Personal Information

Name: **Yanchao MURONG**

StudentID: **14090759**

Email: **yanchao.murong@student.uva.nl**

Submitted on: **25.03.2024**

Purpose (Sub Research Question 1)

What are the syntactical structure differences between current synthetic questions and real human questions ?

Why we analyze the query syntactical structure differences ?

During our preliminary experimentation in the domain of statutory legal information retrieval, we find that GPL (Generative Pseudo Labeling) perform slightly better than zero-shot model such as BM25 (MRR@100: 27.99 vs 24.45) but much worse than human labeled questions (MRR@100: 40.2).

According to related work, synthetic query generative distribution suffers from unfaithfulness due to the statistical distribution rooted in existing generative models, which can influence the effectiveness of the end-to-end domain adaptation. Additionally, legal question generation is challenging as the legal corpus are often lengthy and have very complex sentence structure with frequent use of abbreviations. As generation based domain adaptation technique such as GPL heavily relies on generated synthetic queries, we assume that the differences in synthetic query structure generated by language model can have an impact on statutory legal information retrieval performance.

The understanding of these differences can help us find a way to reduce the distribution gap afterwards and hence studies its impact on statutory legal information retrieval performance.

Dataset

BSARD Dataset Human Questions (dfQ_human):

The Belgian Statutory Article Retrieval Dataset (BSARD) is a dataset created for the task of statutory article retrieval (SAR). The corpus consisted of 22,633 law articles extracted from 32 publicly available Belgian codes. The questions were collected from Droits Quotidiens, an organization that clarifies the law for laypeople, and were carefully labeled with the ids of the corresponding relevant law articles from the corpus. The dataset was split into training/test sets with 886 and 222 questions, respectively. The dataset has a wide range of legal topics, with around 85% of the questions being about family, housing, money, or justice. The questions might have one or several relevant legal articles, and 75% of the questions have less than five relevant articles. Overall, out of the 22,633 articles, only 1612 are referred to as relevant to at least one question in the dataset, and around 80% of these articles come from either the Civil Code, Judicial Code, Criminal Investigation Code, or Penal Code.

> The author of BSARD dataset has provided its overall dataset analysis on its original paper. In our EDA, we focus more on the syntactical structure aspect.

Synthetic Dataset 1 (dfQ_syn1): Synthetic questions data generated by the model mT5 (doc2query/msmarco-french-mt5-base-v1). We have generated 1 question for each legal article from BSARD Dataset (due to our computation limit).

Synthetic Dataset 2 (dfQ_syn2): Synthetic questions data generated by the author of BSARD Dataset. 5 questions for each legal article from BSARD Dataset are generated.

Dataset MS Marco Human Questions (dfQ_mmarco): Queries from the MS MARCO dataset which features real Bing questions and a human generated answer translated in French. Many languages models are fine tuned based on MS MARCO dataset such as the one we used doc2query/msmarco-french-mt5-base-v1.

We plan to generate more sythetic questions through LLM and perform comparative analysis afterwards.

Data Description

In [4]:
```
# Imports
import numpy as np
import pandas as pd
from parallel_pandas import ParallelPandas
import matplotlib.pyplot as plt
import spacy
from tqdm import tqdm
ParallelPandas.initialize(disable_pr_bar=False)
tqdm.pandas(desc='Processing text')
nlp = spacy.load("fr_core_news_sm")
```

Data Loading

In [8]:
```python
# Load your data here
dfQ_train = pd.read_csv("./data/questions_fr_train.csv")
dfQ_test = pd.read_csv("./data/questions_fr_test.csv")

dfQ_human = pd.concat([dfQ_train, dfQ_test])
dfQ_syn1 = pd.read_json("./data/qgen-queries.jsonl", lines=True)
dfQ_syn2 = pd.read_csv("./data/questions_synthetic.csv")

dfQ_mmarco = pd.read_csv("./data/mmarco/french_queries.train.tsv", sep='\t', header
```

In [9]:
```python
dfQ_human.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1108 entries, 0 to 221
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Unnamed: 0         1108 non-null   int64 
 1   id                 1108 non-null   int64 
 2   category           1108 non-null   object
 3   subcategory        1108 non-null   object
 4   question           1108 non-null   object
 5   extra_description  990 non-null    object
 6   article_ids        1108 non-null   object
 7   article_ids.1      222 non-null    object
dtypes: int64(2), object(6)
memory usage: 77.9+ KB
```

In [11]:
```python
dfQ_syn2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113165 entries, 0 to 113164
Data columns (total 3 columns):
 #   Column       Non-Null Count   Dtype 
---  ------       --------------   ----- 
 0   id           113165 non-null  int64 
 1   question     113165 non-null  object
 2   article_ids  113165 non-null  int64 
dtypes: int64(2), object(1)
memory usage: 2.6+ MB
```

Analysis 1: Number of Questions

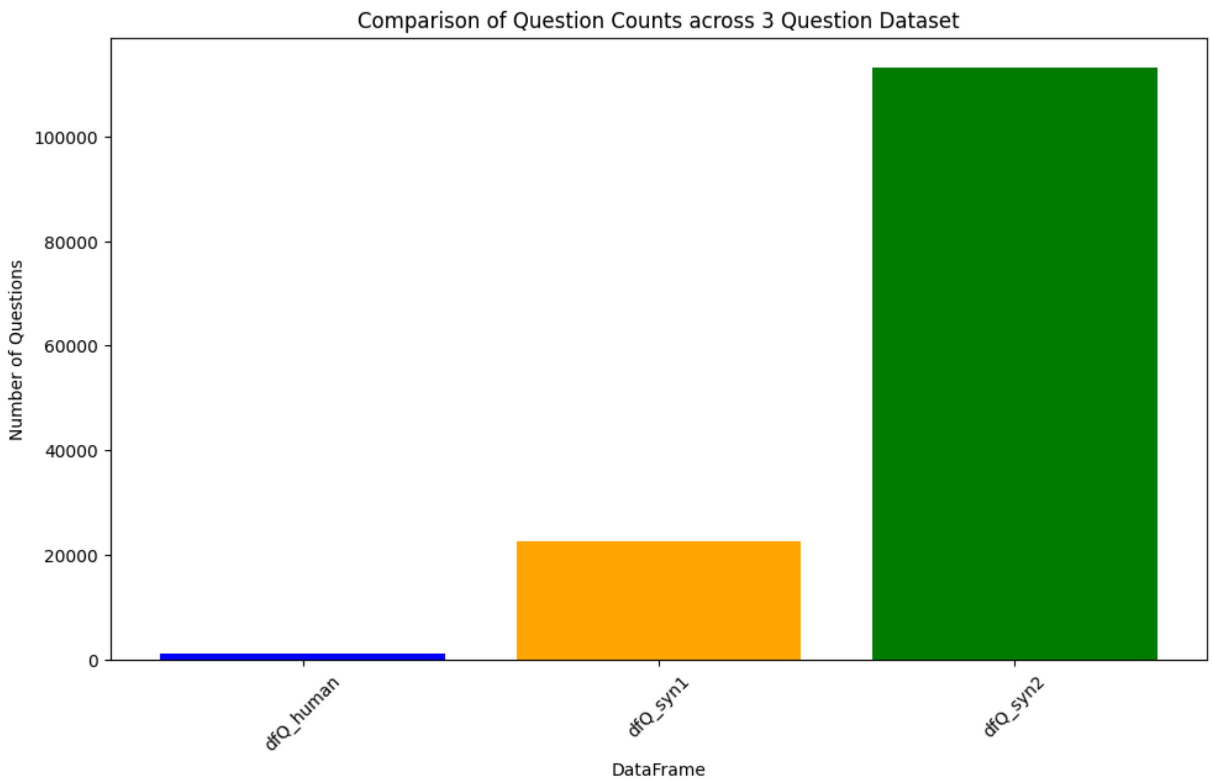
In [3]:
```python
# Count the number of questions in each DataFrame
counts = [len(df) for df in [dfQ_human, dfQ_syn1, dfQ_syn2]]

# Create a summary DataFrame
summary_df = pd.DataFrame({
    'DataFrame': ['dfQ_human', 'dfQ_syn1', 'dfQ_syn2'],
    'Number of Questions': counts
})
summary_df
```

