# 1.常用的大模型微调方法

## 1.Prompt Engineering

### 简介

Prompt engineering 主要是指设计有效的提示（prompt）来指导预训练语言模型的行为，从而使其产生特定类型的输出。这种方法不需要对模型本身的参数进行任何更改或微调。

### 流程

1. **定义目标**：明确您希望通过模型完成的具体任务，比如生成故事、翻译文本、解答问题等。确定模型输出的质量标准，比如准确性、流畅性、创造性等。
2. **设计初始提示**：根据任务类型创建一个基本的提示模板。初始提示应该足够清晰，以便模型能够理解任务的要求。
3. **添加示例**：在提示中加入一些示例输入和对应的预期输出，以帮助模型更好地理解任务。示例可以是实际的案例或者是虚构的例子，但要确保它们与任务紧密相关。
4. **测试和评估**：使用设计好的提示向模型发出请求，并评估模型的响应是否符合预期。分析模型输出的质量和一致性。
5. **迭代改进**：根据测试结果调整提示，比如增加或修改示例、明确特定的格式要求等。持续迭代直至达到满意的性能水平。
6. **优化提示结构**：探索不同的提示结构，比如使用指令词、特定的语气、格式化标记等。实验不同的提示长度，有时简短的提示比冗长的提示更有效。
7. **利用反馈**：如果可能的话，收集用户的反馈来进一步改进提示。也可以通过A/B测试不同版本的提示来确定哪个版本效果最好。
8. **部署**：将最终优化的提示用于实际应用中。监控模型的表现，并准备在未来根据需要继续调整提示。
9. **持续监控和更新**：随着时间的推移，语言模型可能会更新，因此需要定期检查提示的有效性。如果发现模型的表现下降，或者有新的任务出现，应及时调整提示。
10. **文档记录**：记录下所有尝试过的提示版本以及相应的性能指标。这样可以方便以后参考，并且有助于团队成员之间的知识共享。

### 优势

**成本效益**：不需要额外的数据标注或模型训练，减少了时间和金钱成本。可以迅速测试和部署，适用于快速原型设计和迭代。

**灵活性**：可以灵活地改变提示的内容来适应不同的任务需求。通过修改提示中的指令或例子，可以轻松地调整模型的行为。

**易于实现**：相较于微调模型，prompt engineering 更简单易行。不需要深入理解模型内部机制就能开始使用。

**减少过拟合风险**：由于不需要对模型进行额外训练，因此可以减少过拟合的风险。

特别是在小样本数据集上表现更佳。

**快速迭代**：可以快速测试不同的提示变体，以找到最有效的版本。这种迭代速度对于产品开发非常重要。

**适应性**：可以通过调整提示来适应不同的应用场景和目标用户群体。有助于定制化服务，满足特定领域的需求。

**促进多模态任务**：可以通过构建包含文本和其他模态（如图像）的复杂提示来处理多模态任务。例如，可以通过在提示中加入图像描述来引导模型生成与图像相关的内容。

**扩展模型功能**：通过巧妙设计提示，可以让模型执行原本没有经过训练的任务。例如，让一个语言模型执行数学计算或逻辑推理等。

### 缺点

**难以量化效果**：Prompt engineering 的效果往往依赖于设计者的直觉和经验，很难通过定量分析来评估其优劣。设计良好的提示可以显著改善模型的性能，但设计不佳的提示则可能导致模型输出质量下降。

**缺乏系统性**：提示的设计通常是一个试错的过程，缺乏一套系统的理论或方法论来指导设计。这意味着即使对于类似的任务，设计出的有效提示也可能有很大的差异。

**泛化能力有限**：即使是精心设计的提示，在应用于未见过的新数据或新任务时，其效果可能会大打折扣。提示通常针对特定任务或数据集进行优化，这限制了它们在其他场景下的适用性。

**调试困难**：当提示没有达到预期效果时，很难诊断问题所在。由于提示的设计是非参数化的，因此很难通过简单的调整来解决问题。

**可解释性差**：提示是如何影响模型生成过程的机理往往不够透明。这使得理解和解释模型为何会产生特定输出变得困难。

**效率低下**：对于复杂的任务，可能需要大量的尝试才能找到合适的设计。这种不断试验的方法可能会消耗较多的时间和资源。

**难以自动化**：提示的设计和优化过程很难完全自动化，需要人工干预。这对于大规模应用而言是一个挑战，尤其是在需要频繁更新提示的情况下。

## 2.Prompt-tuning

### 简介

Prompt-tuning 利用提示（prompt）技术修改预训练模型的输入，以引导模型产生期望的结果。这种方法显著提高了模型性能并减少了过度依赖和过拟合的风险。

