Building a RAG Chatbot for Product and Policy Support in Retail

1. Objective

- Develop a Retrieval-Augmented Generation (RAG) chatbot that helps retail employees and customers quickly get accurate answers about product details, store policies, and return/exchange guidelines for a large retailer (e.g., Walmart, Amazon).
- The chatbot should reference in-memory retail documents and generate clear, human-friendly answers using Azure OpenAI.

2. Problem Statement

- Retail staff and customers often have questions about product specifications, warranties, in-store services, and return/exchange policies.
- Without instant access to information, staff spend time searching manuals or waiting for approvals, and customers face longer wait times.
- By deploying a RAG chatbot, retail operations can deliver fast, reliable, and context-aware answers that improve service efficiency and customer satisfaction.

3. Inputs / Shared Artifacts

- Azure OpenAI Resource: API key, endpoint URL, and deployment name (to be set as environment variables).
- You are given a mock dataset of Walmart policy and product documents (hardcoded in code for this exercise):

id	content
1	"Walmart customers may return electronics within 30 days with a receipt and original packaging."
2	"Grocery items at Walmart can be returned within 90 days with proof of purchase, except perishable products."
3	"Walmart offers a 1-year warranty on most electronics and appliances. See product details for exceptions."
4	"Walmart Plus members get free shipping with no minimum order amount."
5	"Prescription medications purchased at Walmart are not eligible for return or exchange."
6	"Open-box items are eligible for return at Walmart within the standard return period, but must include all original accessories."
7	"If a Walmart customer does not have a receipt, most returns are eligible for store credit with valid photo identification."
8	"Walmart allows price matching for identical items found on Walmart.com and local competitor ads."
9	"Walmart Vision Center purchases may be returned or exchanged within 60 days with a receipt."
10	"Returns on cell phones at Walmart require the device to be unlocked and all personal data erased."
11	"Walmart gift cards cannot be redeemed for cash except where required by law."
12	"Seasonal merchandise at Walmart (e.g., holiday decorations) may have modified return windows, see in-store signage."
13	"Bicycles purchased at Walmart can be returned within 90 days if not used outdoors and with all accessories present."
14	"For online Walmart orders, customers can return items in store or by mail using the prepaid label."

id	content
15	"Walmart reserves the right to deny returns suspected of fraud or abuse."

4. Expected Outcome

- The chatbot must accept any user's natural language question (e.g., "Can I return a Walmart bicycle if I've ridden it?").
- It should **retrieve relevant passages** from the document set using embeddings/vectorstore.
- It should **generate an answer** that references the retrieved information, making clear, concise, and contextually correct recommendations.

Example Input → Output: (for Amazon, see below)

User Question	Retrieved Context	Generated Answer
"Can I return an opened Amazon Kindle?"	- Amazon allows returns of Kindles within 30 days of delivery, provided the device is in new condition. - Open-box items are eligible for return at Amazon within the standard return window but must include all original accessories.	You may return an opened Amazon Kindle within 30 days if it is in new condition and includes all accessories. See Amazon policy for details.

5. Concepts Covered

- Azure OpenAI API
- Prompt Engineering
- Vector Store, Embedding
- Retrieval-Augmented Generation (RAG)

