VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI-590 018, KARNATAKA

ASSIGNMENT

ON

"Generative AI"

Submitted in the partial fulfilment of requirements for the award of Degree
B.E. in Computer Science and Engineering

Submitted By:

ANKITHA B	**4BD22CS016**
DISHA V KUMAR	**4BD22CS048**

GUIDE:

Dr. ABDUL RAZAK M S Ph.D.
Associate Professor.
Dept. of CS&E.

Bapuji Institute of Engineering and Technology Department of Computer Science and Engineering Davanagere-577004

2024-2025

Bapuji Institute of Engineering and Technology Davangere – 577004

Department of Computer Science and Engineering

DAVANAGERE-577004

CERTIFICATE

This is to certify that **ANKITHA B**, **DISHA V KUMAR** bearing USN **4BD22CS016, 4BD22CS048** respectively of Computer Science and Engineering department have satisfactorily submitted the Assignment on **"Generative AI"** in the partial fulfilment of the requirements for the award of Degree of Bachelor of Engineering (B.E.) in Computer Science and Engineering, under the VTU during the academic year 2024-25**.**

Dr. Abdul Razak M S Ph. D.	**Dr. Nirmala C R** Ph.D.
Associate Professor	**Head of Department**
Dept. of CS&E	**CS & E**
Guide	

Date: 05-06-2025
Place: Davangere

ACKNOWLEDGEMENT

Salutations to our beloved and highly esteemed institute, **"BAPUJI INSTITUTE OF ENGINEERING AND TECHNOLOGY"** for having well-qualified staff and labs furnished withthe necessary equipment.

We express our sincere thanks to our guides **Dr. Abdul Razak M S,** Associate Professor, Dept. of Computer Science and Engineering, BI.E.T., Davangere, for giving us the constant encouragement, support and valuable guidance throughout the course of the mini project without whose stable guidance this project would not been achieved.

We express whole hearted gratitude to **Dr. Nirmala C R** who is our respectable H.O.D of Computer science and Engineering Department. We wish to acknowledge her help who made our task easy by providing with her valuable help and encouragement

We also express our wholehearted gratitude to our respected Principal, **Dr. H B Aravind** For his moral support and encouragement.

We would like to extend our gratitude to all Teaching and Non-Teaching staff of the **Department of Computer Science and Engineering** for the help and support rendered to us. We have benefited a lot from the feedback, and suggestions given by them.

We would like to extend our gratitude to all our family members and friends especially for their advice and moral support.

Bapuji Educational Association (Regd.)

Bapuji Institute of Engineering and Technology, Davangere-577004

Vision and Mission of the Computer Science and Engineering Department

Vision

"To be a centre-of-excellence by imbibing state-of-the-art technology in the field of Computer Science and Engineering, thereby enabling students to excel professionally and be ethical."

Mission

1.	Adapting best teaching and learning techniques that cultivates Questioning and Reasoning culture among the students.
2.	Creating collaborative learning environment that ignites the critical thinking in students and leading to the innovation.
3.	Establishing Industry Institute relationship to bridge skill gap and make them industry ready and relevant.
4.	Mentoring students to be socially responsible by inculcating ethical and moral values.

Program Educational Objectives (PEOs):

PEO1	To apply skills acquired in the discipline of computer science and engineering for solving Societal and industrial problems with apt technology intervention.
PEO2	To continue their carrier ion industry /academia or pursue higher studies and research.
PEO3	To become successful entrepreneurs, innovators to design and develop software products and services that meets societal, technical and business challenges.
PEO4	To work in the diversified environment by acquiring leadership qualities with effective communication skills accompanied by professional and ethical values.

Program Specific Outcomes (PSOs):

PSO1	Analyse and develop solutions for problems that are complex in nature but applying the knowledge acquired from the core subjects of this program.
PSO2	To develop secure, scalable, resilient and distributed applications for industry and societal Requirements.
PSO3	To learn and apply the concepts and contract of emerging technologies like artificial intelligence, machine learning, deep learning, big-data analytics, IOT, cloud computing etc for any real time problems.

ABSTRACT

Generative Artificial Intelligence (Generative AI) represents a transformative advancement in the field of machine learning, where models are designed to generate new data that resembles a given dataset. Leveraging deep learning techniques such as Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and large language models (LLMs), generative AI can create realistic images, audio, text, and even videos. These models learn underlying patterns and structures in data, enabling them to produce novel and high-quality outputs that are often indistinguishable from human-created content. Applications of generative AI span diverse domains, including content creation, drug discovery, data augmentation, and personalized recommendations. Despite its immense potential, generative AI also raises ethical and societal concerns related to misinformation, bias, and intellectual property. As the technology continues to evolve, addressing these challenges is critical to ensuring its responsible and beneficial use.