### 流程

1. **准备数据集**：收集或创建一个包含输入文本和对应期望输出的数据集。数据集中的每一条样本都应该有一个明确的目标输出，用于指导模型生成特定类型的文本。
2. **设计初始提示**：根据任务需求设计一个初始提示模板。这个模板通常包含一些占位符，它们会在训练过程中被具体的文本替换。初始提示的设计非常重要，好的设计可以显著提高最终结果的质量。
3. **构建可学习的参数**：将设计好的提示转换为模型可以理解的形式，通常是一组可学习的向量或者文本序列。这些可学习的参数将会在训练过程中被优化以更好地匹配目标输出。
4. **训练过程**：使用数据集中的样本来训练这些可学习的参数。训练的目标是使模型根据给定的提示生成尽可能接近目标输出的结果。可以使用监督学习方法来实现这一目标，比如最小化预测输出和实际目标之间的损失函数。
5. **评估与调整**：在训练过程中定期评估模型的表现，检查是否达到预期的效果。如果效果不佳，可能需要调整提示的设计或者训练参数等。
6. **部署应用**：当模型训练完成并且达到满意的性能后，就可以将它部署到实际的应用场景中了。此时模型可以根据给定的提示来生成高质量的文本输出。

### 优势

**成本效益**：相比从头开始训练一个全新的模型，Prompt-tuning可以在较少的数据和计算资源下对预训练模型进行微调以适应特定任务。

**灵活性**：可以通过简单的文本提示来引导模型执行不同的任务，而无需修改模型本身的架构。

**泛化能力**：良好的提示设计可以使模型更好地理解和适应新任务，提高其在未见过的数据上的表现。

**减少过拟合风险**：由于不需要对整个模型进行大量的权重更新，因此可以减少过拟合的风险。

**快速适应新任务**：通过调整提示，可以在较短时间内使模型适应新的应用场景，而无需长时间的训练过程。

**知识迁移**：预训练模型已经学习了大量的语言结构和模式，通过prompt可以有效地利用这些知识来解决新任务。

**低资源需求**：对于一些资源有限的情况（如数据量小或计算能力受限），prompt-tuning是一个可行的选择。

### 缺点

**性能限制**：虽然Prompt-tuning可以在少量数据上有效，但在某些情况下可能无法达到传统微调方法的性能水平。

**提示设计难度**：找到有效的提示往往需要经验和试错，这可能会消耗大量时间和精力。

**泛化性挑战**：虽然好的提示可以帮助模型更好地泛化，但错误的设计可能导致模型在新数据上的表现不佳。

**可解释性不足**：提示如何影响模型内部的工作机制通常难以解释，这可能会影响其在某些领域的应用。

**资源效率**：尽管Prompt-tuning比全模型微调节省资源，但对于大规模预训练模型来说，生成高质量的提示仍然需要一定的计算资源。

**依赖预训练模型**：Prompt-tuning高度依赖于预训练模型的质量，如果基础模型不够强大，则可能无法获得满意的结果。

**稳定性问题**：不同提示之间的效果差异可能很大，且模型的表现可能随着提示的变化而不稳定。

## 3.LoRA

### 简介

LoRa 是一种轻量级的微调方法，旨在通过调整模型中的一小部分参数来适应新的任务，而保留大部分模型的原始权重不变。这种方法可以减少训练所需的时间和资源消耗。

### 流程

**1.准备基础模型**：选择一个预训练的大规模语言模型作为基础模型。例如，您可以选择使用 GLM-4 这样的模型作为基础。

**2.添加 LoRA 层**：在选定的基础模型的关键层（如线性层、注意力层等）中插入 LoRA 层。LoRA 层由两个小矩阵 U*U* 和 V*V* 组成，其中 U⋅VT*U*⋅*VT* 代表一个低秩矩阵。这些矩阵的大小远小于原始层的权重矩阵，因此可以显著减少额外参数的数量。

**3.冻结主模型参数**：冻结基础模型中的大多数参数，只训练插入的 LoRA 层中的参数 U*U* 和 V*V*。这样可以避免破坏预训练模型的通用知识。

**4.准备数据集**：准备用于微调的特定领域或任务的数据集。这些数据集通常比原始预训练数据集要小得多。

**5.微调模型**：使用新的数据集对模型进行微调，仅更新 LoRA 层中的参数。可以通过传统的监督学习方式进行训练，例如使用交叉熵损失函数。根据任务类型选择合适的损失函数。对于分类任务，可以使用交叉熵损失；对于回归任务，则可以使用均方误差损失。

**6.调整超参数**：调整学习率、批次大小、训练轮数等超参数以优化训练过程。

对于 LoRA 特定的参数，如 LoRA 层的秩（rank），也需要进行调整以找到最佳配置。

**7.监控训练过程**：使用验证集来监控模型的训练进度，确保模型不会过拟合。定期保存检查点，以便在需要时恢复训练。

**8.评估模型**：在特定任务上评估微调后的模型性能。这通常涉及到使用验证集或测试集来进行评估。评估指标取决于任务类型，例如准确率、F1 分数、平均绝对误差等。

**9.部署模型**：将微调后的模型部署到实际应用中。由于只修改了部分参数，模型的体积相对较小，易于部署。部署时可以考虑使用模型压缩技术进一步减小模型大小。

**10.可选步骤：合并 LoRA 参数**：如果需要保存或进一步使用完整的模型，可以选择将 LoRA 参数合并到原模型的权重中。这意味着模型将不再需要 LoRA 层，但会稍微增加模型的大小。合并后，模型可以直接用于推断，无需额外加载 LoRA 层。