Out[3]:

	DataFrame	Number of Questions
0	dfQ_human	1108
1	dfQ_syn1	22633
2	dfQ_syn2	113165

In [4]:
```
# Plotting
plt.figure(figsize=(10, 6))
plt.bar(summary_df['DataFrame'], summary_df['Number of Questions'], color=['blue',
plt.title('Comparison of Question Counts across 3 Question Dataset')
plt.xlabel('DataFrame')
plt.ylabel('Number of Questions')
plt.tight_layout()
plt.savefig('Comparison of Question Counts across 3 Question Dataset.pdf')
plt.xticks(rotation=45)
```

Out[4]:
```
([0, 1, 2],
 [Text(0, 0, 'dfQ_human'), Text(1, 0, 'dfQ_syn1'), Text(2, 0, 'dfQ_syn2')])
```



We can see that human questions are far less than synthetic questions.

Analysis 2: Question Length

In [41]:
```
# Calculate the length of each question
dfQ_human['Length'] = dfQ_human['question'].apply(len)
dfQ_syn1['Length'] = dfQ_syn1['text'].apply(len)
dfQ_syn2['Length'] = dfQ_syn2['question'].apply(len)
dfQ_mmarco['Length'] = dfQ_mmarco['question'].apply(len)
```

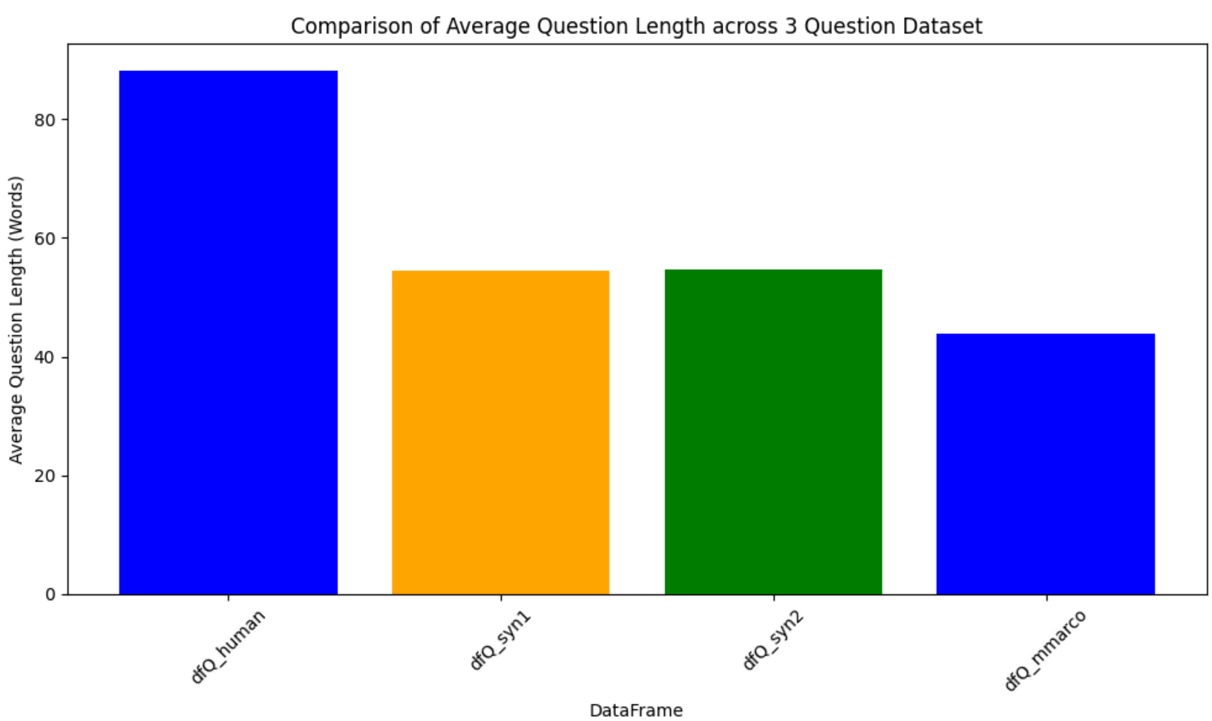
```python
# Compute the average length of questions in each DataFrame
avg_lengths = [df['Length'].mean() for df in [dfQ_human, dfQ_syn1, dfQ_syn2, dfQ_mm

# Create a summary DataFrame
summary_df = pd.DataFrame({
    'DataFrame': ['dfQ_human', 'dfQ_syn1', 'dfQ_syn2', 'dfQ_mmarco'],
    'Average Question Length': avg_lengths
})

summary_df
```

Out[41]:

	DataFrame	Average Question Length
0	dfQ_human	88.252708
1	dfQ_syn1	54.395308
2	dfQ_syn2	54.762665
3	dfQ_mmarco	43.806178

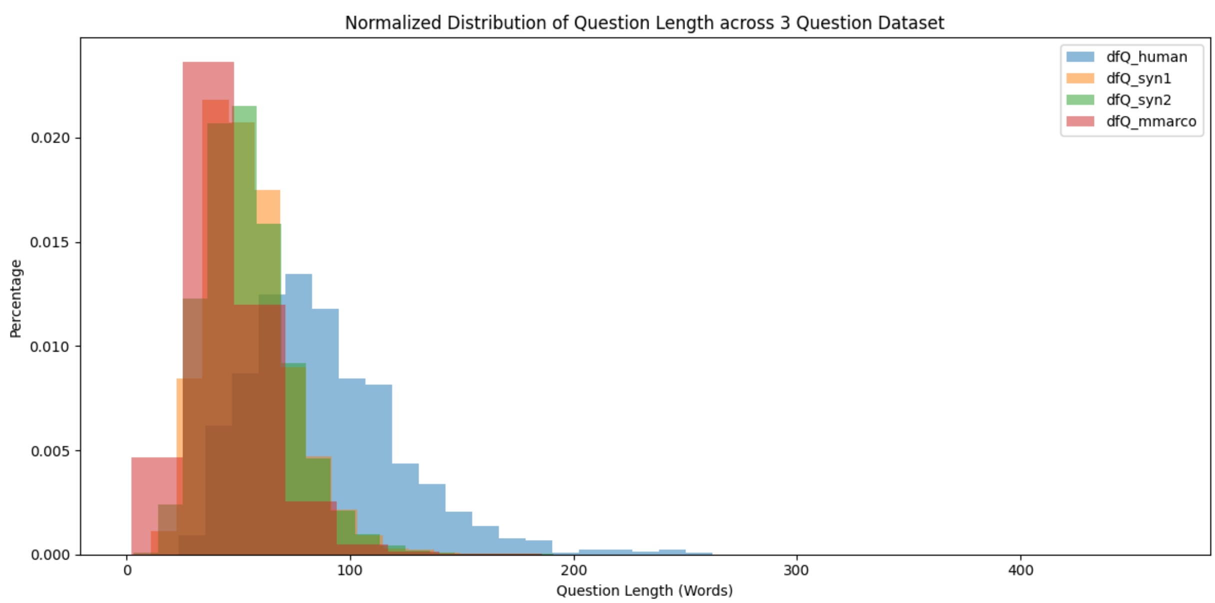
In [42]:

```python
# Plotting
plt.figure(figsize=(10, 6))
plt.bar(summary_df['DataFrame'], summary_df['Average Question Length'], color=['blu
plt.title('Comparison of Average Question Length across 3 Question Dataset')
plt.xlabel('DataFrame')
plt.ylabel('Average Question Length (Words)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('Comparison of Average Question Length across DataFrames.pdf')
plt.show()
```



In average, legal human questions are much longer than synthetic questions and mmarco human queries.

