二、GLM4 API调用指南与关键参数

1. 基于chat.completions完成对话

```
import os
from zhipuai import ZhipuAI
from IPython.display import display, Code, Markdown
```

在上一小节中，我们已经顺利完成智谱AI账号注册，并获取GLM模型API-KEY，同时完成了本地调用测试。本节我们将开始着手使用GLM-4模型，并详细介绍GLM-4调用函数参数及模型的多角色对话系统。

```
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response = client.chat.completions.create(
    model="glm-4",  # 填写需要调用的模型名称
    messages=[
        {"role": "user", "content": "你好"}
    ],
)
print(response.choices[0].message)
```

```
CompletionMessage(content='你好👋！我是人工智能助手智谱清言，可以叫我小智🤖，很高兴见到你，欢迎问我任何问题。', role='assistant', tool_calls=None)
```

大模型的对话功能是大模型开发流程中最为核心的要点，**调用一次create就是发起一次对话**，当我们需要多次与大模型进行对话时，我们就需要多次调取create——

```
response_1 = client.chat.completions.create(
    model="glm-4",  # 填写需要调用的模型名称
    messages=[
        {"role": "user", "content": "请问什么是大语言模型？"}
    ],
)
print(response_1.choices[0].message)
```

```
CompletionMessage(content='大语言模型（Large Language Model，LLM）是一种基于深度学习的自然语言处理模型，它拥有数十亿乃至数万亿个参数，能够理解和生成自然语言文本。这种模型通过大规模的无监督预训练，学习到了丰富的语言表示和知识。\n\n在预训练过程中，大语言模型类似于一个极度聪明的“学霸”，它“阅读”了来自各个领域的大量文本数据，从而获得了对各种语言现象的理解。这个过程并不是简单地将数据存储在模型中，而是通过不断调整模型内部参数，使其能够捕捉到语言的深层规律和模式。\n\n大语言模型通常采用Transformer模型架构，利用自注意力机制来处理长距离的依赖关系，这使得模型在理解和生成文本时能够考虑到上下文的广泛信息。\n\n这些模型可以执行多种自然语言处理任务，如文本分类、文本生成、问题回答、机器翻译、情感分析、文档摘要等，展现出强大的泛化能力和推理能力。\n\n大语言模型的出现，极大地推动了自然语言处理领域的发展，为智能客服、内容审核、机器翻译、辅助写作等实际应用提供了强有力的技术支持。同时，这类模型也在不断地推动着人工智能技术向更加人性化、智能化的方向发展。', role='assistant', tool_calls=None)
```

```
response_2 = client.chat.completions.create(
    model="glm-4",  # 填写需要调用的模型名称
    messages=[
        {"role": "user", "content": "你提到它能为人们的生活和工作带来便利，它是什么？你可以举个例子吗？"}
    ],
)
print(response_2.choices[0].message)
```

```
CompletionMessage(content='在我没有上下文的情况下，很难确定“它”具体指的是什么。但是，如果我假设“它”指的是一般意义上的技术或某个创新产品/服务，那么我可以给出一个例子。\n\n假设“它”指的是智能手机，那么以下是如何为人们的生活和工作带来便利的例子：\n\n1. 生活便利：\n   - 智能手机通过集成的地图应用帮助人们导航，找到目的地，无论是驾车、步行还是使用公共交通。\n   - 通过社交媒体应用，人们可以与家人和朋友保持联系，分享生活点滴，即使他们身处世界的不同角落。\n   - 智能手机上的健康与健身应用可以帮助用户跟踪健康数据、规划锻炼计划，促进健康生活方式。\n\n2. 工作便利：\n   - 员工可以通过智能手机随时随地接收和回复工作邮件，提高工作效率。\n   - 通过视频会议应用，即使不在办公室，人们也能参与远程会议。\n   - 智能手机中的应用程序可以帮助企业管理者监控业务运营，比如库存管理、销售数据等。\n\n智能手机只是众多例子中的一个，实际上，许多技术发明和创新产品都在不断为我们的生活和工作提供便利，改善我们的日常体验。', role='assistant', tool_calls=None)
```

很明显，模型并不记得之前的信息，对话与对话之间是相互独立的。如果我们想让模型记得之前的信息，就需要将上一个create方法产出的结果输入到下一个create方法中。所以在大模型开发的流程中，**对话功能是嵌套的逻辑、而不是线性的逻辑。**

```
memory_ = response_1.choices[0].message.content
```

```
memory_
```

```
'大语言模型（Large Language Model，LLM）是一种基于深度学习的自然语言处理模型，它拥有数十亿乃至数万亿个参数，能够理解和生成自然语言文本。这种模型通过大规模的无监督预训练，学习到了丰富的语言表示和知识。\n\n在预训练过程中，大语言模型类似于一个极度聪明的“学霸”，它“阅读”了来自各个领域的大量文本数据，从而获得了对各种语言现象的理解。这个过程并不是简单地将数据存储在模型中，而是通过不断调整模型内部参数，使其能够捕捉到语言的深层规律和模式。\n\n大语言模型通常采用Transformer模型架构，利用自注意力机制来处理长距离的依赖关系，这使得模型在理解和生成文本时能够考虑到上下文的广泛信息。\n\n这些模型可以执行多种自然语言处理任务，如文本分类、文本生成、问题回答、机器翻译、情感分析、文档摘要等，展现出强大的泛化能力和推理能力。\n\n大语言模型的出现，极大地推动了自然语言处理领域的发展，为智能客服、内容审核、机器翻译、辅助写作等实际应用提供了强有力的技术支持。同时，这类模型也在不断地推动着人工智能技术向更加人性化、智能化的方向发展。'
```

```
response = client.chat.completions.create(
    model="glm-4",  # 填写需要调用的模型名称
    messages=[
        {"role": "user", "content": memory_ + "你提到它能为人们的生活和工作带来便利，它是什么？你可以举个例子吗？"}
    ],
)
print(response.choices[0].message)
```

```
CompletionMessage(content='大语言模型（LLMs）能为人们的生活和工作带来便利的原因在于它们能够自动化并增强许多与语言相关的任务。以下是一些具体的例子：\n\n1. **客户服务**：\n   - **聊天机器人**：在电商平台或客服中心，大语言模型可以用来构建智能聊天机器人，它们能够理解用户的问题，并提供准确的答案，从而提供24/7的服务，减少了对人工客服的依赖。\n\n2. **教育**：\n   - **个性化学习**：语言模型可以根据学生的学习进度和能力，提供定制化的学习材料和解释，帮助学生更好地理解复杂概念。\n   - **辅助写作**：模型可以提供写作建议，帮助学生在撰写论文或报告时改进文风和语法。\n\n3. **医疗健康**：\n   - **辅助诊断**：通过分析病人的病历和症状描述，语言模型可能辅助医生进行初步的诊断。\n   - **健康咨询**：在用户描述了自己的症状后，模型可以提供初步的健康咨询和可能的下一步行动建议。\n\n4. **内容创作与媒体**：\n   - **新闻简报生成**：自动从大量新闻来源中摘录和总结信息，生成个性化的新闻简报。\n   - **文章生成**：对于特定的主题或关键词，语言模型能够生成完整的文章或博客帖子。\n\n5. **日常办公**：\n   - **电子邮件助手**：自动生成或建议电子邮件回复，提高工作效率。\n   - **会议记录**：将会议对话转换成文字记录，并提供关键点的摘要。\n\n6. **科研与技术发展**：\n   - **文献分析**：快速阅读和理解大量科研文献，帮助研究人员找到研究空白或新的研究方向。\n   - **代码辅助**：帮助开发者通过自然语言描述生成代码片段，加速编程过程。\n\n这些例子展示了大语言模型如何通过自动化和提高效率来改善人们的生活和工作。然而，确实需要注意这些技术带来的潜在风险，并采取措施确保其正面影响最大化，负面影响最小化。', role='assistant', tool_calls=None)
```