### 优势

**参数效率**：LoRA通过低秩矩阵分解来修改模型的部分权重，这意味着它只需要训练和存储少量额外的参数，从而大大减少了内存消耗和计算成本。

**快速适应新任务**：由于LoRA仅需要对模型的一小部分进行更新，因此可以更快地适应新的任务或者领域，这对于持续学习和在线学习场景非常有用。

**减少过拟合风险**：通过限制模型权重的变化范围，LoRA有助于减少模型在新数据集上的过拟合风险。

**易于集成**：LoRA可以很容易地集成到现有的深度学习框架中，几乎不需要改变原有的训练流程。

**灵活的任务适应**：LoRA可以应用于各种类型的神经网络模型，并且能够根据不同任务的需求灵活调整其适应性。

### 缺点

**性能限制**：因为LoRA仅调整模型的一部分权重，这可能会限制模型在某些复杂任务上的表现能力。对于那些需要较大调整幅度的任务，LoRA可能无法达到最佳性能。

**泛化能力**：虽然LoRA有助于减少过拟合的风险，但在某些情况下，它的特定调整方式可能会限制模型在未见过的数据上的泛化能力。

**超参数调整**：LoRA引入了额外的超参数（如低秩矩阵的大小），这些超参数的选择可能会对最终模型的表现产生重要影响。不当的设置可能会导致性能下降。

**初始模型依赖性**：LoRA的效果在很大程度上依赖于预训练模型的质量。如果初始模型不够强大或不适合目标任务，LoRA可能无法有效提升性能。

**特定任务适用性**：LoRA可能不是所有任务的最佳选择，特别是对于那些需要对模型进行全面调整的任务。

## 4.SFT

### 简介

SFT (Supervised Fine-Tuning) 是一种常见的微调方法，它利用带有标签的数据集来进一步训练预训练模型，使其适应特定的任务或领域。通过这种方式，模型可以学习到更具体的知识和技能。

### 流程

1. **准备数据**:收集并准备针对特定任务的数据集。数据通常需要进行预处理，比如清洗、标注等。
2. **选择预训练模型**:选择一个已经在大量数据上预训练过的模型作为基础模型。这个模型可以是语言模型、图像识别模型等，取决于具体的应用场景。
3. **定义任务**:明确微调的目标，例如文本分类、问答系统等。根据任务需求，可能需要添加额外的输出层或其他组件。
4. **微调模型**:使用新的数据集来微调预训练模型。调整学习率、批次大小等超参数以优化训练过程。训练过程中可能会冻结某些层以减少计算成本或防止过拟合。
5. **评估模型**:在验证集上评估模型性能。可能需要调整模型结构或训练策略以提高性能。
6. **部署模型**:将最终训练好的模型部署到生产环境。模型可以被集成到应用程序中，提供实际的服务或功能。

### 优势

**提高模型性能**:预训练模型在大量通用数据上训练，通过微调可以使其更好地适应特定任务的数据分布，从而提高模型在该任务上的表现。

**减少数据需求**:相对于从零开始训练模型，监督微调通常需要较少的标记数据就能达到较好的性能。

**节省计算资源**:利用预训练模型的初始权重，可以减少训练时间和计算成本。

**快速迭代**:微调可以在较短的时间内完成，使得开发者能够快速测试不同的假设和配置。

**易于实现**:大多数深度学习框架支持微调操作，因此实施起来相对简单。

**迁移学习的有效应用**:监督微调是迁移学习的一种形式，它利用了预训练模型学到的一般特征，然后针对特定任务进行调整。

**灵活性**:开发者可以选择微调整个模型或仅微调某些层，这提供了很大的灵活性。

**可扩展性**:如果需要进一步改进模型，可以通过增加更多数据或进行额外的微调来实现。

### 缺点

**过拟合风险**:如果微调数据集较小，模型可能会过度拟合这些数据，导致泛化能力下降。

**计算资源消耗**:虽然相对于从头开始训练模型所需的资源要少，但微调仍然需要一定的计算资源，特别是对于大型模型。

**参数调整难度**:找到合适的微调参数（如学习率、批量大小等）可能需要一些试验和错误，这可能会耗费时间。

**性能不稳定**:微调的效果有时会因为数据分布的变化而变得不稳定，特别是在目标任务与预训练任务差异较大的情况下。

**知识转移限制**:预训练模型可能没有覆盖所有相关领域或概念，这意味着某些特定领域的知识可能无法很好地转移到微调后的模型中。

**数据偏见**:如果微调数据集中存在偏差，那么这种偏差可能会被放大，影响模型的公正性和准确性。

**训练时间**:对于非常大的模型，即使是微调也可能需要相当长的时间来完成。

## 5.CT

### 简介

CT（Curriculum Tuning）是一种模仿人类学习过程的方法，它逐步引入复杂度递增的数据集来训练模型，类似于教育中的课程设置从简单到复杂的顺序。这种训练策略可以帮助模型更好地学习和理解数据。

