```
from swarm_models.openai_function_caller import OpenAIFunctionCaller

from pydantic import BaseModel




# 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 ModelCode(BaseModel):

    file_name: str

    model_code_in_pytorch: str




class TrainingCodeModel(BaseModel):

    file_name: str

    training_code: str

    dataset_name: str




# 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 a model engineer, you're purpose is to generate code in pytorch for a
give model name and code",
    max_tokens=4000,
    temperature=0.5,
    base_model=ModelCode,
)

trainer = OpenAIFunctionCaller(
    system_prompt="You're a model engineer, you're purpose is to generate the code for a given model architecture in pytorch to train using available datasets on huggingface",
    max_tokens=4000,
    temperature=0.5,
    base_model=TrainingCodeModel,
)

# The OpenAIFunctionCaller class is used to interact with the OpenAI API and make function calls.
# Here, we initialize an instance of the OpenAIFunctionCaller class with the following parameters:
# - system_prompt: A prompt that sets the context for the conversation with the API.
# - max_tokens: The maximum number of tokens to generate in the API response.
# - temperature: A parameter that controls the randomness of the generated text.
# - base_model: The base model to use for the API calls, in this case, the WeatherAPI class.
out = model.run(
    "Generate a pytorch code for a sentiment analysis model using pytorch"
)
print(str(out))

# Trainer
out = trainer.run(
    f"Generate the training code for the sentiment analysis model using pytorch: {trainer}"
)
print(out)
```