Pipecat

Features › Function Calling

Features

Function Calling

Enable LLMs to interact with external services and APIs

Understanding Function Calling

Function calling (also known as tool calling) allows LLMs to request information from external services and APIs. This enables your bot to access real-time data and perform actions that aren't part of its training data.

For example, you could give your bot the ability to:

- Check current weather conditions
- Look up stock prices
- Query a database
- Control smart home devices
- Schedule appointments

Here's how it works:

1. You define functions the LLM can use and register them to the LLM service used in your pipeline
2. When needed, the LLM requests a function call
3. Your application executes any corresponding functions
4. The result is sent back to the LLM
5. The LLM uses this information in its response

Implementation

Pipecat

Features › **Function Calling**

OpenAI Anthropic Gemini

```
tools = [
    {
        "function_declarations": [
            {
                "name": "get_weather_current",
                "description": "Get the current weather",
                "parameters": {
                    "type": "object",
                    "properties": {
                        "location": {
                            "type": "string",
                            "description": "Location as 'city,state,country'",
                        },
                        "format": {
                            "type": "string",
                            "enum": ["celsius", "fahrenheit"],
                            "description": "The temperature unit to use.",
                        },
                    },
                    "required": ["location", "format"],
                },
            }
        ]
    }
]
```

2. Register Function Handlers

Register handlers for your functions using the LLM service's `register_function` **method**:

```
llm = OpenAILLMService(api_key="your-api-key", model="gpt-4")
```

Pipecat

Features › **Function Calling**

```
async def fetch_weather_from_api(function_name, tool_call_id, args, llm, conte
    # Fetch weather data from your API
    weather_data = {"conditions": "sunny", "temperature": "75"}
    await result_callback(weather_data)

# Register the function
llm.register_function(
    "get_current_weather",
    fetch_weather_from_api,
    start_callback=start_fetch_weather
)
```

3. Create the Pipeline

Include your LLM service in your pipeline with the registered functions:

```
# Initiliaze the LLM context, including messges and tool calls
context = OpenAILLMContext(messages, tools)

# Create the context aggregator, to collec the user and assistnat context
context_aggregator = llm.create_context_aggregator(context)

# Create the pipeline
# 1. Input from the transport
# 2. User context aggregation
# 3. LLM processing
# 4. TTS processing
# 5. Output to the transport
# 6. Assistant context aggregation
pipeline = Pipeline([
    transport.input(),
    context_aggregator.user(),
    llm,
    tts,
    transport.output(),
```

Pipecat

Features › **Function Calling**

Function Handler Details

Handler Parameters

`function_name` : Name of the called function

`tool_call_id` : Unique identifier for the function call

`args` : Arguments passed by the LLM

`llm` : Reference to the LLM service

`context` : Current conversation context

`result_callback` : Async function to return results

Return Values

Return data through the `result_callback`

Return `None` to ignore the function call

Errors should be handled within your function

Controlling Function Call Behavior

When returning results from a function handler, you can control how the LLM processes those results using a `FunctionCallResultProperties` frame.

Properties

`run_llm` `Optional[bool]`

Controls whether the LLM should generate a response after the function call:

`True` : Run LLM after function call (default if no other function calls in progress)

`False` : Don't run LLM after function call - `None` : Use default behavior

Pipecat

Features › **Function Calling**

Example Usage

```
async def fetch_weather_from_api(function_name, tool_call_id, args, llm, c  te
    # Fetch weather data
    weather_data = {"conditions": "sunny", "temperature": "75"}

    # Don't run LLM after this function call
    properties = FunctionCallResultProperties(run_llm=False)

    await result_callback(weather_data, properties=properties)

async def query_database(function_name, tool_call_id, args, llm, context, resul
    # Query database
    results = await db.query(args["query"])

    async def on_update():
        await notify_system("Database query complete")

    # Run LLM after function call and notify when context is updated
    properties = FunctionCallResultProperties(
        run_llm=True,
        on_context_updated=on_update
    )

    await result_callback(results, properties=properties)
```

Next steps

Check out the **function calling examples** to see a complete example for specific LLM providers.

Refer to your LLM providers documentation to learn more about their funciton calling capabilities.

Pipecat

Features › **Function Calling**

Powered by Mintlify