### 流程

1. **定义学习路径**：根据任务复杂度定义一系列从简单到复杂的训练样本或任务序列。可以基于数据集本身的属性（如长度、语法复杂性等）来确定样本的难度。
2. **初始化模型**：选择合适的模型架构并进行初始化。
3. **开始训练**：从最简单的样本或任务开始训练模型。随着模型能力的提高，逐渐引入更复杂的样本或任务。
4. **评估与调整**：在每个阶段评估模型的表现。根据模型的表现调整学习速率或其他超参数。如果必要，可以微调学习路径本身。
5. **重复步骤**：不断迭代上述过程，直到达到满意的性能水平或者模型收敛。
6. **最终测试**：使用未见过的数据对模型进行最终测试，以评估其泛化能力。

### 优点

**加速学习过程**：通过从简单任务开始，模型可以更快地学习基础特征，从而加快整个训练过程。

**提高最终性能**：在某些情况下，Curriculum Tuning可以帮助模型达到更高的准确性或更好的泛化能力。

**减少过拟合风险**：由于模型最初接触到的是较为简单的例子，这有助于模型建立一个更通用的基础表示，从而减少过拟合的风险。

**增强模型鲁棒性**：模型在处理各种复杂度的任务后，可能会更加鲁棒，能够更好地应对未知数据。

**促进无监督或半监督学习**：在数据标注成本较高时，Curriculum Tuning可以帮助利用未标注数据进行预训练，然后逐渐加入少量标注数据来引导模型学习。

### 缺点

**难以定义学习路径**：确定有效的学习顺序可能很困难，尤其是在没有明显难度层次的情况下。如何量化任务难度以及如何确定合适的进展速度都是挑战。

**增加训练复杂度**：实现Curriculum Tuning需要额外的工作来设计和调整学习路径，这增加了训练的复杂度。

**可能影响收敛速度**：如果学习路径设置不当，可能会导致模型收敛速度变慢或无法达到最优解。

**不一定总是有效**：对于某些特定的任务或模型结构，传统的随机梯度下降训练方法可能就已经足够好，Curriculum Tuning未必能带来显著的好处。

**可移植性差**：Curriculum Tuning的设计往往依赖于具体的任务和数据集，这意味着在不同的场景下可能需要重新设计学习路径。

## 6.RLHF

### 简介

RLHF(Reinforcement Learning with Human Feedback) 是一种结合了强化学习和人类反馈的方法，用于训练模型生成更加符合人类期望的回答或行为。这种方法通常用于对话系统和文本生成等领域。

### 流程

1. **数据收集**：收集基础的数据集，可以是随机策略生成的行为序列或其他简单的策略产生的数据。这些数据用于初始化强化学习环境中的代理行为。
2. **初始模型训练**：使用初始数据集训练一个基线模型（如语言模型）。此模型将作为后续强化学习过程的基础。
3. **人类评估**：从模型生成的样本中选择一些让人类进行评价。人类根据某些标准（如质量、相关性等）对这些样本进行排序或评分。
4. **奖励模型训练**：使用人类的反馈来训练一个奖励模型，该模型预测给定输入下哪种输出更受人类偏好。这个奖励模型实际上是在学习如何量化人类的偏好。
5. **强化学习**：利用奖励模型作为强化学习过程的一部分，来指导代理的学习。代理通过与环境交互来最大化预期的累积奖励。
6. **迭代优化**：重复上述步骤多次，每次迭代都使用更新后的奖励模型来进一步优化代理的行为。随着迭代的进行，代理的行为会逐渐接近人类期望的结果。
7. **最终模型部署**：经过多轮迭代后，获得的模型能够较好地模拟人类偏好，并可用于实际应用中。

