Chosen Scenario: Business Development Automation

The goal is to design an AI agent that automates business development tasks, such as generating personalized outreach messages, scheduling follow-ups, and tracking responses. Below is the proposed design.

Part A: Agent Prompt Design

Agent Design

- Agent Capabilities:
    - Data Gathering Agent: Collects customer information (e.g., industry, pain points, goals) from CRM systems, websites, and other sources.
    - RAG (Retrieval-Augmented Generation) System Component: Provides tailored product or solution recommendations aligned with the customer's pain points by leveraging the company's internal knowledge base. This knowledge base includes
        - Product catalog & documentation
        - Early product offerings
        - Case studies and success stories
        - Sales and marketing collateral
        - Competitor benchmarking data
    - Lead Scoring Agent: Analyzes customer data and interaction history to assign a score that represents the likelihood of the lead converting. This score is based on metrics such as:
        - Engagement levels (e.g., email opens, click rates).
        - Customer fit (e.g., alignment with product/service offering).
        - Historical data from similar leads.
        - Recency and frequency of customer interactions.
    - Outreach Drafting Agent: Generates personalized outreach emails/messages based on customer data.
    - QA Agent: Reviews drafted content to ensure:
        - Accuracy: Email content aligns with customer data.
        - Grammar, Vocabulary, and Tone: Emails are professional, error-free, and appropriately engaging.
        - Guardrails Against Bias: Verifies that content avoids socially or ethically biased language, stereotypes, or inappropriate assumptions.
    - Scheduling Agent: Suggests and books follow up meetings or calls.
    - Response Tracker Agent: Monitors responses and updates customer interaction history by categorizing and summarizing replies.
    - Human-in-the-Loop: Enables review/approval for:
        - Outreach drafts
        - High-priority customer responses
        - Lead Scoring overrides or adjustments for special cases

Example Prompts

Prompt 1: **Outreach Drafting Agent**

Task: Generate a personalized outreach message for a customer.

```
f"""You are an AI agent tasked with drafting a professional and
personalized outreach email. The email should address
the customer's pain points and suggest our solution.

Customer Details:
    - Name: {customer_data['name']}
    - Industry: {customer_data['industry']}
    - Pain Points: {customer_data['pain_points']}
    - Goals: {customer_data['goals']}
    - Product/Service: {customer_data['product']}

Requirements:
    - Subject: Brief and relevant to the customer's industry.
    - Email Body: Address the customer's challenges and introduce the
                  product/service as a solution.
    - Call to Action: Schedule a call or request a reply.

Output the email in the following format:
    Subject: [Your Subject]
    Body: [Your Email Body]"""
```

Prompt 2: **Response Tracker Agent**

Task: Track customer responses and update status.

```
f"""You are an AI response tracker. Your role is to categorize and
summarize email responses from customers.

Use the following inputs:
    - Response Text: "response_text}}"
    - Categories: Interested, Not Interested, Need More Info, Follow-Up
Required.

Output:
    - Category of response.
    - Summary of the customer's reply in 2-3 sentences.
    - Suggested next step (e.g., send more information, escalate to
sales team, or no action needed).
"""
```

Testing Procedures

1) Agent Output Validation:

   • Test outputs against sample inputs to ensure:

   • Relevance to customer details.

   • Adherence to format and tone guidelines.

   • Validate the response categorization accuracy (Response Tracker).

2) Human Review Checkpoints:

   • Ensure QA Agent checks grammar, tone, and alignment with brand messaging.

   • Allow humans to approve/disapprove content at key steps.

3) Iteration

   • Improve prompts based on feedback from human reviewers and test runs.

Information Structuring

To optimize performance:

- Maintain a structured database for customer data using fields like: Name, Industry, Pain Points, Goals, Preferred Contact Time.
- Use embeddings for matching customer pain points with relevant solutions.
- Employ API integrations (e.g., OpenAI API) to enhance context understanding for agents.

Part B: Process Flow Design

This section focuses on creating a seamless workflow for the automation process by detailing how agents interact with each other, incorporating human-in-the-loop checkpoints, and optimizing efficiency.

