```
# Cell 1: Install Required Dependencies (NO GOOGLE PACKAGES)
!pip install transformers
!pip install torch
!pip install sentence-transformers
!pip install qdrant-client
!pip install pypdf
!pip install PyPDF2
!pip install pdfplumber
!pip install pandas
!pip install numpy
```

```
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from httpx>=0.20.0->httpx[http2]>=0.20.0->qdran
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx>=0.20.0->httpx[http2]>=0.20.0-
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages (from httpx>=0.20.0->httpx[http2]>=0.20.0->qdrant-c
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx>=0.20.0->httpx[http
Requirement already satisfied: h2<5,>=3 in /usr/local/lib/python3.12/dist-packages (from httpx[http2]>=0.20.0->qdrant-client) (4.3
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic!=2.0.*,!=2.1.*,!=2
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.12/dist-packages (from pydantic!=2.0.*,!=2.1.*,!=2.
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.12/dist-packages (from pydantic!=2.0.*,!=2.1.*,
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from pydantic!=2.0.*,!=2.1.*,!
Requirement already satisfied: hyperframe<7,>=6.1 in /usr/local/lib/python3.12/dist-packages (from h2<5,>=3->httpx[http2]>=0.20.0-
Requirement already satisfied: hpack<5,>=4.1 in /usr/local/lib/python3.12/dist-packages (from h2<5,>=3->httpx[http2]>=0.20.0->qdra
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.12/dist-packages (from anyio->httpx>=0.20.0->httpx[http2]>=0
Downloading qdrant_client-1.15.1-py3-none-any.whl (337 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 337.3/337.3 kB 6.6 MB/s eta 0:00:00
Downloading portalocker-3.2.0-py3-none-any.whl (22 kB)
Installing collected packages: portalocker, qdrant-client
Successfully installed portalocker-3.2.0 qdrant-client-1.15.1
Collecting pypdf
  Downloading pypdf-6.0.0-py3-none-any.whl.metadata (7.1 kB)
Downloading pypdf-6.0.0-py3-none-any.whl (310 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 310.5/310.5 kB 5.5 MB/s eta 0:00:00
Installing collected packages: pypdf
Successfully installed pypdf-6.0.0
Collecting PyPDF2
  Downloading pypdf2-3.0.1-py3-none-any.whl.metadata (6.8 kB)
Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 5.1 MB/s eta 0:00:00
Installing collected packages: PyPDF2
Successfully installed PyPDF2-3.0.1
Collecting pdfplumber
  Downloading pdfplumber-0.11.7-py3-none-any.whl.metadata (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.8/42.8 kB 2.1 MB/s eta 0:00:00
Collecting pdfminer.six==20250506 (from pdfplumber)
  Downloading pdfminer_six-20250506-py3-none-any.whl.metadata (4.2 kB)
Requirement already satisfied: Pillow>=9.1 in /usr/local/lib/python3.12/dist-packages (from pdfplumber) (11.3.0)
Collecting pypdfium2>=4.18.0 (from pdfplumber)
  Downloading pypdfium2-4.30.0-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (48 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 48.5/48.5 kB 3.2 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from pdfminer.six==20250506->
Requirement already satisfied: cryptography>=36.0.0 in /usr/local/lib/python3.12/dist-packages (from pdfminer.six==20250506->pdfpl
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.12/dist-packages (from cryptography>=36.0.0->pdfminer.six==202
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.12->cryptography>=36.0.0->pdfmin
Downloading pdfplumber-0.11.7-py3-none-any.whl (60 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.0/60.0 kB 4.9 MB/s eta 0:00:00
Downloading pdfminer_six-20250506-py3-none-any.whl (5.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.6/5.6 MB 59.5 MB/s eta 0:00:00
Downloading pypdfium2-4.30.0-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.8/2.8 MB 68.9 MB/s eta 0:00:00
Installing collected packages: pypdfium2, pdfminer.six, pdfplumber
Successfully installed pdfminer.six-20250506 pdfplumber-0.11.7 pypdfium2-4.30.0
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
```

```
# Cell 2: Import Required Libraries
import os
import json
import re
from typing import List, Dict, Any
from pathlib import Path
import pandas as pd
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# PDF processing
import PyPDF2
import pdfplumber

# Transformers and embeddings
```

```
from transformers import pipeline, AutoTokenizer, AutoModel
from sentence_transformers import SentenceTransformer
import torch

# Vector operations
import numpy as np

print("All libraries imported successfully!")