### 优点

**提高质量和相关性**：RLHF通过直接利用人类反馈来调整模型输出的质量和相关性，使得模型更加符合人类的期望和偏好。

**减少偏差**：由于人类反馈通常比纯自动评估更为准确，因此RLHF可以帮助减少模型中的偏差。

**灵活性**：RLHF方法可以灵活地应用于各种任务，包括但不限于文本生成、对话系统、游戏策略等。

**可解释性**：人类可以直接参与训练过程，这有助于理解模型是如何学习的，以及为什么会产生特定的输出。

**持续改进**：通过迭代过程，模型可以持续得到优化，逐步逼近最优解。

### 缺点

**成本高昂**：获取高质量的人类反馈往往需要大量时间和资源，特别是在大规模项目中。

**效率问题**：人类反馈的速度通常远低于自动算法的速度，这可能会限制训练过程的效率。

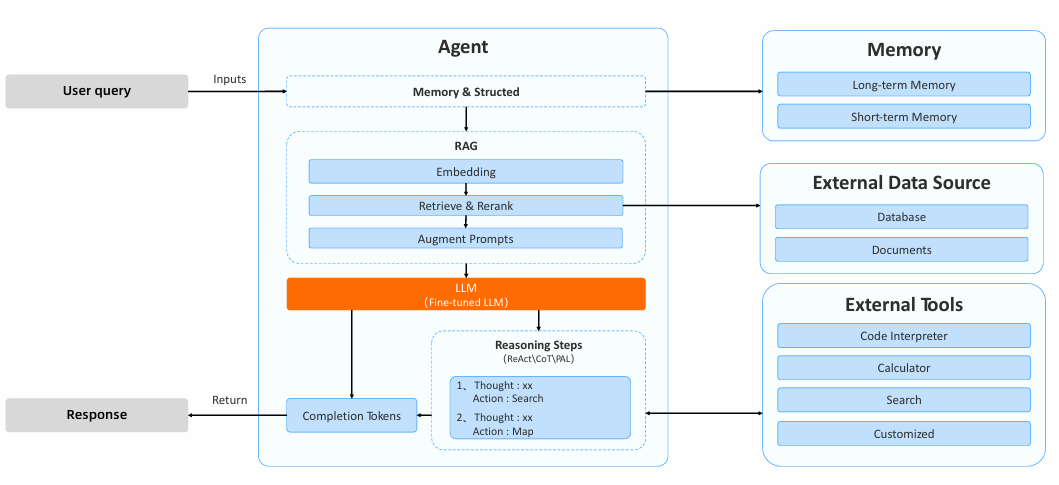
**主观性和一致性问题**：不同的人类评估者可能会有不同的偏好，导致反馈的一致性问题。此外，即使是同一评估者也可能随着时间推移而改变其偏好。

**可扩展性挑战**：在大规模数据集上获取足够的人类反馈是非常具有挑战性的。

**潜在的滥用风险**：如果没有适当的监管和道德考虑，RLHF可能会被用于不道德的目的，比如生成误导性或有害的内容。

**反馈偏差**：人类反馈可能受到个人偏见的影响，如果未得到妥善管理，可能会引入新的偏差。

## 总结



每种微调方法都有其适用场景和优缺点，针对业务场景的分析，比较适用的是Prompt Engineering，Prompt-tuning,LoRA这三种微调方法。整个的知识问答场景的模型应用到业务场景应该使用基础模型+微调+Prompt Engineering+Agent这套流程。技术方案使用的优先级：提示词工程(PE),检索增强生成(RAG）,微调(FT)。

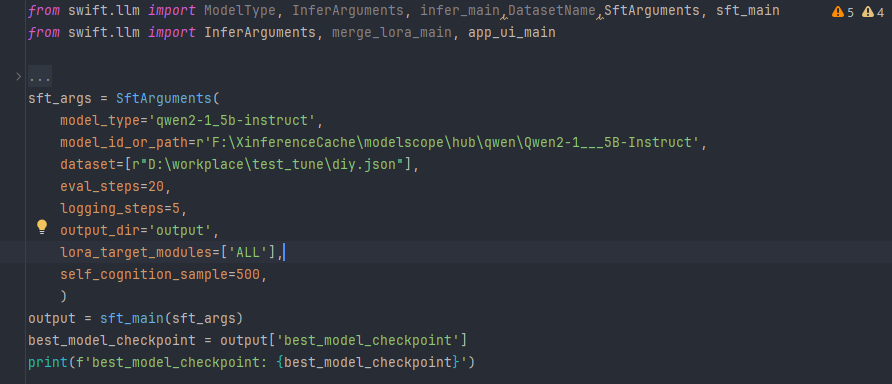
# 2.已经验证的微调方法

## 1.Prompt Engineering

对于一些小规模的业务场景可以通过Prompt Engineering覆盖，在mapgis\_datastore的几个api接口上进行了测试，在用户输入的自然语言与提示词内容匹配度高的情况下，基本可以得到正确的结果。优点是调节过程方便，可以用于高度定制化的小型产品，迭代速度快。缺点是无法覆盖大型的业务场景，提示词过长，会超出模型的context\_length,及其占用显存。

## 2.采用swift对LLM采用自己的数据集进行lora对话微调

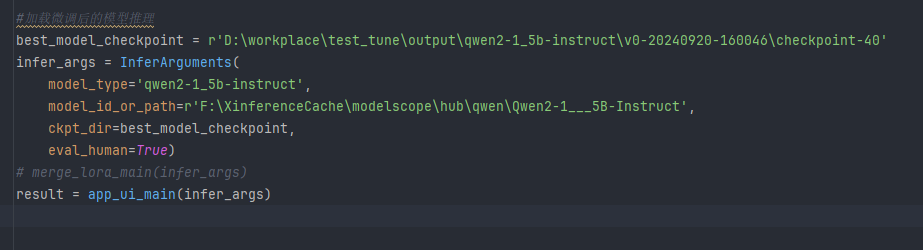
1.微调训练模型



具体参数说明，查看文档[命令行参数 — swift 2.5.0.dev0 文档](https://swift.readthedocs.io/zh-cn/latest/Instruction/命令行参数.html#sft)

