```
from typing import Any, List

from docstring_parser import parse
from pydantic import BaseModel
from swarms.utils.loguru_logger import initialize_logger

logger = initialize_logger("pydantic_to_json")


def _remove_a_key(d: dict, remove_key: str) -> None:
    """Remove a key from a dictionary recursively"""
    if isinstance(d, dict):
        for key in list(d.keys()):
            if key == remove_key and "type" in d.keys():
                del d[key]
            else:
                _remove_a_key(d[key], remove_key)


def check_pydantic_name(pydantic_type: type[BaseModel]) -> str:
    """
    Check the name of the Pydantic model.

    Args:
        pydantic_type (type[BaseModel]): The Pydantic model type to check.
```

```
    Returns:
        str: The name of the Pydantic model.

    """
    try:
        return type(pydantic_type).__name__
    except AttributeError as error:
        logger.error(
            f"The Pydantic model does not have a name. {error}"
        )
        raise error


def base_model_to_openai_function(
    pydantic_type: type[BaseModel],
    output_str: bool = False,
) -> dict[str, Any]:
    """
    Convert a Pydantic model to a dictionary representation of functions.

    Args:
        pydantic_type (type[BaseModel]): The Pydantic model type to convert.

    Returns:
        dict[str, Any]: A dictionary representation of the functions.
```

```
"""
schema = pydantic_type.model_json_schema()

# Fetch the name of the class
name = type(pydantic_type).__name__

docstring = parse(pydantic_type.__doc__ or "")
parameters = {
    k: v
    for k, v in schema.items()
    if k not in ("title", "description")
}

for param in docstring.params:
    if (name := param.arg_name) in parameters["properties"] and (
        description := param.description
    ):
        if "description" not in parameters["properties"][name]:
            parameters["properties"][name][
                "description"
            ] = description

parameters["type"] = "object"

if "description" not in schema:
    if docstring.short_description:
```

```
        schema["description"] = docstring.short_description
    else:
        schema["description"] = (
            f"Correctly extracted `{name}` with all "
            f"the required parameters with correct types"
        )

_remove_a_key(parameters, "title")
_remove_a_key(parameters, "additionalProperties")

if output_str:
    out = {
        "function_call": {
            "name": name,
        },
        "functions": [
            {
                "name": name,
                "description": schema["description"],
                "parameters": parameters,
            },
        ],
    }
    return str(out)

else:
```

```
        return {
            "function_call": {
                "name": name,
            },
            "functions": [
                {
                    "name": name,
                    "description": schema["description"],
                    "parameters": parameters,
                },
            ],
        }


def multi_base_model_to_openai_function(
    pydantic_types: List[BaseModel] = None,
    output_str: bool = False,
) -> dict[str, Any]:
    """
    Converts multiple Pydantic types to a dictionary of functions.

    Args:
        pydantic_types (List[BaseModel]]): A list of Pydantic types to convert.

    Returns:
        dict[str, Any]: A dictionary containing the converted functions.
```

```
"""
functions: list[dict[str, Any]] = [
    base_model_to_openai_function(pydantic_type, output_str)[
        "functions"
    ][0]
    for pydantic_type in pydantic_types
]

return {
    "function_call": "auto",
    "functions": functions,
}
```