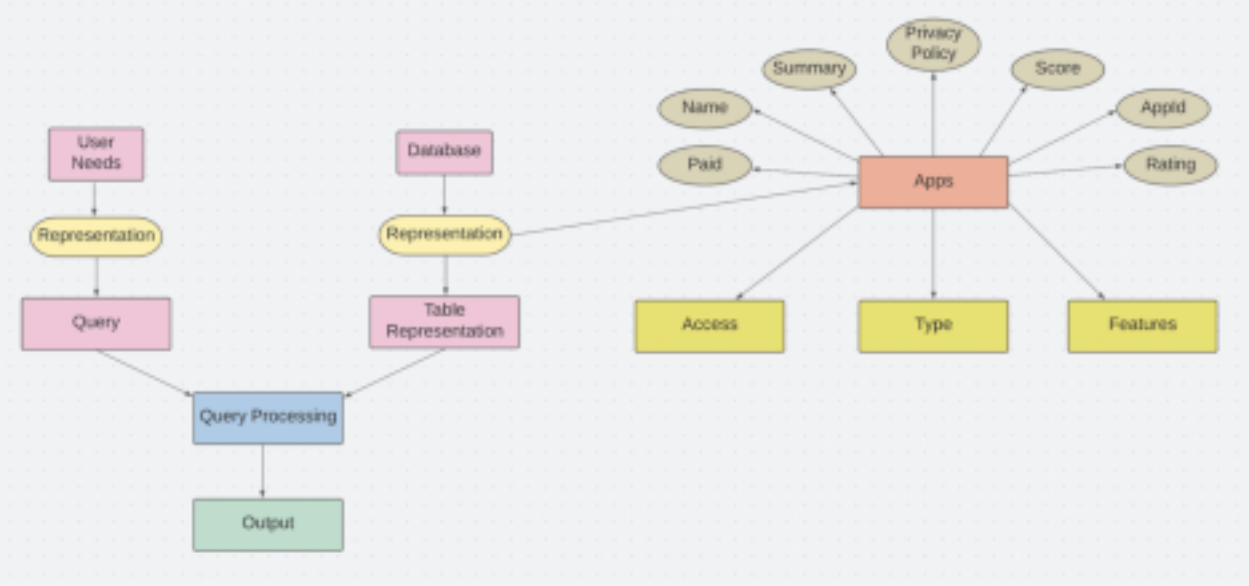
Group 31 - IR PROJECT README

How to Run:
Go to ‘website copy’ and run ‘python app.py’.

Database:-
We created a database in **MySQL**, storing the details of various applications such as their privacy policy, score, features, permissions required, etc. We have mentioned the tables below with the work-flow diagram:



Tables

1. Apps

App ID	TypeID		Privacy Policy Summary Score	Ra	

This is the main table that references TypeID from the **Type** Table.

2. Features

Feature ID	Feature

This is the table storing various features provided by applications.

3. Type

ID	Type

This is the table storing different types of applications.

4. Permission

Id	Perm	Explanation

This is the table storing different accesses asked by applications.

5. App_join_permission

AppID	PermissionID

This is the table mapping various apps to the permissions they require. The appID references from the **Apps** table and permissionID references from the **Permission** table.

6. App_join_feature

AppID	Feature ID

This is the table mapping various apps to the permissions they require. The appID references from the **Apps** table and featureID references from the **Feature** table.

Indexing and query processing:-

We have used positional indexing on the privacy policies of the apps. The key to the positional indexing is the words/tokens of the privacy policies. The indexing values are the names of the apps in the database, with those words present in the privacy policies

of apps.

eg.
postDict[‘third party’’] = {

‘Whatspp’ : [0, 5, 8, 14],
‘Twitter’ : [2, 6, 9],
‘Tumbler’ : [3, 4, 8, 23]
}

```
posDict['sharing']
✓ 0.0s

Output exceeds the size limit. Open the full output data in a text editor
{'Whatsapp': [768, 768],
 'WeChat': [1922, 1967, 2189, 2840, 1922, 1967, 2189, 2840],
 'Instagram': [953, 969, 1631, 1829, 1848, 953, 969, 1631, 1829, 1848],
 'Facebook': [977, 993, 1673, 1876, 1895, 977, 993, 1673, 1876, 1895],
 'Telegram': [817, 817],
 'Tumblr': [547,
  818,
  1753,
```