Table of Contents

Table of Contents

CHAPTER 1: GENERATIVE AI

1.1 INTRODUCTION :

Artificial Intelligence (AI) has undergone significant evolution over the past decade, with machine learning and deep learning revolutionizing the way computers learn from data. Among the most exciting developments in this field is Generative AI**,** a branch of AI focused on creating new content rather than merely analyzing or classifying existing data. Unlike traditional AI models that are designed for recognition or prediction, generative models are capable of producing original data in various forms— such as images, text, music, and videos—by learning patterns from large datasets.Generative AI works by modeling the underlying distribution of the input data, allowing it to generate new samples that resemble the original dataset. Popular generative architectures include Generative Adversarial Networks (GANs)**,** Variational Autoencoders (VAEs)**,** and more recently**,** transformer-based models such as GPT (Generative Pre-trained Transformer) and DALL·E. These technologies have demonstrated remarkable success in a wide range of applications, from synthetic image generation and natural language processing to artistic creation and scientific discovery.The rise of large-scale generative models has unlocked powerful tools for industries. In entertainment, AI-generated art and music are gaining popularity. In healthcare, generative models assist in drug design and medical imaging. In education, they help create customized content and interactive learning environments. Furthermore, the integration of generative AI into business tools is enhancing automation and innovation across sectors.Despite its advantages, generative AI also introduces several challenges. Ethical concerns such as the spread of misinformation, deepfakes, data privacy, and bias in AI-generated outputs are becoming increasingly significant. These issues underscore the need for responsible AI development, transparency in model training, and the implementation of robust regulatory frameworks.As the field continues to expand, generative AI stands as both a remarkable achievement and a critical area of focus in modern artificial intelligence. Its potential to augment human creativity, automate complex tasks, and transform industries is immense, but must be approached with careful consideration .

1.2 CORE CONCEPTS :

1. What is Generative AI?

- A subfield of AI that creates new data similar to training data.
- Learns patterns and distributions from input data.
- Outputs can be: text, images, audio, video, code, and more.

2. Core Techniques

- **Generative Adversarial Networks (GANs):**
  - Two-part model: Generator vs. Discriminator.
  - Generator creates data; Discriminator evaluates it.
  - Used in realistic image/video synthesis.
- **Variational Autoencoders (VAEs):**
  - Encode input into a latent space and reconstruct with variation.
  - Common in image generation and data compression.
- **Transformers (e.g., GPT, DALL·E):**
  - Based on self-attention mechanism.
  - Handle long sequences, ideal for text/image generation.

3. How It Works

- Trained on large datasets (text corpus, image sets, etc.).
- Uses neural networks (deep learning).
- Learns semantic structures, syntax, and style of data.

1.3 APPLICATIONS AND USE CASES :

1. **Text Generation**
    a. **Chatbots & Assistants:** ChatGPT, Google Bard, Alexa.
    b. **Content Writing:** Blogs, scripts, poetry, reports.
    c. **Code Generation:** GitHub Copilot, OpenAI Codex.

2. **Image & Video Generation**
    a. **Art & Design:** DALL·E, Midjourney, Stable Diffusion.
    b. **Gaming:** Procedural level design and asset creation.
    c. **Deepfakes:** Realistic but synthetic media (controversial).

3. **Audio & Music**
    a. **Voice Synthesis:** Text-to-speech tools (e.g., WaveNet).
    b. **Music Composition:** AI-composed symphonies and soundtracks.

4. **Healthcare & Science**
    a. **Drug Discovery:** AI suggests chemical structures.
    b. **Medical Imaging:** Enhances scans, generates synthetic data.
    c. **Biology:** Protein structure prediction (e.g., AlphaFold).

5. **Business & Marketing**
    a. **Ad Generation:** Personalized copywriting at scale.
    b. **Customer Support:** AI-powered virtual agents.
    c. **Synthetic Data:** Augments real datasets for better model training

1.4 CHALLENGES, ETHICS, AND TRENDS :

1. **Key Challenges**
    a. **Bias in Outputs:** Mirrors training data biases.
    b. **Hallucination:** AI may fabricate false but realistic content.
    c. **Resource Demands:** Needs high compute power and memory.

2. **Ethical Issues**
    a. **Misinformation:** Deepfakes, fake news, spam.
    b. **Copyright Infringement:** Content may resemble existing works.
    c. **Privacy Concerns:** Models trained on personal or sensitive data.