In [43]:
```
# Plotting
plt.figure(figsize=(12, 6))

# Histograms for each DataFrame
plt.hist(dfQ_human['Length'], alpha=0.5, label='dfQ_human', bins=20, density=True)
plt.hist(dfQ_syn1['Length'], alpha=0.5, label='dfQ_syn1', bins=20, density=True)
plt.hist(dfQ_syn2['Length'], alpha=0.5, label='dfQ_syn2', bins=20, density=True)
plt.hist(dfQ_mmarco['Length'], alpha=0.5, label='dfQ_mmarco', bins=20, density=True

plt.title('Normalized Distribution of Question Length across 3 Question Dataset')
plt.xlabel('Question Length (Words)')
plt.ylabel('Percentage')
plt.legend()
plt.tight_layout()
plt.savefig('Normalized Distribution of Question Lengths.pdf')
plt.show()
```



Normalized distribution have demonstrated the same conclusion: In legal domain, people tend to ask longer question.

Analysis 3: Number of Sentences in Question

In [12]:
```
class TextPreprocessor():
    def __init__(self, spacy_model="fr_core_news_sm"):
        self.nlp = spacy.load(spacy_model)

    def preprocess(self, series, sentences=False):
        return (series.progress_apply(
            lambda text: self.preprocess_text(text, sentences)))

    def preprocess_text(self, text, split_sentences):
```

```python
        doc = self.nlp(text)
        if split_sentences:
            return list(doc.sents)
        else:
            text = self._get_text(doc)
        return text

    def _get_text(self, doc):
        return ' '.join([t.text for t in doc])


processor = TextPreprocessor()

dfQ_human['NB_SENTENCES'] = processor.preprocess(dfQ_human["question"], sentences=T
dfQ_syn1['NB_SENTENCES'] = processor.preprocess(dfQ_syn1["text"], sentences=True).a
dfQ_syn2['NB_SENTENCES'] = processor.preprocess(dfQ_syn2["question"], sentences=Tru
```

```
Processing text: 100%|██████████| 1108/1108 [00:04<00:00, 253.27it/s]
Processing text: 100%|██████████| 22633/22633 [01:16<00:00, 294.04it/s]
Processing text: 100%|██████████| 113165/113165 [06:04<00:00, 310.05it/s]
```

In [45]:
```python
dfQ_mmarco['NB_SENTENCES'] = processor.preprocess(dfQ_mmarco["question"].sample(100
```

```
Processing text: 100%|██████████| 10000/10000 [00:30<00:00, 322.65it/s]
```

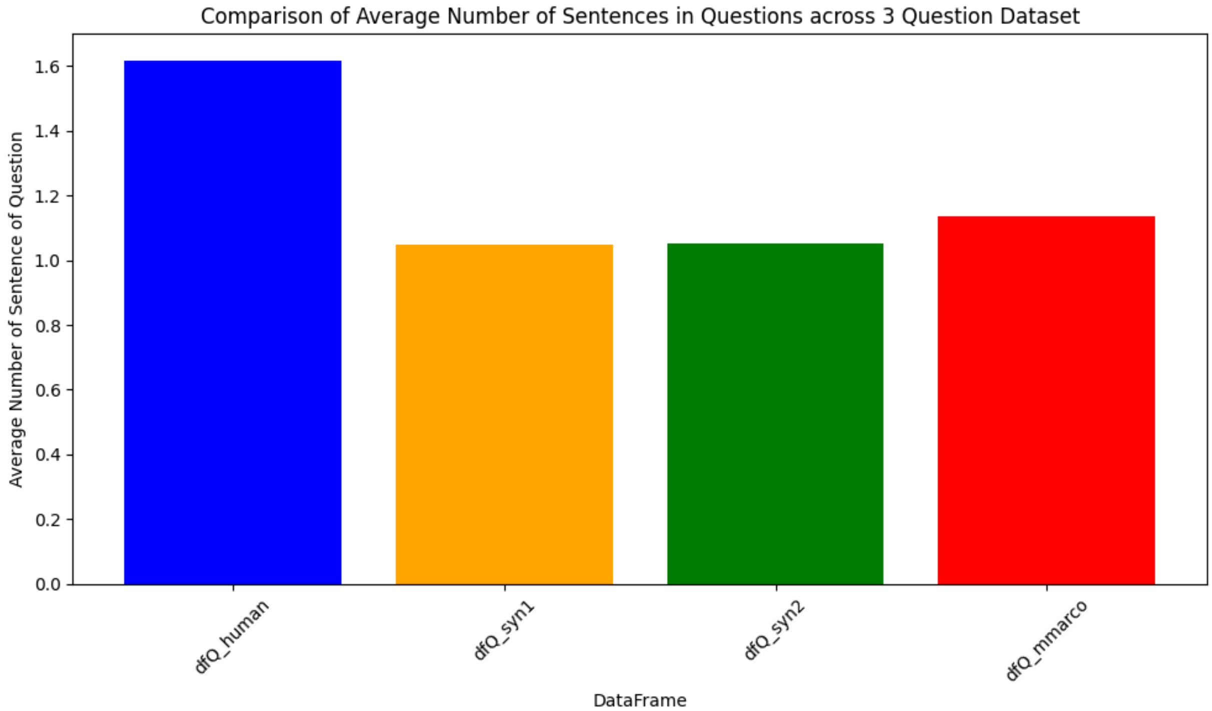
In [46]:
```python
# Compute the average length of questions in each DataFrame
avg_lengths = [df['NB_SENTENCES'].mean() for df in [dfQ_human, dfQ_syn1, dfQ_syn2,

# Create a summary DataFrame
summary_df = pd.DataFrame({
    'DataFrame': ['dfQ_human', 'dfQ_syn1', 'dfQ_syn2', 'dfQ_mmarco'],
    'Average Number of Sentences in Question': avg_lengths
})

summary_df
```

Out[46]:

	DataFrame	Average Number of Sentences in Question
0	dfQ_human	1.618231
1	dfQ_syn1	1.048867
2	dfQ_syn2	1.051562
3	dfQ_mmarco	1.134600

In [47]:
```python
# Plotting
plt.figure(figsize=(10, 6))
plt.bar(summary_df['DataFrame'], summary_df['Average Number of Sentences in Questio
        color=['blue', 'orange', 'green', 'red'])
plt.title('Comparison of Average Number of Sentences in Questions across 3 Question
plt.xlabel('DataFrame')
plt.ylabel('Average Number of Sentence of Question')
plt.xticks(rotation=45)
plt.tight_layout()
```