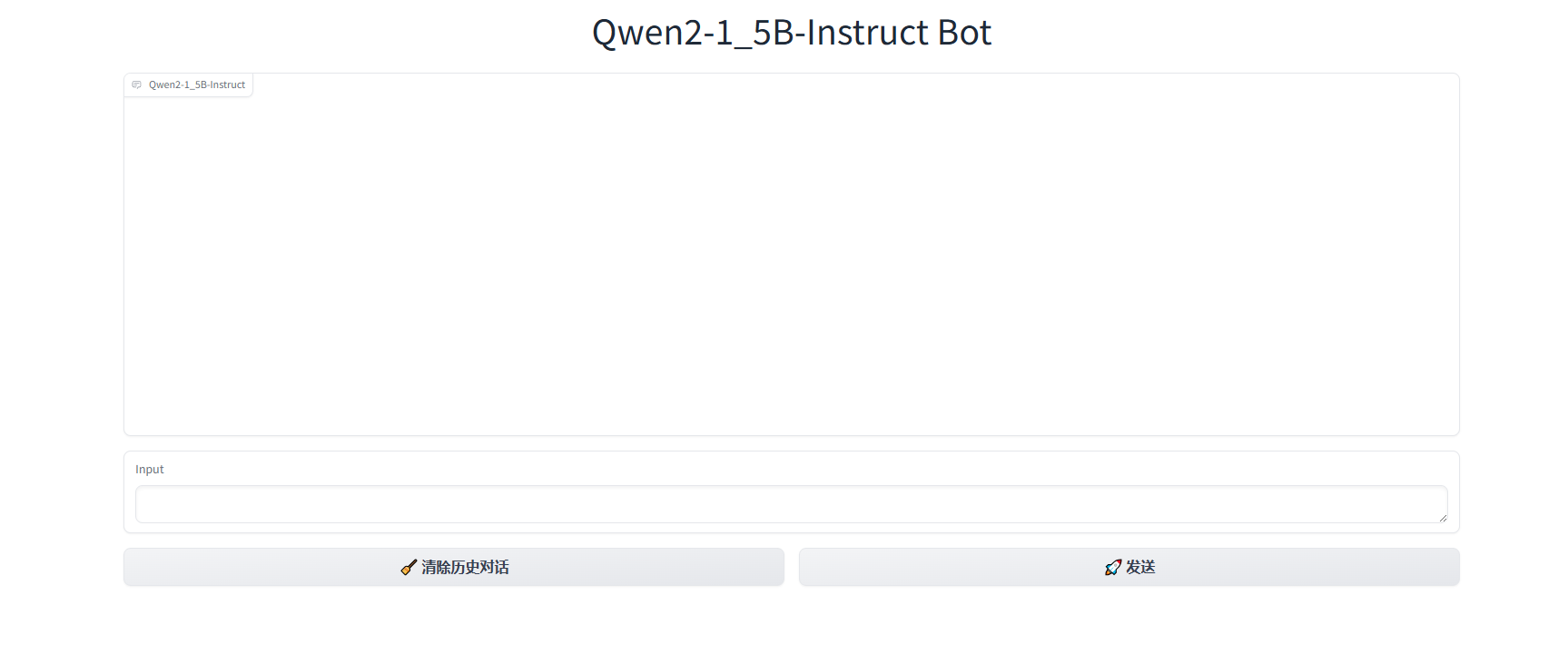
断点续训：resume\_from\_checkpoint指定该参数

2.加载训练后的模型进行推理

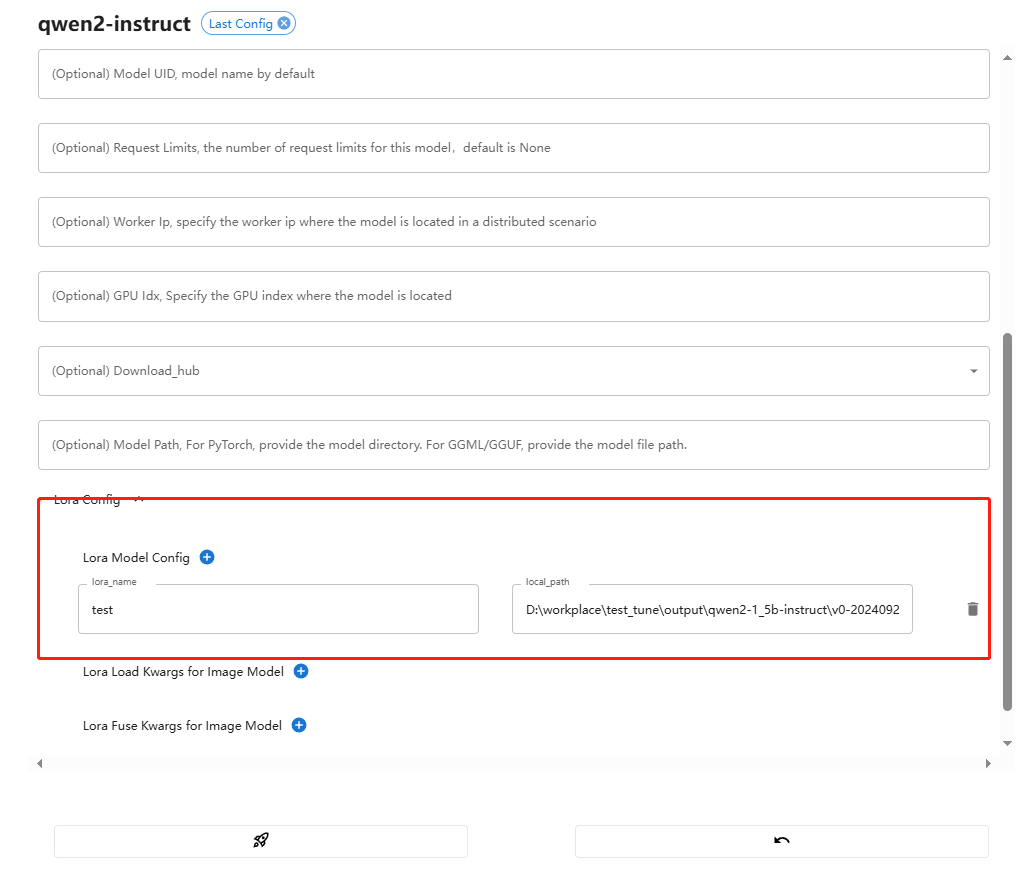


运行上述代码，可以访问webui

# 



## 3.采用xinference部署lora微调后的模型



在xinference的ui界面打开原始模型的页面，点击Optional Configurations，划到下面，找到loraconfig,设置lora模型名称，可以随便填，然后设置lora模型的路径，例如：D:\workplace\test\_tune\output\qwen2-1\_5b-instruct\v0-20240920-160046\checkpoint-40

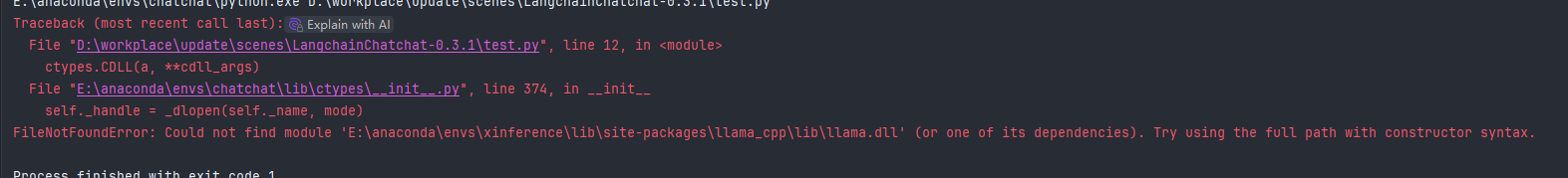
# 3.遇到的问题以及验证结论

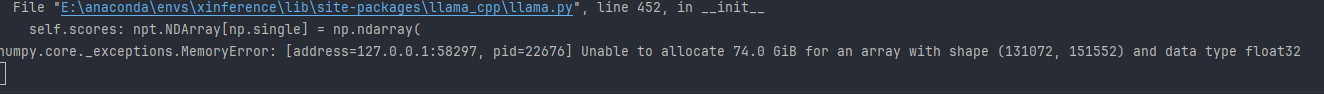
|  |  |
| --- | --- |
| 问题 | 解决方案 |
| 1.有部分api的参数，描述太过于类似，LLM无法做出区分  2.有的api里面两个字段的值，描述过于相近，无法让LLM正确识别出  3.目前提示词过长，需要简化  4.对于同一产品，不同版本的api也无法分清该使用哪个  5.