Here's a complete **step-by-step code guide** to develop a **Multi-Agent AI Assistant** using:

- ✅ **CrewAI** (multi-agent coordination)
- ✅ **LangChain** (LLM orchestration)
- ✅ **FAISS** (vector database for company data)
- ✅ **Ollama + LLaMA** (local LLM)
- ✅ **Web interface (Gradio or Streamlit)**
- ✅ **Cross-channel support**: Chat, Email, Social Media
- ✅ **Trainable** on your company's own PDFs/CSVs/Docs

🔧 OVERVIEW

Feature	Tool/Method
AI Framework	CrewAI + LangChain
Data Search	FAISS + Ollama Embeddings
LLM	LLaMA3 via Ollama (local)
UI	Gradio or Streamlit
Channel Integration	WhatsApp (via n8n), Email, Facebook
Custom Company Data	PDFs, CSVs, Docs
Modular Agent Design	FAQ, Scheduling, Sales, Support

🚀 STEP-BY-STEP CODE GUIDE

---

1. 📄 Install Required Libraries

pip install crewai langchain faiss-cpu pdfplumber pandas openpyxl gradio

2. 🗃️ Load and Embed Company Data

```
from langchain.document_loaders import PyPDFLoader, CSVLoader
```

```
from langchain.embeddings import OllamaEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import FAISS

embedding_model = OllamaEmbeddings(model="llama3")

def load_and_embed_docs(file_paths):
    docs = []
    for path in file_paths:
        if path.endswith(".pdf"):
            loader = PyPDFLoader(path)
        elif path.endswith(".csv"):
            loader = CSVLoader(file_path=path)
        loaded = loader.load()
        splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=50)
        docs.extend(splitter.split_documents(loaded))
    db = FAISS.from_documents(docs, embedding_model)
    db.save_local("company_knowledge_faiss")
    return db
```

3. Setup CrewAI Agents

```
from crewai import Agent
from langchain.vectorstores import FAISS
from langchain.llms import Ollama

llm = Ollama(model="llama3")
vectorstore = FAISS.load_local("company_knowledge_faiss", embedding_model,
allow_dangerous_deserialization=True)
retriever = vectorstore.as_retriever()

faq_agent = Agent(
    role="FAQ Bot",
    goal="Answer common questions based on company policy",
    backstory="Knows the full company handbook and website FAQs.",
    tools=[retriever],
    llm=llm
)

scheduler_agent = Agent(
    role="Scheduler",
    goal="Help users schedule meetings or calls with our team.",
    backstory="Has access to the team's calendar availability.",
    llm=llm
)
```

```
sales_agent = Agent(
    role="Sales Assistant",
    goal="Answer product-related queries and convince leads.",
    backstory="Trained on product descriptions and sales brochures.",
    tools=[retriever],
    llm=llm
)

support_agent = Agent(
    role="Support Helper",
    goal="Assist customers with common troubleshooting steps.",
    backstory="Knows common issues from support documentation.",
    tools=[retriever],
    llm=llm
)
```

4. 🧠 Assign Tasks to Agents

```
from crewai import Task, Crew

faq_task = Task("Respond to customer FAQs: {query}", agent=faq_agent)
schedule_task = Task("Schedule a meeting based on: {query}",
agent=scheduler_agent)
sales_task = Task("Provide product/sales info for: {query}",
agent=sales_agent)
support_task = Task("Help solve this issue: {query}", agent=support_agent)

crew = Crew(
    agents=[faq_agent, scheduler_agent, sales_agent, support_agent],
    tasks=[faq_task, schedule_task, sales_task, support_task],
    verbose=True
)
```