6. Example: Step-by-Step Implementation (Python, LangChain, Azure OpenAI)

```
# Step 0: Imports & Environment Setup

import os
from typing import TypedDict, Optional
from langchain_openai import AzureOpenAIEmbeddings, AzureChatOpenAI
from langchain_community.docstore.in_memory import InMemoryDocstore
from langchain_community.vectorstores import FAISS
from langchain_core.documents import Document
from langchain_core.prompts import ChatPromptTemplate
from langgraph.graph import StateGraph, END

# Step 1: Example Amazon Documents
os.environ["AZURE_OPENAI_EMBEDDING_API_KEY"] = ""
os.environ["AZURE_OPENAI_EMBEDDING_ENDPOINT"] = "https://aiportalapi.stu-
platform.live/jpe"
os.environ["AZURE_OPENAI_EMBED_MODEL"] = "text-embedding-3-small"

os.environ["AZURE_OPENAI_LLM_API_KEY"] = ""
os.environ["AZURE_OPENAI_LLM_ENDPOINT"] = "https://aiportalapi.stu-
platform.live/jpe"
os.environ["AZURE_OPENAI_LLM_MODEL"] = "GPT-4o-mini"

# --- Step 2: Sample Documents ---
docs = [
    Document(
        page_content="Amazon customers can return most items within 30 days
of receipt for a full refund."
    ),
    Document(
        page_content="Digital products purchased from Amazon, such as eBooks
or apps, are not eligible for return once downloaded."
    ),
    Document(
        page_content="Amazon Prime members receive free two-day shipping on
eligible items."
    ),
    Document(
```

```
        page_content="Electronics purchased from Amazon must be returned in
new condition with all original packaging and accessories."
    ),
    Document(
        page_content="Amazon Kindle devices may be returned within 30 days of
delivery, provided the device is in new condition."
    ),
]

# --- Step 3: Typed State for LangGraph ---
class RAGState(TypedDict):
    question: str
    context: Optional[str]
    answer: Optional[str]

# --- Step 4: Embedding & Vector Store ---
embeddings = AzureOpenAIEmbeddings(
    azure_endpoint=os.environ["AZURE_OPENAI_EMBEDDING_ENDPOINT"],
    api_key=os.environ["AZURE_OPENAI_EMBEDDING_API_KEY"],
    model=os.environ["AZURE_OPENAI_EMBED_MODEL"],
    api_version="2024-07-01-preview",
)
vectorstore = FAISS.from_documents(
    docs,
    embeddings,
    docstore=InMemoryDocstore({str(i): doc for i, doc in enumerate(docs)}),
)

retriever = vectorstore.as_retriever(search_kwargs={"k": 2})

# --- Step 5: Azure Chat Model ---
llm = AzureChatOpenAI(
    azure_endpoint=os.environ["AZURE_OPENAI_LLM_ENDPOINT"],
    api_key=os.environ["AZURE_OPENAI_LLM_API_KEY"],
    deployment_name=os.environ["AZURE_OPENAI_LLM_MODEL"],
    api_version="2024-07-01-preview",
    temperature=0,
)

# --- Step 6: Prompt Template ---
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You are a helpful Amazon support assistant. Use the provided
information to answer product and policy questions. Always cite the retrieved
info in your answer.",
        ),
        ("human", "{context}\n\nUser question: {question}"),
    ]
)

# --- Step 7: Define Nodes ---
def retrieve_node(state: RAGState) -> RAGState:
    docs = retriever.get_relevant_documents(state["question"])
```

```
    context = "\n".join([doc.page_content for doc in docs])
    return {**state, "context": context}

def generate_node(state: RAGState) -> RAGState:
    formatted_prompt = prompt.format(
        context=state["context"], question=state["question"]
    )
    answer = llm.invoke(formatted_prompt)
    return {**state, "answer": answer.content}

# --- Step 8: Build LangGraph ---
builder = StateGraph(RAGState)

builder.add_node("retrieve", retrieve_node)
builder.add_node("generate", generate_node)

builder.set_entry_point("retrieve")
builder.add_edge("retrieve", "generate")
builder.set_finish_point("generate")

rag_graph = builder.compile()
if __name__ == "__main__":
    user_question = "Can I return an opened Amazon Kindle?"
    result = rag_graph.invoke({"question": user_question})

    print("📚 Retrieved Context:\n", result["context"])
    print("\n💬 Generated Answer:\n", result["answer"])
```

7. Final Submission

- Source code implementing the RAG chatbot as above (with 15 data entries for your chosen brand) in .py file.
- At least one realistic Q&A demo showing input question, retrieved context, and generated answer.
- Use a dummy input list (auto input). Do not use manual input (input() built-in).
- Brief reflection on how RAG improves retail support vs. traditional static FAQs or keyword search.
- (Optional) Suggestions for expanding to product recommendations or multi-turn conversations.