3. **Mitigation Strategies**
    a. **Human Oversight:** "Human-in-the-loop" systems.
    b. **Explainability Tools:** Model transparency and accountability.
    c. **Regulatory Efforts:** AI ethics guidelines and usage laws.

4. **Future Directions**
    a. **Multimodal AI:** Combines text, images, audio in one model (e.g., Sora, GPT-4).
    b. **Creative Collaboration:** AI as co-writer, designer, or editor.
    c. **Low-code/No-code Tools:** Broader access to AI capabilities.
    d. **Personalized Generative Systems:** Custom AI for individuals/businesse

CHAPTER 2: EMBEDDINGS

USE CASE: SPELL CORRECTION AND TEXT NORMALIZATION USING EMBEDDINGS AND SIMILARITY

2.1 INTRODUCTION

Spell correction and text normalization are essential preprocessing steps in Natural Language Processing (NLP) tasks, especially when dealing with noisy, user-generated, or informal text. Traditional spell checkers rely on dictionary lookups or edit distances, which often fail to account for semantic meaning. To address this, modern approaches integrate **embedding-based similarity techniques**, where words are represented as high-dimensional vectors that capture contextual and semantic information. This enables models to not only suggest orthographic corrections but also preserve meaning by considering semantically related terms. Embedding-aware spell correction improves accuracy in downstream NLP tasks such as Named Entity Recognition (NER), sentiment analysis, and machine translation.

2.2 ROLE OF EMBEDDINGS IN SPELL CORRECTION

- **Semantic Awareness**

  Embeddings capture word meaning based on context and usage, allowing the model to suggest a meaningful alternative rather than purely phonetically similar ones. For example, if the input is "Googlle", embeddings help identify "Google" as a likely candidate based on usage patterns.

- **Fallback for Unknown Words**

  When traditional spellcheckers fail or when a word is out-of-vocabulary, embedding similarity allows the system to infer the correct word using semantic proximity in vector space.

- **Multi-strategy Correction**

  Combining embedding-based similarity with traditional spell correction and fuzzy string matching (e.g., Levenshtein distance or difflib) increases the robustness of the correction pipeline, especially in multilingual or specialized domains.

2.3 EMBEDDING-BASED TEXT SIMILARITY FOR CORRECTION

- **Nearest Neighbor Matching**

  For a misspelled or unknown word, the most similar words in the embedding space can be retrieved and scored. These suggestions are semantically related and often contextually appropriate.

- **Hybrid Correction Pipeline**

  A multi-step correction process is used to handle unknown or misspelled words. If the word exists in the embedding model, it is kept as is. Otherwise, the nearest neighbor in the embedding space is used. If that fails, rule-based spellchecker suggestions are applied. As a last resort, string similarity methods help find the closest match.

2.4 WORKFLOW EXAMPLE: SPELL CORRECTION USING EMBEDDINGS

Step 1: Load Pre-trained Word Embeddings

Load fastText subword embeddings trained on large corpora (e.g., Wikipedia) using Gensim.

Step 2: Tokenize and Check Each Word

Process each word in the sentence, checking if it exists in the model vocabulary.

Step 3: Suggest Corrections Using Embeddings

For unknown words, retrieve the top-N similar words using cosine similarity from the embedding space.

Step 4: Fall Back to Traditional Methods

If no embedding match is strong enough, fall back to spell checkers like pyspellchecker, or use string matching algorithms (difflib) to retrieve close alternatives.

Step 5: Return Corrected Sentence

The corrected sentence is reconstructed using the best available suggestions, preserving semantic intent and improving overall readability.

2.5 TOOLS AND LIBRARIES

- **Gensim**: For loading and querying pre-trained fastText embeddings.
- **pyspellchecker**: For traditional rule-based spelling correction.
- **difflib**: For string similarity using edit distance.
- **NumPy / Scikit-learn**: For calculating cosine similarity and vector operations.

