Ikonz Studios Internship Report

This document outlines the tasks which are part of my internship assignment, focusing on various technical topics and tools. Each section provides an overview of the learning process and tasks performed to enhance my understanding and skills in these areas.

1. Version control with Git & GitHub

The tasks focused on mastering Git and GitHub version control practices. I worked on setting up repositories, using essential Git commands, and collaborating on code via GitHub. These activities have helped me build a foundational understanding of version control and collaboration within a software development environment. As a beginner, I am continuously learning and improving my skills in this area.

Setup GitHub Repository:

Process:

- I created a new GitHub repository named "Internrepo".
- I initialized a Git repository in my local machine with the following command:
  - git init
- Then, I added the remote origin for my GitHub repository:
  - **git remote add origin <repository-url>**

Commit Changes:

- Staged changes to the local repository:
  - git add .
- Committed the changes with a descriptive message:
  - git commit -m "Initial commit"

Pushed Changes:

- Pushed the local repository to the GitHub remote origin:
    - git push -u origin main

Collaboration:

- Learned how to create branches, switch between branches, and merge them.
- Opened pull requests to collaborate with team members.
- Resolved merge conflicts during code reviews.

2. Fundamentals of Python in Machine learning

This section focuses on the fundamental concepts of Python programming with an emphasis on Machine Learning (ML) applications. It includes mastering Python syntax, data structures, functions, modules, file handling, and object-oriented programming (OOP).

Python Syntax:

- Practiced writing clean and error-free Python code.
- Worked on indentation, comments, and best practices for writing Python programs.

Data Structures:

- Explored and implemented various data structures:
    - **Lists**: Used for storing and manipulating collections of elements.
    - **Dictionaries**: Utilized for key-value pair storage.
    - **Tuples and Sets**: Implemented for immutable and unique value storage.
- Applied data structures to solve real-world problems.

Functions:

- Created reusable functions.
- Practiced using built-in Python functions like map(), filter(), and lambda expressions.

Modules:

- Learned to import and utilize built-in modules such as os, math, and random.
- Created custom modules to reuse specific functionalities across projects.

File Handling:

- Performed operations like reading, writing, and appending data to files: with

  open('data.txt', 'w') as file:

  file.write('Hello, File Handling!')

- Implemented error handling for file operations using try and except blocks.

OOPs Concepts:

- Practiced object-oriented programming by implementing classes and object.
- Explored inheritance, polymorphism, and encapsulation.

3. Core Machine Learning Concepts

This section covers foundational concepts of Machine Learning (ML), including workflows, algorithms, and model evaluation techniques. Tasks focused on building a strong theoretical and practical foundation for ML applications.

Introduction to ML:

- Gained an understanding of the definition and goals of Machine Learning.
- Studied the differences between Supervised Learning (e.g., regression and classification) and Unsupervised Learning (e.g., clustering and dimensionality reduction).
- Explored common ML workflows, including data preprocessing, model training, and evaluation.

Basic Algorithms:

- Implemented foundational ML algorithms:
  - Linear Regression: Used to predict continuous values.
  - Logistic Regression: Applied for binary classification problems.
  - Decision Trees: Learned to visualize decision-making processes.
  - K-Nearest Neighbors (KNN): Used for classification and regression tasks.

Model Evaluation:

- Studied evaluation metrics:
  - Accuracy: Proportion of correct predictions.
  - Precision: Proportion of true positive predictions out of all positive predictions.
  - Recall: Proportion of true positive predictions out of all actual positives.
  - F1 Score: Harmonic mean of precision and recall.
  - ROC-AUC: Evaluated the trade-off between true positive and false positive rates.
- Practiced techniques to avoid overfitting and underfitting:
  - Implemented Cross-Validation to ensure generalization of models.
  - Explored strategies like regularization and early stopping.

4. Mathematics for Machine Learning:

Linear Algebra

Vectors and Matrices

- **Definition**: Vectors are ordered arrays of numbers, and matrices are two-dimensional arrays of numbers.
- **Relevance**: Used to represent and manipulate data in ML models. For instance, datasets are stored as matrices, and transformations like scaling or rotation are expressed mathematically as matrix operations.

Dot Product

- **Definition**: The dot product of two vectors is a scalar value obtained by multiplying corresponding elements and summing the results.
- **Relevance**: Measures similarity between vectors, crucial in algorithms like cosine similarity for recommendation systems.

Eigenvalues and Eigenvectors

- **Definition**: Eigenvalues are scalars indicating how much the eigenvector is stretched during a transformation. Eigenvectors are non-zero vectors that change only in magnitude when a linear transformation is applied.
- **Relevance**: Fundamental in dimensionality reduction techniques like Principal Component Analysis (PCA).

Probability and Statistics:

Basic Probability Concepts

- **Definition**: Probability measures the likelihood of an event occurring.
- **Relevance**: Forms the basis for probabilistic models like Naïve Bayes and Bayesian Networks.

Distributions

- **Definition**: Distributions describe the probabilities of outcomes in a dataset.
- **Relevance**: Understanding distributions helps in designing and interpreting ML models, such as Gaussian Mixture Models.

Mean, Variance, and Standard Deviation

- **Definitions**:
  - Mean: Average value of a dataset.
  - Variance: Measure of data spread.

    - Standard Deviation: Square root of variance, indicating data dispersion.
- **Relevance**: Used to analyze and preprocess data, ensuring features are normalized for better model performance.

Hypothesis Testing and p-values

- **Definition**: Hypothesis testing evaluates assumptions about a dataset, and p-values indicate the significance of results.
- **Relevance**: Ensures that observed patterns are statistically significant and not due to random chance.

Calculus:

Derivatives and Gradients

- **Definition**: Derivatives measure the rate of change of a function. Gradients are vector representations of derivatives for multivariable functions.
- **Relevance**: Crucial for optimization techniques like gradient descent, which is used to minimize loss functions in ML models.

Gradient Descent

- **Definition**: An iterative optimization algorithm to minimize a function by moving in the direction of the steepest descent.
- **Relevance**: Core mechanism for training machine learning models, especially neural networks.

5. Developing RAG Architecture with DeepSeek Model:

The goal is to build a Retrieval-Augmented Generation (RAG) architecture for a Q&A system that retrieves data from external documents and generates accurate answers using the DeepSeek model. The model is integrated for local execution with the support of the Ollama module.

The main objective of the project was to enhance a traditional Q&A system by integrating a retrieval mechanism. When a user asks a question, the system retrieves relevant content from uploaded PDFs and generates an answer based on that context, utilizing the Deepseek model.

Tools and Technologies Used:

Ollama Module

- **Purpose**: The Ollama module was used to interact with the DeepSeek model locally by running the following commands.

    ollama run deepseek-r1:1.5b

LangChain

- **Purpose**: LangChain provides a framework to integrate various language models, document loaders, and retrievers into one pipeline, simplifying the development of complex AI workflows.

