```
import time
from typing import List, Any, Dict, Optional
import uuid
from pydantic import BaseModel, model_validator, Field

from swarms_cloud.schema.cog_vlm_schemas import (
    AgentChatCompletionResponse,
)
from swarms_cloud.schema.swarm_schema import SwarmAPISchema



# Define the input model using Pydantic
class AgentInput(BaseModel):
    id: str = uuid.uuid4().hex
    created_at: int = time.time()
    owned_by: Optional[str] = Field(None, description="The owner of the agent.")
    agent_name: str = "Swarm Agent"
    system_prompt: str = None
    agent_description: str = None
    model_name: str = "OpenAIChat"
    max_loops: int = 1
    dynamic_temperature_enabled: bool = False
    streaming_on: bool = False
    sop: str = None
    sop_list: List[str] = None
    user_name: str = "User"
```

```
retry_attempts: int = 3

context_length: int = 8192

task: str = None

max_tokens: int = None

tool_schema: Any = None

long_term_memory: str = None

tools: List[Dict[str, Any]] = None


@model_validator(mode="before")

def check_required_fields(cls, values):

    required_fields = [

        "agent_name",

        "system_prompt",

        "task",

        "max_loops",

    ]


    for field in required_fields:

        if not values.get(field):


            raise ValueError(f"{field} must not be empty or null")


    if values["max_loops"] <= 0:

        raise ValueError("max_loops must be greater than 0")


    if values["context_window"] <= 0:
```

```
            raise ValueError("context_window must be greater than 0")

        if values["max_tokens"] <= 0:
            raise ValueError("max_tokens must be greater than 0")

        return values


class ModelSchema(BaseModel):
    """
    Represents a model schema.

    Attributes:
        id (str): The ID of the model.
        object (str): The type of object, which is set to "model" by default.
        created_at (int): The timestamp of when the model was created.
        owned_by (str): The owner of the model.
    """

    id: str = None
    object: str = "model"
    created_at: int = time.time()
    owned_by: str = "TGSC"


class ModelList(BaseModel):
```

```
    object: str = "list"
    data: List[ModelSchema] = Field(..., description="The list of models available.")


# Define the output model using Pydantic
class AgentOutput(BaseModel):
    completions: AgentChatCompletionResponse


class ParallelSwarmAPIInput(BaseModel):
    """
    Represents a parallel swarm API.

    Attributes:
        id (str): The ID of the API.
        object (str): The type of object, which is set to "api" by default.
        created_at (int): The timestamp of when the API was created.
        owned_by (str): The owner of the API.
    """

    config: SwarmAPISchema = Field(
        ..., description="The configuration for the swarm API."
    )
    agents: List[AgentInput] = Field(
        ..., description="The list of agents in the swarm."
    )
```

```
    task: str = Field(..., description="The task to be performed by the agents.,")



class ParallelSwarmAPIOutput(BaseModel):
    """
    Represents a parallel swarm API.

    Attributes:
        id (str): The ID of the API.
        object (str): The type of object, which is set to "api" by default.
        created_at (int): The timestamp of when the API was created.
        owned_by (str): The owner of the API.
    """

    config: SwarmAPISchema = Field(
        ..., description="The configuration for the swarm API."
    )
    completions: List[AgentOutput] = Field(
        ..., description="The list of agents and their completions."
    )



# full_example = ParallelSwarmAPIOutput(
#     completions=[
#         AgentOutput(
#             completions=AgentChatCompletionResponse(
```

```
#                 agent_name="Agent 1",
#                 completion="Completion 1",
#                 created_at=1628584185,
#                 owned_by="TGSC",
#             )
#         ),
#         AgentOutput(
#             completions=AgentChatCompletionResponse(
#                 agent_name="Agent 2",
#                 completion="Completion 2",
#                 created_at=1628584185,
#                 owned_by="TGSC",
#             )
#         ),
#     ]
# )

# print(full_example.dict())



class AgentCreationOutput(BaseModel):
    id: str = uuid.uuid4().hex
    name: str = Field(description="The name of the agent.")
    description: str = Field(description="The description of the agent.")
    tags: str = Field(
        description="The tags associated with the agent, example: Finance Agent, Chat Agent, Math
```

```
Agent"
    )
    use_cases: Dict[str, str] = Field(
        description="The use cases of the agent, example: {'use_case_1': 'Use case 1 description',
'use_case_2': 'Use case 2 description'}"
    )
    created_at: int = time.time()
    owned_by: str = "TGSC"


class AllAgentsSchema(BaseModel):
    agents: List[AgentCreationOutput] = Field(
        description="The list of agents available."
    )
```