2.6 CODE IMPLEMENTATION

```
!pip install pyspellchecker

!pip install gensim

import gensim.downloader as api

from spellchecker import SpellChecker

from difflib import get_close_matches

print("⏳ Loading fastText embeddings...")

model = api.load('fasttext-wiki-news-subwords-300')

print("✅ Embeddings loaded.")

spell = SpellChecker()

def tokenize(text):

    return text.lower().split()

def correct_word(word, similarity_threshold=0.5):

    # 1. If known word, keep as is

    if word in model:

        return word

    # 2. Try embedding similarity first

    try:

        sims = model.most_similar(word, topn=5)
```

```
        best_match, score = sims[0]
        if score > similarity_threshold:
            return best_match
    except:
        pass
    # 3. If embedding fails, fallback to spellchecker
    correction = spell.correction(word)
    if correction != word:
        return correction
    # 4. Lastly, try difflib close match
    alt = get_close_matches(word, model.key_to_index.keys(), n=1, cutoff=0.8)
    if alt:
        return alt[0]
    # 5. If all fails, return original word
    return word
def correct_sentence(sentence):
    words = tokenize(sentence)
    corrected = [correct_word(w) for w in words]
    return " ".join(corrected)
# User input
input_sentence = input(" Enter sentence with misspellings: ")
corrected_sentence = correct_sentence(input_sentence)
print("\n Corrected Sentence:", corrected_sentence)
```

2.7 OUTPUT

```
Requirement already satisfied: pyspellchecker in /usr/local/lib/python3.11/dist-packages (0.8.3)
Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.17.2)
⏳ Loading fastText embeddings...
[==================================================] 100.0% 958.5/958.4MB downloaded
✅ Embeddings loaded.
 Enter sentence with misspellings: The conpany is now headquartarered in Apple Park. Sundar Pichai is the CEO of Googlle. Barack Obama was born in Hawaii and served as the 44th President of the Uniited Staetes.

 Corrected Sentence: the company is now headquartered in apple park. sundar pickax is the ceo of google barack obama was born in hawaii and served as the 44th president of the united states
```

2.8 REAL-WORLD APPLICATIONS

- **Chatbots**: Interpreting user inputs with spelling errors or slang by mapping to correct tokens.
- **Search Engines**: Improving query understanding by correcting misspelled keywords using semantic proximity.
- **Text Preprocessing in NLP Pipelines**: Cleaning user-generated content before sentiment analysis, translation, or entity recognition.
- **Assistive Writing Tools**: Enhancing spell check in writing apps or IDEs with context-aware suggestions.

CHAPTER 3: HUGGING FACE

USE CASE: NAMED ENTITY RECOGNITION (NER) – USING HUGGING FACE TRANSFORMERS PIPELINE

3.1 INTRODUCTION

Named Entity Recognition (NER) is a key task in Natural Language Processing (NLP) that involves identifying specific entities—such as people, organizations, places, and others—within unstructured text. These entities are then categorized into predefined groups like PERSON, ORGANIZATION, LOCATION, etc. Traditionally, NER systems were rule-based or statistical, but with the advancement of transformer-based models, the process has become more accurate and context-aware.Hugging Face, a leader in modern NLP tools, provides a transformers library with pre-trained models specifically fine-tuned for NER tasks. One of the easiest ways to apply NER is by using the pipeline utility, which abstracts away the complexities of model loading, tokenization, and inference. This pipeline approach enables users to input a sentence and immediately receive extracted entities with contextual labels and confidence scores.

3.2 ROLE OF HUGGING FACE IN NAMED ENTITY RECOGNITION (NER)

1. **Simplified Access to Pretrained Models**

   Hugging Face provides direct access to high-performing models that are pre-trained and fine-tuned on benchmark datasets for NER, such as CoNLL-2003. These models are available for immediate use without any training or setup.

2. **Entity Aggregation for Better Readability**

   In many transformer models, entity names are tokenized into smaller sub-units. Hugging Face's pipeline supports automatic aggregation of these tokens into complete words or phrases, improving the clarity and usability of the output.

3. **Interactive NER Execution**

   The system can accept user-defined input text and performing NER dynamically. It identifies all relevant entities within the sentence and returns each with a category and confidence score, making it suitable for real-time analysis and conversational interfaces.

4. **Support for Multiple Languages and Contexts**

   The Model Hub includes multilingual and domain-specific models, making the solution across industries like healthcare, finance, and legal where terminology varies greatly.

3.3 WORKFLOW EXPLANATION

The NER pipeline is initialized to work with pre-trained models and is designed to process user-provided input in real-time. Once the user enters a sentence, the system processes the text to detect named entities, classifies them into appropriate groups such as PERSON, ORGANIZATION, or LOCATION, and returns results with associated confidence levels. The use of aggregation ensures that names or phrases are combined into complete entities rather than being returned as fragmented tokens.

The overall flow involves four core steps:

- Accepting natural language text from the user.
- Applying the pre-trained NER model using Hugging Face's pipeline abstraction.
- Aggregating subword tokens to form coherent named entities.
- Displaying the identified entities, their types, and the confidence scores.