5. Route User Query to Right Agent

```
def route_query(query):
    lower = query.lower()
    if any(word in lower for word in ["faq", "policy", "how"]):
        return crew.run(faq_task.format(query=query))
    elif any(word in lower for word in ["meeting", "schedule",
"appointment"]):
        return crew.run(schedule_task.format(query=query))
```

```
    elif any(word in lower for word in ["price", "buy", "offer", "product"]):
        return crew.run(sales_task.format(query=query))
    elif any(word in lower for word in ["error", "issue", "not working",
"problem"]):
        return crew.run(support_task.format(query=query))
    else:
        return "Sorry, I couldn't classify your query."
```

6. 🌐 Web Interface with Gradio

```
import gradio as gr

def chat(query):
    return route_query(query)

gr.Interface(fn=chat, inputs="text", outputs="text", title="Company AI
Assistant").launch()
```

7. 🔌 Connect with Channels

Channel	Integration Tool	Description
Chat	Gradio / Streamlit	Web-based chat UI
WhatsApp	n8n + WhatsApp Cloud API	Auto-forward queries to chatbot
Email	IMAP/SMTP or Gmail API	Parse inbound emails and auto-respond
Facebook	Meta Graph API	Respond to page DMs using webhook

🧠 Add Memory (Optional)

```
from langchain.memory import ConversationBufferMemory

memory = ConversationBufferMemory(return_messages=True)
```

🗃️ File Structure Suggestion

```
company_agent/
│
├── main.py                    # Gradio App + Router
├── agents.py                  # CrewAI agent setup
├── vectorstore.py             # FAISS + RAG logic
├── data/                      # Company PDFs/CSVs
└── faiss_store/               # Auto-created vector DB
```

✅ You Now Have:

- Modular CrewAI agents for each task
- Vector-trained on your own company data
- Local LLM (LLaMA via Ollama)
- Web UI (chat)
- Ready for channel automation via APIs

Full above code together to apply it main.py :-

```
# main.py - Multi-Agent AI Assistant using CrewAI, LangChain, FAISS, and
Ollama

# 1. Imports and Installation
# pip install crewai langchain faiss-cpu pdfplumber pandas openpyxl gradio

from crewai import Agent, Task, Crew
from langchain.vectorstores import FAISS
from langchain.embeddings import OllamaEmbeddings
from langchain.document_loaders import PyPDFLoader, CSVLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.llms import Ollama
import gradio as gr

# 2. Load and Embed Company Data
embedding_model = OllamaEmbeddings(model="llama3")

def load_and_embed_docs(file_paths):
    docs = []
    for path in file_paths:
        if path.endswith(".pdf"):
            loader = PyPDFLoader(path)
        elif path.endswith(".csv"):
            loader = CSVLoader(file_path=path)
        loaded = loader.load()
        splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=50)
```

```
        docs.extend(splitter.split_documents(loaded))
    db = FAISS.from_documents(docs, embedding_model)
    db.save_local("company_knowledge_faiss")
    return db

# Load the vectorstore (after saving once)
vectorstore = FAISS.load_local("company_knowledge_faiss", embedding_model,
allow_dangerous_deserialization=True)
retriever = vectorstore.as_retriever()

# 3. Setup Agents
llm = Ollama(model="llama3")

faq_agent = Agent(
    role="FAQ Bot",
    goal="Answer common questions based on company policy",
    backstory="Knows the full company handbook and website FAQs.",
    tools=[retriever],
    llm=llm
)

scheduler_agent = Agent(
    role="Scheduler",
    goal="Help users schedule meetings or calls with our team.",
    backstory="Has access to the team's calendar availability.",
    llm=llm
)

sales_agent = Agent(
    role="Sales Assistant",
    goal="Answer product-related queries and convince leads.",
    backstory="Trained on product descriptions and sales brochures.",
    tools=[retriever],
    llm=llm
)

support_agent = Agent(
    role="Support Helper",
    goal="Assist customers with common troubleshooting steps.",
    backstory="Knows common issues from support documentation.",
    tools=[retriever],
    llm=llm
)

# 4. Define Tasks
faq_task = Task("Respond to customer FAQs: {query}", agent=faq_agent)
schedule_task = Task("Schedule a meeting based on: {query}",
agent=scheduler_agent)
```

```
sales_task = Task("Provide product/sales info for: {query}",
agent=sales_agent)
support_task = Task("Help solve this issue: {query}", agent=support_agent)

# 5. Setup Crew
crew = Crew(
    agents=[faq_agent, scheduler_agent, sales_agent, support_agent],
    tasks=[faq_task, schedule_task, sales_task, support_task],
    verbose=True
)

# 6. Query Router
def route_query(query):
    lower = query.lower()
    if any(word in lower for word in ["faq", "policy", "how"]):
        return crew.run(faq_task.format(query=query))
    elif any(word in lower for word in ["meeting", "schedule",
"appointment"]):
        return crew.run(schedule_task.format(query=query))
    elif any(word in lower for word in ["price", "buy", "offer", "product"]):
        return crew.run(sales_task.format(query=query))
    elif any(word in lower for word in ["error", "issue", "not working",
"problem"]):
        return crew.run(support_task.format(query=query))
    else:
        return "Sorry, I couldn't classify your query."

# 7. Gradio Web UI
def chat(query):
    return route_query(query)

gr.Interface(fn=chat, inputs="text", outputs="text", title="Company AI
Assistant").launch()
```