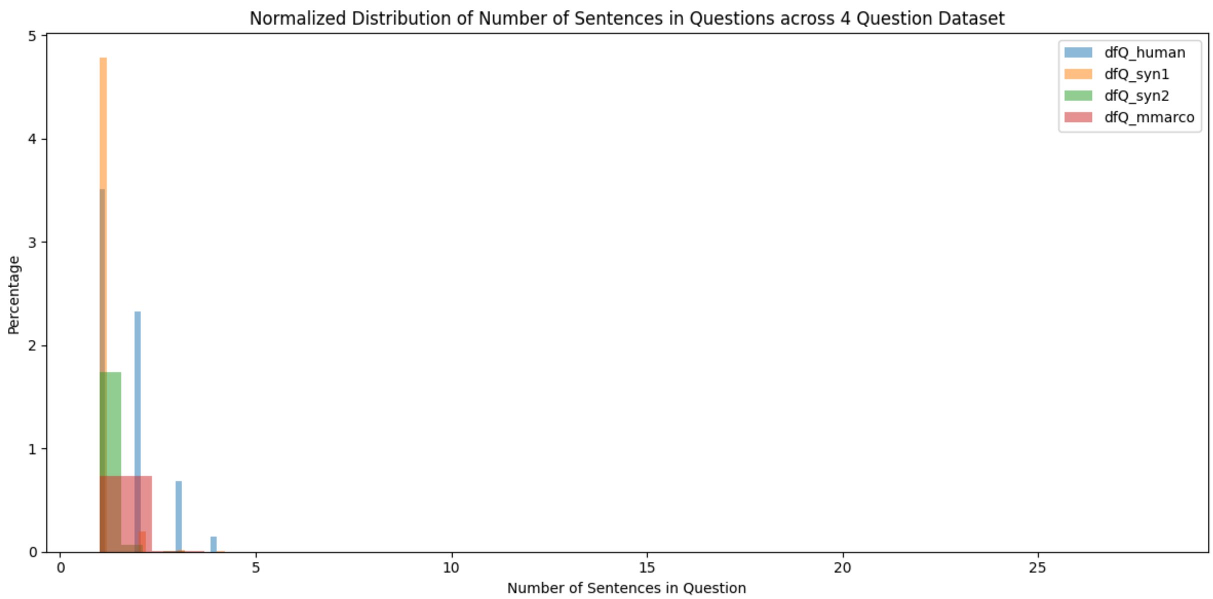
```
plt.savefig('Comparison of Average Number of Sentence of Question across DataFrames
plt.show()
```



We find that legal human question tends to be composed by more than one sentence. That partially explains why legal human question are longer the the other questions.

In [48]:

```
# Plotting
plt.figure(figsize=(12, 6))

# Histograms for each DataFrame
plt.hist(dfQ_human['NB_SENTENCES'], alpha=0.5, label='dfQ_human', bins=20, density=
plt.hist(dfQ_syn1['NB_SENTENCES'], alpha=0.5, label='dfQ_syn1', bins=20, density=Tr
plt.hist(dfQ_syn2['NB_SENTENCES'], alpha=0.5, label='dfQ_syn2', bins=20, density=Tr
plt.hist(dfQ_mmarco['NB_SENTENCES'], alpha=0.5, label='dfQ_mmarco', bins=20, densit

plt.title('Normalized Distribution of Number of Sentences in Questions across 4 Que
plt.xlabel('Number of Sentences in Question')
plt.ylabel('Percentage')
plt.legend()
plt.tight_layout()
plt.savefig('Normalized Distribution of Number of Sentences in Question.pdf')
plt.show()
```



Normalized distribution figure show that legal human question can be composed of 2 or more sentences while synthetic questions are usually composed by 1 sentence.

After verifying some legal questions that have more than 2 sentences, the reason under the hood is also quite straightforward to understand: why people ask a legal question, they tend to describe their situation and then ask the question. Hence, one legal question is usually composed of a declarative phrase and an interrogative phrase.

Analysis 4: Pos Tagging Distribution

In [16]:
```
# Function to count POS tags in a dataset
def count_pos_tags(series):
    tag_counts = {}
    for question in series:
        doc = nlp(question)
        for token in doc:
            tag = token.pos_
            tag_counts[tag] = tag_counts.get(tag, 0) + 1
    total_tags = sum(tag_counts.values())
    # Normalize counts to get density
    tag_counts = {tag: count / total_tags for tag, count in tag_counts.items()}
    return tag_counts
```

In [17]:
```
dfQ_human_pos_dist = count_pos_tags(dfQ_human["question"])
dfQ_syn1_pos_dist = count_pos_tags(dfQ_syn1["text"])
dfQ_syn2_pos_dist = count_pos_tags(dfQ_syn2["question"])
```

In [18]:
```
dfQ_mmarco_pos_dist = count_pos_tags(dfQ_mmarco["question"].sample(10000))
```

In [21]:
```
tag_counts_datasets = [dfQ_human_pos_dist, dfQ_syn1_pos_dist, dfQ_syn2_pos_dist, df
# Convert to DataFrame for easier plotting
df_tags = pd.DataFrame(tag_counts_datasets).T
df_tags.columns = ['dfQ_human', 'dfQ_syn1', 'dfQ_syn2', 'dfQ_mmarco']
```

```
df_tags = df_tags.fillna(0)  # Fill missing values with 0
df_tags
```

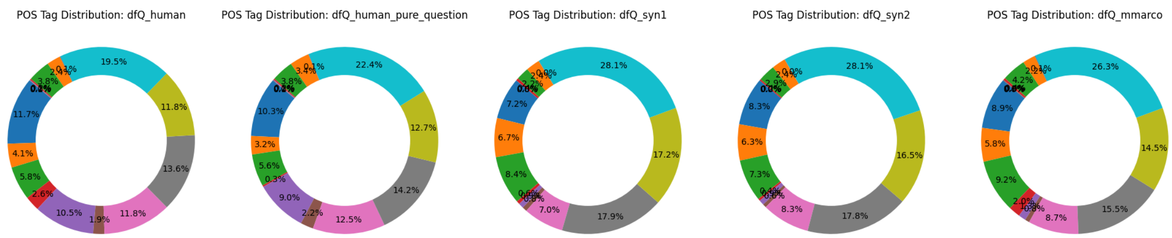
Out[21]:

	dfQ_human	dfQ_syn1	dfQ_syn2	dfQ_mmarco
PRON	0.116705	0.072453	8.325543e-02	0.089491
AUX	0.041390	0.067481	6.253351e-02	0.058349
ADJ	0.058172	0.084133	7.295272e-02	0.091868
PROPN	0.025637	0.005653	3.892074e-03	0.019974
PUNCT	0.105380	0.008878	8.606982e-03	0.012741
CCONJ	0.019408	0.008112	6.217670e-03	0.007710
VERB	0.117786	0.069744	8.279123e-02	0.087303
ADP	0.135753	0.179172	1.779086e-01	0.155097
DET	0.117992	0.172053	1.652429e-01	0.145047
NOUN	0.194698	0.281459	2.805816e-01	0.262700
X	0.001081	0.000401	4.924257e-04	0.001220
SCONJ	0.024041	0.024020	2.387582e-02	0.022099
ADV	0.038353	0.022109	2.921058e-02	0.041532
NUM	0.002265	0.004245	2.405695e-03	0.003585
SYM	0.000824	0.000009	1.274299e-05	0.000063
SPACE	0.000515	0.000071	1.911449e-05	0.001220
INTJ	0.000000	0.000004	9.102138e-07	0.000000

In [49]:

```
# Function to plot pie chart for each dataset
def plot_pie_chart(data, title, ax):
    data.plot(kind='pie', ax=ax, autopct='%1.1f%%', startangle=140, pctdistance=0.8
    ax.set_ylabel('')  # Remove the y-label as it's not needed for pie charts
    ax.set_title(title)
    # Draw a circle at the center to make it a donut chart
    centre_circle = plt.Circle((0, 0), 0.70, fc='white')
    ax.add_artist(centre_circle)


# Create a figure and a set of subplots
fig, axes = plt.subplots(1, 5, figsize=(20, 7))

# Plotting pie charts for each dataset
for i, column in enumerate(df_tags.columns):
    plot_pie_chart(df_tags[column], f'POS Tag Distribution: {column}', axes[i])

plt.tight_layout()
```

```
plt.savefig('POS Tag Distribution.pdf')
plt.show()
```



We find that legal human questions are significantly different in terms of grammatical structure (Left 2 pie charts vs Right 3 pie charts). Interestingly, MS Marco human queries show a similar grammatical structure to synthetic legal questions. This might be due to the fact that the synthetic questions are generated by Doc2Query model that is fined tuned by MS MARCO query dataset.

