```
from swarm_models.openai_function_caller import OpenAIFunctionCaller
from pydantic import BaseModel, Field
from swarms.tools.prebuilt.code_executor import CodeExecutor
from swarms.structs.concat import concat_strings


# Pydantic is a data validation library that provides data validation and parsing using Python type hints.
# It is used here to define the data structure for making API calls to retrieve weather information.
class CodeSpec(BaseModel):
    summary: str = Field(
        ...,
        description="The summary of the code",
    )
    algorithmic_pseudocode: str = Field(
        ...,
        description="The pseudocode of the code",
    )
    code: str = Field(
        ...,
        description="The code for the algorithm.",
    )


def clean_model_code(model_code_str: str) -> str:
    """
    Cleans up the generated model code string.


    Args:
        model_code_str (str): The raw model code as a string.


    Returns:
        str: The cleaned-up model code.
    """
    cleaned_code = model_code_str.replace("\\n", "\n").replace(
        "\\'", "'"
    )
    return cleaned_code.strip()



# The WeatherAPI class is a Pydantic BaseModel that represents the data structure
# for making API calls to retrieve weather information. It has two attributes: city and date.


# Example usage:
# Initialize the function caller
model = OpenAIFunctionCaller(
    system_prompt="You're the code interpreter agent, your purpose is to generate code given a task 
and provide a summary, pseudocode, and code for the algorithm.",
    max_tokens=3400,
    temperature=0.5,
    base_model=CodeSpec,
    parallel_tool_calls=False,
)



def run_model_and_generate_code(max_loops: int = 2):
    question = "What is the task for the code interpreter agent?"
    task = input(question)
    responses = []
    responses.append(question)
    responses.append(task)

    for i in range(max_loops):
        task = concat_strings(task)

        out = model.run(task)
        summary = out["summary"]
        print("\nSummary: ", summary)
        pseudocode = out["algorithmic_pseudocode"]
        code = clean_model_code(out["code"])

        output = f"{summary}\n\n{pseudocode}\n\n{code}"
        responses.append(output)

        # Code Executor
        executor = CodeExecutor()

        # Execute the code
        result = executor.execute(code)

        if "error" in result:
            print(f"Error: {result}")
            break

        print("\nCode Output: ", result)

        task = input(
            "\nEnter the next task for the code interpreter agent (or 'exit' to stop): "
        )
        responses.append(task)

    return responses


run_model_and_generate_code()
```