Function call

为什么要用function call?

以前的LLM只能依靠自己已有的知识回答问题，无法直接获取实时数据、也无法与外部系统交互。

Function Call 是一种实现大型语言模型连接外部工具的机制。通过 API 调用 LLM 时，调用方可以描述函数，包括函数的功能描述、请求参数说明、响应参数说明，让 LLM 根据用户的输入，合适地选择调用哪个函数，同时理解用户的自然语言，并转换为调用函数的请求参数（通过 JSON 格式返回）。调用方使用 LLM 返回的函数名称和参数，调用函数并得到响应。最后，如果需要，把函数的响应传给 LLM，让 LLM 组织成自然语言回复用户。

大模型的function call能力是如何获得的?

主要通过对基础模型进行sft获得，基础模型需要先具备良好的指令遵循和代码/结构化数据生成能力。

sft的核心思想，是要教会LLM两件事：

1、识别意图：理解用户的请求是否需要借助外部工具/函数来完成，而不是直接生成文本回答。

2、参数提取与格式化：如果需要调用函数，需要能够正确地从用户请求中抽取出所需的参数，并按照预先定义的格式（通常是json）生成函数调用的指令。

sft的过程如下：

- 步骤 1：数据集构建：构建包含 Function Calling 场景的指令微调数据集，每条数据样本包含用户输入（可能需调用函数或直接回答的请求）、可用函数 / 工具描述（函数用途、参数类型等结构化文本）、期望输出（需调用函数时为含函数名与参数的 JSON，否则为直接文本回答）。
- 步骤 2：选择基础模型：选用具备强大指令遵循能力的预训练大模型（如 Llama、GPT、Qwen 等）。
- 步骤 3：格式化训练数据：将 “用户输入” 与 “可用函数描述” 拼接为模型输入（Prompt），“期望输出”（JSON 函数调用或文本回答）作为目标输出（Completion/Target），通过特定分隔符或模板区分。
- 步骤 4：进行微调：使用标准 SFT 方法（全参数微调或 PEFT 如 LoRA）在数据集上训练，优化目标为最小化预测输出与期望输出的差异（如交叉熵损失），使模型学会根据输入与函数描述，决定直接回答或生成特定格式的函数调用 JSON。

通过上述监督微调流程，大模型掌握识别意图（判断是否需调用外部工具）与参数提取格式化（正确抽取参数并生成规范函数调用指令）的能力，从而获得 Function Call 能力。

Function - Call 数据集的基本结构包含哪些部分?

Function - Call 数据集基本结构通常包含:

- [系统提示 / 全局指令]（可选）：设定角色、能力边界等。
- [可用函数 / 工具描述区]：详细列出每个可用函数的结构化描述。
- [对话历史]（可选，多轮对话重要）：记录用户（User）和助理（Assistant）的交互历史及当前用户请求。
- [触发指令 / 分隔符]：提示模型开始思考或生成，如 “Assistant:”。

Function - Call数据集中可用函数/工具描述区的格式是怎样的?

通常使用JSON列表或结构化文本，包含以下核心字段:

代码块

```
[
    {
        "name": "函数名",
        "description": "函数功能描述",
        "parameters": {
            "type": "object",
            "properties": {
                "参数名": {
                    "type": "数据类型",
                    "description": "参数含义描述"
                }
            },
            "required": ["必填参数名列表"]
        }
    }
]
```

例如:

代码块

```
{
    "name": "get_weather",
    "description": "查询指定城市和日期的天气信息",
    "parameters": {
        "type": "object",
        "properties": {
            "city": {
```

```
                "type": "string",
                "description": "需要查询天气的城市名称，例如：北京"
            },
            "date": {
                "type": "string",
                "description": "需要查询的日期，例如：今天、明天、2023 - 10 - 26"
            }
        },
        "required": ["city", "date"]
    }
}
```

Function - Call数据集的关键要素有哪些?

- **name**：函数的唯一标识符。
- **description**：用自然语言清晰描述函数的功能和适用场景，是模型判断何时调用的关键。
- **parameters**：定义函数接受的参数，包含：
  - **type**：通常为 `"object"`。
  - **properties**：列出每个参数的名称、数据类型（如 `string`、`integer`）和描述。
  - **required**：必须提供的参数名称列表。

Function - Call数据集中对话流程的格式是怎样的?

1. **用户请求**：用户发出指令（如“帮我查一下明天上海的天气，然后给张三发邮件”）。
2. **模型首次响应（Function Call）**：模型识别后生成调用函数的JSON（如 `get_weather`）。
3. **外部执行**：应用程序调用实际工具（如天气API）。
4. **结果喂回模型**：将工具执行结果格式化后再次输入模型（如天气结果 `{"temperature": "25°C", "condition": "晴朗"}`）。
5. **模型再次响应**：可能再次调用函数（如 `send_email`）或生成最终回答（如“已查询并发送邮件”）。

如何将下游工具、插件转化为模型可理解的方式?

核心是**标准化描述**和**执行对接**：

- **标准化描述（Standardized Description）**：
  - 为工具设计符合Function Call格式的结构化描述（如JSON Schema），包含唯一名称（Name）、功能描述（Description）、参数定义（Parameters）。

    - 描述语言自然准确，避免歧义。
- **执行对接（Execution Bridging）**：
    - 将工具描述作为上下文传递给模型。
    - 解析模型输出的Function Call JSON，调用实际工具，处理参数校验和结果，再将结果反馈给模型。

简述Function Call的工作原理

1. **LLM接收用户提示**：用户输入请求。
2. **LLM决定所需工具**：根据提示判断调用哪些工具。
3. **程序处理调用请求**：开发者实现逻辑，接收LLM的工具调用请求并准备参数。
4. **执行函数调用**：将带参数的函数调用传递给后端服务执行，结果再反馈给LLM用于后续处理。