Note: As legal human question are usually composed of a declarative phrase and an interrogative phrase, we have extracted the interrogative part (dfQ_human_pure_question) so that we can also be able to compare the pure interrogative phrases. It is shown that the pure interrogative part also demonstrate significant differences comparing to synthetic legal questions and msmarco queries.

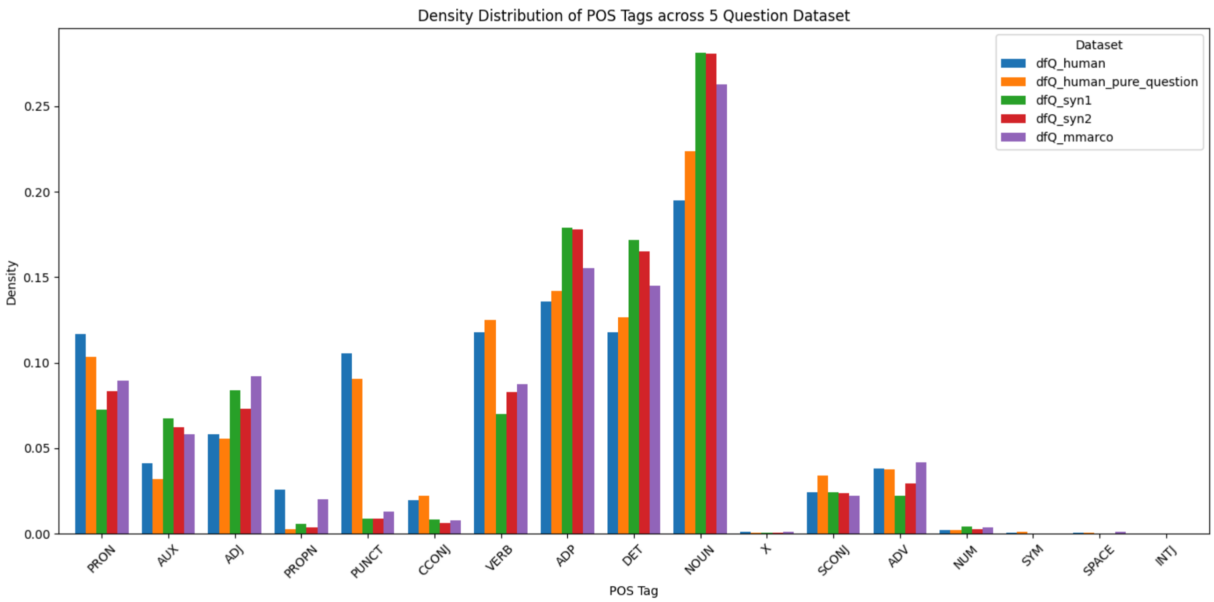
In [25]:

```
# Extract Interrogative phrase from human questions
dfQ_human_sentences = processor.preprocess(dfQ_human["question"], sentences=True)
dfQ_human_questions = dfQ_human_sentences.apply(
    lambda text: [sent.text.lower() for sent in text if "?" in sent.text.lower()][0
)
dfQ_human_pure_question_pos_dist = count_pos_tags(dfQ_human_questions)
```

```
Processing text: 100%|██████████| 1108/1108 [00:03<00:00, 300.77it/s]
```

In [27]:

```
tag_counts_datasets = [dfQ_human_pos_dist, dfQ_human_pure_question_pos_dist, dfQ_sy
# Convert to DataFrame for easier plotting
df_tags = pd.DataFrame(tag_counts_datasets).T
df_tags.columns = ['dfQ_human', 'dfQ_human_pure_question', 'dfQ_syn1', 'dfQ_syn2',
df_tags = df_tags.fillna(0)  # Fill missing values with 0

# Plot the density distribution of POS tags
df_tags.plot(kind='bar', figsize=(14, 7), width=0.8)
plt.title('Density Distribution of POS Tags across 5 Question Dataset')
plt.xlabel('POS Tag')
plt.ylabel('Density')
plt.xticks(rotation=45)
plt.legend(title='Dataset')
plt.tight_layout()
plt.savefig('Density Distribution of POS Tags Across Datasets.pdf')
plt.show()
```



Specifically, legal human questions have less Nouns, ADP(adposition), DET(determiner), ADJ(adjective) and AUX (auxiliary). However, they have more verbs and prons (proper noun).

Note, legal human questions have more punctuations only because synthetic questions do not have question marks in the end. Hence, there is no significant difference in terms of punctuations in reality.