The multi-agent business development automation system is designed to be modular and scalable. It starts by identifying the customer, analyzing their pain points, and recommending relevant products or services. This data is then used to generate personalized outreach emails. These emails undergo a validation process, and if any issues are detected, the emails are sent back to be re-crafted. The scheduling and tracking of customer responses are then handled automatically, with feedback loops incorporated to enhance performance.

- **Data Flow**
  - Data Gathering Agent: The Data Gathering Agent pulls data from CRM or marketing platforms, such as name, contact information, and previous interactions. This data is standardized and structured using JSON schemas, ensuring that information like customer preferences, pain points, and historical data are captured in a consistent format. Once the customer data is collected, it's passed on to the RAG System for further context enrichment.
  - RAG System for Pain Point Solution Proposal: The RAG System helps identify products and solutions) to be offered by the company for the customer to address its pain points. It achieves this by asking the following discrete questions based on the customer's pain points:
    - Which product can be applied to this pain point?
    - How can the product be leveraged to address the pain point?
    - Are there any testimonials about the product being applied to this specific pain point?

    Based on the customer's responses, the RAG system assigns a red, amber, or green label to each pain point, indicating the urgency and alignment with available solutions. The outcomes are enriched with relevant products and services that match the customer's pain points.

- Lead Scoring Agent: The Lead Scoring Agent evaluates customer data and interaction history to generate a score that represents the likelihood of conversion. The score is based on:
    - Engagement levels (e.g., email opens, click-through rates).
    - Customer fit (alignment with the product or service offering).
    - Historical data from similar leads.
    - Recency and frequency of customer interactions. The lead score output will flag the conversion likelihood, influencing the urgency of the response and the need for human intervention.
- Outreach Drafting Agent: The Outreach Drafting Agent creates a highly personalized outreach email. The email is designed to:
    - Address the customer's specific pain points.
    - Introduce the most relevant product/service.
    - Provide a clear call-to-action (CTA) to encourage engagement.
    - The email draft is structured using a JSON schema, ensuring all necessary elements are included, such as greetings, product descriptions, and contact information.
- QA Agent (with Loops): The QA Agent reviews the email content to ensure it meets quality standards, such as:
    - Grammar and spelling checks.
    - Brand tone and messaging consistency.
    - Compliance with privacy policies and legal guidelines.
    - Guard rails ensuring email is free of social and racial biases.
    - Ground the email against common SPAM email standards

  If the email passes the review, it is forwarded to the Email Scheduling Agent. If the QA Agent identifies any issues (e.g., spelling errors or inconsistent tone), the email is flagged as "failed" and sent back with comments (reasons to mark the email failed) to the Content Generation Agent for revisions. This loop ensures that emails meet high standards before being sent.

- Scheduling Agent: Once the email is validated, The Scheduling Agent handles the responsibilities of both scheduling follow-up meetings or calls as well as outreach email sending schedules. This agent is designed to:
    - Suggest and book follow-up meetings or calls based on customer interactions and engagement patterns.
    - Schedule emails based on customer preferences, engagement data (e.g., previous email opens and clicks), and time zones to maximize open and click rates.

  The email is then sent at the scheduled time.

- Response Tracker Agent: The Response Tracking Agent monitors how customers interact with the email (e.g., whether they open it, click on links, or reply). Interaction data is captured in a structured format (JSON) for easy analysis and future decision-making. The agent tracks key metrics like open rates, click-through rates, and customer responses, which are forwarded to the Insights and Feedback Agent.
- Insights and Feedback Agent: The Insights and Feedback Agent processes the data gathered from customer interactions to identify trends and patterns. The agent helps optimize future email campaigns by analyzing response rates, customer sentiment, and

engagement. If a significant issue or opportunity is detected (e.g., a complaint or a request for more information), the system triggers a Human-in-the-Loop intervention for manual handling.