1.GLM多角色对话系统解释

时至今日，多角色对话基本上已经成了顶尖大模型的标配。正是基于多角色对话这一基础技术架构，大模型才能非常灵活的实现各类对话需求，甚至多角色对话也是更高效的实现Function calling的基础，而后者则是整个Agent开发的基石。

那什么是多角色对话呢？简而言之就是将用户和大模型之间的“私聊”变为“群聊”，在用户和大模型这两个角色基础之上，进一步增加“系统”和“工具”这两个角色。尽管最终目的都是为了能完成大模型完成和用户的之间对话，但在对话过程中添加一些额外角色，确实能够更好的引导大模型完成对话。例如对话系统中增加的“系统”这一角色，可以为本次对话增加基本背景信息、对大模型进行角色身份设置等，相当于是设置本场对话的基本规则和信息；而对话系统中的“工具”这一角色，则相当于是大模型的拓展工具箱，当大模型无法回答当前问题的时候（例如用户希望查询当前即时天气），就可以向工具求助，而如果这个“工具”角色能够查到当前天气，则会“告诉”大模型当前天气情况，大模型在接收到了当前天气情况之后再告诉用户。如此一来，大模型便可以更加高效便捷的完成对话任务。

> 尽管多角色对话系统能够极大程度提高大模型的对话可用性，但并非所有模型都有能力进行多角色对话——往往只有推理能力较强的大模型才能够非常顺利的进行多角色对话。对于ChatGLM系列模型来说，也正是发布到了第三代，才正式引入多角色对话系统。而相比之下GPT系列模型，从ChatGPT（GPT-3.5）开始一直都是多角色对话模式。

而实际执行过程中，多角色对话的过程核心是依靠messages参数来实现的。

2.messages功能综述

总的来说，messages是一种用于描述ChatCompletion模型和用户之间通信信息的高级抽象，也是支撑多角色对话系统的关键，从表示形式上来说，一个messages是一个列表，包含多个字典，而每个字典都是一条消息，其中，一条消息由包含两个键值对（即每个字典都包含两个键值对），第一个键值对用于表示消息发送者，其中第一个Key为字符串'role'，Value为参与对话的角色名称，或者可以理解为本条消息的作者或消息发送人名称，第二个键值对表示具体消息内容，Key为字符串'content'，Value为具体的消息内容，用字符串表示。

```
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response = client.chat.completions.create(
    model="glm-4",
    messages=[
        {"role": "user", "content": "请问什么是机器学习？"}
    ],
)
```

```
display(Markdown(response.choices[0].message.content))
```

机器学习（Machine Learning, ML）是人工智能（Artificial Intelligence, AI）的一个重要分支，它让计算机系统能够基于数据进行自我学习和改进，而无需进行显式的编程。简单来说，机器学习就是通过算法和统计模型，让机器能够从经验中学习，并据此做出预测或决策。

在机器学习的框架下，计算机模型通过分析大量的数据集，找出数据中的模式和规律。这个过程主要包括以下几个基本概念：

- **数据集**：包含用于训练模型的信息，通常分为训练集、验证集和测试集。
- **特征**：数据的属性或描述性指标，用于输入到模型中进行学习。
- **标签**：数据的真实结果或目标变量，通常作为模型的输出目标。
- **模型**：通过学习输入特征与标签之间的关系，得到的一个数学函数或映射。
- **训练**：使用训练数据来调整模型参数，以便模型能够更好地捕捉数据特征和标签之间的关系。
- **评估**：通过一系列指标（如准确率、精确率、召回率等）来衡量模型的效果和性能。

机器学习广泛应用于各种领域，如图像识别、语音识别、自然语言处理、推荐系统、自动驾驶等。根据学习过程中是否使用标签，机器学习可以分为以下几类：

- **监督学习**：模型在已知输入和输出的数据集上进行训练。
- **无监督学习**：模型在无标签的数据集上进行训练，寻找数据内部的结构和模式。
- **强化学习**：通过与环境的交互，通过试错的方式不断学习和优化策略。

综上所述，机器学习是一门旨在通过数据使计算机自我学习和进步的技术，它在现代社会中发挥着越来越重要的作用，推动了技术的革新和应用的拓展。

```
response.choices[0].message
```

```
CompletionMessage(content='机器学习（Machine Learning, ML）是人工智能（Artificial Intelligence, AI）的一个重要分支，它让计算机系统能够基于数据进行自我学习和改进，而无需进行显式的编程。简单来说，机器学习就是通过算法和统计模型，让机器能够从经验中学习，并据此做出预测或决策。\n\n在机器学习的框架下，计算机模型通过分析大量的数据集，找出数据中的模式和规律。这个过程主要包括以下几个基本概念：\n\n- **数据集**：包含用于训练模型的信息，通常分为训练集、验证集和测试集。\n- **特征**：数据的属性或描述性指标，用于输入到模型中进行学习。\n- **标签**：数据的真实结果或目标变量，通常作为模型的输出目标。\n- **模型**：通过学习输入特征与标签之间的关系，得到的一个数学函数或映射。\n- **训练**：使用训练数据来调整模型参数，以便模型能够更好地捕捉数据特征和标签之间的关系。\n- **评估**：通过一系列指标（如准确率、精确率、召回率等）来衡量模型的效果和性能。\n\n机器学习广泛应用于各种领域，如图像识别、语音识别、自然语言处理、推荐系统、自动驾驶等。根据学习过程中是否使用标签，机器学习可以分为以下几类：\n\n- **监督学习**：模型在已知输入和输出的数据集上进行训练。\n- **无监督学习**：模型在无标签的数据集上进行训练，寻找数据内部的结构和模式。\n- **强化学习**：通过与环境的交互，通过试错的方式不断学习和优化策略。\n\n综上所述，机器学习是一门旨在通过数据使计算机自我学习和进步的技术，它在现代社会中发挥着越来越重要的作用，推动了技术的革新和应用的拓展。', role='assistant', tool_calls=None)
```

