```
import inspect
from typing import (
    Callable,
    Type,
    Any,
    get_type_hints,
    Union,
    List,
    Dict,
    Tuple,
)
from types import NoneType
from pydantic import BaseModel, create_model, Field
from loguru import logger


# Utility functions to handle complex types
def get_origin(tp):
    return getattr(tp, "__origin__", None)


def get_args(tp):
    return getattr(tp, "__args__", ())


def function_to_pydantic_schema(
```

```
    func: Callable[..., Any], model_name: str = "FunctionParamsModel"
) -> Type[BaseModel]:
    """
    Create a production-grade Pydantic BaseModel schema from a function's parameters.

    This function inspects the given function's parameters and their type hints
    to create a corresponding Pydantic BaseModel schema. It handles complex types,
    including Optional, Union, List, Dict, and Tuple.

    Args:
        func (Callable[..., Any]): The function to create a schema for.
                model_name (str, optional): The name for the created model. Defaults to "FunctionParamsModel".

    Returns:
        Type[BaseModel]: A new Pydantic BaseModel subclass with fields based on the function's parameters.

    Raises:
        ValueError: If the function has no parameters, if type hints are missing, or if unsupported types are used.

    Example:
        >>> def example_function(name: str, age: Optional[int], tags: List[str] = None):
        ...     pass
        >>> ParamsModel = function_to_pydantic_schema(example_function)
```

```
    >>> print(ParamsModel.schema_json(indent=2))
"""
logger.info(
    f"Creating Pydantic schema '{model_name}' from function '{func.__name__}'"
)

signature = inspect.signature(func)
parameters = signature.parameters

if not parameters:
    logger.error("Function has no parameters")
    raise ValueError(
        "Cannot create a schema for a function with no parameters"
    )

type_hints = get_type_hints(func)

field_definitions = {}
for name, param in parameters.items():
    if name not in type_hints:
        logger.error(f"Missing type hint for parameter '{name}'")
        raise ValueError(
            f"Type hint missing for parameter '{name}'"
        )

    field_type = type_hints[name]
```

```
default = (
    param.default if param.default != param.empty else ...
)
field_info = {}

# Handle Optional types
if get_origin(field_type) is Union and NoneType in get_args(
    field_type
):
    field_type = get_args(field_type)[
        0
    ]  # Get the non-None type
    if default is ...:
        default = None

# Handle Union types
if get_origin(field_type) is Union:
    field_info["description"] = (
        f"Union of {', '.join([arg.__name__ for arg in get_args(field_type)])}"
    )

# Handle List, Dict, and Tuple types
if get_origin(field_type) in (
    List,
    list,
    Dict,
```

```
    dict,
    Tuple,
    tuple,
):
    container_type = get_origin(field_type).__name__
    content_types = ", ".join(
        [arg.__name__ for arg in get_args(field_type)]
    )
    field_info["description"] = (
        f"{container_type} of {content_types}"
    )

# Add default value to description if it exists
if default is not ...:
    field_info["description"] = (
        field_info.get("description", "")
        + f" (default: {default})"
    )

logger.debug(
    f"Field '{name}' of type {field_type} with default {default}"
)
field_definitions[name] = (
    field_type,
    Field(default=default, **field_info),
)
```

```
    logger.success(
        f"Successfully created schema '{model_name}' with {len(field_definitions)} fields"
    )
    return create_model(model_name, **field_definitions)


# # Example usage with more complex types
# def complex_function(
#     name: str,
#     age: Optional[int],
#     is_student: bool = False,
#     grades: List[float] = None,
#     metadata: Dict[str, Any] = None,
#     tags: Union[List[str], Tuple[str, ...]] = (),
# ):
#     pass


# try:
#     ComplexSchema = function_to_pydantic_schema(
#         complex_function, "ComplexFunctionSchema"
#     )
#     print(ComplexSchema.schema_json(indent=2))
# except ValueError as e:
#     logger.error(f"Error creating schema: {e}")
```