- **Human-in-the-Loop Triggers**
  - Trigger 1: The Customer Identification Agent fails to retrieve sufficient customer data, requiring human intervention to correct the information.
  - Trigger 2: If the RAG System identifies a pain point that doesn't align with any product, the system escalates for manual review to identify new solutions.
  - Trigger 3: The QA Agent flags the email as failed after multiple validation attempts, and human intervention is needed to approve the draft.
  - Trigger 4: A critical customer issue arises that requires direct human communication, such as complaints or urgent inquiries.
  - Trigger 5: The Insights and Feedback Agent detects an anomaly, such as an unusually high number of opt-outs or responses, prompting a manual review.
- **Agentic Pipelines**

Here's the adapted Agentic Pipeline for the updated agent design:

  - Data Gathering Agent:
    - ***Input**:* Customer name, company website, CRM records, previous interactions, historical data.
    - ***Output***: Structured JSON with fields like name, industry, pain_points, goals, and other relevant customer data.
  - RAG System for Pain Point Solution Proposal:
    - ***Input:*** Structured JSON from Data Gathering Agent.
    - ***Output:*** Solution proposals, including matching products/services and context about how they align with the pain points.
  - Lead Scoring Agent:
    - ***Input:*** Enriched customer data from the RAG System.
    - ***Output:*** A lead score indicating the likelihood of conversion, and a flag marking whether human intervention is needed (e.g., if the score is high or urgent).
  - Outreach Drafting Agent:
    - ***Input:*** Enriched data (products/services for pain points) and lead score from the Lead Scoring Agent.
    - **Output:** A draft outreach email.
  - QA Agent (with Loops):
    - ***Input:*** Outreach email draft from the Outreach Drafting Agent.
    - ***Output:*** Approved email for sending or feedback for revisions.
  - Scheduling Agent:
    - ***Input:*** Approved email from the QA Agent.
    - ***Output:*** Scheduled email or meeting invitation.
  - Response Tracker Agent:
    - ***Input:*** Customer's interaction with the email (e.g., opens, clicks, replies).
    - ***Output:*** Categorized response data and next-step recommendations.
  - Insights and Feedback Agent:
    - ***Input:*** Response tracking data from the Response Tracker Agent.
    - ***Output:*** Optimized recommendations for future email campaigns or flags for Human-in-the-Loop intervention.

The whole workflow in the Business Development Automation agent can be summarized using the Figure below.