Query processing:-

The positional indexing has been made on the privacy policies of all the apps. After the indexing has been done, we can use **phrase query** to get the name of the apps with the phrase present in the privacy policies of the apps.

```
import query as pt
query = input("Enter the INPUT phrase query: ")
ans = pt.query_processing(posDict, query, AppID)
ans
✓ 5.6s

Query Using POSITIONAL INVERTED INDEX:
No of documents retrived:  5
```

We have defined the positional_query method to process the phrase queries and retrieve all the names of the apps with these phrases in their privacy policies.

```
def positional_query(query, posDict):

    potential = list(posDict[query[0]].keys())

    for i in range(1, len(query)):
        potential = AND(potential, list( posDict[query[i]].keys()) )

    ans = []

    for file in potential:
        temp = posDict[query[0]][file]
        for word in range(1, len(query)):
            temp = helper(temp, posDict[query[word]][file] )
        if( len(temp) != 0 ):
            ans.append(file)


    print("\nQuery Using POSITIONAL INVERTED INDEX: ")
    print("No of documents retrived: ", len(ans))

    return ans
```

Score calculation:-

we have defined a list of keywords, such as
data sharing, third-party, advertisement, camera, location, contact, photos, personal information, document

```
def helper_function():
    query = ['data sharing', 'third party', 'advertisement', 'photo',
             'location', 'gallery', 'document', 'personal information']
    ans = []
    for word in query:
        temp = pt.query_processing(posDict, word, AppID)
        ans.append(temp)

    return score(ans, len(ans[0]))
```

Method:-

In the score function, we take the number of apps 'n' as an input and a list of dictionaries. In this list, the first dictionary has the app names as critical values and the first 'special word' occurrence as its value. These special words are words with high information about the threat level of an app. The intermediate weights for a specific word and app are calculated by dividing that word's occurrence in the app's privacy policy by the total number of occurrences of all the special words. After that, for each

particular word, these intermediate weights are multiplied by their occurrences in all different apps and then summed up. With this, we get weights for each specific word returned in the 'score_list.'

Score Range:-
We have normalised the scores of the different apps between the range 0 - 10.
To compare the scores of different apps.

0 :- safest apps
10:- Dangerous/Bad apps

Recommender System:-

Our app has an inbuilt recommendation system, which gives suggestions for similar but safer apps present on installing any new app. We use the Jaccard similarity between the feature vectors (a vector of all the different feature IDs present in the app) of all the apps in the database with the newly installed app. After that, we use the score calculated by our system, which classifies an app as safe or unsafe, to recommend apps with the most similarity and better scores. The lesser the score, the safer the app, and in this way, we suggest alternatives to the user.
Eg.
If you want to see the apps that provide similar services to the app ‘WhatsApp’ are safer than it. Then our recommendation system will recommend apps such as

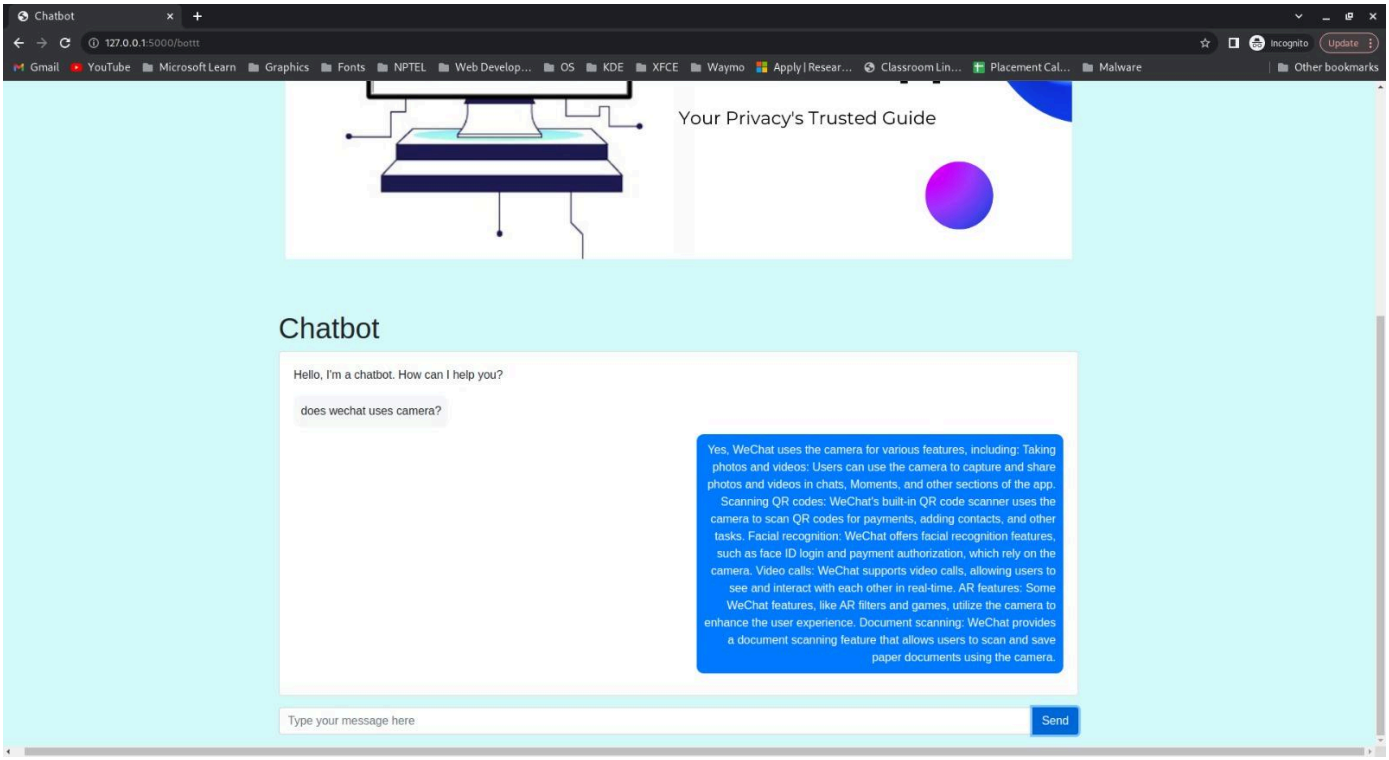
‘Telegram’, ‘signal’, ‘Facebook’, ‘Tumbler’.