```

```
All libraries imported successfully!
```

```
#Cell 3: Configure Hugging Face Models
print("Setting up Hugging Face models...")

# Initialize question-answering model (best for manual queries)
try:
    qa_pipeline = pipeline(
        "question-answering",
        model="distilbert-base-cased-distilled-squad",
        device=0 if torch.cuda.is_available() else -1
    )
    print("Question-answering model loaded successfully")
except Exception as e:
    print(f"Error loading Q&A model: {e}")

# Initialize embeddings model
try:
    embedding_model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")
    print("Embeddings model loaded successfully")
except Exception as e:
    print(f"Error loading embeddings: {e}")

print("Models configured successfully!")
```

```
Setting up Hugging Face models...
config.json: 100%  473/473 [00:00<00:00, 34.5kB/s]
model.safetensors: 100%  261M/261M [00:07<00:00, 38.8MB/s]
tokenizer_config.json: 100%  49.0/49.0 [00:00<00:00, 2.78kB/s]
vocab.txt: 100%  213k/213k [00:00<00:00, 2.34MB/s]
tokenizer.json: 100%  436k/436k [00:00<00:00, 9.22MB/s]
Fetching 0 files: 0/0 [00:00<?, ?it/s]
Fetching 1 files: 100%  1/1 [00:00<00:00, 74.68it/s]
Fetching 0 files: 0/0 [00:00<?, ?it/s]
Device set to use cpu
Question-answering model loaded successfully
modules.json: 100%  349/349 [00:00<00:00, 30.0kB/s]
config_sentence_transformers.json: 100%  116/116 [00:00<00:00, 9.76kB/s]
README.md: 10.5k/? [00:00<00:00, 485kB/s]
sentence_bert_config.json: 100%  53.0/53.0 [00:00<00:00, 2.80kB/s]
config.json: 100%  612/612 [00:00<00:00, 30.1kB/s]
model.safetensors: 100%  90.9M/90.9M [00:01<00:00, 85.7MB/s]
tokenizer_config.json: 100%  350/350 [00:00<00:00, 19.4kB/s]
vocab.txt: 232k/? [00:00<00:00, 12.2MB/s]
tokenizer.json: 466k/? [00:00<00:00, 16.4MB/s]
special_tokens_map.json: 100%  112/112 [00:00<00:00, 6.17kB/s]
config.json: 100%  190/190 [00:00<00:00, 14.3kB/s]
Embeddings model loaded successfully
Models configured successfully!