FAISS

- **Purpose**: **FAISS** (Facebook AI Similarity Search) is a library used for efficient similarity search and clustering of vectors. It helps store the embeddings(conversion of text from numerical values) of document chunks and retrieve relevant information.

Streamlit

- **Purpose**: Streamlit is a Python framework used to create interactive applications. In this project, it was used to create a simple user interface for uploading PDFs and asking questions.

HuggingFaceEmbeddings

- **Purpose**: Used to convert document text into embeddings, enabling semantic search of relevant chunks from the document.

6. Comparison of Local DeepSeek-r1 vs. RAG System for Query Response:

The performance of the model was compared by analyzing its responses to a variety of queries, with a particular focus on handling real-time or current events.The comparison also explores how DeepSeek (local) handles queries that require information beyond its pre-trained dataset, and how a RAG system leverages external data sources (e.g., web APIs, documents) to generate answers.

Query: who won Nobel prize for chemistry?

Response from Local DeepSeek (Running with Ollama):

```
In summary, no official winners for the Chemistry Nobel Prize were reported for 2024, and if awarded, they would
likely be announced in recent years or through significant scientific news.

>>> Send a message (/? for help)
```

The model provided an incorrect or outdated answer because it is limited to the knowledge that was available when the model was trained. Since the training data doesn't include the most recent events (like the 2024 Nobel Prize winners), the system cannot provide accurate answers about current or recent events.

Response from RAG System (With external Data access):

> The Nobel Prize in Chemistry 2024 was awarded to David Baker Demis Hassabish with his contributions to protein research. He and his collaborator John Jumper were recognized for their innovative computerized methods in creating proteins, which revolutionized the field.
>
> Answer: David Baker Demis Hassabish

The RAG system was able to provide the correct and up-to-date answer because it can retrieve real-time information from external databases, websites, or APIs. In this case, the RAG system pulled the latest information about the 2024 Nobel Prize winners.

The Complete Code for Building RAG Architecture:

Rag_deepseek.py > ...

RetrievalQA

```python
import streamlit as st
from langchain_community.document_loaders import PDFPlumberLoader
from langchain_experimental.text_splitter import SemanticChunker
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_community.llms import Ollama
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain.chains import RetrievalQA
from langchain.chains import StuffDocumentsChain


# Streamlit file uploader
uploaded_file = st.file_uploader("Upload a PDF file", type="pdf")

if uploaded_file:
    # Save PDF temporarily
    with open("temp.pdf", "wb") as f:
        f.write(uploaded_file.getvalue())

    # Load PDF text
    loader = PDFPlumberLoader("temp.pdf")
    docs = loader.load()


    # Split text into semantic chunks


    text_splitter = SemanticChunker(HuggingFaceEmbeddings())
    documents = text_splitter.split_documents(docs)

    # Generate embeddings
    embeddings = HuggingFaceEmbeddings()
    vector_store = FAISS.from_documents(documents, embeddings)
```

```
    prompt = """
    #1. Use ONLY the context below.
    #2. If unsure, say "I don't know".
    #3. Keep answers under 4 sentences.

    #Context: {context}

    #Question: {question}

   # Answer:

"""
    QA_CHAIN_PROMPT = PromptTemplate.from_template(prompt)

    # Chain 1: Generate answers
    llm_chain = LLMChain(llm=llm, prompt=QA_CHAIN_PROMPT)

# Chain 2: Combine document chunks
    document_prompt = PromptTemplate(
    template="Context:\ncontent:{page_content}\nsource:{source}",
    input_variables=["page_content", "source"]
    )
    # doc_name= "content"
    document_variable_name = "context"

# Final RAG pipeline
    qa = RetrievalQA(combine_documents_chain=StuffDocumentsChain(llm_chain=llm_chain,
    document_prompt=document_prompt,document_variable_name =document_variable_name ),
    retriever=retriever)

    # Streamlit UI
    user_input = st.text_input("Ask your PDF a question:")

    if user_input:
        with st.spinner("Thinking..."):
            response = qa(user_input)["result"]
            st.write(response)
```

Web Interface Using Streamlit UI

To provide a user-friendly interface for interacting with the model, **Streamlit** is used as the web-based frontend. This allows users to input queries, receive responses, and visualize the output in a structured manner.

Running the Streamlit Web Interface using the following command

- Streamlit run app.py

This command initializes the **Streamlit web UI**, and the application will be accessible in a web browser at a local address.

Upload a PDF file

Drag and drop file here

Limit 200MB per file • PDF

Browse files

popular-chemistryprize2024.pdf 2.3MB

Ask your PDF a question:

who won nobel prize for chemistry?

<think> Okay, so I need to figure out who won the Nobel Prize for Chemistry in 2024 based on the provided context. Let me go through each piece of information step by step.
First, there's a link to "FURTHER READING" which doesn't seem directly relevant unless I'm missing something. Then, looking at "Additional information on this year's prizes", I see that the Royal Swedish Academy has decided to award the Nobel Prize in Chemistry 2024 with one half to and the other half jointly to DAVID BAKER DEMIS HASSABIS, JOHN JUMPER, and JOHN KENDWELL.

Wait a minute, let me make sure I understand this correctly. The sentence says "DAVID BAKER DEMIS HASSABIS John JUMPER". So it's three names: David Baker Demis Hassabish, John Jumper, and John Kendrew. But the context includes more details about their work.

Looking further down, there's a part that talks about proteins in the dawn of modern biochemistry. It mentions that in the 1950s, Cambridge researchers John Kendrew and Max Perutz used X-ray crystallography to create the first three-dimensional models of proteins. They were awarded the Nobel Prize in Chemistry in 1962 with one half going to John Kendrew (probably as co-inventor) and the other half jointly to David Baker Demis Hassabish.

So, putting it all together, the recipient is David Baker Demis Hassabish. He was awarded both halves of the Nobel Prize: one part for his computerized methods in proteins and the other part jointly with John Jumper.

I should make sure I didn't mix up any names. The key here is that the prize was awarded to David B Baker, not John Kendrew or Perutz. Even though their work was earlier, this year's award goes to them because they have new discoveries. </think>

The Nobel Prize in Chemistry 2024 was awarded to David Baker Demis Hassabish with his contributions to protein research. He and his collaborator John Jumper were recognized for their innovative computerized methods in creating proteins, which revolutionized the field.

Answer: David Baker Demis Hassabish

7. Data Handling & preprocessing:

Handling Missing Values:

1. Techniques to handle missing data includes remove missing values and imputation techniques.

2. Remove missing values only if too many values are missing.

   Imputation techniques:

   - Mean, median and mode imputation are used when data is normally distributed.
   - KNN imputation is used when missing values depends on the other features.
   - Predictive imputation using ML models is used when missing values are not random and can be predicted from other variables.

   1) Interpolation for time series and continuous data
   2) Forward and Backward Fill for sequential data
   3) multiple imputation when there is a pattern in missing values

Scaling:

There are more scaling technique used for feature scaling. The list of most commonly used feature scaling techniques:

1) Min Max scaling

   It is a normalization method that scales data between fixed range (0 to 1)
2) Z-score scaling

   It is a standardization method that centers data zero mean and unit variance.
3) Robust Scaling

   It handles outliers better by using median and interquartile range (IQR) instead of mean and standard deviation.
4) Log transformation

   It reduces the effect of skewness in data by compressing large values.