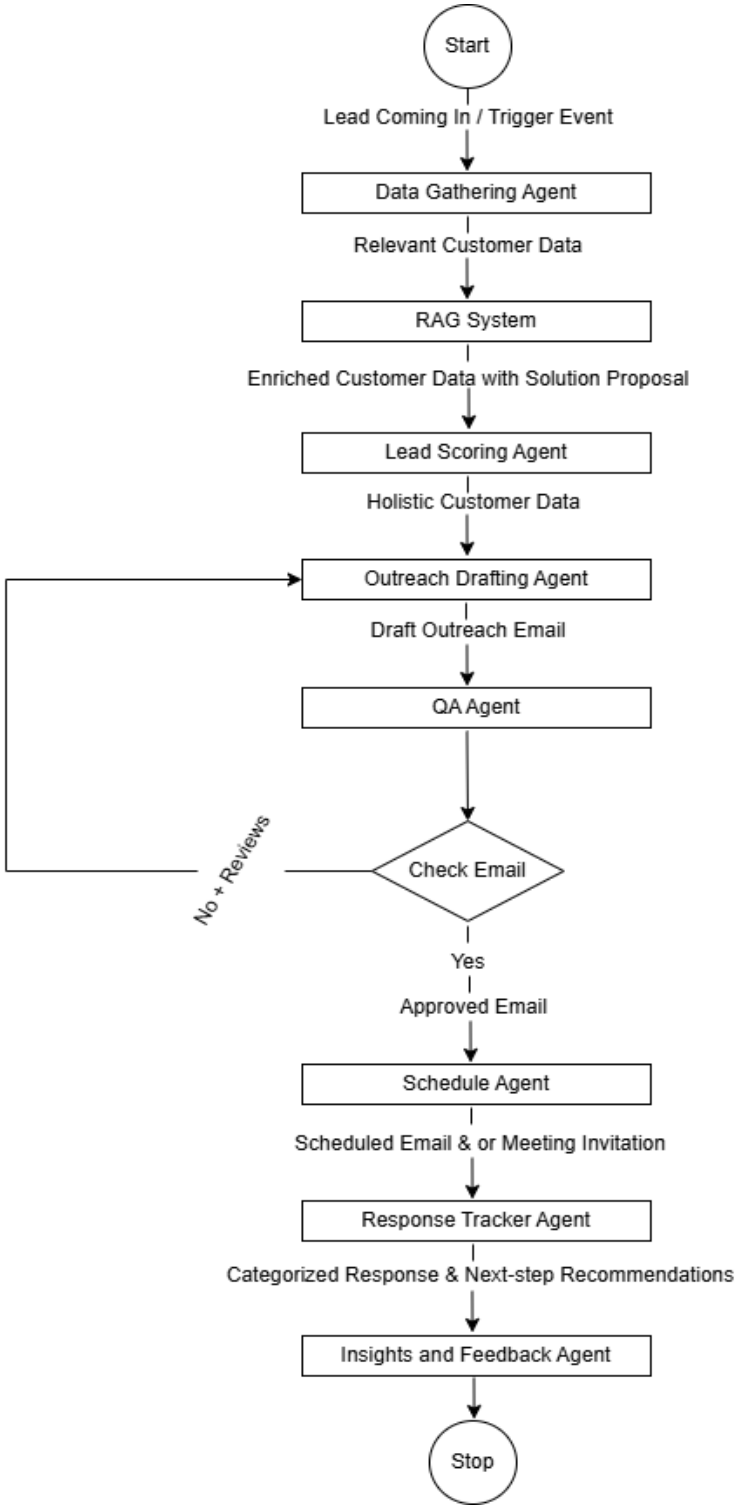

Fig. 1, Agent Workflow Pipeline

Part C: Integration of Use Cases for Inspiration

- **Review & Analysis of Provided Use Cases**
  - **Document Research Assistant for Blog Creation with NVIDIA NIM microservices**
    The system is structured into two primary phases: setup and querying.
    - Setup Phase:
      - Document Parsing: Utilizing LlamaParse, the system converts complex document formats (e.g., PDFs, Word documents, PowerPoints, spreadsheets) into Markdown text, facilitating easier processing by Large Language Models (LLMs).
      - Embedding and Indexing: The parsed text is transformed into vector embeddings using NVIDIA's NeMo Retriever embedding NIM microservice. These embeddings are then stored in a vector database, enabling efficient semantic search and retrieval.
    - Querying Phase:
      - Outline Generation: An agent creates an initial outline for the blog post based on the user's query.
      - Question Formulation: A subsequent agent develops specific questions aimed at gathering detailed information to support the outline.
      - Information Retrieval: Another agent queries the RAG database using the formulated questions, retrieves relevant context, and generates corresponding answers.
      - Content Creation: With the outline, gathered information, and original query, an agent drafts the blog post.
      - Quality Assessment: A final agent reviews the draft to ensure it comprehensively addresses the initial query. If the content is deemed insufficient, the system iteratively refines the blog post by generating additional questions and incorporating new information, with a maximum of three iterations.
  - **Developer Project Spotlight: Building an AI Sales Assistant with LlamaIndex and NVIDIA NIM**

    The AI Sales Assistant is engineered to assist sales teams by:

    - Streamlining access to customer insights.
    - Automating lead qualification and personalized outreach.
    - Improving sales efficiency and reducing manual effort.