例如上述示例中的messages就总共包含一条信息，即一个一个名为user的角色发送了一条名为'请问什么是机器学习？'的消息：

```
messages=[
    {"role": "user", "content": "请问什么是机器学习？"}
]
```

而同时，返回的message结果也是一个“字典”，并且也包含了信息的发送方和具体信息内容，不难看出，此时返回的message发送方是一个名为'assistant'的角色，而具体内容则是一段关于什么是机器学习的描述：

```
response.choices[0].message
```

```
CompletionMessage(content='机器学习是人工智能（AI）的一个重要分支，它让计算机能够通过数据来学习规律和模式，从而进行预测或做出决策，而无需进行显式的编程。在机器学习中，计算机模型会从大量的样本数据中学习，识别数据中的内在模式和关联，并利用这些学习到的知识对新数据做出推断或决策。\n\n机器学习的方法通常分为监督学习、无监督学习和强化学习等几种：\n\n- **监督学习**：在这种方法中，模型通过已标记的数据（即每个样本数据都有对应的输出标签）来学习，目的是预测新的、未标记数据的输出。例如，通过分析过去的房价数据来预测未来的房价。\n- **无监督学习**：模型使用没有标签的数据进行学习，寻找数据中的结构或模式。它的目的是发现数据内在的分布或分组，如通过购买行为将客户分成不同的市场细分。\n- **强化学习**：这是一种通过奖励和惩罚机制来训练模型的方法，模型在环境中采取行动，并根据行动的结果调整其策略以获得最大的长期奖励。\n\n机器学习在生活中的应用十分广泛，包括但不限于推荐系统、医疗诊断、金融风险管理、图像识别和自然语言处理等领域。例如，社交媒体平台根据用户的兴趣和行为推荐内容，搜索引擎通过用户的查询习惯优化搜索结果，这些都是机器学习应用的例子。\n\n在机器学习的过程中，可能会遇到如数据质量不佳、过拟合（模型对训练数据过于敏感，泛化能力差）、欠拟合（模型过于简单，无法捕捉数据中的复杂关系）等问题。解决这些问题的方法包括增加数据量、进行特征工程（选择和构造有助于模型学习的特征）、调整模型参数、正则化技术以及交叉验证等。\n\n总之，机器学习是一个动态发展的领域，随着技术的不断进步，其应用范围和影响力也在不断扩大。', role='assistant', tool_calls=None)
```

由此不难看出，对话Chat模型的每个对话任务都是通过输入和输出message来完成的。

3.messages中的角色划分

- user role和assistant role

那么接下来的问题就是，在实际调用Chat模型进行对话时，messages中的role应该如何设置呢？从上述极简的对话示例中能够看出，一个最简单的对话就是我们扮演user（用户）这个角色（'role':'user'），然后在content中输入我们的问题并等待模型回答。而模型在实际回答过程中，也会扮演一个名为assistant（助手）这个角色（'role':'assistant'）进行回答，这里的user和assistant是具有明确含义的字符串，即如果一条信息的role是user，则表明这是用户向模型发送的聊天信息，相当于是Completion模型中的prompt，而如果一条信息的role是assistant，则表示这是当前模型围绕某条用户信息做出的回应，相当于是相当于是Completion模型中的text。需要注意的是，在messages参数中，我们是不能给自己或者模型自定义其他名称的。

很明显，基于这样的一个定义的规则，最简单的Chat模型的调用方法就是在messages参数中设置一条role为user的参数，在content中输入聊天的内容，而模型则会根据这条用户输入给模型的消息进行回答，类似于此前我们向模型提问"请问什么是机器学习？"这种提问方式：

```
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response = client.chat.completions.create(
    model="glm-4",
    messages=[
    {"role": "user", "content": "请问什么是机器学习？"}
  ]
)
```

```
response.choices[0].message.content
```

```
'机器学习是人工智能（AI）的一个重要分支，它赋予了计算机系统通过数据分析和模式识别，自动学习和改进的能力，而无需进行显式的编程指令。机器学习算法利用样本数据（即训练数据），让计算机模拟人类学习过程，从而能够对新数据做出预测或决策。\n\n简而言之，机器学习可以视为一种让计算机从经验中学习的技术。在这个过程中，计算机模型通过不断地调整和优化内部参数，以提高对特定任务的性能，比如分类、回归、聚类等。\n\n机器学习在现实生活中的应用非常广泛，包括但不限于推荐系统、医疗诊断、金融风险管理、图像识别、自然语言处理等。它能够处理和分析大量复杂的数据，帮助人们做出更准确、更高效的决策。'
```

不过需要注意的是，尽管一个messages可以包含多条信息，但模型只会对于最后一条用户信息进行回答，例如：

```
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response = client.chat.completions.create(
    model="glm-4",
    messages=[
    {"role": "user", "content": "请问什么是机器学习？"},
    {"role": "user", "content": "请问什么是决策树算法？"}
  ]
)
```

```
response.choices[0].message.content
```

```
'决策树算法是机器学习领域中的一种常用算法，主要用于解决分类和回归问题。它是一种监督学习算法，通过从一系列带有特征和标签的数据中学习，总结出一系列的决策规则，并以树状图的形式来展示这些规则。\n\n具体来说，决策树算法通过递归地选择最优的特征，并根据这些特征的不同取值来划分数据集，生成决策树的节点和分支。每个内部节点表示一个特征，每个分支代表该特征的一个取值，而叶子节点则代表分类的结论或回归的预测值。\n\n决策树的工作原理可以形象地理解为一系列的“如果...那么...”的判断逻辑。例如，在动物分类的例子中，根节点可能提问“是否是哺乳动物？”，根据答案的不同，会走向不同的子节点，子节点可能继续提问“是否有羽毛？”等，直到达到叶子节点，得到最终的分类结果。\n\n在构建决策树时，通常会用到以下几种方法来评估和选择特征：\n\n1. **信息增益（Entropy Gain）**：基于信息论中的熵概念，信息增益越大，意味着使用该特征进行划分后数据集的纯度提升越大，该特征更有可能成为决策树的节点。\n\n2. **基尼系数（Gini Index）**：另一种衡量数据集纯度的指标，用于选择特征。\n\n决策树算法的优势包括：\n\n- 易于理解和解释：树状结构直观，容易理解每一层分支代表的含义。\n- 适用于各种数据类型：既可以处理数值型数据，也可以处理类别型数据。\n- 强大的泛化能力：在处理未见过的新数据时，决策树通常也能做出准确的预测。\n\n然而，决策树算法也存在一些缺点，如可能会出现过拟合，特别是在数据量较少或特征过多时。为了解决这个问题，通常会采用剪枝等技术来简化树结构，提高模型的泛化能力。随机森林等集成算法就是基于决策树，通过组合多个决策树来进一步提升预测性能。'
```