If in case a user enters an app that is not in our database, the privacy policy of the said app is fetched using OpenAI’s API. Further, this privacy policy is summarised, and a privacy score is calculated.

Chatbot:-
we have also incorporated an NLP and ML model chatbot. This is a question-answering generative model chatbot.
Our application has a chatbot based on the Gemini API that has been fine-tuned using our privacy policy data. This enables the chatbot to provide

users with accurate and efficient responses to their queries while also reflecting our commitment to protecting their personal data.

We preprocessed our privacy policy data in JSONL format to facilitate prompt and accurate responses. Our chatbot has developed a deep understanding of our privacy policy guidelines, features and permissions by training the Gemini model on this preprocessed data.



In addition, we employed a survey mechanism to gather user feedback and used another evaluation metric. We created a Google Form and distributed it among our peers, friends, and family to gather basic information about their privacy policy and data breach behaviour. Our survey aimed to determine if users would be willing to utilise an application with the functionalities that **App-Police** aims to provide.

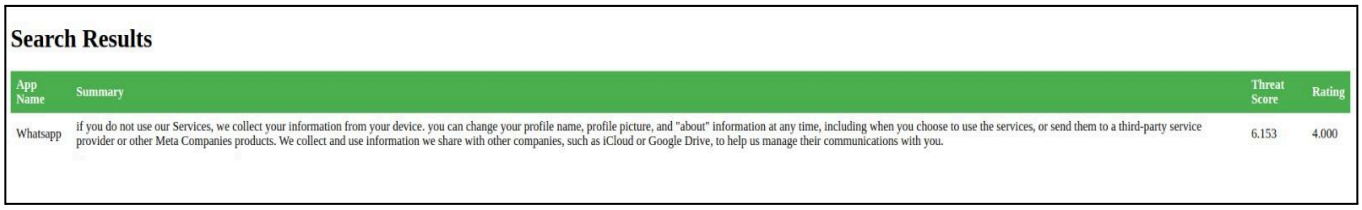
Link to the form - Google form link

Summarizer:-

T5 (Text-to-Text Transfer Transformer) is a transformer-based language model that Google Brain developed, achieving state-of-the-art performance on various NLP benchmarks. T5 is designed to solve various NLP tasks using a single neural network architecture, trained end-to-end.

The T5 small model is a smaller version of the T5 architecture, which contains 60 million parameters compared to the 220 million parameters of the base model. This model is more lightweight, faster to train, and suitable for smaller-scale NLP tasks.