    The system focuses on integrating advanced LLM capabilities with structured knowledge and semantic search to create a dynamic, autonomous agent pipeline. The proposed system comprises the following components and workflows:

    - Multi-Agent Workflow: The system adopts a modular, multi-agent workflow, where each agent specializes in a discrete task to fulfill the sales workflow objectives.
    - RAG-Powered Information Retrieval

- **Case Study: Netchex: More Efficient HR Operations with LlamaIndex Powered AskHR + Netchex AI**
  Netchex's clients, primarily small to medium-sized businesses with limited HR staff, faced challenges such as:
  - Overwhelmed HR teams due to a high volume of employee queries, leading to delayed responses and decreased productivity.
  - Difficulty in efficiently utilizing comprehensive documentation (e.g., employee handbooks, benefit plan summaries) to provide prompt answers to staff inquiries.
  - Existing solutions lacking user-friendly interfaces, making it hard for employees to find information independently.

  To tackle these challenges, Netchex developed AskHR + Netchex AI, leveraging several key features of LlamaIndex:
  - Document Ingestion and Processing: Utilized LlamaIndex's ingestion pipeline to process various document types, including MS Office files and PDFs. Employed chunking strategies on large HR documents to manage and retrieve information efficiently. Integrated with Azure OpenAI to generate embeddings, enhancing the system's understanding of the documents.
  - Retrieval Augmented Generation (RAG): Implemented advanced RAG pipelines to ensure AI responses were grounded in the company's specific HR documents. Customized retriever classes to fine-tune the relevance of retrieved information. Used a flexible query engine architecture to process employee questions and synthesize accurate, context-aware responses. Employed a citation feature to provide direct links to source documents and page numbers in AI responses, enhancing transparency and trust.
  - Scalability and Semantic Routing: Leveraged LlamaIndex's scalable architecture to handle varying loads across Netchex's diverse client base. Implemented semantic routing to efficiently direct queries to the most appropriate knowledge bases or response generation pipelines.

- **How Use Cases Shaped the Design**

To create a robust and practical solution for the Business Development Automation Scenario, real-world workflows were analyzed, and insights from provided use cases, such as document research assistants and AI sales tools, were incorporated. These include:

- Modular Agent Design: Inspired by the multi-agent architecture of AI Sales and Netchex AI, each agent in the pipeline has been designed with a specific responsibility, minimizing bottlenecks and improving scalability.
- Retrieval-Augmented Generation (RAG): RAG concepts from Document Research Assistants influenced the integration of customer data retrieval in outreach drafting. Agents rely on retrieval-based insights to craft targeted content.
- Human-in-the-Loop Triggers: Inspired by compliance-focused applications in Netchex AI, human approval checkpoints were incorporated to review drafts and critical responses.
- Scalability Considerations: Real-world examples emphasized the importance of scaling workflows to handle increased customer loads, which influenced the following:
  - Structuring JSON schemas for data handling.
  - Ensuring API endpoints are scalable for seamless agent communication.

Part D: Practical Coding Task

Below are Python scripts and JSON schemas for the agents involved in the Business Development Automation scenario.

- **Data Gathering Agent**

  Python Script

```python
import requests

def gather_customer_data(customer_name, company_website):
    # Simulating data retrieval from CRM
    crm_data = {
        "name": customer_name,
        "industry": "Technology",
        "pain_points": "Difficulty scaling infrastructure",
        "goals": "Adopt cloud-based solutions",
        "preferred_contact_time": "Morning",
        "contact_email": "example@company.com"
    }

    # Simulating scraping website information
    website_info = f"Website overview for {company_website}"

    # Combine CRM data and website info
    customer_data = {
        "name": crm_data["name"],
        "industry": crm_data["industry"],
        "pain_points": crm_data["pain_points"],
        "goals": crm_data["goals"],
        "preferred_contact_time": crm_data["preferred_contact_time"],
        "contact_email": crm_data["contact_email"],
        "website_info": website_info
    }

    return customer_data

customer_name = "TechCorp Inc."
company_website = "www.techcorp.com"
data = gather_customer_data(customer_name, company_website)
print(data)
```

  JSON Schema

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "CustomerData",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "Name of the customer"
    },
    "industry": {
      "type": "string",
```