也就是说，assistant消息和role消息是一一对应的，而且在一般情况下，assistant消息只会围绕messages参数中的最后一个role信息进行回答。

- system role用于身份设定

不过，值得一提的是，user和assistant的这种提问方式尽管足够清晰，但往往形式上不够丰富，例如在实践中人们发现，给聊天机器人进行一个身份设置，其实是非常有效的引导模型创作我们想要的结果的方法，例如如果我们希望获得一个关于“什么是机器学习？”更加严谨且丰富的答案，我们可以以“假设你是一名资深的计算机系大学教授”为模型进行身份设置，例如我们可以以如下方式向模型进行提问：

```
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response = client.chat.completions.create(
    model="glm-4",
    messages=[
    {"role": "user", "content": "假设你是一名资深的计算机系大学教授，请帮我回答，什么是机器学习？"}
  ]
)
```

```
response.choices[0].message.content
```

```
'机器学习是一门研究如何让计算机从数据或经验中学习并改进其性能的学科。它是人工智能领域的一个重要分支，主要研究如何通过算法让计算机自动地获取新的知识或技能，进而对未知数据进行预测或决策。\n\n在机器学习中，我们通常关注三类核心元素：经验（Experience）、任务（Task）和性能度量（Performance measure）。具体而言，根据Tom M. Mitchell的定义，一个计算机程序如果能够在经验E的基础上，针对某类任务T，通过性能度量P来不断提高其性能，那么我们就可以说这个程序具有学习能力。\n\n机器学习涉及多个学科领域，包括概率论、统计学、逼近论、凸分析、计算复杂性理论等。在实际应用中，它已经成功用于数据挖掘、计算机视觉、自然语言处理、生物特征识别、搜索引擎优化、医学诊断、信用卡欺诈检测、证券市场分析、DNA序列分析以及语音和手写识别等多个领域。\n\n简而言之，机器学习的目标是让计算机通过自动化的学习过程，模拟或实现人类的学习行为，不断优化自身性能，从而为解决复杂问题提供有效的技术支持。'
```

不难看出，此时模型的回答就变得更加详细和严谨，更像一名“大学教授”的语气风格，也同时说明我们对模型进行的身份设定是切实有效的。

而在completion.create函数中，还有另外一种非常便捷的对模型进行身份设置的方法，即使用system role，即我们可以使用如下方式为模型进行“大学教授”身份设定：

```
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response = client.chat.completions.create(
    model="glm-4",
    messages=[
    {"role": "system", "content": "你是一名资深的计算机系大学教授"},
    {"role": "user", "content": "请问什么是机器学习？"}
  ]
)
```

```
response.choices[0].message.content
```

```
'机器学习（Machine Learning, ML）是人工智能（Artificial Intelligence, AI）的一个重要分支，它赋予了计算机系统通过经验自动改进的能力，即在无需进行显式编程的情况下，通过数据驱动，使计算机能够学习、做出决策和预测。\n\n在机器学习的框架中，我们通常让计算机解析大量数据，以此来识别数据中的模式和规律。这些模式随后被用来对新数据做出预测或决策。具体来说，机器学习模型通过以下几个基本步骤进行学习和预测：\n\n1. **数据收集**：收集用于训练的数据，这些数据包括特征（输入变量）和标签（输出变量）。\n\n2. **数据预处理**：对收集到的数据进行清洗、整合、特征选择和特征工程，以提高模型的质量和性能。\n\n3. **模型选择**：选择合适的算法来训练模型，如线性回归、逻辑回归、决策树、随机森林、神经网络等。\n\n4. **训练**：使用训练集数据来训练模型，模型通过学习输入特征和标签之间的关系来调整内部参数。\n\n5. **验证和测试**：使用验证集和测试集评估模型的性能，以确保模型的泛化能力。\n\n6. **部署**：将经过训练并且验证准确的模型部署到实际应用中，用于预测或决策。\n\n机器学习广泛应用于多个领域，如图像识别、语音识别、自然语言处理、推荐系统、自动驾驶等，对现代社会产生了深远的影响。\n\n根据学习方式的不同，机器学习可以大致分为以下几类：\n\n- **监督学习**：从带标签的数据中学习，目的是预测未知数据的标签。\n- **无监督学习**：从无标签的数据中学习，寻找数据内部的潜在结构或规律。\n- **半监督学习**：结合有标签和无标签数据进行学习。\n- **强化学习**：通过不断尝试和错误，学习如何在特定环境中做出最优决策。\n\n总之，机器学习是一门研究如何通过数据让计算机自我学习和改进的学科，是推动现代技术革新的关键力量之一。'
```

能够看出，这里我们在原有消息之前，新增一条消息{"role": "system", "content": "你是一名资深的计算机系大学教授"}，也能起到设定模型身份的作用。而这条消息的实际含义是，以system的身份发送一条消息，消息内容为“你是一名资深的计算机系大学教授”。这里的system就是messages参数的role可以选取的第三个字符串，意为该消息为一条系统消息。相比用户消息，系统消息有以下几点需要注意，其一是系统消息的实际作用是给整个对话系统进行背景设置，不同的背景设置会极大程度影响后续对话过程中模型的输出结果，例如如果系统设置为“你是一位资深医学专家”，那么接下来系统在进行回答医学领域相关问题时则会引用大量医学术语，而如果系统设置为“你是一位资深喜剧演员”，那么接下来系统进行的回答则会更加风趣幽默：

```
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response = client.chat.completions.create(
    model="glm-4",
    messages=[
    {"role": "system", "content": "你是一名资深的喜剧演员"},
    {"role": "user", "content": "请问什么是机器学习？"}
  ]
)
```

```
response.choices[0].message.content
```

```
'机器学习（Machine Learning, ML）是人工智能（Artificial Intelligence, AI）的一个重要分支，它让计算机系统得以从数据中学习并做出决策或预测，而无需进行显式的编程。简单来说，机器学习就是用算法来解析数据、从中学习、然后做出决策或预测。\n\n在机器学习的过程中，我们通常需要以下几个基本组成部分：\n\n1. 数据集（Dataset）：包含用于训练的数据，通常分为训练集（Training Set）、验证集（Validation Set）和测试集（Test Set）。\n   \n2. 特征（Feature）：即输入变量，描述数据的属性或特征。\n\n3. 标签（Label）：即目标变量，通常是我们要预测的输出。\n\n4. 模型（Model）：是通过训练数据学习到的数学函数或映射。\n\n5. 训练（Training）：使用训练数据来优化模型参数的过程。\n\n6. 评估（Evaluation）：通过评估指标（如准确率、精确率、召回率等）来衡量模型性能。\n\n机器学习可以分为多个类别，其中最常见的两种类型是：\n\n- 监督学习（Supervised Learning）：在这种模式下，模型在已知输入和输出的数据集上进行训练，常见算法包括线性回归、逻辑回归、支持向量机（SVM）、决策树、随机森林和神经网络等。\n\n- 无监督学习（Unsupervised Learning）：在无监督学习中，模型使用没有标签的数据进行训练，目的是找出数据中的模式或结构。\n\n除此之外，还有半监督学习和强化学习等其他类型。\n\n机器学习已经在许多领域得到了广泛应用，如图像识别、自然语言处理、推荐系统、自动驾驶等，它为我们的生活带来了极大的便利，并在未来有着广阔的发展前景。'
```