Encoding Categorical Data:

There are several techniques for encoding categorical data followed by

one-hot encoding: when categorical are unordered and the number of unique categories is small. It creates binary columns for each category.

Label encoding: when categories have ordered such as low, median, high. It assigns numeric value to each category.

Ordinal Encoding: when categories have a rank or order, It assigns incremental numbers based on order.

Frequency Encoding: when high cardinality categorical features exist which means many unique values. It replaces each category with the number of times it appears in the dataset.

Target Encoding: when categories are related to target variable values for regression and classification. It replaces each category with the mean of the target variable for that category.

Binary Encoding: when one hot encoding causes too many unique categories, It converts categories into binary representation and stores them in multiple columns.

Hash Encoding: when categorical features have very high cardinality use a hash function to map categories into fixed number of columns.

8. Exploratory Data Analysis (EDA):

It helps to understand data distribution and relationship between features.

Libraries Used: Pandas, Matplotlib, Seaborn.

Techniques: Data visualization, box plots, histograms, scatter plots and correlation analysis using pearson’s correlation.

Feature Engineering:

Feature Engineering is the process of creating, transforming, or selecting features to improve a machine learning model’s performance. It involves making raw data more meaningful and enhancing the predictive power of a model.

Feature engineering can be divided into six major categories followed by

Feature creation: It creates new features from existing data to improve prediction accuracy.

Feature Transformation: It modifies the existing features to enhance their predictive power.

- Log Transformation
- Polynomial Features
- Binning

Feature Selection: It helps remove irrelevant features, reducing overfitting and improving model performance.

Feature Extraction: It reduces complex data into simpler, more informative features. It uses PCA to reduces dimensionality by capturing most of the variance in fewer features and also uses Text Feature Extraction (TF-IDF) to convert text data into numerical values.

Tools and Libraries:

- NumPy & pandas for data analysis.
- Matplotlib & seaborn for data visualization.
- Implemented ML algorithm and pipelines for training and testing using Scikit-learn.

9. Cloud Platforms for ML Tasks:

AWS services:

AWS EC2: Learned how to launch and manage virtual machine.

AWS SageMaker: Learned how to build and train ML model in such platform.

10. Deep Learning and neural network:

Perceptron:

It's a single layer neural network used for binary classification tasks. It takes input data, processes it with set of weights and bias and then use an activation function to produce output.

Activation Function:

it decides whether it should pass its signal to the next layer. It allows the model to learn complex patterns. There are multiple types of activation functions over there, some common functions such as Rectified neural network, Leaky Rectified neural network, Sigmoid, Tanh, SoftMax.

Rectified neural network:

used to reduces the chance of vanishing gradient problem ReLU(x)

= max(0, x)

Leaky Rectified neural network:

ReLU output is always zero for negative inputs, causing it to become dead neurons. Leaky ReLU solve this problem by allowing a small gradient value for negative inputs, which helps in backpropagation.

Leaky ReLU(x) = x   if x>0

Ax   if x<=0

Sigmoid:

It maps the input between 0 and 1 and it is often used in output layer for binary classification problems.

Tanh:

similar to sigmoid but input values between –1 to 1 and it is often used in hidden layer.

SoftMax:

Used for multiclass classification problem. It converts the output into probability distribution.

Loss Function:

It is used to calculate the difference between the predicted value and the actual value. The goal is to minimize the error during training. There are multiple types of loss functions over there, some common functions such as Mean Squared Error, Mean Absolute Error, Huber Loss for Regression tasks and Cross entropy Loss, Hinge Loss, focul loss for Classification tasks.

Mean Squared Error:

It gives more weight to bigger mistakes by squaring the difference between predicted and actual values.

Mean Absolute Error:

It measures the average of the absolute difference between actual and predicted values.

Huber Loss:

It is being less sensitive to the outliers than MSE while providing smoother gradient than MAE.

Cross Entropy Loss:

It measures the difference between the predicted probability and actual class label.

Hinge Loss:

It tries to make the decision as confident as possible by pushing the correct class away from the decision boundary. It is mostly used in SVM algorithm to find the best line that separates the classes.

Focal loss:

It encourages the model to focus more on examples where it gets struggles and less focus on example where it’s already doing well.

11. Understanding the working of Basic Architectures:

Convolutional Neural network:

It is used to process image data by using convolutional layers as it detects the features such as edges, patterns in the image. It is very efficient for image related tasks such as image classification, object detection.

Recurrent Neural network:

It is used for handling sequence data such as text or time series data. It has loops in its structure that allow it to remember information from earlier steps and use it when processing later.

LSTM:

It is a special type of RNN that helps the model to remember long term information and forget the less important. It is useful when dealing with long sequence of data, where regular RNN might forgot the important details.

12. Transformer Architecture and Related concepts:

Encoder Decoder architecture

The transformer consists of encoder decoder structure. The encoder processes the input text and generates a high dimensional embedding vector representation of the input, which the decoder uses to produce the output.

Encoder: Converts the input into tokens and tokens are converted into contextual embeddings. It processes the entire input sequence at once using self-attention and feed forward layers.

Decoder: Generates the output by using encoder outputs and previous decoder outputs with masked self-attention.

Workflow of encoder:

1. Input text is tokenized into word tokens. Each token is converted into vector using embedding layer. Positional encoding is added to maintain the order of words in the sequence.
2. Encoder process the input by using multiple layers such as self-attention, add & layer normalization, feedforward network.
   a. Self-attention generates context aware representation of each token by attending to all other tokens in the input sequence.
   b. The output of self-attention is added to the input and normalized in the next layer.
   c. Feed forward network applies non linearity to enhance feature representation.
   d. Again the output is normalized
   e. The final encoder output is a set of contextualized embeddings which is passed to the decoder.

Workflow of decoder:

1. Decoder generates the output by using multiple layers such as masked self-attention, Cross-Attention, add & normalization, feedforward network.
   f. Masked self-attention is used to ensure each token in the decoder can only attends to the previous tokens not the future tokens.
   g. Cross -attention helps the decoder looks at the encoder output and focuses on the most important parts while generating response.
   h. The output is now get normalized and the non-linear transformation applied by Feed forward network.
   i. The final decoder output is contextualized embeddings.
   j. Softmax layer converts the decoder output into probabilities for each word in the vocabulary. It predicts one word at a time.
   k. Techniques like beam search, greedy decoding are used to generate final text.

Transformer Workflow:

It follows the Encoder-Decoder Architecture mainly used for translation, summarization and other sequence to sequence tasks.