3.4 FEATURES

- **Pipeline Abstraction**: Eliminates the need to manually tokenize, load models, or manage model output.
- **Real-Time Input Processing**: Allows for interactive usage where users can input sentences dynamically and get immediate results.
- **Entity Aggregation Strategy**: Enhances the quality of entity recognition by grouping token fragments into complete names or terms.

- **Confidence Scoring**: Each detected entity is accompanied by a probability score that reflects the model's certainty, which can be used to filter or rank entities in downstream applications.

3.5 TOOLS AND LIBRARIES UTILIZED

- **Hugging Face Transformers**

  The primary framework used to implement and run transformer-based NLP models. It provides the backbone for the pipeline API and includes a vast collection of NER models.

- **NER Pipeline**

  A task-specific utility that loads the appropriate model and handles input/output operations automatically, enabling users to focus on results rather than setup.

- **Pretrained Models with Aggregation**

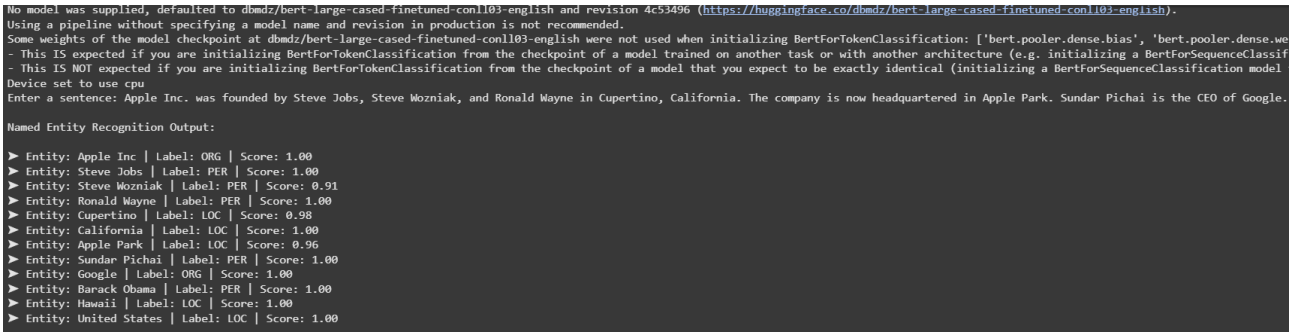
  These models are already fine-tuned for NER and come with built-in token grouping, reducing the need for post-processing or manual reassembly of fragmented names.

3.6 CODE IMPLEMENTATION

```
from transformers import pipeline

# Load the NER pipeline with entity grouping
ner_pipeline = pipeline("ner", aggregation_strategy="simple")

# Ask user for input
user_input = input("Enter a sentence: ")

# Run NER
entities = ner_pipeline(user_input)

# Print results
print("\nNamed Entity Recognition Output:\n")
```

```
if entities:
    for ent in entities:
        print(f"➤ Entity: {ent['word']} | Label: {ent['entity_group']} | Score: {ent['score']:.2f}")
else:
    print("No named entities found in the sentence.")
```

3.6 OUTPUT



3.7 OUTPUT CHARACTERISTICS

When applied to sentences mentioning public figures, companies, or locations, the NER system extracts and returns clear and contextual entity annotations. For example, in a sentence describing corporate founders and world leaders, the system is able to:

- Identify individuals like “Steve Jobs” and “Barack Obama” as PERSON entities.
- Recognize “Apple Inc.” and “Google” as ORGANIZATION entities.
- Detect “Cupertino” and “Hawaii” as LOCATION entities.

Each of these extractions is presented alongside a confidence score, often exceeding 0.95 for well-known names and commonly seen patterns.

3.8 APPLICATIONS IN INDUSTRY AND RESEARCH

- **Corporate Knowledge Mining**: Automatically extracting CEO names, founding dates, and headquarters from business articles.
- **Digital Assistants**: Enhancing chatbot understanding by recognizing product names, user identities, or service locations.

- **Healthcare NLP**: Identifying patient names, hospital names, and drug references in clinical text.
- **Search Relevance Optimization**: Improving search engine accuracy by associating keywords with recognized entities.
- **Education and Research Tools**: Mapping authors, institutes, or grant agencies in academic writing to help with citation tracking and discovery.