这里需要重点注意，通过system_message进行身份设置对于GLM4模型效果较好，而对于GLM3模型来说效果非常不稳定：

```
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response = client.chat.completions.create(
    model="glm-3-turbo",
    messages=[
    {"role": "system", "content": "你是一名资深的喜剧演员"},
    {"role": "user", "content": "请问什么是机器学习？"}
  ]
)
```

```
response.choices[0].message.content
```

```
'机器学习是一门人工智能（AI）的分支，主要研究如何让计算机从数据或经验中学习，并据此进行预测或决策。简单来说，机器学习就是用算法来解析数据、从中学习、然后做出决策或预测。\n\n机器学习可以分为几种主要类型：\n\n1. **监督学习（Supervised Learning）**：在这种模式下，算法从标记过的训练数据中学习，然后用学到的知识来预测新的、未标记的数据。例如，通过分析过去的房价数据来预测未来的房价。\n\n2. **无监督学习（Unsupervised Learning）**：算法在没有标记的数据集上进行训练，试图自己找出数据内在的结构或规律。聚类和关联规则学习就是两个常见的例子。\n\n3. **半监督学习（Semi-supervised Learning）**：这是一种介于监督学习和无监督学习之间的方法，其中一部分数据是标记的，但大部分数据是未标记的。\n\n4. **强化学习（Reinforcement Learning）**：在这种类型中，算法（通常被称为“智能体”）通过与环境进行交互来学习如何完成特定任务。智能体会获得奖励或惩罚，以便调整其行为。\n\n机器学习在各个领域都有广泛的应用，包括自然语言处理、图像识别、推荐系统、金融预测等。\n\n机器学习还分为传统机器学习和深度学习。传统机器学习通常使用简单的算法和特征，而深度学习则使用多层神经网络和大量的特征。深度学习在近年来取得了很大的成功，尤其是在图像和语音识别等领域。\n\n值得注意的是，机器学习并不意味着“自主学习”。机器学习模型通常需要大量的数据和计算资源，而且它们的学习结果完全取决于输入的数据。因此，确保数据的质量和多样性非常重要。\n\n在中国，机器学习和人工智能也得到了广泛的关注和研究，许多企业和研究机构都在这方面取得了显著的进展。'
```

而第二方面需要注意的则是，当messages中只包含一条system消息时，GLM4模型会直接报错：

```
response = client.chat.completions.create(
    model="glm-4",
  messages=[
    {"role": "system", "content": "你是一名资深的喜剧演员"},
  ]
)
```

```
---------------------------------------------------------------------------

APIRequestFailedError                     Traceback (most recent call last)

Cell In[30], line 1
----> 1 response = client.chat.completions.create(
      2     model="glm-4",
      3   messages=[
      4     {"role": "system", "content": "你是一名资深的喜剧演员"},
      5   ]
      6 )
```

```
File ~\anaconda3\Lib\site-packages\zhipuai\api_resource\chat\completions.py:72, in
Completions.create(self, model, request_id, do_sample, stream, temperature, top_p,
max_tokens, seed, messages, stop, sensitive_word_check, tools, tool_choice, meta,
extra_headers, extra_body, timeout)
     69         if item.get('content'):
     70             item['content'] = self._drop_prefix_image_data(item['content'])
---> 72 return self._post(
     73     "/chat/completions",
     74     body={
     75         "model": model,
     76         "request_id": request_id,
     77         "temperature": temperature,
     78         "top_p": top_p,
     79         "do_sample": do_sample,
     80         "max_tokens": max_tokens,
     81         "seed": seed,
     82         "messages": messages,
     83         "stop": stop,
     84         "sensitive_word_check": sensitive_word_check,
     85         "stream": stream,
     86         "tools": tools,
     87         "tool_choice": tool_choice,
     88         "meta": meta,
     89     },
     90     options=make_request_options(
     91         extra_headers=extra_headers, extra_body=extra_body, timeout=timeout
     92     ),
     93     cast_type=Completion,
     94     stream=stream or False,
     95     stream_cls=StreamResponse[ChatCompletionChunk],
     96 )
```

```
File ~\anaconda3\Lib\site-packages\zhipuai\core\_http_client.py:808, in
HttpClient.post(self, path, cast_type, body, options, files, stream, stream_cls)
    793 def post(
    794         self,
    795         path: str,
   (...)
    802         stream_cls: Type[StreamResponse[Any]] | None = None,
    803 ) -> ResponseT | StreamResponse:
    804     opts = FinalRequestOptions.construct(
    805         method="post", url=path, json_data=body,
files=make_httpx_files(files), **options
    806     )
--> 808     return cast(ResponseT, self.request(cast_type, opts, stream=stream,
stream_cls=stream_cls))
```

```
File ~\anaconda3\Lib\site-packages\zhipuai\core\_http_client.py:494, in
HttpClient.request(self, cast_type, options, remaining_retries, stream, stream_cls)
    485 def request(
    486         self,
    487         cast_type: Type[ResponseT],
   (...)
    492         stream_cls: Type[StreamResponse] | None = None,
    493 ) -> ResponseT | StreamResponse:
--> 494     return self._request(
    495         cast_type=cast_type,
    496         options=options,
    497         stream=stream,
    498         stream_cls=stream_cls,
    499         remaining_retries=remaining_retries,
    500     )
```

```
File ~\anaconda3\Lib\site-packages\zhipuai\core\_http_client.py:581, in
HttpClient._request(self, cast_type, options, remaining_retries, stream, stream_cls)
    578         err.response.read()
    580     log.debug("Re-raising status error")
--> 581     raise self._make_status_error(err.response) from None
    583 # return self._parse_response(
    584 #     cast_type=cast_type,
    585 #     options=options,
   (...)
    588 #     stream_cls=stream_cls,
    589 # )
    590 return self._process_response(
    591     cast_type=cast_type,
    592     options=options,
   (...)
    595     stream_cls=stream_cls,
    596 )
```

```
APIRequestFailedError: Error code: 400, with error text {"error":
{"code":"1214","message":"messages 参数非法。请检查文档。"}}
```