T5 small is designed to perform text summarisation tasks. The task of text summarisation involves condensing a large amount of text into a shorter version while retaining the essential meaning of the original text. The T5 small model is



Search Results

App Name	Summary	Threat Score	Rating
Whatsapp	if you do not use our Services, we collect your information from your device. you can change your profile name, profile picture, and "about" information at any time, including when you choose to use the services, or send them to a third-party service provider or other Meta Companies products. We collect and use information we share with other companies, such as iCloud or Google Drive, to help us manage their communications with you.	6.153	4.000

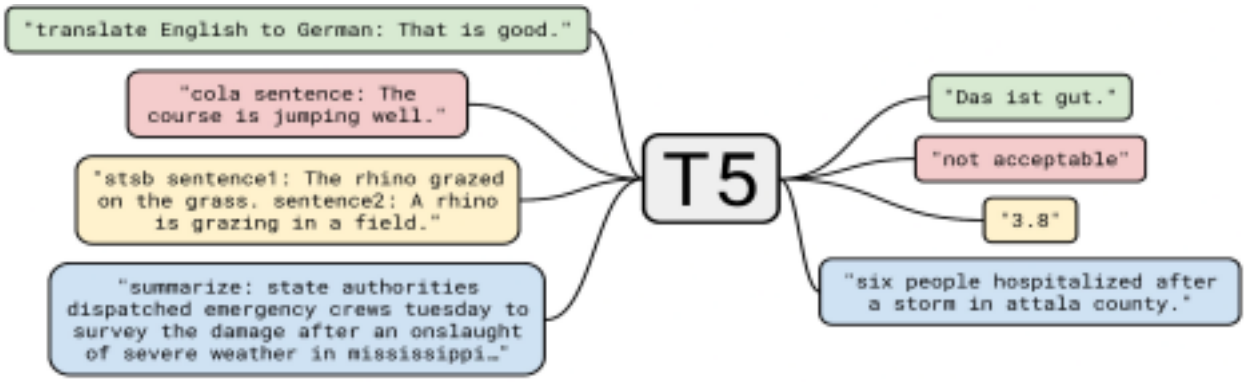
trained to perform this task using unsupervised and supervised learning.

The T5 small model is trained on a large corpus of text data, where the input text and corresponding summary pairs are used to train the model. During training, the model learns to generate a summary from the input text by encoding the input text into a sequence of vectors using the transformer architecture and then decoding these vectors into a summary sequence.

The T5 small model generates summaries using a sequence-to-sequence architecture with an attention mechanism. The input text is tokenised and fed into the encoder, which outputs a sequence of vectors representing the encoded input. The decoder takes this sequence of vectors and generates a sequence of tokens, which are then converted into the summary text.

During inference, the T5 small model takes a large text document as input and generates a summary of the document by decoding the input text into a summary

sequence. The quality of the summary generated by the model is evaluated based on metrics such as ROUGE (Recall-Oriented Understudy for Gisting Evaluation), which measures the overlap between the generated summary and the reference summary.



Summary Evaluation:-

several evaluation metrics can be used to evaluate the summary quality generated by a T5 model for summarisation. Some commonly used metrics are:

ROUGE (Recall-Oriented Understudy for Gisting Evaluation): ROUGE is a set of metrics that measures the overlap between the generated summary and the reference summary in terms of n-gram matches. It computes precision, recall, and F1 scores.

ROUGE SCORE:- 0.64

```
from rouge_score import rouge_scorer
```

```
rouge = load_metric('rouge', 'rouge-l')
rouge
```

```
ref1 = [ s[0] for s in references]
```

```
print(rouge.compute(predictions = summary, references = ref_summary))
```

```
0.64
```

BLEU (Bilingual Evaluation Understudy): BLEU measures the similarity between the generated and reference summaries regarding n-gram matches. It computes precision, recall, and a geometric mean of these two scores.

BLEU SCORE:- 0.68

```
# Load the BLEU metric
metric_bleu = load_metric('bleu')

# Calculate BLEU-1 score
bleu_score_1 = metric_bleu.compute(predictions=summary, references=original, max_order=1)['bleu']
# Calculate BLEU-2 score
bleu_score_2 = metric_bleu.compute(predictions=summary, references=original, max_order=2)['bleu']
```

```
bleu_score_1, bleu_score_2
```

```
0.68
```

Dynamic Database:-

Our database is dynamic, i.e. we can add new data to our database during run time. If an app is not present in our database, the user can enter the name of the app and our chatbot model will fetch the database from online sources and append it to the central database. Now new positional indexing file will be made according to the updated database, and new scores will be calculated and will be shown to the user. This new app will be appended permanently to our database, and the next time a user searches for the same app, the score and summary will be fetched from the database.

As more and more users use our web app, our database will grow and get better and will show better results.

By group 31