ToolAgent Documentation

The `ToolAgent` class is a specialized agent that facilitates the execution of specific tasks using a model and tokenizer. It is part of the `swarms` module and inherits from the `Agent` class. This agent is designed to generate functions based on a given JSON schema and task, making it highly adaptable for various use cases, including natural language processing and data generation.

The `ToolAgent` class plays a crucial role in leveraging pre-trained models and tokenizers to automate tasks that require the interpretation and generation of structured data. By providing a flexible interface and robust error handling, it ensures smooth integration and efficient task execution.

Parameters

Parameter	Type	Description
`name`	`str`	The name of the tool agent. Default is "Function Calling Agent".
`description`	`str`	A description of the tool agent. Default is "Generates a function based on the input json schema and the task".
`model`	`Any`	The model used by the tool agent.
`tokenizer`	`Any`	The tokenizer used by the tool agent.
`json_schema`	`Any`	The JSON schema used by the tool agent.

|

| `max_number_tokens`| `int` | The maximum number of tokens for generation. Default is 500. |

| `parsing_function` | `Optional[Callable]` | An optional parsing function to process the output of the tool agent. |

| `llm` | `Any` | An optional large language model to be used by the tool agent. |

| `*args` | Variable length argument list | Additional positional arguments. |

| `**kwargs` | Arbitrary keyword arguments | Additional keyword arguments. |

Attributes

Attribute	Type	Description
`name`	`str`	The name of the tool agent.
`description`	`str`	A description of the tool agent.
`model`	`Any`	The model used by the tool agent.
`tokenizer`	`Any`	The tokenizer used by the tool agent.
`json_schema`	`Any`	The JSON schema used by the tool agent.

Methods

`run`

```python
def run(self, task: str, *args, **kwargs) -> Any:
```

Parameters:

Parameter	Type	Description
`task`	`str`	The task to be performed by the tool agent.
`*args`	Variable length argument list	Additional positional arguments.
`**kwargs`	Arbitrary keyword arguments	Additional keyword arguments.

Returns:

- The output of the tool agent.

Raises:

- `Exception`: If an error occurs during the execution of the tool agent.

Functionality and Usage

The `ToolAgent` class provides a structured way to perform tasks using a model and tokenizer. It initializes with essential parameters and attributes, and the `run` method facilitates the execution of the specified task.

Initialization

The initialization of a `ToolAgent` involves specifying its name, description, model, tokenizer, JSON schema, maximum number of tokens, optional parsing function, and optional large language model.

```python
agent = ToolAgent(
    name="My Tool Agent",
    description="A tool agent for specific tasks",
    model=model,
    tokenizer=tokenizer,
    json_schema=json_schema,
    max_number_tokens=1000,
    parsing_function=my_parsing_function,
    llm=my_llm
)
```

Running a Task

To execute a task using the `ToolAgent`, the `run` method is called with the task description and any additional arguments or keyword arguments.

```python
result = agent.run("Generate a person's information based on the given schema.")
print(result)
```

```

### Detailed Examples

#### Example 1: Basic Usage

```python
from transformers import AutoModelForCausalLM, AutoTokenizer
from swarms import ToolAgent

model = AutoModelForCausalLM.from_pretrained("databricks/dolly-v2-12b")
tokenizer = AutoTokenizer.from_pretrained("databricks/dolly-v2-12b")

json_schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "age": {"type": "number"},
        "is_student": {"type": "boolean"},
        "courses": {
            "type": "array",
            "items": {"type": "string"}
        }
    }
}
```
```

task = "Generate a person's information based on the following schema:"

agent = ToolAgent(model=model, tokenizer=tokenizer, json_schema=json_schema)

generated_data = agent.run(task)

print(generated_data)

```

#### Example 2: Using a Parsing Function

```python

def parse_output(output):