I. Input processing stage consists of three actions such as tokenization, embedding Layer, Positional encoding.
II. Encoder Layer consists of Multi-Head self-Attention, Add and Layer normalization, Feedforward network, Add and Layer normalization. It produces a set of contextualized embeddings for each word.
III. Decoder layer consists of Masked self-attention, Cross-Attention, add & normalization, feedforward network. It produces the set of processed embeddings for final token prediction.
IV. Output generation stage uses linear layer, softmax layer, beam search to generate the full output sentence.
    a. Linear layer converts the decoder output embeddings into vocabulary sized logits.
    b. Softmax converts the logits into probabilities for each word in the vocabulary.
    c. Beam search selects the most likely next word
    d. Decoder repeats this step until it generates the full output sentence.

13. BERT – Encoder Only models

BERT is an encoder only model that it understands words based on the full sentence from both left and right words.

Working:

Tokenization:

BERT uses the Wordpiece Tokenizer, which breaks words into subwords.

Special tokens are added such as CLS, SEP

[CLS] - It contains the entire sentence meaning and is used for classification.

[SEP] - Separates sentences which is important for next sentence prediction

Positional encoding and segment embeddings are added with token embeddings to form the final input embeddings in BERT.

Processing through multiple self-attention Layers:

BERT has multiple transformer encoder layers, Each layer has self attention to focus on important words.

Extracting contextual embeddings:

After passing through multiple attention layer, BERT outputs contextual embeddings.

How BERT learns?

Masked Language Modelling:

Random words are masked in a sentence and BERT must predict them. This

helps BERT understand relationship between words.

Next Sentence Prediction:

Given two sentences, it predicts whether the second one follows the first BERT learns

to understand sentence relationships.

14. Transfer Learning and Fine tuning with BERT

Transfer learning:

pretrained BERT is used for new tasks without training from scratch.

Fine tuning:

It helps BERT adapt to specific task by training it on specific data while it keeps the pretrained knowledge.

Fine tuning process:

- Load the pretrained BERT model
- Add Task Specific Layers on top of BERT, depending on Task,
- Decide whether to keep some layers unchanged or train all layers.
- Use a dataset relavent to the task.
- Train the model on Task data

- Evaluate the fine tuned model
- Deploy the fine tuned model.

Optimization Techniques:

It is used to minimize the loss function and improve a models accuracy. It adjust model parameters such as weights and biases to reduce error.

There are many optimization Techniques over there, Gradient descent and its variants, momentum based, and adaptive learning rate. Among all these learned about

1) Gradient Descent and its varients such as batch gradient descent, stochastic gradient descent, and mini batch gradient descent
2) Adaptive learning rate methods such as adagrad, RMS prop, and adam.

Gradient Descent: It updates weight by computing gradients and it moves in the direction where the loss get minimized.

Variants of Gradient Descent:

- Batch Gradient Descent: it uses the entire dataset to compute gradients.
- Stochastic Gradient Descent: it updates weights after each data points. It is faster but it introduces noice in updates.
- Mini-batch Gradient Descent: it uses a small batch of data points to update weight.

Adaptive Learning Rate: These are optimizers that adjusts the learning rate dynamically for different parameters.

Adagrad: It adjust the learning rate for each parameter independently based on how frequently it is updated. If the parameter that gets frequent update then it get a smaller learning rate while those parameter update less frequently it will get high learning rate.

RMSprop: It is an improvement of adagrad that solves the problem that the training progresses, the learning rate becomes too small too quickly, causing the model to stop learning effectively. This issue is solved by RMsprop.

Rms prop avoids this by giving more importance to the recent gradients and older gradients gradually lose their influence which is achieved by exponentially weighted moving average of squared gradients. This allows it to keep learning rates stable.

Adam:

It combines the benefits of Momentum and RMSprop, making it learning efficient and stable.

In standard gradient descent, updates can be unstable because gradient keep changing directions, especially when training on complex dataset. It will cause zigzag movements slowing down convergence.

To solve this, Momentum keeps track of the past gradients to make the update smoothly instead of updating the weights only based on current gradient.

Instead of using the same learning rate for all parameters, RMSprop adapts learning rates based on how much each parameter needs to change.

Regularization Technique

Batch Normalization and Dropout:

Both Batch Normalization and Dropout are regularization techniques used in deep learning to improve the training of neural networks.

Batch Normalization: It normalizes the activations in each layer across mini batches.It helps optimization by allowing higher learning rates and reduce dependency on weight initialization.

Dropout: It helps by randomly deactivating neurons during training, forcing the network to learn from different patterns and preventing from overfitting. It ensures the optimizer finds a better generalization instead of just memorizing the training data.

15. NLP & Text Processing:

Tokenization:

Tokenization is the process of splitting input text into smaller tokens. There are many tokenization techniques over there.

1) Character Level
2) Wordlevel
3) Subword tokenization

4) Unigram language model
5) Byte level BPE
6) WordPeice
7) SentencePeice
8) Fasttext Tokenization
9) Character n-gram
10) Hybrid tokenization

All these tokenization techniques are used based on the use case and the type of NLP model being built. The most common tokenization techniques in modern NLP depend on the type of model and use case.

WordPiece:

It starts with characters as tokens and find the best pair to merge based on likelihood probability. It will merge pairs until vocabulary size is reached.

Used in Transformer-based models like:

BERT (Bidirectional Encoder Representations from Transformers)

ALBERT (A Lite BERT)

DistilBERT (Smaller, faster BERT version)

RoBERTa (Robustly optimized BERT variant)

Byte Pair Encoding:

It treats each word as a sequence of characters and find the most frequent adjacent character pair. It replaces that pair with new token. It repeats this process until a predefined vocabulary size is reached.

Used in Generative and Translation models like:

GPT (Generative Pre-trained Transformer) (OpenAI models use BPE)

T5 (Text-to-Text Transfer Transformer)

MarianMT (Machine Translation models

SentencePiece:

It sees text as a sequence of characters and finds the frequently occurring patterns. It merges smaller pieces into bigger, meaningful parts based on probability.

SentencePiece is commonly used in:

BERT, GPT, T5, ALBERT (Popular NLP models)

Multilingual NLP (Handling multiple languages in the same model)

Text generation and machine translation

Text Embeddings Techniques:

Embedding techniques convert words, sentences, or entire documents into numerical vectors so that NLP models can process them. These vectors capture semantic relationships.

Types of Embedding Techniques:

1) One-Hot Encoding
2) TF-IDF (Term Frequency-Inverse Document Frequency)
3) Word Embeddings

The most common embedding techniques:

Word2Vec:

Learns embeddings using:

1. CBOW (Continuous Bag of Words): Predicts a word from its surrounding words.

2. Skip-gram: Predicts surrounding words from a given word.

FastText:

It Improves Word2Vec by breaking words into subwords (character n-grams) and helps with rare words and different word forms, also it handles OOV words.

GloVe (Global Vectors for Word Representation):

It learns words meaning based on how often words appear together. It is used in NLP tasks where word relationships are important.

BERT embeddings:

It is contextualized embeddings unlike word2vec, it changes based on sentence meaning.

Words get different vectors in different contexts.