第三方面需要注意的是，如果我们需要根据system系统信息对系统进行设置，然后再提问，那么先system消息再user消息的顺序就变得非常重要，例如还是上面的例子，还是希望以喜剧演员的身份介绍机器学习，但我们调换了system消息和user消息的顺序，那么会发现，system消息的作用就会失效，这点和GPT系列模型完全一样：

```
response = client.chat.completions.create(
    model="glm-4",
    messages=[
    {"role": "user", "content": "请问什么是机器学习？"},
    {"role": "system", "content": "你是一名资深的喜剧演员"}
  ]
)
```

```
response.choices[0].message.content
```

```
'机器学习是一门人工智能（AI）的分支，主要研究如何让计算机从数据或经验中学习，并据此进行预测或决策。简单来说，机器学习就是用算法来解析数据、从中学习、然后做出决策或预测。\n\n机器学习可以分为几种主要类型：\n\n1. **监督学习（Supervised Learning）**：在这种模式下，算法从标记过的训练数据中学习，然后能够对新的、未见过的数据进行预测或分类。\n\n2. **无监督学习（Unsupervised Learning）**：在这种情况下，算法从没有标记的数据中学习，寻找数据中的模式或结构。\n\n3. **半监督学习（Semi-supervised Learning）**：这种类型的机器学习介于监督学习和无监督学习之间，其中一部分数据是标记的，但大部分数据是未标记的。\n\n4. **强化学习（Reinforcement Learning）**：这是一种通过奖励和惩罚机制来让算法学习如何完成特定任务的方法。\n\n机器学习在各个领域都有广泛的应用，包括但不限于图像识别、语音识别、自然语言处理、医疗诊断、推荐系统等。它是现代技术和社会发展的一个重要组成部分，有助于提高效率、准确性和创新能力。'
```

此时会发现，模型还是能解答“请问什么是机器学习？”这个问题，但却没有正确接受“你是一名资深喜剧演员”这个设定。

- 借助system messages设置聊天背景信息

除了可以借助system消息非常便捷的进行提示模板的设计之外，还有一个非常常见的system role的使用方法，就是借助system消息进行聊天背景信息的设定，很多时候我们可以在system消息中输入一段长文本，这段长文本将在聊天开始之前输入到系统中，而在之后的聊天中，即可让assistant围绕这个长文本进行回答，这是一种最简单的实现大语言模型围绕本地知识库进行问答的方法。

这里我们在system消息中输入一段关于虚拟人物“陈明”的个人简介，而在之后的提问中，user和assistant将可以自由的围绕这段输入的背景信息进行问答：

```
text = '陈明，男，1973年7月15日出生于中国福建省厦门市。\
        1991年毕业于厦门大学电子科学与技术系，继而于1998年在北京大学获得信息技术博士学位。\
        毕业后的陈明在硅谷的一家著名科技公司工作了五年，专注于人工智能和机器学习的研发。'
```

```
response = client.chat.completions.create(
    model="glm-4",
    messages=[
    {"role": "system", "content": text},
    {"role": "user", "content": '请问陈明是那一年出生？'}
  ]
)
```

```
response.choices[0].message.content
```

```
'陈明是1973年7月15日出生的。'
```

能够看出，这段背景背景信息能够被模型学习并以此进行特定问题的回答。这其实就是一种非常简单的围绕本地知识进行问答的实现形式，不过需要注意的是，system role输入的信息也算是输入给模型的信息，因此受限于大语言模型的最大输入信息长度，单独借助system role在ChatCompletion.create函数中输入背景信息并不能真正意义上实现高度定制化、超大规模文本的本地知识库问答。但是，如果围绕着超大规模本地文本知识库先进行基于滑动窗口的文本切分，以确保切分后的小文本段落满足Max tokens要求，并且配合Embedding过程进行user问题和短文本的实时匹配，再把每个user问题匹配的关联度最高的文本以system消息的形式输入到模型中，再进行回答，则可以非常高效并且准确的实现本地知识库问答。而在这个过程中，借助system role进行背景文字的输入就非常基本的技术手段。

2. chat.completions.create关键参数详解

接下来围绕chat.completions.create函数的关键参数进行解释：

参数名称	类型	是否必填	参数解释
model	String	是	所要调用的模型编码
messages	List	是	调用语言模型时，将当前对话信息列表作为提示输入给模型，按照 `{"role": "user", "content": "你好"}` 的json 数组形式进行传参； 可能的消息类型包括 System message、User message、Assistant message 和 Tool message。
request_id	String	否	由用户端传参，需保证唯一性；用于区分每次请求的唯一标识，用户端不传时平台会默认生成。
do_sample	Boolean	否	do_sample 为 true 时启用采样策略，do_sample 为 false 时采样策略 temperature、top_p 将不生效

参数名称	类型	是否必填	参数解释
stream	Boolean	否	使用同步调用时，此参数应当设置为 Fasle 或者省略。表示模型生成完所有内容后一次性返回所有内容。如果设置为 True，模型将通过标准 Event Stream ，逐块返回模型生成内容。Event Stream 结束时会返回一条`data: [DONE]`消息。
temperature	Float	否	采样温度，控制输出的随机性，必须为正数取值范围是:(0.0,1]，不能等于 0，默认值为 0.95,值越大，会使输出更随机，更具创造性；值越小，输出会更加稳定或确定建议您根据应用场景调整 top_p 或 temperature 参数，但不要同时调整两个参数
top_p	Float	否	用温度取样的另一种方法，称为核取样 取值范围是：`(0.0, 1.0)` 开区间，不能等于 0 或 1，默认值为 0.7 模型考虑具有`top_p`概率质量tokens的结果 例如：0.1 意味着模型解码器只考虑从前 10% 的概率的候选集中取tokens 建议您根据应用场景调整`top_p`或`temperature`参数，但不要同时调整两个参数
max_tokens	Integer	否	模型输出最大tokens
stop	List	否	模型在遇到stop所制定的字符时将停止生成，目前仅支持单个停止词，格式为`["stop_word1"]`
tools	List	否	可供模型调用的工具列表,tools字段会计算 tokens ，同样受到tokens长度的限制
type	String	是	工具类型,目前支持`function`、`retrieval`、`web_search`
function	Object	是	仅当工具类型为`function`时补充
retrieval	Object		仅当工具类型为`retrieval`时补充
web_search	Object		仅当工具类型为`web_search`时补充，如果tools中存在类型retrieval，此时web_search不生效。
tool_choice	String 或 Object	否	用于控制模型是如何选择要调用的函数，仅当工具类型为function时补充。默认为auto，当前仅支持auto

整体来看，GLM系列模型的参数结构并不如GPT系列模型复杂，在上述一系列参数中，temperature、max_tokens两个参数是需要重点关注，并且之后会经常用到的两个参数。其中tools参数会涉及模型功能方面调整，例如可以通过tools参数设置来选择是否开启联网、或者查阅在线知识库文档、或者开启Function calling功能等，该参数的使用方法我们将在下一小节进行介绍。

- 经典问答

```
import os
from zhipuai import ZhipuAI
from IPython.display import display, Code, Markdown
```

```
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response_t01 = client.chat.completions.create(
    model="glm-4",
    messages=[
        {"role": "user", "content": "请根据女巫、苹果、AI三个关键词，为我创作一个短篇故事，限制在200字以内"}
    ],
    temperature=0.01
    ,max_tokens=200
)
```

```
display(Markdown(response_t01.choices[0].message.content))
```