```

```
# Cell 4: Document Processing Functions
def extract_section_heading(text: str, page_num: int) -> str:
    """Extract section heading from document text"""
    lines = text.split('\n')
    for line in lines[:10]:
        line = line.strip()
        if line and len(line) < 100 and (line.isupper() or line.istitle()):
```

```
            return line
    return f"Page {page_num}"

def try_multiple_pdf_loaders(pdf_path: str):
    """Try multiple PDF loading strategies"""
    # Strategy 1: PyPDF2
    try:
        docs = []
        with open(pdf_path, 'rb') as file:
            pdf_reader = PyPDF2.PdfReader(file)
            for page_num, page in enumerate(pdf_reader.pages):
                try:
                    text = page.extract_text()
                    if text.strip():
                        docs.append({
                            'page_content': text,
                            'metadata': {
                                'page': page_num + 1,
                                'source': pdf_path,
                                'document_name': Path(pdf_path).stem
                            }
                        })
                except Exception:
                    continue

        if docs:
            print(f"Successfully loaded {pdf_path} with PyPDF2")
            return docs
    except Exception as e:
        print(f"PyPDF2 failed for {pdf_path}: {str(e)}")

    # Strategy 2: pdfplumber
    try:
        docs = []
        with pdfplumber.open(pdf_path) as pdf:
            for page_num, page in enumerate(pdf.pages):
                try:
                    text = page.extract_text()
                    if text and text.strip():
                        docs.append({
                            'page_content': text,
                            'metadata': {
                                'page': page_num + 1,
                                'source': pdf_path,
                                'document_name': Path(pdf_path).stem
                            }
                        })
                except Exception:
                    continue

        if docs:
            print(f"Successfully loaded {pdf_path} with pdfplumber")
            return docs
    except Exception as e:
        print(f"pdfplumber failed for {pdf_path}: {str(e)}")

    # If all fail, create placeholder
    print(f"All methods failed for {pdf_path}")
    return [{
        'page_content': f"Error loading PDF: {pdf_path}",
        'metadata': {'page': 1, 'source': pdf_path, 'error': True}
    }]

def load_and_process_pdfs(pdf_paths: List[str]) -> List[Dict]:
    """Load PDFs with multiple fallback strategies"""
    all_documents = []
    successful_loads = 0

    for pdf_path in pdf_paths:
        print(f"Processing: {pdf_path}")
        docs = try_multiple_pdf_loaders(pdf_path)

        if docs and not docs[0]['metadata'].get('error', False):
            successful_loads += 1

        doc_name = Path(pdf_path).stem

        for doc in docs:
            page_num = doc['metadata'].get('page', 0)
            section_heading = extract_section_heading(doc['page_content'], page_num)

            doc['metadata'].update({
```

```
                'document_name': doc_name,
                'page_number': page_num,
                'section_heading': section_heading,
                'source_file': pdf_path,
                'processed_date': datetime.now().isoformat()
            })

            all_documents.append(doc)

    print(f"Processing complete: {successful_loads}/{len(pdf_paths)} documents loaded")
    print(f"Total pages: {len(all_documents)}")

    return all_documents

def create_semantic_chunks(documents: List[Dict], chunk_size: int = 800, overlap: int = 150) -> List[Dict]:
    """Create semantic chunks"""
    chunks = []

    for doc in documents:
        content = doc['page_content']

        # Simple sentence-based chunking
        sentences = content.split('. ')
        current_chunk = ""

        for sentence in sentences:
            if len(current_chunk + sentence) < chunk_size:
                current_chunk += sentence + ". "
            else:
                if current_chunk.strip():
                    chunks.append({
                        'page_content': current_chunk.strip(),
                        'metadata': {
                            **doc['metadata'],
                            'chunk_id': len(chunks),
                            'chunk_length': len(current_chunk),
                            'chunk_type': 'sentence-based'
                        }
                    })
                current_chunk = sentence + ". "

        # Add remaining chunk
        if current_chunk.strip():
            chunks.append({
                'page_content': current_chunk.strip(),
                'metadata': {
                    **doc['metadata'],
                    'chunk_id': len(chunks),
                    'chunk_length': len(current_chunk),
                    'chunk_type': 'sentence-based'
                }
            })

    print(f"Created {len(chunks)} semantic chunks")
    return chunks

print("Document processing functions defined!")
```

```
Document processing functions defined!
```

```
# Cell 5: Simple Vector Store Implementation
class SimpleVectorStore:
    """Simple vector store using numpy for similarity search"""

    def __init__(self, embedding_model):
        self.embedding_model = embedding_model
        self.documents = []
        self.embeddings = []

    def add_documents(self, chunks: List[Dict]):
        """Add documents to vector store"""
        print(f"Adding {len(chunks)} chunks to vector store...")

