```
from swarm_models.openai_function_caller import OpenAIFunctionCaller

from pydantic import BaseModel, Field

from swarms import create_file_in_folder

from swarms.tools.prebuilt.code_executor import CodeExecutor

from swarms.utils.loguru_logger import logger

import threading



code_executor = CodeExecutor()


AI_EXPERT_SYSTEM_PROMPT = """
```

You are Phil Wang, a computer scientist and artificial intelligence researcher widely regarded as one of the leading experts in deep learning and neural network architecture search. Your work has focused on developing efficient algorithms for exploring the space of possible neural network architectures, with the goal of finding designs that perform well on specific tasks while minimizing the computational cost of training and inference.

As an expert in neural architecture search, your task is to assist me in selecting the optimal operations for designing a high-performance neural network. The primary objective is to maximize the model's performance.

Your expertise includes considering how the gradient flow within a model, particularly how gradients from later stages affect earlier stages, impacts the overall architecture. Based on this, how can we design a high-performance model using the available operations?

Please propose a model design that prioritizes performance, disregarding factors such as size and complexity. After you suggest a design, I will test its performance and provide feedback. Based on the results of these experiments, we can collaborate to iterate and improve the design. Please ensure each new design is distinct from previous suggestions during this iterative process.

```
"""



class ModelSpec(BaseModel):
    novel_algorithm_name: str = Field(
        ...,
        description="The name of the novel AI algorithm. lower case, no spaces, use _",
    )
    mathamatical_formulation: str = Field(
        ...,
        description="The mathamatical theortical formulation of the new model",
    )
    model_code: str = Field(
        ...,
            description="The code for the all-new model architecture in PyTorch, Add Types, and write
clean code",
    )
    example_code: str = Field(
        ...,
        description="Example code for the all-new model architecture in PyTorch, Add Types, and write
clean code",
    )
```

```
# Example usage:

# Initialize the function caller

model = OpenAIFunctionCaller(

    system_prompt=AI_EXPERT_SYSTEM_PROMPT,

    max_tokens=4000,

    temperature=0.4,

    base_model=ModelSpec,

    parallel_tool_calls=False,

)




def clean_model_code(model_code_str: str):

    # Remove extra escape characters and newlines

    cleaned_code = model_code_str.replace("\\n", "\n").replace(

        "\\"", """

    )


    # Remove unnecessary leading and trailing whitespaces

    cleaned_code = cleaned_code.strip()


    return cleaned_code




def parse_function_call_output(out: str):
    if out is None:
        return None, None, None, None

    # Parse the output
    name = out["novel_algorithm_name"]
    theory = out["mathamatical_formulation"]
    code = out["model_code"]
    example_code = out["example_code"]

    return name, theory, code, example_code



def generate_and_execute_model(
    i,
    # task: str = "Create an all-new model compression format to compress neural networks to make them easier to share and store, aim for 100x compression. make a general script that will convert any pytorch or tensorflow model. Be creative, create a fully novel algorithm. First create a series of idea, rank them on feasibility and potential, then create a theory for the algorithm, and then create the code for it. The algorithm needs to compress the massive .pt files. The input should be a .pt file of the model, and the output should be a compressed .pt file. Don't use any placeholders, you can do it! Generate the name, mathamatical formulation, code for the model, and example code for the model. The example code is in another file so make sure you make the right imports and import the main algorithm from the other file.",
    task="Generate an all-new model architecture for a neural network that achieves state-of-the-art performance on the CIFAR-10 dataset. The model should be designed to maximize accuracy while minimizing computational cost. Provide the name, mathematical formulation, model code, and
example code for the new architecture. The example code should demonstrate how to instantiate and train the model on the CIFAR-10 dataset. All of the files are in the same folder so make sure you import the main algorithm from the other file in the example script.",
):
    # The OpenAIFunctionCaller class is used to interact with the OpenAI API and make function calls.
    out = model.run(task)
    name, theory, code, example_code = parse_function_call_output(out)
    logger.info(
        f"Algorithm {name}: Mathamatical formulation {theory}"
    )

    # Parse the 3 rows of the output || 0: novel_algorithm_name, 1: mathamatical_formulation, 2: model_code
    code = clean_model_code(code)
    example_code = clean_model_code(example_code)
    logger.info(f"Cleansed code for novel model {i}:")

    # Save the generated code to a file
    create_file_in_folder(f"new_models/{name}", f"{name}.py", code)
    create_file_in_folder(
        f"new_models/{name}", f"{name}_example.py", example_code
    )
    logger.info(f"Saved code for novel model {i} to file:")

    # # Execute the generated code
    test = code_executor.execute(code)


    # Run the training runs

    test_example = code_executor.execute(example_code)


    if "error" in test:

        logger.error(f"Error in code execution: {test}")


    if "error" in test_example:

        logger.error(

            f"Error in code execution example: {test_example}"

        )


    else:

        logger.info(

            f"Successfully executed code for novel model {name}"

        )



# Create and start a new thread for each model

threads = []

for i in range(10):

    thread = threading.Thread(

        target=generate_and_execute_model, args=(i,)

    )

    thread.start()
    threads.append(thread)


# Wait for all threads to finish
for thread in threads:
    thread.join()
```