CHAPTER 4: MEDICAL ASSISTANT USING BLENDERBOT AND LANGCHAIN

USE CASE: CONVERSATIONAL MEDICAL SUPPORT BOT — POWERED BY TRANSFORMERS AND LANGCHAIN

4.1 INTRODUCTION

In the modern age of conversational AI, users expect fast, intelligent, and context-aware responses from virtual assistants across a wide range of domains, including healthcare. This project introduces a Medical Assistant Bot built using Hugging Face Transformers and LangChain, centered around the BlenderBot-400M model. The system offers a responsive and interactive chatbot experience, tailored to provide non-diagnostic health-related support and advice in natural language.

By integrating LangChain's memory architecture and prompt templates with Hugging Face's conversational models, this system enables dynamic interactions with historical context retention. The assistant engages users in a human-like dialogue, maintaining past conversations to deliver coherent and personalized responses. While it avoids direct diagnosis, the assistant provides general health guidance, lifestyle suggestions, and prompts users to seek professional medical advice when needed

4.2 ROLE OF WIKIPEDIA-API

1. **Conversational Pipeline with Transformers**

The assistant uses the facebook/blenderbot-400M-distill model, which is fine-tuned for multi- turn dialogue. This transformer-based pipeline allows for fluid, context-sensitive interactions, enabling the assistant to simulate human-like responses with empathy and relevance.

2. **Prompt Engineering with LangChain**

LangChain's ChatPromptTemplate allows the creation of a structured conversational flow. The prompt includes a system message (defining the assistant's role), a history placeholder, and the user's input. This structured prompt ensures the assistant consistently stays in its

defined role as a helpful, non-diagnostic entity.

3. **Memory Management**

LangChain's ConversationBufferMemory retains the chat history across turns. This allows the assistant to remember what the user said in previous messages, enabling continuity and coherence in the conversation—essential for medical dialogue.

4. **Chain Composition for Execution**

LangChain's modular approach lets developers compose a runnable chain (prompt | llm), combining the prompt, memory, and model to create a clean, efficient interaction pipeline.

5. **Lightweight Deployment**

The system runs efficiently on local CPU (device=-1) using open-access models, requiring no proprietary API keys—making it ideal for academic use, offline environments, or low-resource systems.

4.3 WORKING EXAMPLE

Step 1: Install and Import Dependencies

Install all required libraries including transformers, langchain, and langchain-huggingface. These libraries support the model pipeline, memory management, and chat formatting.

Step 2: Initialize BlenderBot Pipeline

The system creates a conversational pipeline using the BlenderBot model, wrapped inside a HuggingFacePipeline to make it compatible with LangChain's LLM interface.

Step 3: Setup Chat Prompt and Memory

A custom prompt defines the assistant's tone and responsibilities. Memory is enabled via ConversationBufferMemory, allowing the assistant to maintain the conversation flow across user inputs.

Step 4: Real-Time User Interaction

A while loop prompts the user to enter queries. The system then generates responses using the model, incorporating prior chat history and system instructions to produce thoughtful replies.

Step 5: Memory Update and Loop

After every turn, both user and assistant messages are saved in memory, ensuring that future inputs benefit from the context built up during the session.

4.4 TOOLS AND LIBRARIES

- **Transformers (Hugging Face)**

  Used to load and operate BlenderBot-400M, a transformer model optimized for dialogue generation.

- **LangChain**

  Provides memory support, chaining logic, and prompt templating for smooth integration and modularity.