        self.documents = chunks
        contents = [chunk['page_content'] for chunk in chunks]

        # Create embeddings
        self.embeddings = self.embedding_model.encode(contents, show_progress_bar=True)

        print(f"Vector store setup complete with {len(chunks)} documents")

    def similarity_search(self, query: str, k: int = 5) -> List[Dict]:
```

```
        """Search for similar documents"""
        if not self.documents:
            return []

        # Encode query
        query_embedding = self.embedding_model.encode([query])

        # Calculate similarities
        similarities = np.dot(self.embeddings, query_embedding.T).flatten()

        # Get top k results
        top_indices = np.argsort(similarities)[-k:][::-1]

        results = []
        for idx in top_indices:
            results.append({
                'page_content': self.documents[idx]['page_content'],
                'metadata': self.documents[idx]['metadata'],
                'similarity_score': similarities[idx]
            })

        return results

print("Vector store functions defined!")
```

```
Vector store functions defined!
```

```
# Cell 6: Hugging Face Chatbot with Memory
class HuggingFaceChatbot:
    """Chatbot using Hugging Face models with conversational memory"""

    def __init__(self, qa_pipeline, vector_store):
        self.qa_pipeline = qa_pipeline
        self.vector_store = vector_store
        self.conversation_history = {}

    def _format_context(self, docs: List[Dict]) -> str:
        """Format retrieved documents as context"""
        context_parts = []
        for i, doc in enumerate(docs, 1):
            metadata = doc['metadata']
            content = doc['page_content'][:500] + "..." if len(doc['page_content']) > 500 else doc['page_content']

            context_part = f"""Document {i}:
Source: {metadata.get('document_name', 'Unknown')}
Page: {metadata.get('page_number', 'Unknown')}
Content: {content}
---"""
            context_parts.append(context_part)

        return "\n\n".join(context_parts)

    def chat(self, session_id: str, user_input: str, show_sources: bool = False) -> Dict[str, Any]:
        """Main chat function"""

        # Initialize session history
        if session_id not in self.conversation_history:
            self.conversation_history[session_id] = []

        # Retrieve relevant documents
        docs = self.vector_store.similarity_search(user_input, k=3)

        if docs:
            # Use Q&A pipeline with retrieved context
            context = self._format_context(docs)

            try:
                # Use the best document as context for Q&A
                best_doc_content = docs[0]['page_content']

                result = self.qa_pipeline(
                    question=user_input,
                    context=best_doc_content[:2000]  # Limit context length
                )

                answer = result['answer']
                confidence = result.get('score', 0)

                # Format response based on confidence
                if confidence > 0.5:
                    response = f"{answer}"
                elif confidence > 0.2:
```

```
                    response = f"{answer} (Note: This answer has moderate confidence - please verify in the manual)"
                else:
                    response = "I found some relevant information but couldn't provide a confident answer. Please check the manual sect

                # Add source information
                source_info = f" (Source: {docs[0]['metadata'].get('document_name', 'Unknown')}, Page {docs[0]['metadata'].get('page_nur
                response += source_info

            except Exception as e:
                response = f"I found relevant manual sections but encountered an error processing your question. Please check the retri
        else:
            # No relevant documents found
            response = "I couldn't find relevant information in the available manuals for your question. Please try rephrasing or ask al

        # Store conversation
        self.conversation_history[session_id].append({
            'question': user_input,
            'response': response,
            'timestamp': datetime.now().isoformat()
        })

        # Prepare result
        result = {
            "response": response,
            "sources": [
                {
                    "document": doc['metadata'].get('document_name', 'Unknown'),
                    "page": doc['metadata'].get('page_number', 'Unknown'),
                    "section": doc['metadata'].get('section_heading', 'Unknown'),
                    "content_preview": doc['page_content'][:200] + "...",
                    "similarity": doc.get('similarity_score', 0)
                }
                for doc in docs
            ]
        }

        if show_sources:
            print(f"Retrieved {len(docs)} relevant chunks:")
            for i, source in enumerate(result["sources"], 1):
                print(f"{i}. {source['document']} (Page {source['page']})")
                print(f"   Section: {source['section']}")
                print(f"   Similarity: {source['similarity']:.3f}")
                print(f"   Preview: {source['content_preview']}\n")

        return result

    def get_session_summary(self, session_id: str) -> str:
        """Get summary of conversation history"""
        if session_id not in self.conversation_history:
            return "No conversation history found."