```
      "description": "Industry the customer operates in"
    },
    "pain_points": {
      "type": "string",
      "description": "Key challenges faced by the customer"
    },
    "goals": {
      "type": "string",
      "description": "Goals or objectives the customer wants to
achieve"
    },
    "product": {
      "type": "string",
      "description": "The product/service being offered"
    },
    "contact_email": {
      "type": "string",
      "format": "email",
      "description": "Customer's email address"
    },
    "preferred_contact_time": {
      "type": "string",
      "description": "Customer's preferred time for contact (e.g.,
morning, afternoon)"
    }
  },
  "required": [
    "name",
    "industry",
    "pain_points",
    "goals",
    "product",
    "contact_email"]}
```

- **RAG System**

Python Script

```
import openai

openai.api_key = 'api-key'

def retrieve_and_generate_insights(customer_data):
    prompt = f"""
    You are an AI agent tasked with retrieving relevant information
    to match customer pain points with solutions. Use the following
    customer data to craft insights:

    Customer Details:
    - Name: {customer_data['name']}
    - Industry: {customer_data['industry']}
    - Pain Points: {customer_data['pain_points']}
    - Goals: {customer_data['goals']}
    - Product/Service: {customer_data['product']}
```

```
    Provide a solution that specifically addresses the customer's
challenges.
    """

    response = openai.Completion.create(
        engine="text-davinci-004",
        prompt=prompt,
        max_tokens=300,
        temperature=0.7
    )
    return response.choices[0].text.strip()

customer_data = {
    "name": "Jane Doe",
    "industry": "Healthcare",
    "pain_points": "High administrative costs and slow patient
onboarding.",
    "goals": "Reduce overhead and improve operational efficiency.",
    "product": "AI-powered workflow automation tools"
}

insights = retrieve_and_generate_insights(customer_data)
print(insights)
```

JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "InsightsData",
  "type": "object",
  "properties": {
    "customer_name": {
      "type": "string",
      "description": "Name of the customer"
    },
    "pain_points": {
      "type": "string",
      "description": "Challenges faced by the customer"
    },
    "solution": {
      "type": "string",
      "description": "Solution generated by the RAG system"
    }
  },
  "required": ["customer_name", "pain_points", "solution"]
}
```

- **Lead Scoring Agent**

Python Script

```
def calculate_lead_score(customer_data):
    engagement = 8 # Simulating engagement score (out of 10)
    fit = 9         # Simulating customer fit score (out of 10)
    interaction_history = 7 #Simulating interaction recency (out of 10)
```

```
    # Calculating lead score
    lead_score = (engagement * 0.4) + (fit * 0.4) +
    (interaction_history * 0.2)
    category = "Hot Lead" if lead_score >= 8 else
    "Warm Lead" if lead_score >= 5 else "Cold Lead"

    return {
        "lead_score": round(lead_score, 2),
        "category": category,
        "notes": "Customer is highly engaged and a great fit for our
services."
    }

customer_data = {"name": "John Doe", "industry": "Finance"}
lead_score_info = calculate_lead_score(customer_data)
print(lead_score_info)
```

JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "LeadScore",
  "type": "object",
  "properties": {
    "lead_score": {
      "type": "number",
      "description": "Calculated lead score"
    },
    "category": {
      "type": "string",
      "enum": ["Hot Lead", "Warm Lead", "Cold Lead"],
      "description": "Lead categorization based on score"
    },
    "notes": {
      "type": "string",
      "description": "Additional notes or observations about the lead"
    }
  },
  "required": ["lead_score", "category", "notes"]
}
```

- **Outreach Drafting Agent**

Python Script

```
import openai

openai.api_key = 'api-key'

def generate_outreach_email(customer_data):
    prompt = f"""
    You are an AI agent tasked with drafting a professional and
    personalized outreach email. The email should address
    the customer's pain points and suggest our solution.

    Customer Details:
```

```
    - Name: {customer_data['name']}
    - Industry: {customer_data['industry']}
    - Pain Points: {customer_data['pain_points']}
    - Goals: {customer_data['goals']}
    - Product/Service: {customer_data['product']}

    Requirements:
    - Subject: Brief and relevant to the customer's industry.
    - Email Body: Address the customer's challenges and introduce the
                  product/service as a solution.
    - Call to Action: Schedule a call or request a reply.