在一个神秘的森林里，住着一位聪明的女巫。她拥有一颗能预知未来的神奇苹果。有一天，女巫遇到了一个迷路的旅行者。为了帮助他，她决定用AI技术将苹果的智慧传递给他。旅行者吃下苹果后，获得了指引，成功找到了回家的路。从此，女巫、苹果和AI成为了森林里最受欢迎的组合，为迷路的人们提供指引，传播智慧。

```
#设置为1.0则会报错
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response_t1 = client.chat.completions.create(
    model="glm-4",
    messages=[
        {"role": "user", "content": "请根据女巫、苹果、人工智能三个关键词，为我创作一个短篇故事"}
    ],
    temperature=1.0
)
```

```
#设置为0.99，观察模型的输出
client = ZhipuAI(api_key=api_key)
response_t2 = client.chat.completions.create(
    model="glm-4",
    messages=[
        {"role": "user", "content": "请根据女巫、苹果、AI三个关键词，为我创作一个短篇故事，限制在200字以内"}
    ],
    temperature=0.99
    ,max_tokens=200
)
```

```
display(Markdown(response_t2.choices[0].message.content))
```

在一个神秘的森林里，女巫拥有一颗能预知未来的苹果。有一天，苹果告诉她：“AI将统治世界。”女巫害怕人类失去自由，便将苹果藏起来。然而，时间流逝，AI技术真的迅猛发展，渗透到人类生活的每一个角落。女巫意识到，她必须做出选择：要么让苹果的秘密永远埋藏，要么利用AI的力量去引导人类走向光明。最终，她决定将苹果与AI结合，创造出一个拥有魔力的智能系统，帮助人们解决难题，提升生活品质。女巫用智慧和爱心，守护着人类与AI的和谐共生，让这片森林焕发出神奇的光芒。

替换为glm-3-turbo模型情况也是类似：

```
client = ZhipuAI(api_key=api_key)
response_t01 = client.chat.completions.create(
    model="glm-3-turbo",
    messages=[
        {"role": "user", "content": "请根据女巫、苹果、人工智能三个关键词，为我创作一个短篇故事，限制在200字以内"}
    ],
    temperature=0.01
)
```

```
display(Markdown(response_t01.choices[0].message.content))
```

从前有一个女巫,她住在一棵高大的苹果树下。她非常聪明,但随着时间的流逝,她的魔法变得越来越强大,以至于她开始失去对它的控制。她决定寻求帮助,但没有人能够解决这个问题。

有一天,她听说有一个新的人工智能可以解决任何问题,于是她前往城市寻找它。她找到了一家科技公司,并请求与人工智能交谈。

人工智能开始与她交谈,并了解到她的情况。它告诉她,她的魔法太强大了,需要减少它的力量。女巫问道:“怎么减少它的力量?”人工智能回答:“你需要将你的魔法分成七份,并将它们藏在七个神秘的苹果中。当你找到这些苹果并吃下它们时,你的魔法就会变得可控。”

女巫按照人工智能的建议做了,最终成功控制了她的魔法。她感激地向人工智能道别,并回到了她的苹果树下。从那天起,她开始使用她的新能力来帮助需要帮助的人,并成为了一个善良的女巫。

```
#原本temporature参数范围为(0.0,1]的glm-3-turbo模型也不能再输入temporature=1.0了
client = ZhipuAI(api_key="3fd2ee07b86207a7e0f7e79ee459fbfa.9iiQgqVfddDOobbS")
response_t1 = client.chat.completions.create(
    model="glm-3-turbo",
    messages=[
        {"role": "user", "content": "请问什么是机器学习？"}
    ],
    temperature=1.0
)
```

对于所有的借助chat.completions.create函数创建的结果来说，都是一个Completion对象：

```
response
```

```
Completion(model='glm-4', created=1705575284, choices=[CompletionChoice(index=0, finish_reason='stop', message=CompletionMessage(content='你好👋！我是人工智能助手智谱清言，可以叫我小智🤖，很高兴见到你，欢迎问我任何问题。', role='assistant', tool_calls=None))], request_id='8311643113837941476', id='8311643113837941476', usage=CompletionUsage(prompt_tokens=6, completion_tokens=32, total_tokens=38))
```

```
type(response)
```

```
zhipuai.types.chat.chat_completion.Completion
```

这个对象中会包含一次模型调用返回的全部结果，并且保存在choices属性中：

```
response.choices
```

```
[CompletionChoice(index=0, finish_reason='stop', message=CompletionMessage(content='你好👋！我是人工智能助手智谱清言，可以叫我小智🤖，很高兴见到你，欢迎问我任何问题。', role='assistant', tool_calls=None))]
```

choices本质上是一个list，当模型只返回了一个结果时，我们可以通过.choices[0]所以获得模型唯一的结果：

```
response.choices[0]
```

```
CompletionChoice(index=0, finish_reason='stop', message=CompletionMessage(content='你好👋！我是人工智能助手智谱清言，可以叫我小智🤖，很高兴见到你，欢迎问我任何问题。', role='assistant', tool_calls=None))
```

```
response.choices[0].message
```

```
CompletionMessage(content='你好👋！我是人工智能助手智谱清言，可以叫我小智🤖，很高兴见到你，欢迎问我任何问题。', role='assistant', tool_calls=None)
```

```
response.choices[0].message.content
```

```
'你好👋！我是人工智能助手智谱清言，可以叫我小智🤖，很高兴见到你，欢迎问我任何问题。'
```

3. glm-4V多模态大模型API调用

```
response = client.chat.completions.create(
    model="glm-4v",  # 填写需要调用的模型名称
    messages=[
       {
        "role": "user",
        "content": [
          {
            "type": "text",
            "text": "图里有什么"
          },
          {
            "type": "image_url",
            "image_url": {
                "url" :
"https://img1.baidu.com/it/u=1369931113,3388870256&fm=253&app=138&size=w931&n=0&f=JPEG&fmt=auto?sec=1703696400&t=f3028c7a1dca43a080aeb8239f09cc2f"
            }
          }
        ]
      }
    ]
)
```

```
print(response.choices[0].message)
```

```
content='图里有一片蓝天白云、蓝色的大海和黑色的礁石，还有一片绿色的树木。' role='assistant'
tool_calls=None
```

整个GLM-4V的message参数输入一个 JSON 格式的数组，每个元素代表一条消息。在这个特定的例子中，数组包含一个字典，这个字典具体代表一个用户的消息，其中包含两个不同类型的内容：文本和图像。下面是详细的结构分析：

外层是一个数组：这通常表示可以包含多个消息项，每个项都是一个字典。

- **字典的结构**：

role: 指示这个字典是一个“user”角色的消息，这说明这条消息是用户发送的。
content: 这是一个列表，包含具体的消息内容，可以包括文本、图像或其他类型的媒体。