You can run this script locally after:

- Setting up your PDFs/CSVs in the same folder.
- Installing the required packages and Ollama with LLaMA3.

Before running run ollama:-

Ollama run llama3

🔌 7. Connect Channels to Agents — Step-by-Step Guide

Channel	Integration Tool	Method
Chat (Web)	Gradio / Streamlit	Direct Python call
WhatsApp	n8n.io + WhatsApp Cloud API	Webhook → Python Agent API
Email	IMAP/SMTP or Gmail API	Read → classify → respond
Facebook	Meta Graph API	DM webhook → Python API

✅ Now Your full AI assistant code **with all channel integrations** (Chat, WhatsApp, Email, Facebook) is ready!

✅ Recap

You now have:

- 🧠 Agents trained on your docs
- 🕹️ Web + WhatsApp + Email + Facebook integration
- 🔄 All routed to the correct CrewAI agent

```
# final-code.py - Multi-Agent AI Assistant using CrewAI, LangChain, FAISS, and
Ollama

# 1. Imports and Installation
# pip install crewai langchain faiss-cpu pdfplumber pandas openpyxl gradio
flask requests

from crewai import Agent, Task, Crew
from langchain.vectorstores import FAISS
from langchain.embeddings import OllamaEmbeddings
from langchain.document_loaders import PyPDFLoader, CSVLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.llms import Ollama
import gradio as gr
from flask import Flask, request, jsonify
import imaplib, smtplib, email
from email.mime.text import MIMEText
import requests

# 2. Load and Embed Company Data
embedding_model = OllamaEmbeddings(model="llama3")
```

```
def load_and_embed_docs(file_paths):
    docs = []
    for path in file_paths:
        if path.endswith(".pdf"):
            loader = PyPDFLoader(path)
        elif path.endswith(".csv"):
            loader = CSVLoader(file_path=path)
        loaded = loader.load()
        splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=50)
        docs.extend(splitter.split_documents(loaded))
    db = FAISS.from_documents(docs, embedding_model)
    db.save_local("company_knowledge_faiss")
    return db

# Load the vectorstore (after saving once)
vectorstore = FAISS.load_local("company_knowledge_faiss", embedding_model,
allow_dangerous_deserialization=True)
retriever = vectorstore.as_retriever()

# 3. Setup Agents
llm = Ollama(model="llama3")

faq_agent = Agent(role="FAQ Bot", goal="Answer FAQs", backstory="Knows the
handbook.", tools=[retriever], llm=llm)
scheduler_agent = Agent(role="Scheduler", goal="Book meetings",
backstory="Knows calendar.", llm=llm)
sales_agent = Agent(role="Sales Assistant", goal="Convert leads",
backstory="Trained on product info.", tools=[retriever], llm=llm)
support_agent = Agent(role="Support Helper", goal="Troubleshoot issues",
backstory="Knows common issues.", tools=[retriever], llm=llm)

# 4. Define Tasks
faq_task = Task("Respond to FAQs: {query}", agent=faq_agent)
schedule_task = Task("Schedule based on: {query}", agent=scheduler_agent)
sales_task = Task("Sales query: {query}", agent=sales_agent)
support_task = Task("Support issue: {query}", agent=support_agent)

# 5. Setup Crew
crew = Crew(agents=[faq_agent, scheduler_agent, sales_agent, support_agent],