对于不同的api，对参数的后处理不同，无法形成配置项  6.LLM调用了两次，将tool的结果又输入到了LLM  7.tool返回的结果过多，导致显存溢出 | 简化api接口，针对业务场景去掉不用的字段，将不用户用不到的字段设置默认值写在配置文件，【】  同时在可以替换api中一些产生冲突的参数名，  但是对于query的字段无法做出处理。通过以上方法可以解决1-4中的绝大部分情况。  对于不同的api,需要做出不同的后处理，LLM对于存储服务字段无法正确转换成对应的英文名称，尝试过修改提示词，但效果均不好，但至少能正确提取出中文名称，于是增加后处理，将中文替换成英文的。  对于阀门数据集这种字段，LLM会识别成阀门数据集，但需要的只是阀门两个字，因此需要处理把数据集给去掉。  对于基于地图文档的查询这个接口，doc的值需要与用户输入相匹配，需要通过键值对，以及正则匹配的方式来处理替换。  对于6，7问题，需要阅读langchain文档，查看有无提供接口中断此过程。如果无此接口，考虑采用过滤数据的方式 |

## 3.1部署llama.cpp引擎，以及功能验证



在66服务器上成功部署，但该引擎存在问题，初始化时会申请大量内存，使用glm4初始化申请了74G内存，而且gpu版本与xinference不兼容，目前xinfernce官方还没给出有效解决方案，尝试目前的办法，均不起作用，一直报错：





