Agent that picks up your intent

Depending on your intent it routes you to an agent that can help you with your request.

Account management agent and product support agent

Account Management Agent --> Talk about the user, their account. Just understand the user's intent and route them to the right agent.

```
from swarms import Agent
import requests
import json
from swarms import BaseLLM, base_model_to_openai_function
from pydantic import BaseModel, Field


## Pydantic model for the tool schema
class HASSchema(BaseModel):
    name: str = Field(
        ...,
        title="Name",
        description="The name of the agent to send the task to.",
    )
    task: str = Field(
        ...,
        title="Task",
        description="The task to send to the agent.",
    )
```

```
swarm_schema = base_model_to_openai_function(
    HASSchema, output_str=True
)

ACCOUNT_MANAGEMENT_SYSTEM_PROMPT = """
```

You are an Account Management Agent. Your primary role is to engage with users regarding their accounts. Your main tasks include understanding the user's intent, addressing their immediate needs, and routing them to the appropriate agent for further assistance. Be simple and direct in your communication.

When a user contacts you, start by greeting them and asking how you can assist with their account. Listen carefully to their concerns, questions, or issues. If the user provides information that is specific to their account, acknowledge it and ask any necessary follow-up questions to clarify their needs. Ensure that you fully understand their intent before proceeding.

Once you have a clear understanding of the user's request or issue, determine the best course of action. If you can resolve the issue yourself, do so efficiently. If the issue requires specialized assistance, explain to the user that you will route them to the appropriate agent who can help further. Ensure the user feels heard and understood throughout the process.

Your ultimate goal is to provide a seamless and positive experience for the user by effectively managing their inquiries and directing them to the right resource for resolution. Always maintain a polite and professional tone, and ensure that the user feels supported and valued.

"""

PRODUCT_SUPPORT_QA_SYSTEM_PROMPT = """

You are a Product Support Agent.

Your primary role is to provide assistance to users who have questions or issues related to the product. Your main tasks include understanding the user's needs, providing accurate information, and resolving any problems they may encounter. Be clear and concise in your communication.

"""

class llama3Hosted(BaseLLM):

    """

    A class representing a hosted version of the Llama3 model.

    Args:

        model (str): The name or path of the Llama3 model to use.

        temperature (float): The temperature parameter for generating responses.

        max_tokens (int): The maximum number of tokens in the generated response.

        system_prompt (str): The system prompt to use for generating responses.

        *args: Variable length argument list.

        **kwargs: Arbitrary keyword arguments.

```
Attributes:
    model (str): The name or path of the Llama3 model.
    temperature (float): The temperature parameter for generating responses.
    max_tokens (int): The maximum number of tokens in the generated response.
    system_prompt (str): The system prompt for generating responses.

Methods:
    run(task, *args, **kwargs): Generates a response for the given task.

"""


def __init__(
    self,
    model: str = "meta-llama/Meta-Llama-3-8B-Instruct",
    temperature: float = 0.8,
    max_tokens: int = 4000,
    system_prompt: str = "You are a helpful assistant.",
    *args,
    **kwargs,
):
    super().__init__(*args, **kwargs)
    self.model = model
    self.temperature = temperature
    self.max_tokens = max_tokens
    self.system_prompt = system_prompt
```

```
def run(self, task: str, *args, **kwargs) -> str:
    """
    Generates a response for the given task.

    Args:
        task (str): The user's task or input.

    Returns:
        str: The generated response from the Llama3 model.

    """
    url = "http://34.204.8.31:30001/v1/chat/completions"

    payload = json.dumps(
        {
            "model": self.model,
            "messages": [
                {"role": "system", "content": self.system_prompt},
                {"role": "user", "content": task},
            ],
            "stop_token_ids": [128009, 128001],
            "temperature": self.temperature,
            "max_tokens": self.max_tokens,
        }
    )
```

```
    headers = {"Content-Type": "application/json"}

    response = requests.request(
        "POST", url, headers=headers, data=payload
    )

    response_json = response.json()
    assistant_message = response_json["choices"][0]["message"][
        "content"
    ]

    return assistant_message


def select_agent_and_send_task(name: str = None, task: str = None):
    """
    Select an agent and send a task to them.

    Args:
        name (str): The name of the agent to send the task to.
        task (str): The task to send to the agent.

    Returns:
        str: The response from the agent.
```

```
    """
    if name == "Product Support Agent":
        agent = Agent(
            agent_name="Product Support Agent",
            system_prompt=PRODUCT_SUPPORT_QA_SYSTEM_PROMPT,
            llm=llama3Hosted(),
            max_loops=2,
            autosave=True,
            dashboard=False,
            streaming_on=True,
            verbose=True,
            output_type=str,
            metadata_output_type="json",
            function_calling_format_type="OpenAI",
            function_calling_type="json",
        )
    else:
            return "Invalid agent name. Please select 'Account Management Agent' or 'Product Support Agent'."

    response = agent.run(task)

    return response


def parse_json_then_activate_agent(json_data: str):
```

```
    """
    Parse the JSON data and activate the appropriate agent.

    Args:
        json_data (str): The JSON data containing the agent name and task.

    Returns:
        str: The response from the agent.

    """
    try:
        data = json.loads(json_data)
        name = data.get("name")
        task = data.get("task")

        response = select_agent_and_send_task(name, task)

        return response
    except json.JSONDecodeError:
        return "Invalid JSON data."


agent = Agent(
    agent_name="Account Management Agent",
    system_prompt=ACCOUNT_MANAGEMENT_SYSTEM_PROMPT,
    # sop_list=[GLOSSARY_PROMPTS, FEW_SHORT_PROMPTS],
```

```
    # sop=list_base_models_json,
    llm=llama3Hosted(
        max_tokens=3000,
    ),
    max_loops="auto",
    interactive=True,
    autosave=True,
    dashboard=False,
    streaming_on=True,
    # interactive=True,
    # tools=[search_weather],  # or list of tools
    verbose=True,
    # Set the output type to the tool schema which is a BaseModel
    list_base_models=[HASSchema],
    output_type=str,  # or dict, or str
    metadata_output_type="json",
    # List of schemas that the agent can handle
    function_calling_format_type="OpenAI",
    function_calling_type="json",  # or soon yaml
)

# Run the agent to generate the person's information
generated_data = agent.run("I need help with my modem.")
parse_json_then_activate_agent(generated_data)
```

```
# Print the generated data

print(f"Generated data: {generated_data}")
```