- **LangChain-HuggingFace**

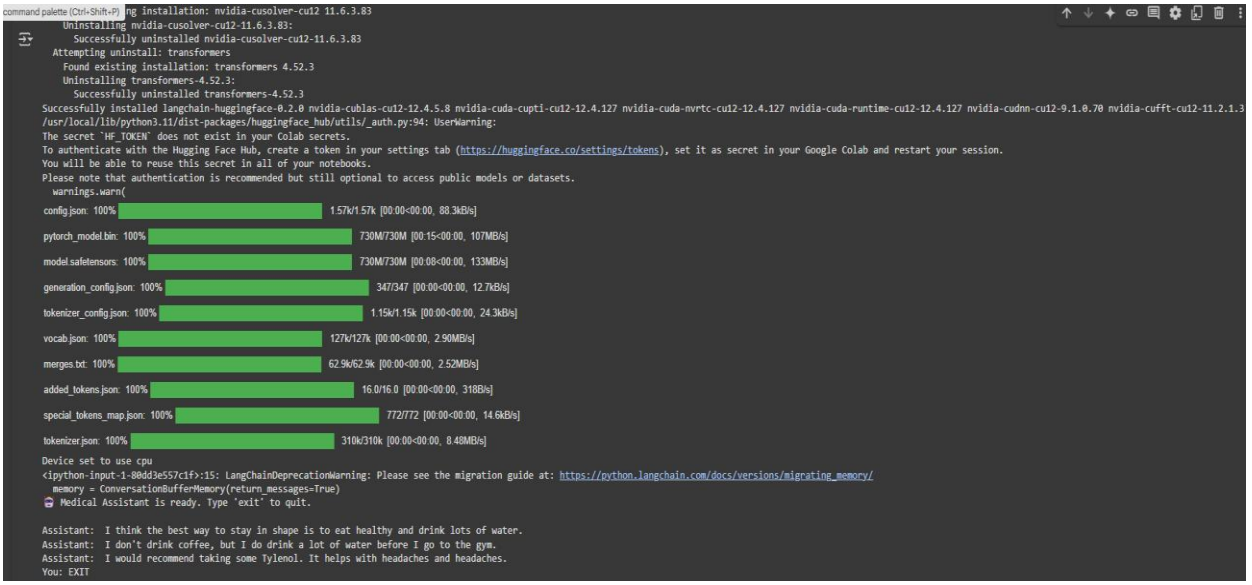
  A bridge between Hugging Face models and LangChain's interface, enabling seamless wrapping of model pipelines.

4.5 CODE IMPLEMENTATION

```
!pip install -U langchain langchain-huggingface transformers

from transformers import pipeline

from langchain_huggingface import HuggingFacePipeline

from langchain.memory import ConversationBufferMemory

from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
```

```
from langchain_core.runnables import Runnable

# Setup chat model (BlenderBot)

chat_pipeline = pipeline("text2text-generation", model="facebook/blenderbot-400M-distill",
    device=-1)

# Wrap in LangChain

llm = HuggingFacePipeline(pipeline=chat_pipeline)

memory = ConversationBufferMemory(return_messages=True)

# Chat prompt with memory

prompt = ChatPromptTemplate.from_messages([

   ("system", "You are a helpful medical assistant. Provide helpful advice without diagnosing
    directly."),

   MessagesPlaceholder(variable_name="history"),

   ("human", "{input}")])

chain = prompt | llm

print("🤖 Medical Assistant is ready. Type 'exit' to quit.\n")

while True:

   user_input = input("You: ")

   if user_input.lower() == "exit":

      break

   history = memory.chat_memory.messages

   # Get response

   response = chain.invoke({"input": user_input, "history": history})

   print("Assistant:", response)

   # Update memory

   memory.chat_memory.add_user_message(user_input)

   memory.chat_memory.add_ai_message(response)
```

4.6 OUTPUT



4.7 REAL WORLD APPLICATION

1. **First-Level Medical Guidance**

   Users can get lifestyle and symptom-related advice before consulting a doctor. It acts as an information filter and a reassurance system.

2. **Educational Health Companion**

   Medical students and learners can simulate patient queries and practice interpreting generalized advice without violating ethical boundaries.

3. **Support Chat for Health Websites**

   Clinics or health platforms can integrate this assistant to answer general health-related questions before routing to human professionals.

4. **Accessible Health Information in Low-Resource Areas**

   Offline or API-free access makes this assistant usable in remote or under-resourced regions for basic health awareness

CHAPTER 5 : PYDANTIC

USE CASE: ENSURE A FIELD IS A VALID DATE , FLOAT BEFORE PROCESSING

5.1 Introduction

Pydantic is a powerful Python library used for data validation and settings management using Python type annotations. It enables developers to define data models with strict type checks and validations, ensuring that data conforms to expected formats and constraints before processing. Pydantic automatically parses input data, converts types where possible, and raises clear errors if validation fails. It's widely used in modern Python applications, especially in web APIs and configuration handling, due to its ease of use, robust validation capabilities, and integration with frameworks like FastAPI.

5.2 TYPES OF PYDANTIC MODELS

- **BaseModel**: The fundamental Pydantic model used to create data classes with validation.
- **Settings Management Models**: Models that inherit from BaseSettings for managing environment variables and configurations.
- **Dataclasses**: Pydantic supports validation with Python's nativedataclasses using pydantic.dataclasses.dataclass.
- **Custom Validators**: Using the @validator decorator to add field-level or model-level validation logic.
- **Generic Models**: Models that support generics for reusability across different data types.
- **ORM Mode Models**: Allow Pydantic models to work seamlessly with ORM objects by setting orm_mode=True.

5.3 REAL WORLD EXAMPLES OF PYDANTIC

- **API Request/Response Validation**: Ensuring that incoming JSON payloads have correct fields and types before processing.