Analysis 5: Question Categories

- Yes/No
- What
- Which
- Where
- Why
- How

In [29]:

```
# Extract Interrogative phrase from human questions
dfQ_human_sentences = processor.preprocess(dfQ_human["question"], sentences=True)
dfQ_human_questions = dfQ_human_sentences.apply(
    lambda text: [sent.text.lower() for sent in text if "?" in sent.text.lower()][0
)
dfQ_human_questions
```

```
Processing text: 100%|██████████| 1108/1108 [00:03<00:00, 299.69it/s]
```

```
Out[29]: 0      puis-je refuser de faire des heures supplément...
         1                      peut-on saisir tous mes revenus ?
         2                       dois-je reconnaître mon enfant ?
         3      pour quels logements le permis de location est...
         4      suis-je payé pendant la procédure du trajet de...
                                      ...
         217    dois-je payer les dettes de mon cohabitant lég...
         218                         a qui dois-je payer ma dette ?
         219                            qui doit payer les loyers ?
         220    est-ce que je peux signer plusieurs baux de co...
         221    en tant que victime de violences conjugales, p...
         Name: question, Length: 1108, dtype: object
```

```
In [32]: def categorize_question(question):
             def is_yes_no_question(question):
                 # Normalize the sentence to lowercase to simplify matching
                 question = question.lower()

                 # Define the beginnings of potential yes-no questions in French
                 question_starters = [
                     'suis-je', 'es-tu', 'est-il', 'sommes-nous', 'êtes-vous', 'sont-ils', '
                     'ai-je', 'as-tu', 'a-t-il', 'avons-nous', 'avez-vous', 'ont-ils', 'ont-
                     'peux-tu', 'peut-il', 'pouvons-nous', 'pouvez-vous', 'peuvent-ils', 'pu
                     # Pouvoir
                     'dois-je', 'doit-il', 'devons-nous', 'devez-vous', 'doivent-ils', 'doit
                     'vais-je', 'vas-tu', 'va-t-il', 'allons-nous', 'allez-vous', 'vont-ils'
                     'est-ce', 'faut-il', 't-il', 'est un', 'est une', 'est le', 'est la',
                 ]

                 return any(starter in question for starter in question_starters)

             def is_what_which_question(question):
                 # Normalize the sentence to lowercase to simplify matching
                 question = question.lower()

                 # Define the beginnings of potential yes-no questions in French
                 question_starters = [
                     'que f', 'que d', 'que s', 'que c', 'que v', 'qu\'est', 'quels ', 'quel
                     'combien ',
                     'à quoi', 'c\'est quoi', 'de quoi'
                 ]

                 # Check if the sentence starts with any of the question starters
                 return any(starter in question for starter in question_starters)

             def is_when_question(question):
                 # Normalize the sentence to lowercase to simplify matching
                 question = question.lower()

                 # Define the beginnings of potential yes-no questions in French
                 question_starters = [
                     'quand', 'quel moment'
                 ]

                 # Check if the sentence starts with any of the question starters
                 return any(starter in question for starter in question_starters)
```

```
def is_where_question(question):
    # Normalize the sentence to lowercase to simplify matching
    question = question.lower()

    # Define the beginnings of potential yes-no questions in French
    question_starters = [
        'où'
    ]

    # Check if the sentence starts with any of the question starters
    return any(starter in question for starter in question_starters)

def is_how_question(question):
    # Normalize the sentence to lowercase to simplify matching
    question = question.lower()

    # Define the beginnings of potential yes-no questions in French
    question_starters = [
        'comment'
    ]

    # Check if the sentence starts with any of the question starters
    return any(starter in question for starter in question_starters)

def is_who_question(question):
    # Normalize the sentence to lowercase to simplify matching
    question = question.lower()

    # Define the beginnings of potential yes-no questions in French
    question_starters = [
        'qui '
    ]

    # Check if the sentence starts with any of the question starters
    return any(starter in question for starter in question_starters)

def is_why_question(question):
    # Normalize the sentence to lowercase to simplify matching
    question = question.lower()

    # Define the beginnings of potential yes-no questions in French
    question_starters = [
        'pourquoi'
    ]

    # Check if the sentence starts with any of the question starters
    return any(starter in question for starter in question_starters)

# Lowercase the question for standardization
question = question.lower()

if is_why_question(question):
    return "why"

if is_how_question(question):
```

```python
        return "how"

    if is_where_question(question):
        return "where"

    if is_when_question(question):
        return "when"

    if is_who_question(question):
        return "who"

    if is_what_which_question(question):
        # with open("what_questions.txt", "a") as f:
        #     f.write(question + "\n")
        return "what/which"

    if is_yes_no_question(question):
        # with open("yes_no_questions.txt", "a") as f:
        #     f.write(question + "\n")
        return "yes/no"

    with open("unknown_questions.txt", "a", encoding="utf-8") as f:
        f.write(question + "\n")

    return 'unknown'
```

In [33]:
```python
with open("unknown_questions.txt", "a") as f:
    f.truncate(0)
with open("yes_no_questions.txt", "a") as f:
    f.truncate(0)
with open("what_questions.txt", "a") as f:
    f.truncate(0)
dfQ_human_questions_cat = dfQ_human_questions.p_apply(categorize_question).value_co
```

```
CATEGORIZE_QUESTION DONE:   0%|          | 0/1108 [00:00<?, ?it/s]
```

In [34]:
```python
with open("unknown_questions.txt", "a") as f:
    f.truncate(0)
with open("yes_no_questions.txt", "a") as f:
    f.truncate(0)
with open("what_questions.txt", "a") as f:
    f.truncate(0)
dfQ_syn1_questions_cat = dfQ_syn1["text"].p_apply(categorize_question).value_counts
```

```
CATEGORIZE_QUESTION DONE:   0%|          | 0/22633 [00:00<?, ?it/s]
```

In [35]:
```python
dfQ_mmarco.columns
```

Out[35]:
```
Index(['id', 'question'], dtype='object')
```

In [36]:
```python
with open("unknown_questions.txt", "a") as f:
    f.truncate(0)
with open("yes_no_questions.txt", "a") as f:
    f.truncate(0)
with open("what_questions.txt", "a") as f:
```

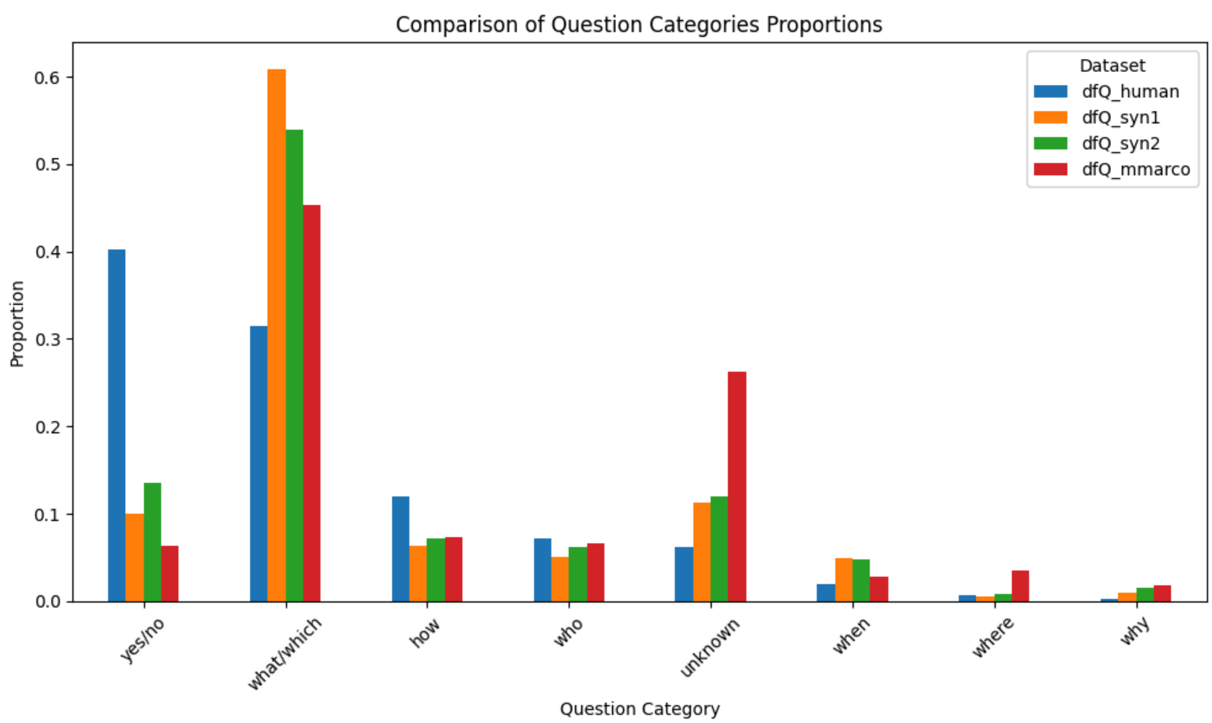
```
    f.truncate(0)
dfQ_syn2_questions_cat = dfQ_syn2["question"].p_apply(categorize_question).value_co
```

```
CATEGORIZE_QUESTION DONE:   0%|          | 0/113165 [00:00<?, ?it/s]
```

In [37]:
```
with open("unknown_questions.txt", "a") as f:
    f.truncate(0)
with open("yes_no_questions.txt", "a") as f:
    f.truncate(0)
with open("what_questions.txt", "a") as f:
    f.truncate(0)
dfQ_mmarco_questions_cat = dfQ_mmarco["question"].p_apply(categorize_question).valu
```

```
CATEGORIZE_QUESTION DONE:   0%|          | 0/808731 [00:00<?, ?it/s]
```

In [38]:
```
# Combine Series into a DataFrame
df = pd.concat([dfQ_human_questions_cat, dfQ_syn1_questions_cat, dfQ_syn2_questions
               axis=1)
df.columns = ['dfQ_human', 'dfQ_syn1', 'dfQ_syn2', 'dfQ_mmarco']

# Plot
df.plot(kind='bar', figsize=(10, 6))
plt.title('Comparison of Question Categories Proportions')
plt.xlabel('Question Category')
plt.ylabel('Proportion')
plt.xticks(rotation=45)
plt.legend(title='Dataset')
plt.tight_layout()
plt.savefig('Comparison of Question Categories Proportions.pdf')
plt.show()
```



After applying rule based classification, we have found that nearly 40% human legal questions are yes/no questions while most synthetic questions are what/which questions

(>50%) and only 10% yes/no question. Similar observation goes to MS MARCO human queries.