        history = self.conversation_history[session_id]
        if not history:
            return "No messages in this session."

        return f"Session has {len(history)} messages. Last message: {history[-1]['question'][:100]}..."

    def clear_session(self, session_id: str):
        """Clear conversation history"""
        if session_id in self.conversation_history:
            del self.conversation_history[session_id]
            print(f"Session {session_id} cleared.")

print("Hugging Face chatbot class defined!")
```

```
Hugging Face chatbot class defined!
```

```
# Cell 7: File Upload and Processing
from google.colab import files

def upload_and_process_manuals():
    """Handle file upload in Colab and process manuals"""
    print("Please upload your PDF manual files...")
    print("Expected files:")
    print("- Samsung Galaxy S23 manual")
    print("- Canon EOS Rebel T7 manual")
    print("- Whirlpool Washing Machine manual")
    print("- Any additional product manual (optional)")

    uploaded = files.upload()

    pdf_paths = []
```

```
    for filename in uploaded.keys():
        if filename.endswith('.pdf'):
            pdf_paths.append(filename)
            print(f"Uploaded: {filename}")

    if len(pdf_paths) < 1:
        print("No PDF files uploaded. Please upload at least one PDF file.")
        return []

    return pdf_paths

# For testing purposes, if files are already in directory
def get_existing_pdfs():
    """Get existing PDF files in current directory"""
    pdf_files = [f for f in os.listdir('.') if f.endswith('.pdf')]
    print(f"Found {len(pdf_files)} PDF files: {pdf_files}")
    return pdf_files

# Try to get existing PDFs first, then upload if none found
try:
    manual_paths = get_existing_pdfs()
    if not manual_paths:
        print("No existing PDFs found. Please upload files.")
        manual_paths = upload_and_process_manuals()
except:
    print("Please upload your PDF manual files using the upload function.")
    manual_paths = []
```

```
Found 0 PDF files: []
No existing PDFs found. Please upload files.
Please upload your PDF manual files...
Expected files:
- Samsung Galaxy S23 manual
- Canon EOS Rebel T7 manual
- Whirlpool Washing Machine manual
- Any additional product manual (optional)
```

Choose Files 3 files

- **galaxy_s25.pdf**(application/pdf) - 4794558 bytes, last modified: 9/5/2025 - 100% done
- **galaxy_s23 (1).pdf**(application/pdf) - 8102742 bytes, last modified: 9/4/2025 - 100% done
- **eos_rebel_t7 (1).pdf**(application/pdf) - 10710061 bytes, last modified: 9/4/2025 - 100% done

```
Saving galaxy_s25.pdf to galaxy_s25.pdf
Saving galaxy_s23 (1).pdf to galaxy_s23 (1).pdf
Saving eos_rebel_t7 (1).pdf to eos_rebel_t7 (1).pdf
Uploaded: galaxy_s25.pdf
Uploaded: galaxy_s23 (1).pdf
Uploaded: eos_rebel_t7 (1).pdf
```

```
# Cell 8: Main Processing Pipeline
def run_huggingface_pipeline(manual_paths: List[str]) -> HuggingFaceChatbot:
    """Run the complete processing pipeline with Hugging Face"""

    print("Starting Hugging Face processing pipeline...")