    Output the email in the following format:
    Subject: [Your Subject]
    Body: [Your Email Body]
    """

    response = openai.Completion.create(
        engine="text-davinci-004",
        prompt=prompt,
        max_tokens=300,
        temperature=0.7
    )
    return response.choices[0].text.strip()

customer_data = {
    "name": "Jane Doe",
    "industry": "Healthcare",
    "pain_points": "High administrative costs and slow patient
onboarding.",
    "goals": "Reduce overhead and improve operational efficiency.",
    "product": "AI-powered workflow automation tools"
}

email = generate_outreach_email(customer_data)
print(email)
```

JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "OutreachEmail",
  "type": "object",
  "properties": {
    "subject": {
      "type": "string",
      "description": "The subject line of the outreach email"
    },
    "body": {
      "type": "string",
      "description": "The main content of the outreach email"
    },
    "call_to_action": {
      "type": "string",
      "description": "Call-to-action included in the email (e.g.,
schedule a call, reply to the email)"
    },
```

```
    "recipient_name": {
      "type": "string",
      "description": "The name of the recipient"
    },
    "recipient_email": {
      "type": "string",
      "format": "email",
      "description": "The recipient's email address"
    },
    "product_or_service": {
      "type": "string",
      "description": "The product or service being introduced"
    },
    "generated_timestamp": {
      "type": "string",
      "format": "date-time",
      "description": "Timestamp when the outreach email was generated"
    }
  },
  "required": ["subject", "body", "call_to_action", "recipient_name",
"recipient_email", "product_or_service", "generated_timestamp"]
}
```

- **QA Agent**

Python Script

```
from textblob import TextBlob

def qa_review(draft_email):
    analysis = TextBlob(draft_email)
    sentiment = analysis.sentiment.polarity
    review = {
        "grammar_check": "Pass"
        if len(analysis.correct()) == len(draft_email) else "Fail",
            "tone_check": "Positive" if sentiment > 0 else
            "Neutral/Negative",
        "status": "Approved" if sentiment > 0.2 else "Requires Edits"
            }

    return review

draft_email = """Hi John,
We understand your pain points in scaling your infrastructure.
Our cloud-based solutions are tailored to meet your needs.
Let's schedule a call to discuss this further.

Best,
TechCorp Team
"""
qa_result = qa_review(draft_email)
print(qa_result)
```

JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "QAReview",
  "type": "object",
  "properties": {
    "grammar_check": {
      "type": "string",
      "enum": ["Pass", "Fail"],
      "description": "Result of grammar check"
    },
    "tone_check": {
      "type": "string",
      "enum": ["Positive", "Neutral/Negative"],
      "description": "Evaluation of the email's tone"
    },
    "status": {
      "type": "string",
      "enum": ["Approved", "Requires Edits"],
      "description": "Final QA review status"
    },
    "review": {
      "type": "string",
      "description": "Review of the Draft Email"
    },
  },
  "required": ["grammar_check", "tone_check", "status"]
}
```

- **Schedule Agent**

Python Script

```
import datetime

def schedule_meeting(customer_name, preferred_time):
    current_time = datetime.datetime.now()
    meeting_time = current_time + datetime.timedelta(days=1)

    if preferred_time == "Morning":
        meeting_time = meeting_time.replace(hour=9, minute=0)
    elif preferred_time == "Afternoon":
        meeting_time = meeting_time.replace(hour=14, minute=0)
    elif preferred_time == "Evening":
        meeting_time = meeting_time.replace(hour=18, minute=0)

    meeting_details = {
        "customer_name": customer_name,
        "meeting_time": meeting_time.strftime("%Y-%m-%d %H:%M:%S"),
        "meeting_link":
      f"https://meetings.company.com/{customer_name.replace(' ',
      '_').lower()}"
    }

    return meeting_details
```

```
meeting = schedule_meeting("John Doe", "Morning")
print(meeting)
```

JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "MeetingDetails",
  "type": "object",
  "properties": {
    "customer_name": {
      "type": "string",
      "description": "Name of the customer"
    },
    "meeting_time": {
      "type": "string",
      "format": "date-time",
      "description": "Scheduled time for the meeting"
    },
    "meeting_link": {
      "type": "string",
      "description": "Link for the scheduled meeting"
    }
  },
  "required": ["customer_name", "meeting_time", "meeting_link"]
}
```