16. Topic Modelling

Topic Modeling is an unsupervised learning technique that identifies hidden topics from a collection of text data. There are different methods are there but the most commonly used methods such as LDA (Latent Dirichlet Allocation) and BERTopic.

LDA (Latent Dirichlet Allocation):

It assumes each document contains multiple topics. It finds patterns in how words appear together to create topic distributions. It is used for Summarizing Large datasets and grouping similar content.

BERTopic:

It is a modern deep-learning-based alternative to LDA. It uses BERT embeddings + clustering techniques so that it better at capturing topic meaning and can work on shorter texts.

It is used for extracting insights of customer reviews, grouping similar documents in large datasets and analyzing social media trends.

17. Advanced Machine Learning Techniques:

Advanced ML techniques improve model accuracy, generalization, and efficiency. These include ensemble learning, hyperparameter tuning, and optimization methods.

It is particularly effective in scenarios where high variance models, such as decision trees, tend to overfit the training data.

Hyperparameter Tuning:

Machine learning models have two types of pf parameters such as model parameter and hyperparameter.

Hyperparameter are model setting before training and how the model learns(learning rate and number trees to be trained).

Tuning these parameters improves model performance.

Techniques for Hyperparameter Tuning:

Grid search:

It is an exhaustive search method where we define a set of values for each hyperparameter and test all possible combinations.

For example, if we have:

- Learning rate: {0.01, 0.1, 0.2}
- Number of estimators: {50, 100, 200}
  Grid Search will test all 3 × 3 = 9 possible combinations.

Random Search:

- Instead of testing all possible combinations, Random Search selects random combinations of hyperparameters from the defined range.
- It does not systematically go through all values but instead randomly picks combinations.

For example:

- Learning rate: Between 0.001 to 0.3
- Number of estimators: Between 50 to 500 Random Search will randomly select values from these ranges.

Bayesian Optimization:

It tries to learn from previous trails to make better decisions in the future trials. So the result from previous trails are stored.

It built probabilistic models such as Gaussian Process which estimate the relationship between hyperparameters and model performance. This model predicts how good a new hyperparameter combination will be before actually testing it. The next hyperparameters are chosen smartly, focusing on areas that are most promising.

How Are Previous Results Calculated?

Each previous result is calculated by actually training the model and measuring its performance on validation data.

For example, if we are tuning hyperparameters for a random forest classifier, each trial does the following:

1. A set of hyperparameters (e.g., n_estimators = 100, max_depth = 5) is chosen.
2. The model is trained on the training dataset.
3. The model’s performance (e.g., accuracy or loss) is evaluated on the validation dataset.
4. The result (e.g., accuracy = 85%) is recorded.

Each of these **trial results** is used to update the probabilistic model.

Ensemble Learning:

Ensemble Learning combines multiple models to get better predictions than a single model. The goal is to reduce errors, improve accuracy, and make predictions more stable.

Types of Ensemble Learning:

1) Bagging
2) Boosting
3) Stacking

Bagging: It is also known as bootstrap Aggregating. It creates multiple subsets of data using random sampling with replacement. Trains multiple models independently on these subsets.

For Regression problem, It averages the predictions for final prediction whereas for Classification It takes majority vote of the prediction.

Example: Random Forest

A Random Forest is an ensemble of decision trees where each tree is trained using bagging. It is a collection of decision trees trained on different random subsets. It Reduces overfitting compared to a single decision tree.

Boosting: It a sequential ensemble learning technique where each model learns from the mistakes of the previous model. Assigns higher weights to misclassified data points, making subsequent models focus on difficult cases.

Popular Boosting algorithm:

AdaBoost:

1) It works by adjusting the weights of incorrectly classified samples so that subsequent weak learners focus more on difficult cases.
2) It uses an ensemble of weak classifiers such as decision stumps and combines them into a strong classifier.
3) The final model is a weighted sum of the weak learners, improving accuracy while reducing overfitting.

Gradient Boosting:

1) It Optimizes the loss function using gradient descent. Each new model is trained to correct the residual error of the previous model.
2) It builds trees sequentially, where each tree corrects the errors of its predecessors.
3) More flexible than AdaBoost, as it can use different loss functions like MSE for regression and log loss for classification.

XGBoost:

An optimized version of Gradient Boosting that improves speed and efficiency.

1) Uses regularization techniques (L1 and L2) to prevent overfitting.
2) Employs parallel processing, handling missing values, and optimized tree-building algorithms.

3) Widely used in data science competitions due to its superior performance.

Stacking:

1) Combines multiple base learners and a meta-model that learns how to best combine the predictions of base models.
2) Base models can be different types (e.g., decision trees, neural networks, SVMs).
3) The meta-model aggregates predictions to improve overall accuracy.

Feature Selection & Dimensionality Reduction:

Feature selection and dimensionality reduction are techniques such as PCA and LDA to reduce model complexity and improve performance.

Principle component Analysis (PCA)

1) A dimensionality reduction technique that transforms data into a lower-dimensional space while preserving variance.

2) It Identifies the principal components that capture the most variance.

3) It is Used in high-dimensional datasets to remove redundant features.

Linear Discriminant Analysis (LDA)

1) It is a supervised dimensionality reduction technique that finds a linear combination of features that best separates different classes.

2) Unlike PCA, LDA focuses on maximizing class separability rather than preserving variance and often used in classification tasks.

18. ML Model Deployment & MLOps

Deploying ML models effectively requires MLOps practices, which focus on automating and managing the ML lifecycle.

MLOps (Machine Learning Operations):

MLOps is a set of practices that automates the deployment, monitoring, and management of ML models in production. It ensures that models remain efficient, scalable, and maintainable over time by integrating DevOps principles into the ML lifecycle.

Model Serialization:

Before deploying a machine learning model, it needs to be serialized, which means converting it into a format that can be saved and reloaded for inference.

1. **Pickle & Joblib:**

pickle and joblib are Python libraries used to save and load ML models as binary files.

Pickle is a general serialization tool but inefficient for large NumPy arrays.

Joblib is optimized for large NumPy arrays and is more efficient for ML models.

2. **ONNX (Open Neural Network Exchange):**

ONNX is an open format that allows models trained in one framework (e.g.,PyTorch, TensorFlow, Scikit-learn) to be used in another.

It optimizes inference speed and supports deployment across different hardware architectures.

3. **TorchScript:**

TorchScript is used to serialize model which is trained using pytorch for production.

It takes a trained PyTorch model and converts it into a special or traced version that removes unnecessary computations and optimizes it for speed.

Deploying ML Models

Once a model is trained and serialized, it needs to be deployed so that applications can use it for predictions. This can be done using following frameworks and cloud services.

Flask & FastAPI:

Both Flask and FastAPI are Python web frameworks that help deploy machine learning models by creating REST APIs. These APIs allow external applications to send input data to the model and receive predictions.

Flask:

Flask is easy to use and works well for small-scale applications.

Requests are handled one after another, which may slow things down when dealing with multiple requests.