    # Step 1: Load and process PDFs
    print("Step 1: Loading PDFs...")
    documents = load_and_process_pdfs(manual_paths)

    # Check if we have valid documents
    valid_docs = [doc for doc in documents if not doc['metadata'].get('error', False)]

    if not valid_docs:
        print("No valid documents loaded. Cannot proceed with pipeline.")
        return None
    elif len(valid_docs) < len(documents):
        print(f"Only {len(valid_docs)}/{len(documents)} documents loaded successfully.")
        documents = valid_docs

    # Step 2: Create chunks
    print(f"Step 2: Creating chunks from {len(documents)} valid pages...")
    chunks = create_semantic_chunks(documents, chunk_size=800, overlap=150)

    if not chunks:
        print("No chunks created. Cannot proceed.")
        return None

    # Step 3: Setup vector store
    print(f"Step 3: Setting up vector store with {len(chunks)} chunks...")
    try:
        vector_store = SimpleVectorStore(embedding_model)
        vector_store.add_documents(chunks)
    except Exception as e:
        print(f"Vector store setup failed: {str(e)}")
        return None
```

```
    # Step 4: Initialize chatbot
    print("Step 4: Initializing Hugging Face chatbot...")
    try:
        chatbot = HuggingFaceChatbot(qa_pipeline, vector_store)
    except Exception as e:
        print(f"Chatbot initialization failed: {str(e)}")
        return None

    print("Pipeline complete! Chatbot ready for use.")

    # Display summary
    successful_manuals = len(set(doc['metadata'].get('document_name', 'Unknown') for doc in documents))
    print(f"""
PROCESSING SUMMARY:
- Manual files processed: {len(manual_paths)}
- Successfully loaded manuals: {successful_manuals}
- Total pages processed: {len(documents)}
- Total chunks created: {len(chunks)}
- Vector store: Ready
- Chatbot: Initialized

Ready to answer questions about your manuals!
""")

    return chatbot

# Execute pipeline
if manual_paths:
    print(f"Found {len(manual_paths)} PDF files to process:")
    for i, path in enumerate(manual_paths, 1):
        print(f"   {i}. {path}")

    chatbot = run_huggingface_pipeline(manual_paths)

    if chatbot:
        print("SUCCESS! Your Hugging Face chatbot is ready!")
    else:
        print("FAILED! Please check the error messages above.")
else:
    print("No PDF files found. Please upload PDF files first.")
    chatbot = None
```

```
Found 3 PDF files to process:
   1. galaxy_s25.pdf
   2. galaxy_s23 (1).pdf
   3. eos_rebel_t7 (1).pdf
Starting Hugging Face processing pipeline...
Step 1: Loading PDFs...
Processing: galaxy_s25.pdf
Successfully loaded galaxy_s25.pdf with PyPDF2
Processing: galaxy_s23 (1).pdf
Successfully loaded galaxy_s23 (1).pdf with PyPDF2
Processing: eos_rebel_t7 (1).pdf
Successfully loaded eos_rebel_t7 (1).pdf with PyPDF2
Processing complete: 3/3 documents loaded
Total pages: 503
Step 2: Creating chunks from 503 valid pages...
Created 832 semantic chunks
Step 3: Setting up vector store with 832 chunks...
Adding 832 chunks to vector store...
Batches: 100%|| 26/26 [01:23<00:00, 1.27s/it]
Vector store setup complete with 832 documents
Step 4: Initializing Hugging Face chatbot...
Pipeline complete! Chatbot ready for use.

PROCESSING SUMMARY:
- Manual files processed: 3
- Successfully loaded manuals: 3
- Total pages processed: 503
- Total chunks created: 832
- Vector store: Ready
- Chatbot: Initialized

Ready to answer questions about your manuals!

SUCCESS! Your Hugging Face chatbot is ready!