tasks=[faq_task, schedule_task, sales_task, support_task], verbose=True)

# 6. Query Router
def route_query(query):
    lower = query.lower()
    if any(word in lower for word in ["faq", "policy", "how"]):
        return crew.run(faq_task.format(query=query))
```

```
    elif any(word in lower for word in ["meeting", "schedule",
"appointment"]):
        return crew.run(schedule_task.format(query=query))
    elif any(word in lower for word in ["price", "buy", "offer", "product"]):
        return crew.run(sales_task.format(query=query))
    elif any(word in lower for word in ["error", "issue", "not working",
"problem"]):
        return crew.run(support_task.format(query=query))
    else:
        return "Sorry, I couldn't classify your query."

# 7. Gradio Web UI
def chat(query):
    return route_query(query)

# 8. Flask API for Webhooks (WhatsApp, Facebook, etc.)
app = Flask(__name__)

@app.route("/chatbot", methods=["POST"])
def chatbot():
    data = request.json
    message = data.get("message")
    reply = route_query(message)
    return jsonify({"reply": reply})

@app.route("/webhook", methods=["POST"])
def fb_webhook():
    data = request.json
    user_message = data["entry"][0]["messaging"][0]["message"]["text"]
    sender_id = data["entry"][0]["messaging"][0]["sender"]["id"]
    reply = route_query(user_message)
    send_fb_message(sender_id, reply)
    return "ok", 200

def send_fb_message(recipient_id, message):
    PAGE_ACCESS_TOKEN = "your_page_token"
    url =
f"https://graph.facebook.com/v18.0/me/messages?access_token={PAGE_ACCESS_TOKEN
}"
    payload = {"recipient": {"id": recipient_id}, "message": {"text":
message}}
    requests.post(url, json=payload)

# 9. Email Listener (IMAP) + Auto-responder
EMAIL = "your-email@gmail.com"
PASSWORD = "your-app-password"

def read_email():
```

```
    mail = imaplib.IMAP4_SSL("imap.gmail.com")
    mail.login(EMAIL, PASSWORD)
    mail.select("inbox")
    status, messages = mail.search(None, "(UNSEEN)")
    mail_ids = messages[0].split()

    for num in mail_ids:
        status, data = mail.fetch(num, "(RFC822)")
        msg = email.message_from_bytes(data[0][1])
        sender = msg["from"]
        content = msg.get_payload(decode=True).decode()
        reply = route_query(content)
        send_email(sender, reply)

def send_email(to, body):
    msg = MIMEText(body)
    msg["Subject"] = "RE: Your Query"
    msg["From"] = EMAIL
    msg["To"] = to
    with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server:
        server.login(EMAIL, PASSWORD)
        server.sendmail(EMAIL, to, msg.as_string())

# 10. Launch both Gradio and Flask
if __name__ == "__main__":
    import threading
    threading.Thread(target=lambda: gr.Interface(fn=chat, inputs="text",
outputs="text", title="Company AI Assistant").launch(share=True)).start()
    app.run(port=5000)
```

This enhanced script includes:

- ✅ CrewAI multi-agent system
- ✅ LLaMA via Ollama
- ✅ FAISS with your company data
- ✅ Web chat UI (Gradio)
- ✅ Flask API for WhatsApp + Facebook
- ✅ Email reading + auto-response