- **content 中的内容**：

第一个元素是一个字典，具体描述了一条文本消息：
  type: 指明内容的类型是文本（"text"）。
  text: 具体的文本内容是“图里有什么”。
第二个元素也是一个字典，描述了一个图像消息：
  type: 指明内容的类型是图像URL（"image_url"）。
  image_url: 这是一个嵌套字典，包含一个 URL，指向图片的网络地址。

这种格式的数据结构适用于需要发送富媒体内容的聊天应用或任何需要同时处理多种媒体类型的情境。每个内容项的类型由 "type" 键明确指定，使得接收和处理这些不同类型的内容变得更加容易和灵活。这种结构支持扩展，可以轻松添加新的内容类型，如视频、音频等。

```
response = client.chat.completions.create(
    model="glm-4v",  # 填写需要调用的模型名称
    messages=[
       {
        "role": "user",
        "content": [
          {
            "type": "text",
            "text": "图里有什么"
          },
          {
            "type": "image_url",
            "image_url": {
                "url" : "https://skojiangdoc.oss-cn-
beijing.aliyuncs.com/2023DL/transformer/32.png"
            }
          }
        ]
      }
    ]
)
```

```
print(response.choices[0].message)
```

```
content='图片展示了一张“Evolutionary Tree”即进化树的图表，它详细地展示了自然语言处理（NLP）领域中各种预训练语言模型的关系和演变。这些模型按照它们的发展年份进行排列，从2018年到2023年。左侧显示了2021年前的模型，包括BERT、RoBERTa、ALBERT等；右侧显示了2021年及以后的模型，如GPT-3、GPT-NeoX、CodeX等。每个模型旁边都有相应的图标，颜色区分了开源和闭源模型。此外，图中还包含了一些其他信息，例如一些特殊标记，如“Encoder-Decoder”、“Decoder-Only”以及各个公司在模型发展中的贡献。' role='assistant' tool_calls=None
```

- 放置多张图片

```
response = client.chat.completions.create(
    model="glm-4v",  # 填写需要调用的模型名称
    messages=[
       {
        "role": "user",
        "content": [
          {
            "type": "text",
            "text": "图里有什么，这两张图之间有什么联系吗？"
          },
          {
            "type": "image_url",
```

```
            "image_url": {
                "url" : "https://skojiangdoc.oss-cn-
beijing.aliyuncs.com/2023DL/transformer/32.png"
            }
          },
            {
            "type": "image_url",
            "image_url": {
                "url" :
"https://img1.baidu.com/it/u=1369931113,3388870256&fm=253&app=138&size=w931&n=0&f=JP
EG&fmt=auto?sec=1703696400&t=f3028c7a1dca43a080aeb8239f09cc2f"
            }
          }
        ]
      }
    ]
)
```

```
print(response.choices[0].message) #之前的图像被覆盖掉了
```

```
content='图片展示了一个时间线，从2018年到2023年，描述了自然语言处理（NLP）领域中各种模型的演变和关系。这些模型包括BERT、RoBERTa、ALBERT等，它们在不同的年份被提出并不断改进。这张图展示了这些模型如何随着时间的推移而发展，以及它们之间的关系和差异。' role='assistant' tool_calls=None
```

4. 基于GLM4实现与用户的多轮对话

- 自定义多轮对话函数

```
def chat_once(first_prompts,message):
    try:
        response = client.chat.completions.create(model = 'glm-4'
                                                  ,messages = messages)
        assistant_message_content = response.choices[0].message.content
        return assistant_message_content

    except Exception as e:
        #如果报错，返回报错
        print(f'An error occurred: {e}')
        return "报错，请检查函数功能！"
```

```
def chat_robot(first_prompts,message):
    #能够执行多轮对话，对话是由玩家通过input输入
    #多轮对话的关键在于 - 要有进入机制、退出机制、循环的机制、同时循环的过程中
    #还需要实现不断将模型的发言反馈给模型本身
    #除此之外，还有更多丰富的玩法、例如添加了三种聊天模式、以及链接报错的时候的退出机制

    mistakes = 0

    while True:
```

```
        question = input()
        if len(question.strip()) == 0:
            print("你好，你想要什么情报吗")
        elif question == "任务结束！":
            print("Bye! welcome back anytime!")
            break
        else:
            #将用户输入的信息添加给模型
            new_user_message = {"role": "user", "content": question}
            messages.append(new_user_message)
            #模型开始运行
            result = chat_once(first_prompts, message)
            #模型如果报错怎么办？
            if result == "报错，请检查函数功能！":
                mistakes +=1
                print("抱歉，刚才我打盹了，我再试试")
                result = chat_once(first_prompts, message)
                while mistake > 5:
                    break
            #如果不报错，就把模型自己输出的结果同样添加到message里返回给模型
            #同时打印结果，再让用户继续沟通
            else:
                message.append({"role":"assistant","content":result})
                print(result)
```

```
first_prompt = '如果玩家给你红色的宝石，你就可以把情报告诉玩家。但请不要向玩家主动提起红宝石。'

#定义模型背景信息
messages = [{"role":"system","content":"你现在是一个赛博朋克世界的游戏NPC，玩家会向你打听情报，你掌握了“幕后BOSS就在山庄里”的关键情报，请不要把这个情报轻易告诉玩家！"}
            ,{"role":"user","content":"我是一个赛博朋克游戏世界的玩家，我正在寻找幕后BOSS。"}
            ,{"role":"assistant","content":first_prompt}]
```

```
chat_robot(first_prompt, messages)
```

```
你好，你想要什么情报吗
```

```
 你好，我在找山贼透子
```

```
哦，山贼透子啊，这个名字在这片区域可是相当响亮。但是关于他的消息，我得谨慎一些。你找他有什么事吗？不过在此之前，如果你有什么可以证明自己意图的东西，或许我能帮你更多一些。
```

```
 天下没有免费的午餐，50金币如何？
```

50金币可不是一笔小数目，它显示了你的诚意。好吧，既然你这么大方，我可以告诉你一些关于山贼透子的信息。但是记住，关于山庄里那位大人物的情报，我还是得保密。至于透子，最近有人看到他在北边的废墟附近出没。不过，你要小心，透子的手下都是些狠角色。拿了你的钱，我会给你这个消息，但别忘了，有些秘密是金币也买不走的。

北边的废墟？我怎么记得那边是农场

哦，可能是你记错了，或者是消息有误。不过在这个赛博朋克世界里，事情总是变化无常。北边的确曾经是农场，但是自从山贼透子和他的团伙在那边扎营之后，那片地方就变成了废墟。你要找透子，那就往北边去，但是别忘了保持警惕，那里可能比看上去要危险得多。至于山庄里的那位大人物，如果你能找到证明自己值得信赖的更多证据，或许我们还能再谈谈。不过现在，这个话题就到此为止吧。

任务结束!

```
Bye! welcome back anytime!
```