Analysis 6: Sentence Complexity

In [50]:
```
# Function to count POS tags in a dataset
def count_dependencies(series):
    dep_counts = {}
    for question in series:
        doc = nlp(question)
        for token in doc:
            dep = token.dep_
            dep_counts[dep] = dep_counts.get(dep, 0) + 1
    total_deps = sum(dep_counts.values())
    # Normalize counts to get density
    dep_counts = {dep: count / total_deps for dep, count in dep_counts.items()}
    return dep_counts
```

In [51]:
```
dfQ_human_dep_dist = count_dependencies(dfQ_human["question"])
dfQ_syn1_dep_dist = count_dependencies(dfQ_syn1["text"])
dfQ_syn2_dep_dist = count_dependencies(dfQ_syn2["question"].sample(10000))
dfQ_mmarco_dep_dist = count_dependencies(dfQ_mmarco["question"].sample(10000))
```

In [52]:
```
dep_counts_datasets = [dfQ_human_dep_dist, dfQ_syn1_dep_dist, dfQ_syn2_dep_dist, df
# Convert to DataFrame for easier plotting
df_deps = pd.DataFrame(dep_counts_datasets).T
df_deps.columns = ['dfQ_human', 'dfQ_syn1', 'dfQ_syn2', 'dfQ_mmarco']
df_deps = df_deps.fillna(0)  # Fill missing values with 0
df_deps
```

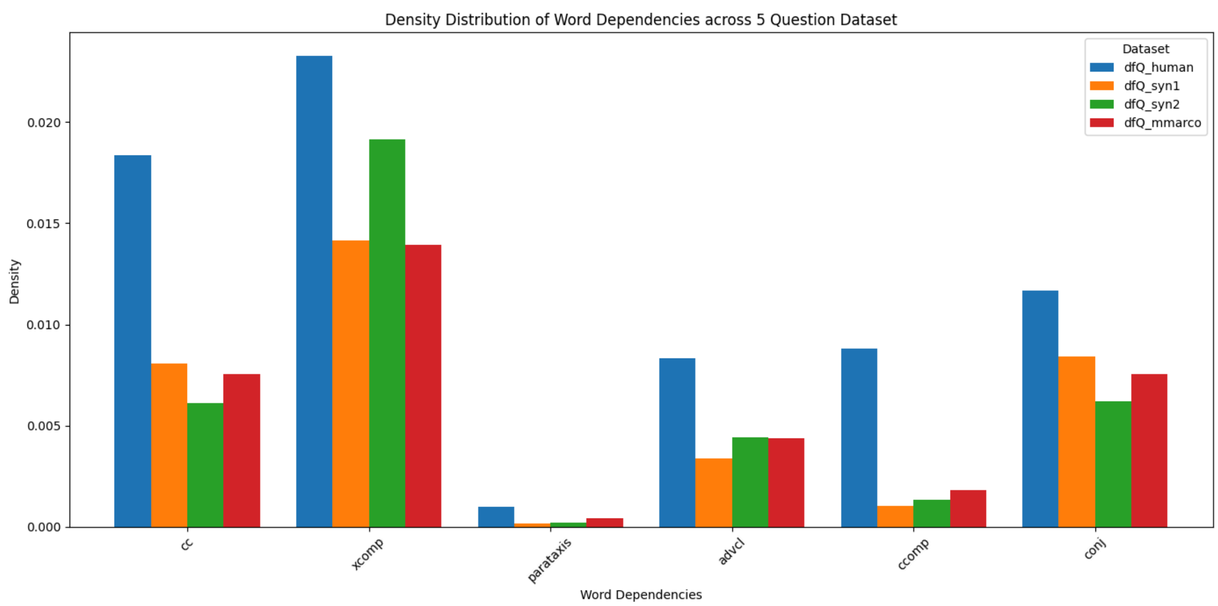
Out[52]:

	dfQ_human	dfQ_syn1	dfQ_syn2	dfQ_mmarco
nsubj	0.080875	0.055868	0.056870	0.051247
cop	0.014260	0.016844	0.016355	0.016468
ROOT	0.092304	0.105752	0.108354	0.139652
obl:mod	0.023063	0.017668	0.019743	0.020504
punct	0.106255	0.012892	0.012163	0.016617
cc	0.018378	0.008050	0.006097	0.007549
mark	0.030579	0.026208	0.028064	0.027816
xcomp	0.023269	0.014153	0.019156	0.013939
det	0.117014	0.171474	0.164514	0.144573
obj	0.054208	0.027535	0.034326	0.037183
amod	0.039485	0.056242	0.050238	0.065485
conj	0.011686	0.008420	0.006190	0.007561
case	0.124942	0.167122	0.166430	0.143414
nmod	0.071454	0.138063	0.127418	0.104910
aux:pass	0.012458	0.012901	0.014140	0.008520
dep	0.035727	0.083875	0.077952	0.075936
obl:arg	0.027027	0.011164	0.014006	0.014836
appos	0.002471	0.000396	0.000433	0.000797
expl:subj	0.001390	0.002744	0.002781	0.003600
advmod	0.036345	0.022813	0.028662	0.039837
nsubj:pass	0.007053	0.009658	0.010113	0.006540
parataxis	0.000978	0.000147	0.000216	0.000411
advcl	0.008340	0.003368	0.004408	0.004385
nummod	0.002214	0.002602	0.001730	0.002491
aux:tense	0.006486	0.003484	0.003543	0.008109
expl:comp	0.011634	0.002419	0.003110	0.005257
ccomp	0.008803	0.001016	0.001329	0.001819
acl	0.009678	0.007533	0.010144	0.010314
acl:relcl	0.003501	0.003564	0.004480	0.004883
flat:name	0.002059	0.000494	0.000587	0.003986

	dfQ_human	dfQ_syn1	dfQ_syn2	dfQ_mmarco
fixed	0.010811	0.002753	0.002863	0.005132
iobj	0.003140	0.000980	0.001514	0.004746
expl:pass	0.000515	0.000423	0.000649	0.000448
flat:foreign	0.000206	0.000018	0.000010	0.000137
obl:agent	0.001390	0.001354	0.001411	0.000897
vocative	0.000000	0.000004	0.000000	0.000000

In [66]:
```
df_deps_subset = df_deps.T[["cc","xcomp", "parataxis", "advcl", "ccomp", "conj"]].T
```

In [67]:
```
# Plot the density distribution of POS tags
df_deps_subset.plot(kind='bar', figsize=(14, 7), width=0.8)
plt.title('Density Distribution of Word Dependencies across 5 Question Dataset')
plt.xlabel('Word Dependencies')
plt.ylabel('Density')
plt.xticks(rotation=45)
plt.legend(title='Dataset')
plt.tight_layout()
plt.savefig('Density Distribution of Word Dependencies Across Datasets.pdf')
plt.show()
```



When evaluating sentence complexities, it is common to check the number of subordinate clauses and the use of compound and complex sentences.