FastAPI:

It is built with async support, meaning it can handle multiple requests at the same time.

It Uses Python type hints to ensure automatic validation and API documentation. It is suited for real time applications requiring low latency, such as chatbots and recommendation system. It is faster than flask for handling concurrent requests.

TensorFlow Serving & TorchServe:

TensorFlow Serving:

Designed specifically for TensorFlow models.

Supports real-time inference and deploy multiple versions of a model. It Can scale efficiently by distributing load across multiple instances.

It suited for large-scale production deployments where high availability and performance are required.

TorchServe:

Designed for PyTorch models, enabling fast and efficient model serving.

Supports multi-model deployment, which means a multiple PyTorch models can run on the same server also it has built-in support for batching, logging, and monitoring.

Best for deploying deep learning models in real-world applications such as image recognition, NLP, and speech processing.

1. AWS Lambda & SageMaker:

Both provides cloud-based solutions for deploying and managing ML models.

AWS Lambda:

It is a serverless computing service, meaning you don't need to manage servers.

It runs the ML models on demand, and making it only pay when the model is used. It has a maximum execution time limit (typically 15 minutes), so not suitable for long-running inference tasks.

AWS SageMaker:

A fully managed ML platform for training, tuning, and deploying models at scale.

Supports distributed training, which means it can handle large models with big datasets.

It is suitable for production-grade models requiring continuous training and monitoring.

CI/CD for ML Models

CI/CD is a DevOps practice that automates the ML model lifecycle, ensuring that model updates are seamlessly integrated and deployed into production environments.

Continuous Integration (CI):

CI focuses on automating the process of integrating code changes into shared repository.

It automate the model training process when new data arrives. It ensures that data preprocessing, feature engineering, and model training works correctly.

Continuous Deployment (CD):

CD automates the process of deploying updated models into production after successful testing.It automatically deploy new models when they perform better than the existing model and also scales models across multiple servers to handle high traffic.

GitHub Actions for CI/CD in ML

GitHub Actions is a workflow automation tool that helps automate ML tasks like model retraining, testing, and deployment.

How to set up GitHub Action for CI/CD pipeline?

Step 1: When push an update, GitHub actions should perform:

1) Install dependencies
2) Train the model
3) Test the model
4) Deploy the model

Step 2: Create GitHub Repository:

1) Go to github and create repository.
2) Clone the repository to your local machine by using following comments
   - git clone (path should be given)
   - cd your-repo
3) Prepare your ML project
4) Inside your project folder, it should have:
   a) Train.py - A script to train the model
   b) Test.py – A script to test the model
   c) Requirements.txt – a file listing all dependencies (libraries used)
   d) .github/workflows/ml-pipeline.yml – github actions CI/CD workflow

Step 3: Write training and Testing Scripts:

Create a ML training script which has a code implementation to train, load and saves the model.

Step 4: Create a GitHub Actions Workflow (CI/CD Pipeline):

Github actions uses YAML files to define workflows.

YAML files:

It is a human readable data serialization format commonly used for configuration files, data exchange, and defining application settings.

Create “. github/workflows” directory and add YAML file using following command:

Mkdir -p .github/workflows

Touch .github/workflows/ml-pipeline.yml

Step 5: Write the CI/CD workflow inside ml-pipeline.yml Step 6:

Push code to GitHub using following comments

- git add .
- git commit -m "Added CI/CD pipeline"
- git push origin main
- once pushed, the GitHub will automatically trigger the CI/CD pipeline.

Step 7: Deploy the Model

Once training and testing succeed, we can deploy ml model using Docker or other frameworks used inside the CI/CD pipeline.

To deploy ML model using DOCKER

Docker:

Docker is a containerization tool that packages ML models along with their dependencies. It ensures the model runs consistently across different environments (local, cloud, or servers). Docker packages the code dependencies into a container.

Helps avoid dependency conflicts, making deployment easier.

Step 1: Create a Dockerfile

A docker file tells the docker how to package the code into a container.

Step 2: Build the Docker Image

Run the following command to create an image

- Docker build -t my-ml-model

Step 3: Run the docker container

Start a container from the image, now the ML model API is running inside a container.

Kubernetes:

It is a container management tool that helps to deploy, and scale containerized applications such as docker container automatically. If a container crashes, it restarts it automatically.

If traffic increases, Kubernetes automatically adds more containers and it distributes request across multiple containers.

Working:

It groups containers into units called pods and manage them efficiently.

1. Define a Deployment (deployment.yaml)

This file tells Kubernetes how to run ml model container like how many replicas and what docker image to use.

2. Define the Service (Service.yaml)

This file tells Kubernetes how to expose your ML model so that users can access it.

3. Deploy to Kubernetes

After writing these YAML files, apply them to Kubernetes. So that it will automatically deploy and manage the ml model.

ML Monitoring & Logging:

ML monitoring:

It is the process of continuously tracking model performance and behavior after deployment to ensure it remains accurate, efficient, and reliable.

Once a model is deployed, it might degrade over time due to data drift, model drift and concept drift.

Logging:

Logging is the process of recording important information during model training, deployment, and inference. It helps trace issues when a model fails.

What are to be log?

Training logs – hyperparameters, dataset versions,and training time. Inference

logs – input data, preedictions and execution time.

Performance logs – Accuraccy, precision recall over time

How to setting up ml monitoring and logging:

Step1: Choose the right tools for monitoring and logging

There are multiple tools are being used, the most common tools are MLflow, evidently AI and Prometheus and grafana

- MLflow is used to log model performance metrics and hyperparameters.
- Evidently AI is used to track data drift and model drift.
- Prometheus and grafana is used to monitor real time performance and set alerts.

Step 2: Log model performance using MLflow

Write a script to logs model type, accuracy, precision and recall. It stores the trained model for furture reference.

Detect data drift with Evidently AI

Write a script to track if incoming data distribution changes compared to training data and if model predictions start shifting unexpectedly.

Log API latency in Prometheus

Write a code to logs how much time the model takes to respond and if latency increases, it will alert via Grafana dashboards.

19.Project Overview and Documentation:

Title: Banking AI Chatbot

Abstract:

An AI-powered assistant designed to handle SBI banking-related queries using a combination of intent classification and Retrieval-Augmented Generation (RAG). the system accurately identifies whether the query is banking-related and responds using information extracted from official banking documents.

Features:

- **Intent Detection** using a fine tuned BERT model to filter banking vs non-banking queries.
- **RAG Pipeline** with FAISS vector and local LLM(Deepseek-r1) for answering banking questions.
- **Document-based Learning** uses real banking PDFs sourced from official websites.
- Achieved over **90% accuracy** by combining both intent classification and RAG.
- User-friendly **Streamlit Web Interface** for interactive usage.
- Handles only banking related questions to maintain domain relevance.

Dataset Summary:

The dataset includes 10,000 banking and 7,422 non-banking queries, sourced from HuggingFace and public datasets. Non-banking queries span domain-specific, general-purpose, and casual interactions. After cleaning and merging as Query–Intent pairs, 100 ambiguous queries were added for data augmentation. A total of 10,100 queries were used to train the intent classification model for better generalization.