- **Response Tracker Agent**

Python Script

```
import openai
from datetime import datetime

openai.api_key = "api-key"

def categorize_response(response_text):
    prompt = f"""
    You are an AI agent tasked with analyzing customer email responses.
    Categorize the response into one of the following categories:
    - Interested
    - Not Interested
    - Need More Info
    - Follow-Up Required

    Then, provide a brief summary of the response and recommend the
next step.

    Response: {response_text}

    Output Format:
    Category: [Category]
    Summary: [Brief summary of the response]
    Next Step: [Recommended next step]
    """
```

```
    response = openai.Completion.create(
        engine="text-davinci-004",
        prompt=prompt,
        max_tokens=150,
        temperature=0.6
    )

    output_lines = response.choices[0].text.strip().split("\n")
    category = output_lines[0].split(":")[1].strip()
    summary = output_lines[1].split(":")[1].strip()
    next_step = output_lines[2].split(":")[1].strip()

    response_analysis = {
        "response_text": response_text,
        "category": category,
        "summary": summary,
        "next_step": next_step,
        "timestamp": datetime.now().isoformat()
    }

    return response_analysis

response_text = "I'm interested in learning more about how your
solution can reduce our costs."
analysis = categorize_response(response_text)
print(analysis)
```

JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "ResponseData",
  "type": "object",
  "properties": {
    "response_text": {
      "type": "string",
      "description": "The actual response from the customer"
    },
    "category": {
      "type": "string",
      "enum": ["Interested", "Not Interested", "Need More Info",
"Follow-Up Required"],
      "description": "Categorization of the customer's response"
    },
    "summary": {
      "type": "string",
      "description": "A brief summary of the customer's response"
    },
    "next_step": {
      "type": "string",
      "description": "Recommended action based on the response"
    },
    "timestamp": {
      "type": "string",
      "format": "date-time",
      "description": "Timestamp of when the response was analyzed"
```

```
    }
  },
  "required": ["response_text", "category", "summary", "next_step",
"timestamp"]
}
```

- **Insights and Feedback Agent**
  Python Script

```
def analyze_campaign_performance(campaign_data):
    total_emails = campaign_data["total_emails_sent"]
    positive_responses = campaign_data["positive_responses"]
    response_rate = (positive_responses / total_emails) * 100
    feedback = "Excellent" if response_rate > 30 else "Good" if
response_rate > 15 else "Needs Improvement"

    return {
        "total_emails_sent": total_emails,
        "positive_responses": positive_responses,
        "response_rate": round(response_rate, 2),
        "feedback": feedback
    }

campaign_data = {"total_emails_sent": 100, "positive_responses": 35}
performance = analyze_campaign_performance(campaign_data)
print(performance)
```

JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "CampaignPerformance",
  "type": "object",
  "properties": {
    "total_emails_sent": {
      "type": "integer",
      "description": "Total number of emails sent during the campaign"
    },
    "positive_responses": {
      "type": "integer",
      "description": "Number of positive responses received"
    },
    "response_rate": {
      "type": "number",
      "description": "Response rate percentage"
    },
    "feedback": {
      "type": "string",
      "description": "Overall feedback based on response rate"
    }
  },
  "required": ["total_emails_sent", "positive_responses",
"response_rate", "feedback"]
}
```

NOTE: The final code of the Business Automation Agent is expected to be a lot different from what is given here due to real-scenario considerations, handling productions inconsistencies and usage of frameworks such as LlamaIndex and LangChain.