但实际上该文件是存在的。cpu版本在实际测试业务问题时也会内存溢出，无法正常使用，因此该引擎目前阶段暂时无法使用。

## 3.2 qwen2-1.5B-instruct模型效果验证

对于qwen2-1.5B-instruct与glm4-7B采取同样的问题提问，qwen2-1.5B-instruct基本没有能回答正确的。下图是qwen2-1.5B-instruct的结果。



下图是glm4-7B的结果。



# 4.大模型相关知识

## 1.系统指令（system) 和用户指令 (prompt)

在 ChatGPT 的世界里面，提示词（prompt）是很重要的，它代表了我们作为用户与大语言模型进行交互的指令。而它又分为两类，一类是系统指令（本模块中通过 system 参数指定），一类是用户指令 （本模块中通过 prompt 参数指定）。

系统指令一般是角色定义，也就是我们系统大语言模型所扮演的角色，它的能力，以及你希望它在跟你对话或者帮助你生成内容时的一些规则。用户指令一般指的是具体你要它帮你做的事情，例如写一篇作文等等。

这两个参数是可以分开用，也可以混合用的。 system 参数会有默认值，所以无论你是否指定，其实它都有一个默认设置。而当你指定了prompt，就意味着你不是要聊天，而是要直接返回文本生成结果了。他们都是字符串的形式，但在具体指定时你有多种选择。

**直接输入**

这是最简单的方式，你可以直接在调用命令时输入，显然它比较适合比较简单的场景，例如：

# 以一个历史学家的角色来开启对话

chat -system "你是一个著名的历史学家，你精通中国上下五千年的历史，并且擅长用小学生听得懂的语言讲解历史，幽默风趣又不失端庄稳重。“

# 直接返回文本生成结果

gpt "中国的首都是哪一个城市?"

# 同时使用 system 和 prompt 两个参数

chat -system "你是一个著名的历史学家，你精通中国上下五千年的历史，并且擅长用小学生听得懂的语言讲解历史，幽默风趣又不失端庄稳重。“ `

-prompt "秦朝为什么会那么快灭亡呢"

**文件输入**

在调用命令时直接输入 system 或 prompt 当然是很直观的， 但是一个事实就是你的指令通常不会这么简单。这时候你就可以用文件来保存他们，然后将文件路径传递给命令。

# 同时使用 system 和 prompt 两个参数, 本例中，为 system 参数传入的其实是一个文件名 expert.md，它会被自动读取。

gpt -system expert.md -prompt "秦朝为什么会那么快灭亡呢"

**在线网页**

另外一种做法是直接读取一个网页地址，当作 system 或 prompt的输入，前提是你的这个文件是可以公开地访问到。

# 同时使用 system 和 prompt 两个参数, 本例中，为 prompt 参数传入的其实是一个网址，它会被自动读取。

gpt -system “你帮我把文字从中文翻译成英文" `

-prompt "https://raw.githubusercontent.com/chenxizhang/openai-powershell/master/README.zh.md"

**模板库**

我更为得意的一个设计就是，你可以直接使用模板库的提示词，快速复用到你的代码中。当前我维护了一个提示词模板库（https://github.com/code365opensource/promptlibrary）, 里面已经有两个模板（fitness 是一个健身教练， gpt-translator-md 是可以把一个md文件进行翻译的模板）。

你可以像下面这样使用它们。

# 同时使用 system 和 prompt 两个参数, 本例中，为 prompt 参数传入的其实是一个网址，它会被自动读取。

gpt -system “lib:gpt-translator-md" `

-prompt "https://raw.githubusercontent.com/chenxizhang/openai-powershell/master/README.zh.md" `

-context @{from\_language="zh";to\_language="en"}

原文链接：[系统指令（system) 和用户指令 (prompt) · chenxizhang/openai-powershell · Discussion #186 (github.com)](https://github.com/chenxizhang/openai-powershell/discussions/186)

# 5.swift、llama-factory支持的模型梳理与对比

SWIFT支持350+ LLM和90+ MLLM(LLM: Qwen2.5, Llama3.2, GLM4, Internlm2.5, Yi1.5, Mistral, Baichuan2, DeepSeek, Gemma2, ...; MLLM: Qwen2-VL, Qwen2-Audio, Llama3.2-Vision, Llava, InternVL2, MiniCPM-V-2.6, GLM4v, Xcomposer2.5, Yi-VL, DeepSeek-VL, Phi3.5-Vision, ...）的训练(预训练、微调、对齐)、推理、评测和部署。

LLaMA Factory 是一个简单易用且高效的大型语言模型训练与微调平台。通过 LLaMA Factory，可以在无需编写任何代码的前提下，在本地完成上百种预训练模型的微调，模型种类：LLaMA、LLaVA、Mistral、Mixtral-MoE、Qwen、Yi、Gemma、Baichuan、ChatGLM、Phi 等等。

swift比LLaMA Factory支持的模型种类更多。