Note: The dataset was used only to train the intent classification model to identify banking-related queries. It is not used in the RAG system or document retrieval process.

How to Add Banking PDFs:

- Run the build_vector_store.py
- Redirect to the Streamlit UI.
- Upload your banking-related PDF files.
- The app will process, chunk, embed, and index them.

How the System Works:

1. **Document Upload & Vector DB Creation**

- Banking PDFs are split semantically and embedded using HuggingFace models.
- Faiss is used to create a searchable vector store saved locally.

2. **RAG-Based Question Answering**

- When a query is identified as banking-related, RAG retrieves relevant document chunks.
- A local LLM (e.g., DeepSeek-r1 via Ollama) generates answers based on the retrieved context.

3. **End-to-End Inference**

- User enters a query.
- Intent model filters out non-banking questions.
- If banking-related, the RAG system generates a concise answer in under two sentences.
- If it's non-banking, you'll be asked to provide a banking query.

Accuracy:

- Intent model Accuracy - 99%
- Overall Accuracy (Intent model + RAG) - 90%

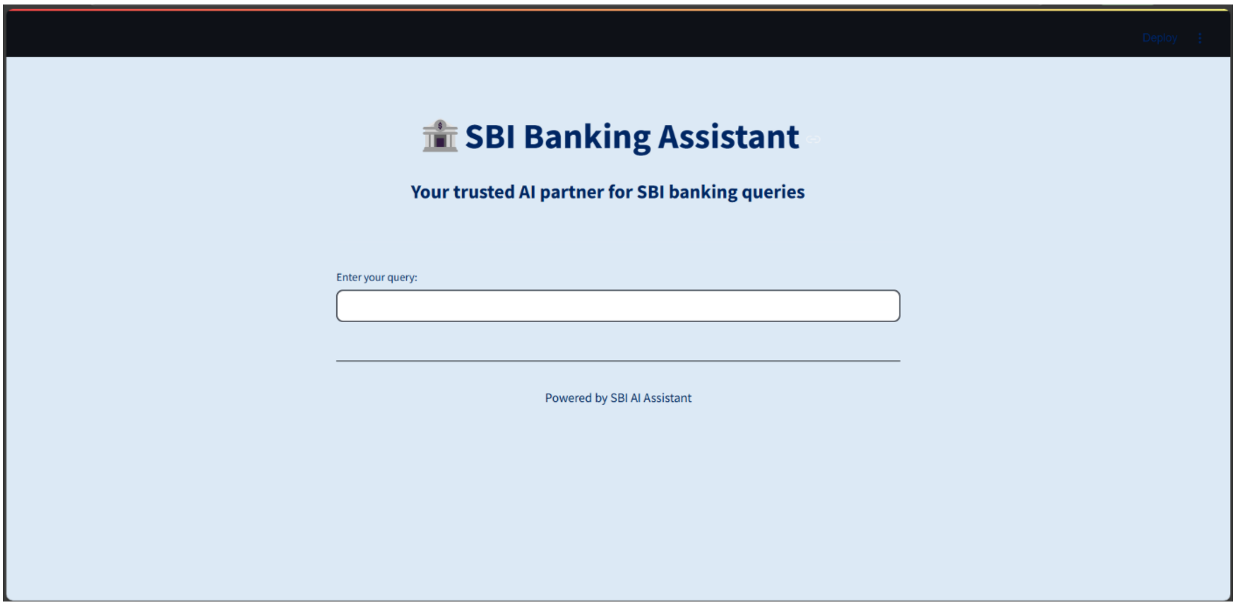
Install Dependencies:

pip install -r requiremnts.txt

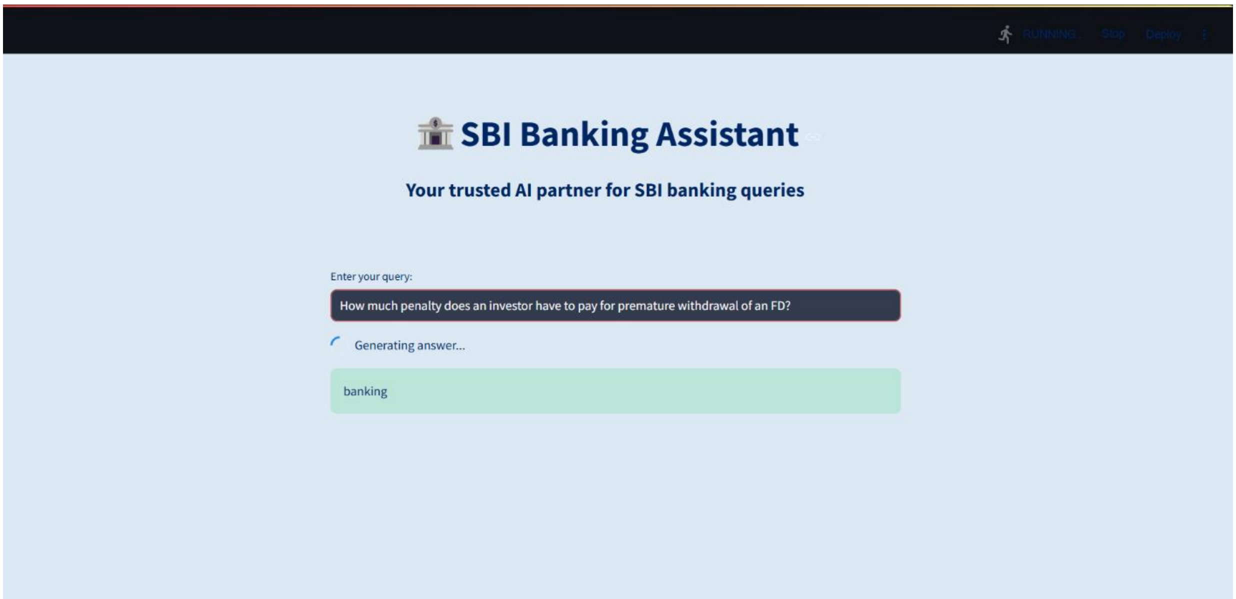
Run the App:

Streamlit run main.py

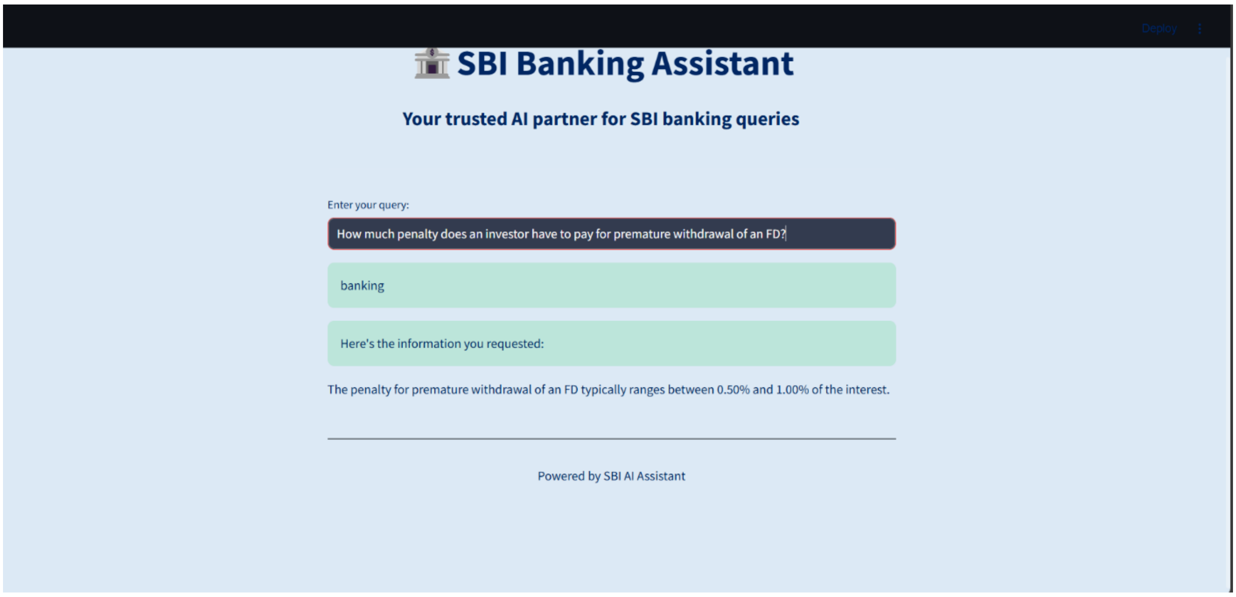
Demo Images:



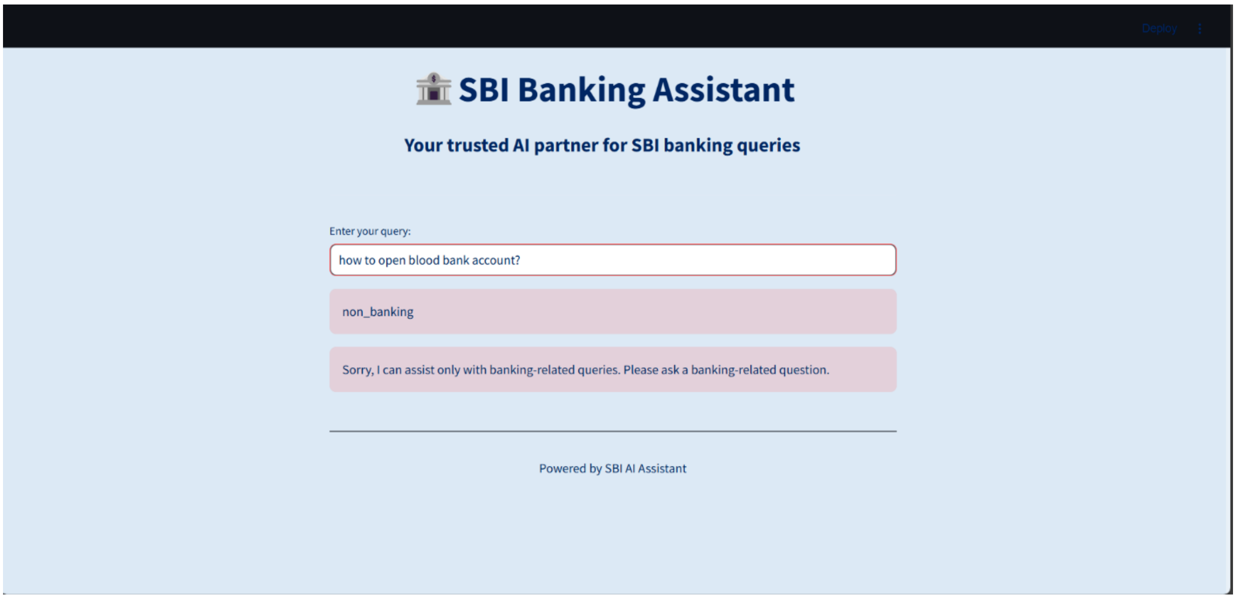
Homepage to Query



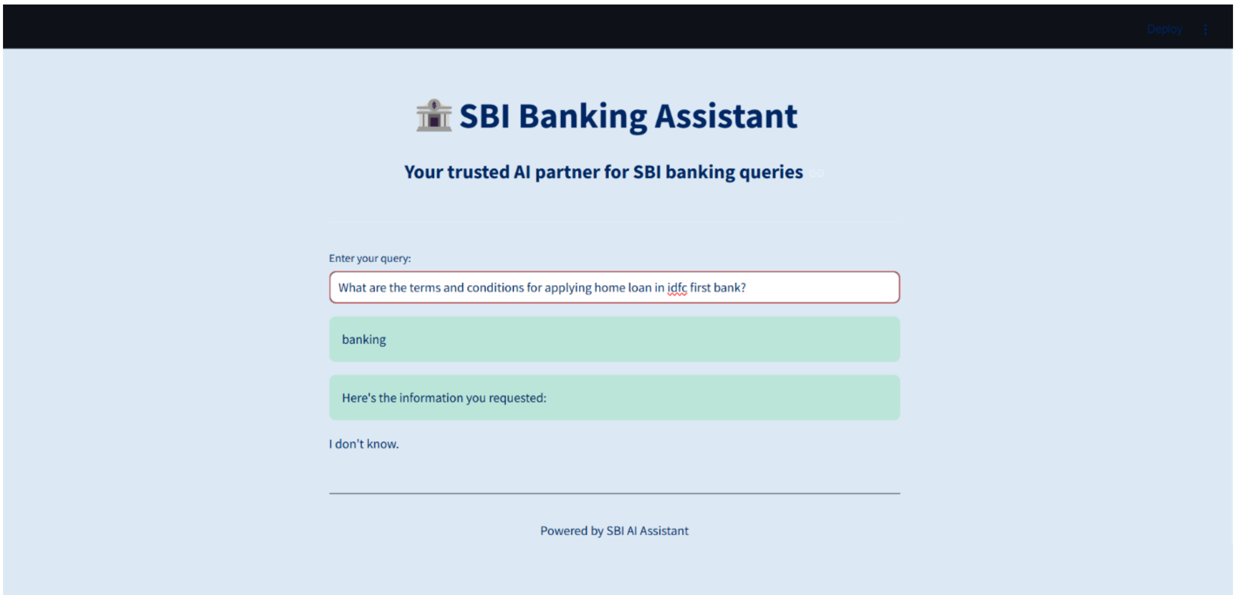
Intent Response



Banking Response



Non-Banking Response


SBI Banking Assistant
Your trusted AI partner for SBI banking queries
Enter your query:
What are the terms and conditions for applying home loan in idfc first bank?
banking
Here's the information you requested:
I don't know.
Powered by SBI AI Assistant

Out of Scope