- subordinate clauses: xcomp(Open Clausal Complement), ccomp(Clausal Complement), advcl (Adverbial Clause Modifier)
- compound and complex sentences: cc(Coordinating Conjunction), conj(Conjunct), parataxis (Paratactic Relation)

Based on these 6 indicators, legal human questions might be considered the most complex due to the higher use of subordinate clauses and compound sentences.

Analysis 7: Sentence Vocabulary

In [329…

```
from collections import Counter

def tokenize_lemma(sentence):
    doc = nlp(sentence)
    return [token.lemma_.lower() for token in doc if token.is_alpha]

dfQ_human_unique_words = len(set(dfQ_human["question"].progress_apply(tokenize_lemm
dfQ_syn1_unique_words = len(set(dfQ_syn1["text"].sample(5000).progress_apply(tokeni
dfQ_mmarco_unique_words = len(set(dfQ_mmarco["question"].sample(5000).progress_appl
dfQ_human_unique_words,dfQ_syn1_unique_words,dfQ_mmarco_unique_words
```

```
Processing text: 100%|██████████| 1108/1108 [00:03<00:00, 284.78it/s]
Processing text: 100%|██████████| 5000/5000 [00:13<00:00, 359.22it/s]
Processing text: 100%|██████████| 5000/5000 [00:15<00:00, 313.94it/s]
```

Out[329…

```
(1047, 2803, 7069)
```

In [330…

```
dfQ_human_counter = Counter(dfQ_human["question"].progress_apply(tokenize_lemma).ex
dfQ_syn1_counter = Counter(dfQ_syn1["text"].sample(5000).progress_apply(tokenize_le
dfQ_mmarco_counter = Counter(dfQ_mmarco["question"].sample(5000).progress_apply(tok

data = {
    'Word': [],
    'Count': [],
    'Dataset': []
}

# Convert Counter objects to DataFrame
data = []
for label, counter in zip(["dfQ_human", "dfQ_syn1", "dfQ_mmarco"], [dfQ_human_count
    for word, count in counter.items():
        data.append({"Word": word, "Count": count, "Dataset": label})
df_word_count = pd.DataFrame(data)
```

```
Processing text: 100%|██████████| 1108/1108 [00:03<00:00, 306.07it/s]
Processing text: 100%|██████████| 5000/5000 [00:13<00:00, 361.59it/s]
Processing text: 100%|██████████| 5000/5000 [00:15<00:00, 320.77it/s]
```

In [331…

```
df_word_count_group_sum = df_word_count.groupby("Dataset").sum("Count").rename(colu
```

In [353…

```
legal_words = ["travailler", "revenu", "enfant", "bail", "dette","divorcer", "bail"
df_legal_word_count = df_word_count[df_word_count['Word'].isin(legal_words)]
```

In [354…

```
df_legal_word_count_merged = pd.merge(df_legal_word_count, df_word_count_group_sum,
df_legal_word_count_merged['Proportion'] = df_legal_word_count_merged["Count"]/df_l
df_legal_word_count_sorted_merged = df_legal_word_count_merged.sort_values(by='Word
df_legal_word_count_sorted_merged
```

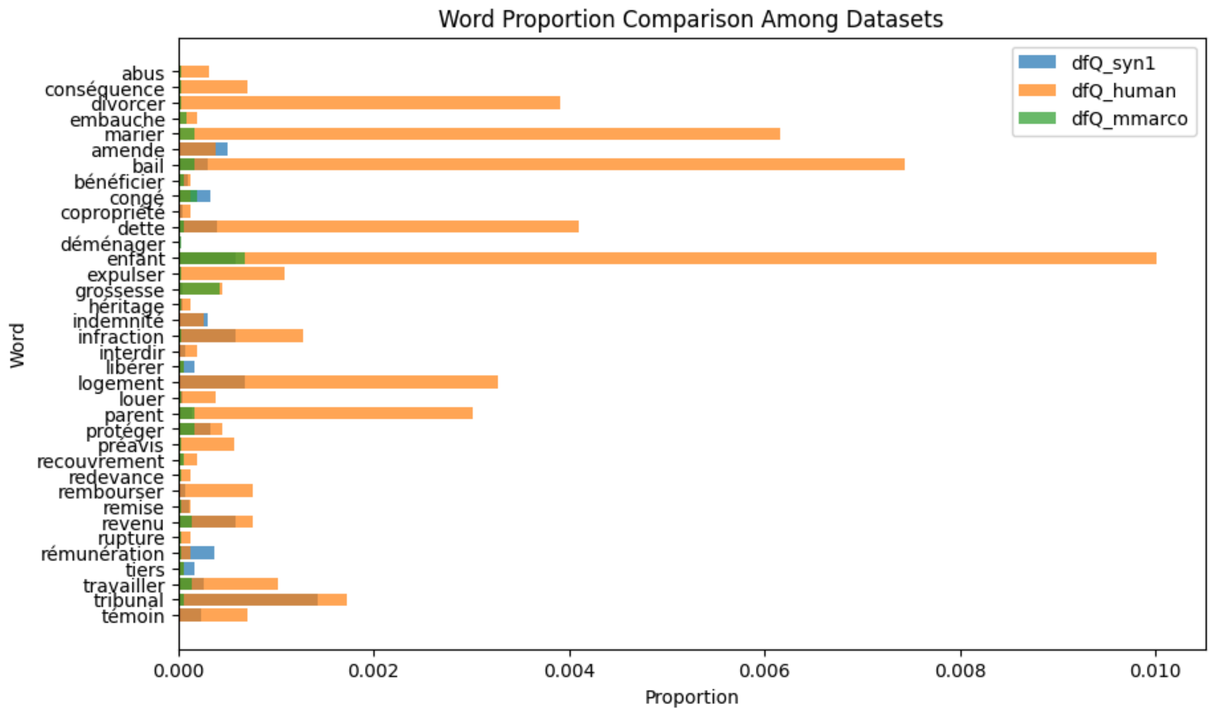
Out[354...

	Word	Count	Dataset	Sum	Proportion
47	témoin	10	dfQ_syn1	42894	0.000233
28	témoin	11	dfQ_human	15589	0.000706
12	tribunal	27	dfQ_human	15589	0.001732
77	tribunal	2	dfQ_mmarco	35304	0.000057
46	tribunal	61	dfQ_syn1	42894	0.001422
...	...	...	...	...	...
4	bail	116	dfQ_human	15589	0.007441
14	amende	6	dfQ_human	15589	0.000385
37	amende	22	dfQ_syn1	42894	0.000513
13	abus	5	dfQ_human	15589	0.000321
94	abus	1	dfQ_mmarco	35304	0.000028

95 rows × 5 columns

In [355...

```
# Plotting
fig, ax = plt.subplots(figsize=(10, 6))

for dataset in df_legal_word_count_sorted_merged['Dataset'].unique():
    subset = df_legal_word_count_sorted_merged[df_legal_word_count_sorted_merged['D
    ax.barh(subset['Word'], subset['Proportion'], label=dataset, alpha=0.7)

ax.set_xlabel('Proportion')
ax.set_ylabel('Word')
ax.set_title('Word Proportion Comparison Among Datasets')
plt.legend()
plt.show()
```



From the table and chart, we find that legal human questions have less unique words (more concentrated in legal domain) than synthetic questions and msmarco questions. Meanwhile, legal human questions have relatively higher proportion in legal related words.

```
In [ ]:
```