```
from swarm_models.openai_function_caller import OpenAIFunctionCaller
from pydantic import BaseModel, Field
from swarms.utils.loguru_logger import logger
import threading
import json
from typing import List, Dict
from datasets import load_dataset
import os


class ModelSpec(BaseModel):
    novel_algorithm_name: str = Field(
        ...,
        description="The name of the novel AI algorithm",
    )
    mathamatical_formulation: str = Field(
        ...,
        description="The mathematical theoretical formulation of the new model",
    )
    model_code: str = Field(
        ...,
        description="The code for the all-new model architecture in PyTorch, with documentation and
clean code",
    )
```

```
class OptimizationSpec(BaseModel):
    errors: str = Field(
        ...,
        description="The errors in the existing model architecture code",
    )
    refined_model_code: str = Field(
        ...,
        description="The refined code for the model architecture in PyTorch",
    )
    step_by_step_instructions: str = Field(
        ...,
        description="The step-by-step instructions on how the model works and how it was refined",
    )


# Initialize the function caller
model = OpenAIFunctionCaller(
        system_prompt="You're an expert model engineer like Lucidrains, you write world-class PhD-level code for deep learning models. Your purpose is to create a novel deep learning model for a research paper. You need to provide the name of the model, the mathematical formulation, and the code for the model architecture in PyTorch. Write clean and concise code that is easy to understand and implement. Write production-grade PyTorch code, add types, and documentation. Make sure you track tensor shapes and write great PyTorch code. Be creative and create models that have never been contemplated before.",
    max_tokens=3500,
    temperature=1.0,
    base_model=ModelSpec,
    parallel_tool_calls=False,
)


# Initialize the function caller
refiner = OpenAIFunctionCaller(
    system_prompt="""
    You're a model refiner, you refine existing deep learning models to improve their performance and
you optimize code and clean it up. You intake a model architecture, and you refine it to make it more
efficient, faster, and more accurate. You need to provide the code for the refined model architecture
in PyTorch. Write clean and concise code that is easy to understand and implement. Write
production-grade PyTorch code, add types, and documentation. Make sure you track tensor shapes
and write great PyTorch code. Be creative and refine models that have never been contemplated
before. Locate all errors in the code and fix them. Provide step-by-step instructions on how the
model works and how it was refined.

    """,
    max_tokens=3500,
    temperature=1.0,
    base_model=OptimizationSpec,
    parallel_tool_calls=False,
)



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()


def generate_novel_model() -> Dict[str, str]:
    """
    Generate a novel neural network model using the OpenAI function caller.

    Returns:
        Dict[str, str]: A dictionary containing the model's name, theory, and code.
    """
    out = model.run(
          "Create an entirely new model architecture by blending backbones like attention, lstms, rnns,
and ssm all into one novel architecture. Provide alternative model architectures to transformers,
ssms, convnets, lstms, and more. Be creative and don't work on architectures that have been done
before. The goal is to create new-ultra high performance nets"
```

```
    )
    name = out["novel_algorithm_name"]
    theory = out["mathamatical_formulation"]
    code = clean_model_code(out["model_code"])

    refined = refiner.run(
        f"Locate all errors in the code and fix them. Provide step-by-step instructions on how the model
works and how it was refined. Name of Algorithm: {name} Code: {code}"
    )
    errors = refined["errors"]
    refined_code = clean_model_code(refined["refined_model_code"])
    instructions = refined["step_by_step_instructions"]

    return {
        "name": name,
        "theory": theory,
        "code": code,
        "errors": errors,
        "refined_code": refined_code,
        "instructions": instructions,
    }


def generate_and_save_model(
    i: int, dataset: List[Dict[str, str]]
) -> None:
    """
    Generate, clean, save, and add the model data to a dataset.

    Args:
        i (int): The iteration number (for logging purposes).
        dataset (List[Dict[str, str]]): The dataset to add the model data to.
    """
    model_data = generate_novel_model()
    # name = model_data["name"]
    # code = model_data["code"]

    # logger.info(f"Generated code for novel model {name}:")
    # create_file_in_folder("new_models", f"{name}.py", code)
    # logger.info(f"Saved code for novel model {i} to file:")

    # Add the model data to the dataset
    dataset.append(model_data)


def save_to_jsonl(
    dataset: List[Dict[str, str]], file_path: str
) -> None:
    """
    Appends the dataset to an existing JSONL file, or creates a new file if it doesn't exist.

    Args:
        dataset (List[Dict[str, str]]): The dataset containing models' data.
        file_path (str): The path to save the JSONL file.
    """
    with open(file_path, "a") as file:  # Open in append mode
        for entry in dataset:
            file.write(json.dumps(entry) + "\n")
    logger.info(f"Dataset appended to {file_path}")


def upload_to_huggingface(
    file_path: str, dataset_name: str, huggingface_token: str
) -> None:
    """
    Uploads the dataset to Hugging Face.

    Args:
        file_path (str): The path to the JSONL file.
        dataset_name (str): The name of the dataset on Hugging Face.
        huggingface_token (str): Your Hugging Face token for authentication.
    """
    dataset = load_dataset(
        "json", data_files=file_path, split="train"
    )
    dataset.push_to_hub(dataset_name, token=huggingface_token)
    logger.info(f"Dataset uploaded to Hugging Face: {dataset_name}")
```

```
def main(
    num_models: int,
    jsonl_file_path: str,
    dataset_name: str,
    huggingface_token: str,
) -> None:
    """
    Main function to generate models, save them to JSONL, and upload to Hugging Face.

    Args:
        num_models (int): The number of models to generate.
        jsonl_file_path (str): The path to save the JSONL file.
        dataset_name (str): The name of the dataset on Hugging Face.
        huggingface_token (str): Your Hugging Face token for authentication.
    """
    dataset = []
    threads = []

    for i in range(num_models):
        thread = threading.Thread(
            target=generate_and_save_model, args=(i, dataset)
        )
        thread.start()
        threads.append(thread)
    for thread in threads:
        thread.join()


    save_to_jsonl(dataset, jsonl_file_path)
    upload_to_huggingface(
        jsonl_file_path, dataset_name, huggingface_token
    )




# Example usage
if __name__ == "__main__":
    num_models = 30  # Number of models to generate
    jsonl_file_path = "novel_models_dataset_new.jsonl"
    dataset_name = "novel_models_architectures_instructions"
    huggingface_token = os.getenv("HUGGINGFACE_TOKEN")


    main(num_models, jsonl_file_path, dataset_name, huggingface_token)
```