['Title'], keep='last')\

11. .reset\_index().drop(columns=['page', 'index'])

The following code will create “user\_table\_chunks” dataframe which only includes the chunks.

1. chunks = {'string\_value':[], 'paper\_id': []}

2. for s in splits:

3. chunks['paper\_id'].append(s.metadata['Title'])

4. chunks['string\_value'].append(s.page\_content)

5.

6. user\_table\_chunks = pd.DataFrame(chunks)

7. user\_table\_chunks['paper\_id'] = pd.factorize(user\_table\_chunks['paper\_id'])[0]

Then the following codes will create the tables and add the data to postgresql on aws. Both vector table and user table and also establishes the foreign keys needed.

1. from sqlalchemy import create\_engine, text

2. from sqlalchemy.exc import SQLAlchemyError

3.

4. connection\_string = "postgresql+psycopg://bsituser:M4pbcMDsbm30zDV6@awseb-e-mmtzduxdgy-stack-awsebrdsdatabase-a1ggrejgeign.cp5mioiwgdbp.ca-central-1.rds.amazonaws.com:5432/postgres"

5.

6. engine = create\_engine(connection\_string, echo=True)

7. connection = engine.connect().execution\_options(isolation\_level="AUTOCOMMIT")

8. try:

9. connection.execute(text("CREATE DATABASE user\_db;"))

10. connection.execute(text("CREATE DATABASE vector\_db;"))

11. except Exception as e:

12. print(f"Error: {e}")

13. finally:

14. connection.close()

15.

16. connection\_string = "postgresql+psycopg://bsituser:M4pbcMDsbm30zDV6@awseb-e-mmtzduxdgy-stack-awsebrdsdatabase-a1ggrejgeign.cp5mioiwgdbp.ca-central-1.rds.amazonaws.com:5432/vector\_db"

17.

18. engine = create\_engine(connection\_string, echo=True)

19.

20. COLLECTION\_NAME = 'state\_of\_union\_vectors'

21. embeddings = OpenAIEmbeddings()

22.

23. vectorstore = PGVector(

24. embedding\_function=embeddings,

25. collection\_name=COLLECTION\_NAME,

26. connection\_string=connection\_string,

27. # use\_jsonb=True,

28. )

29.

30. from tqdm import tqdm

31. batch\_size = 1000 # Adjust this based on your requirements

32. for i in tqdm(range(0, len(splits), batch\_size)):

33. batch = splits[i:i + batch\_size]

34. vectorstore.add\_documents(batch)

35.

36. from sqlalchemy import create\_engine, text

37. from sqlalchemy.exc import SQLAlchemyError

38.

39. connection\_string = "postgresql+psycopg://bsituser:M4pbcMDsbm30zDV6@awseb-e-mmtzduxdgy-stack-awsebrdsdatabase-a1ggrejgeign.cp5mioiwgdbp.ca-central-1.rds.amazonaws.com:5432/user\_db"

40.

41. engine = create\_engine(connection\_string, echo=True)

42.

43. user\_table\_metadata\_with\_topics.to\_sql('metadata', engine, if\_exists='append', index=True, index\_label='id')

44. user\_table\_chunks.to\_sql('chunks', engine, if\_exists='append', index=True, index\_label='id')

45. freq\_table.to\_sql('topics', engine, if\_exists='append', index=True, index\_label='id')

46.

47. create\_primary\_key = text("""

48. ALTER TABLE metadata

49. ADD PRIMARY KEY (id);

50. """

51. )

52.

53. create\_foreign\_key = text("""

54. ALTER TABLE chunks

55. ADD CONSTRAINT fk\_paper\_id

56. FOREIGN KEY (paper\_id)

57. REFERENCES metadata(id);

58. """

59. )

60.

61. change\_column\_type = text("""

62. ALTER TABLE metadata

63. ALTER COLUMN "Topics" TYPE BIGINT;

64. """)

65.

66. checking\_the\_relation = text("""

67. SELECT

68. tc.constraint\_name,

69. tc.table\_name,

70. kcu.column\_name,

71. ccu.table\_name AS foreign\_table\_name,

72. ccu.column\_name AS foreign\_column\_name

73. FROM

74. information\_schema.table\_constraints AS tc

75. JOIN information\_schema.key\_column\_usage AS kcu

76. ON tc.constraint\_name = kcu.constraint\_name

77. JOIN information\_schema.constraint\_column\_usage AS ccu

78. ON ccu.constraint\_name = tc.constraint\_name

79. WHERE

80. tc.constraint\_type = 'FOREIGN KEY' AND tc.table\_name='chunks';

81.

82. """)

83.

84. adding\_unique\_constraint = text("""

85. ALTER TABLE topics

86. ADD CONSTRAINT unique\_topic UNIQUE ("Topic");

87. """)

88.

89. topic\_foreign\_key = text("""

90. ALTER TABLE metadata

91. ADD CONSTRAINT fk\_topic\_num

92. FOREIGN KEY ("Topics")

93. REFERENCES topics("Topic");

94. """)

95.

96.

97. try:

98. with engine.connect() as connection:

99. # Start a transaction

100. with connection.begin():

101. connection.execute(create\_primary\_key)

102. connection.execute(create\_foreign\_key)

103. connection.execute(change\_column\_type)

104. connection.execute(adding\_unique\_constraint)

105. connection.execute(topic\_foreign\_key)

106.

107. # Check the relations

108. result = connection.execute(checking\_the\_relation)

109. for row in result:

110. print(row)

111.

112. print("Successful.")

113. except SQLAlchemyError as e:

114. print(f"An error occurred: {e}")

115. finally:

116. connection.close()

117.

After all these steps run, we can run the app and chat with it.