    # Custom parsing logic

    return output

agent = ToolAgent(

    name="Parsed Tool Agent",

    description="A tool agent with a parsing function",

    model=model,

    tokenizer=tokenizer,

    json_schema=json_schema,

    parsing_function=parse_output

)

task = "Generate a person's information with custom parsing:"

parsed_data = agent.run(task)

```
print(parsed_data)
```

Example 3: Specifying Maximum Number of Tokens

```python
agent = ToolAgent(
    name="Token Limited Tool Agent",
    description="A tool agent with a token limit",
    model=model,
    tokenizer=tokenizer,
    json_schema=json_schema,
    max_number_tokens=200
)

task = "Generate a concise person's information:"
limited_data = agent.run(task)

print(limited_data)
```

Full Usage

```python

from pydantic import BaseModel, Field
```

```
from transformers import AutoModelForCausalLM, AutoTokenizer

from swarms import ToolAgent
from swarms.tools.json_utils import base_model_to_json

# Model name
model_name = "CohereForAI/c4ai-command-r-v01-4bit"

# Load the pre-trained model and tokenizer
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map="auto",
)

# Load the pre-trained model and tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name)


# Initialize the schema for the person's information
class APIExampleRequestSchema(BaseModel):
    endpoint: str = Field(
        ..., description="The API endpoint for the example request"
    )
    method: str = Field(
        ..., description="The HTTP method for the example request"
    )
```

```
    headers: dict = Field(
        ..., description="The headers for the example request"
    )
    body: dict = Field(..., description="The body of the example request")
    response: dict = Field(
        ...,
        description="The expected response of the example request",
    )



# Convert the schema to a JSON string
api_example_schema = base_model_to_json(APIExampleRequestSchema)
# Convert the schema to a JSON string

# Define the task to generate a person's information
task = "Generate an example API request using this code:\n"

# Create an instance of the ToolAgent class
agent = ToolAgent(
    name="Command R Tool Agent",
    description=(
        "An agent that generates an API request using the Command R"
        " model."
    ),
    model=model,
    tokenizer=tokenizer,
```

```
    json_schema=api_example_schema,
)

# Run the agent to generate the person's information
generated_data = agent.run(task)

# Print the generated data
print(f"Generated data: {generated_data}")
```

```

## Jamba ++ ToolAgent

```python
from pydantic import BaseModel, Field
from transformers import AutoModelForCausalLM, AutoTokenizer

from swarms import ToolAgent
from swarms.tools.json_utils import base_model_to_json

# Model name
model_name = "ai21labs/Jamba-v0.1"

# Load the pre-trained model and tokenizer
```
```

```
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map="auto",
)

# Load the pre-trained model and tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name)



# Initialize the schema for the person's information
class APIExampleRequestSchema(BaseModel):
    endpoint: str = Field(
        ..., description="The API endpoint for the example request"
    )
    method: str = Field(
        ..., description="The HTTP method for the example request"
    )
    headers: dict = Field(
        ..., description="The headers for the example request"
    )
    body: dict = Field(..., description="The body of the example request")
    response: dict = Field(
        ...,
        description="The expected response of the example request",
    )
```

```
# Convert the schema to a JSON string
api_example_schema = base_model_to_json(APIExampleRequestSchema)
# Convert the schema to a JSON string

# Define the task to generate a person's information
task = "Generate an example API request using this code:\n"

# Create an instance of the ToolAgent class
agent = ToolAgent(
    name="Command R Tool Agent",
    description=(
        "An agent that generates an API request using the Command R"
        " model."
    ),
    model=model,
    tokenizer=tokenizer,
    json_schema=api_example_schema,
)

# Run the agent to generate the person's information
generated_data = agent(task)

# Print the generated data
print(f"Generated data: {generated_data}")
```

Additional Information and Tips

- Ensure that either the `model` or `llm` parameter is provided during initialization. If neither is provided, the `ToolAgent` will raise an exception.
- The `parsing_function` parameter is optional but can be very useful for post-processing the output of the tool agent.
- Adjust the `max_number_tokens` parameter to control the length of the generated output, depending on the requirements of the task.

References and Resources

- [Transformers Documentation](https://huggingface.co/transformers/)
- [Loguru Logger](https://loguru.readthedocs.io/en/stable/)

This documentation provides a comprehensive guide to the `ToolAgent` class, including its initialization, usage, and practical examples. By following the detailed instructions and examples, developers can effectively utilize the `ToolAgent` for various tasks involving model and tokenizer-based operations.