```

```
# Cell 9: Example Interactions
def run_example_interactions(chatbot: HuggingFaceChatbot):
    """Run example interactions"""

    print("RUNNING EXAMPLE INTERACTIONS")
    print("=" * 50)
```

```
    examples = [
        {
            "type": "Samsung Galaxy S23 Setup",
            "session": "demo_1",
            "question": "How do I set up my Samsung Galaxy S23 for the first time?",
        },
        {
            "type": "Canon Photography",
            "session": "demo_2",
            "question": "What are the different shooting modes on the Canon EOS Rebel T7?",
        },
        {
            "type": "Whirlpool Washing",
            "session": "demo_3",
            "question": "How do I select wash cycles on the Whirlpool washing machine?",
        },
        {
            "type": "Battery Management",
            "session": "demo_4",
            "question": "How can I optimize battery life on my Samsung Galaxy S23?",
        },
        {
            "type": "Camera Settings",
            "session": "demo_5",
            "question": "How do I adjust ISO settings on the Canon T7?",
        }
    ]

    for i, example in enumerate(examples, 1):
        print(f"\n{i}. {example['type']}")
        print(f"Question: {example['question']}")
        print("-" * 40)

        result = chatbot.chat(
            session_id=example['session'],
            user_input=example['question'],
            show_sources=True
        )

        print(f"Response: {result['response']}")
        print("=" * 60)

# Run examples if chatbot is available
if 'chatbot' in locals() and chatbot:
    run_example_interactions(chatbot)
```

```
     2. eos_rebel_t7 (1) (Page 68)
        Section: Page 68
        Similarity: 0.476
        Preview: 66In Basic Zone modes, when the shooting function settings are
     displayed, you can press the < Q> button to display the Quick Control
     screen and can set the functions shown in the table on the next p...

     3. eos_rebel_t7 (1) (Page 87)
        Section: Page 87
        Similarity: 0.468
        Preview: 85k Shooting Movies
     General Movie Shooting Cautions are on pages 91-92.
     If necessary, also read General Live View Shooting Cautions on
     pages 81-82.
     The ISO speed (ISO 100 - ISO 6400), shutter spee...

     Response: Manually (Source: galaxy_s23 (1), Page 59)
     ============================================================
```

```
# Cell 10: Pre-filled Interactive Multi-turn Demo
def interactive_demo_pre_filled(chatbot):
    """
    Runs pre-filled demo for multiple sessions:
    - Direct factual questions
    - Memory-based follow-ups
    - Out-of-scope question
    """
    sessions = {
        "s25_demo": [
            "How do I set up my Samsung Galaxy S25 for the first time?",
            "How can I optimize battery life on my Samsung Galaxy S25?"
        ],
        "canon_demo": [
            "What are the different shooting modes on the Canon EOS Rebel T7?",
            "How do I adjust ISO settings for night photography on the Canon T7?"
        ],
        "out_of_scope_demo": [
            "What is the capital of France?"
        ]
    }

    for session_id, questions in sessions.items():
        print("="*60)
        print(f"Session: {session_id} (demonstrating memory and retrieval)")
        print("="*60)

        for question in questions:
            print(f"\nYou: {question}")
            result = chatbot.chat(session_id, question, show_sources=True)
            print(f"Chatbot: {result['response']}")
            print("-"*60)

# Run the pre-filled demo if chatbot is ready
if 'chatbot' in locals() and chatbot:
    interactive_demo_pre_filled(chatbot)
else:
    print("Chatbot not initialized. Please run the main pipeline first.")
```

```
   Preview: Worldclock
TheWorldclockletsyoukeeptrackofthecurrenttimeinmultiplecitiesaroundthe
globe.
Location
C
enter the globe on
your current
location.City
Access the current
time and add to
your list of ci...

3. galaxy_s23 (1) (Page 77)
   Section: I
   Similarity: 0.183
   Preview: 816x4
3,264
0 l
E!l (fID 0
I
C () % . -.
7 8 9 X
4 5 6 -
1 2 3 +
+/- 0 e
Ill 0 < Apps
Calculator
The Calculator app features both basic and scientific math functions, as well as a unit
con...

Chatbot: New York (Note: This answer has moderate confidence - please verify in the manual) (Source: galaxy_s23 (1), Page 83)
------------------------------------------------------------
```