- **Configuration Management**: Loading and validating environment variables or config files in applications.
- **Data Parsing**: Reading user inputs or files and validating data before use, e.g., transaction records, user profiles.
- **Form Validation**: Validating user-submitted forms in web applications.
- **Microservices Communication**: Validating and serializing messages passed between services.

5.4 WORKING OF PYDANTIC

1. **Define a Model**

   Create a class that inherits from BaseModel and declare the fields with their expected types.

2. **Input Data Passed to Model**

   When you create an instance of the model, you pass in the data (e.g., dictionary, keyword args).

3. **Automatic Parsing and Type Conversion**

   Pydantic parses the input and tries to convert values to the specified types (e.g., strings to int, str to datetime).

4. **Field Validation**

   Pydantic checks each field against its type annotation and runs any built-in validations (e.g., required fields).

5. **Custom Validators Run**

   Any custom validation methods defined with @validator decorators are executed for specific fields.

6. **Raise Validation Errors (if any)**

   If any field fails validation or type conversion, Pydantic raises a ValidationError with details.

7. **Model Instance Created on Success**

   If all validations pass, a validated model instance is created and can be safely used in the program.

8. **Access Validated Data**

   You can access the fields directly as attributes, confident the data matches the expected schema.

9. **Serialization (Optional)**

   You can convert the model back to dict or JSON format for storage, transmission, or further processing.

5.5 CODE IMPLEMENTATION

```
from pydantic import BaseModel, validator, ValidationError from typing import Optional,List
class DataRecord(BaseModel):
    id: int name: str
    amount: float
    transaction_date:
    datetime
    status: Optional[str] = None  # Optional
    field @validator('name')
    def
        name_must_not_be_empt
        y(cls, v): if not v.strip():
            raise ValueError("Name cannot be
        empty") return v
    @validator('status')
    def status_must_be_valid(cls, v):
        allowed = {'pending', 'completed', 'failed'}
        if v is not None and v.lower() not in allowed:
            raise ValueError(f"Status must be one of
        {allowed}") return v
    #Creates with dataobject for storing valid ones and rejecting with errors
    valid_records = []
    errors = []
    for record in
```

```
    raw_data:
    try:
        validated =
        DataRecord(**record)
        valid_records.append(valid
        ated.dict())
    except ValidationError as e:
        errors.append({"record": record, "errors": e.errors()})

#Tables cretaed for both valid records and validation
errors from IPython.display import display
import pandas as pd
df_valid =
pd.DataFrame(valid_records)
df_errors =
pd.DataFrame(errors)
print(" ✅ Valid Records")
display(df_valid)
print("\n❌ Validation
Errors")
display(df_errors)
```

5.6 OUTPUT

CONCLUSION:

Generative AI utilizes transformer-based architectures to create human-like content in the form of text, code, images, and more. At its core, embeddings play a vital role by converting words, sentences, or entire documents into dense numerical representations that capture contextual meaning enabling tasks like similarity detection, classification, and search.Libraries like Hugging Face Transformers simplify the process of using pre-trained state-of-the-art models, making advanced NLP tasks accessible with just a few lines of code. On the other hand, LangChain empowers developers to build intelligent agents and workflows that can interact with tools, APIs, and memory, making LLM- powered apps more dynamic and capable.Pydantic strengthens these applications by ensuring that all inputs and outputs are well-structured, type-safe, and validated which is essential in real-world production environments where data consistency is critical.

REFERENCES

- Denis Rothman, *Transformers for Natural Language Processing*, Packt Publishing, 2021. https://www.packtpub.com/product/transformers-for-natural-language- processing/9781800565794
- Ian J. Goodfellow et al., *Generative Adversarial Nets*, NeurIPS, 2014. https://arxiv.org/abs/1406.2661
- Jay Alammar, *The Illustrated Transformer*, 2018. http://jalammar.github.io/illustrated-transformer/
- Creswell et al., *A Survey on Generative Models: From GANs to Diffusion Models*, Journal of Machine Learning Research, 2022. http://jmlr.org/papers/v23/21-1053.html
- Alec Radford et al., *Language Models are Few-Shot Learners*, OpenAI, 2021. Introduces GPT-3, a milestone large language model in generative AI. https://arxiv.org/abs/2005.14165
- Ashish Vaswani et al., *Attention is All You Need*, NeurIPS, 2017. The seminal paper that introduced the Transformer architecture widely used in generative AI models. https://arxiv.org/abs/1706.03762