第 5 章 构建行业通用能力：继续预训练

随着大语言模型在各个行业的深入应用，如何进一步提升模型在特定领域的表现成为研究的热点。继续预训练(Continuous Pre-training)作为一种有效的技术手段，能够在已有预训练模型的基础上，利用行业特定的数据进一步地优化模型，从而构建出更加强大、通用的行业能力。

5.1 继续预训练概述

继续预训练是一种在已有预训练模型的基础上，利用特定领域数据进行的二次训练技术，其核心目的是通过吸收更多领域特定信息，增强模型在特定任务上的表现力和适应性。这一过程不仅有助于提升模型的性能，还能有效地拓展模型在行业中的应用范围，使其更好地服务于行业特定的需求和挑战。

5.1.1 定义与背景

继续预训练指的是在已有的预训练语言模型的基础上，利用特定领域或任务的数据进一步地进行训练。这一过程旨在使模型更好地满足特定领域或任务的需求，从而提高其在相关应用中的性能。

1. 什么是继续预训练

继续预训练，又称增量预训练或持续预训练，是一种先进的持续学习策略。它针对经过初步预训练的模型，通过不断地引入新数据来对知识进行更新和扩展。这一过程不仅保留了模型原有的知识基础，还使其能够捕捉到最新的发展趋势，从而有效地应对数据时效性带来的知识老化问题。在继续预训练中，模型不再从随机状态开始学习，而是基于已有的预训练成果，进一步吸收新的信息和知识。这种训练方式允许模型在保持原有泛化能力的同时，不断适应新的语言环境和使用场景，实现知识的持续积累和进化。继续预训练原理的示意图如图 5-1 所示。

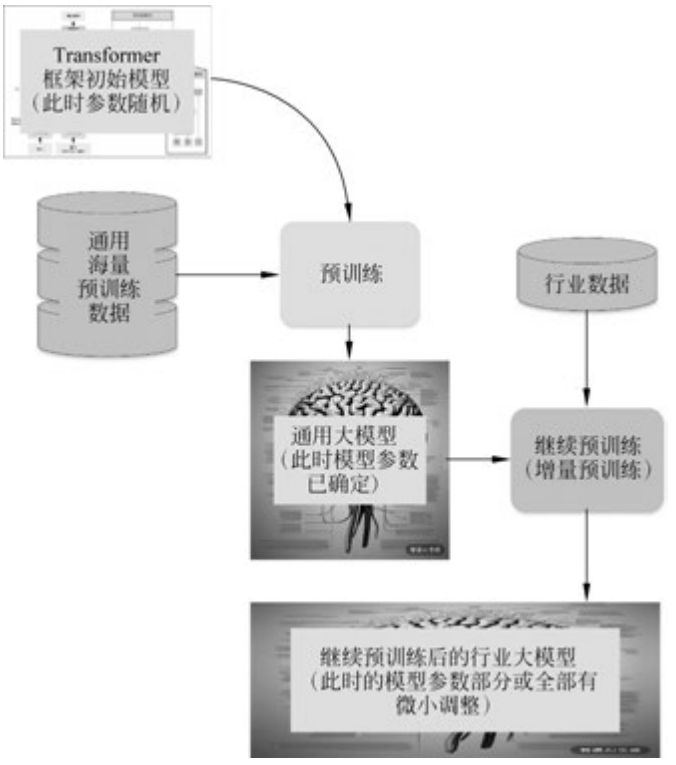

图 5-1　继续预训练原理示意图

2. 继续预训练的背景与意义

以 LLM 为例，随着自然语言处理技术的快速发展，预训练模型在各类任务中取得了显著成果，然而，由于语言和知识的不断演变，静态的预训练模型难以持续适应新的语言现象和知识更新。同理，特定领域或场景下的数据变化也要求模型具备更强的灵活性和适应性。继续预训练主要具有以下的价值与意义。

（1）解决知识老化问题：通过持续引入新数据，继续预训练确保模型能够跟上时代的发展，避免因数据过时而导致性能下降。

（2）增强模型适应性：通过吸收不同风格、主题或领域的文本信息，模型能够更好地适应多样化的语言环境和使用场景。

（3）提高泛化能力：通过知识蒸馏等策略，新模型不仅可以学习新知识，还可以继承旧模型的泛化能力，实现稳定性和连续性的平衡。

（4）推动自然语言处理应用：在新闻摘要、社交媒体分析等领域，继续预训练能够确保模型输出与时代脉搏紧密相连，提高应用的准确性和有效性。

继续预训练作为一种有效的持续学习策略，在自然语言处理领域具有重要的理论和实践意义。它不仅推动了预训练模型的发展，还为各类应用场景提供了更加灵活、强大的语言理解能力。

5.1.2　作用与优势

继续预训练在引导大模型深入理解新知识领域或应对分布外场景方面，扮演着至关重要的角色。以 Llama-3 8b 和 Mistral 7b 为代表的基础模型，虽然已在包含数万亿 Token 的庞大语料库上进行了初步预训练，但在某些特定语言或专业领域（如法律、医学等）的数据覆盖上可能仍显不足，因此通过引入新的 Token 和数据集来进行针对性训练，对于填补模型在特定领域的知识空白，提升其在多样化场景下的表现力和适应性尤为重要。本书将借鉴 SHI 等的分类思想[54]，将继续预训练放在持续学习的框架下进行阐述，并将继续预训练分为横向继续预训练和纵向继续预训练两个维度，以及继续微调和继续人类反馈强化学习，继续学习的总体框架图如图 5-2 所示。

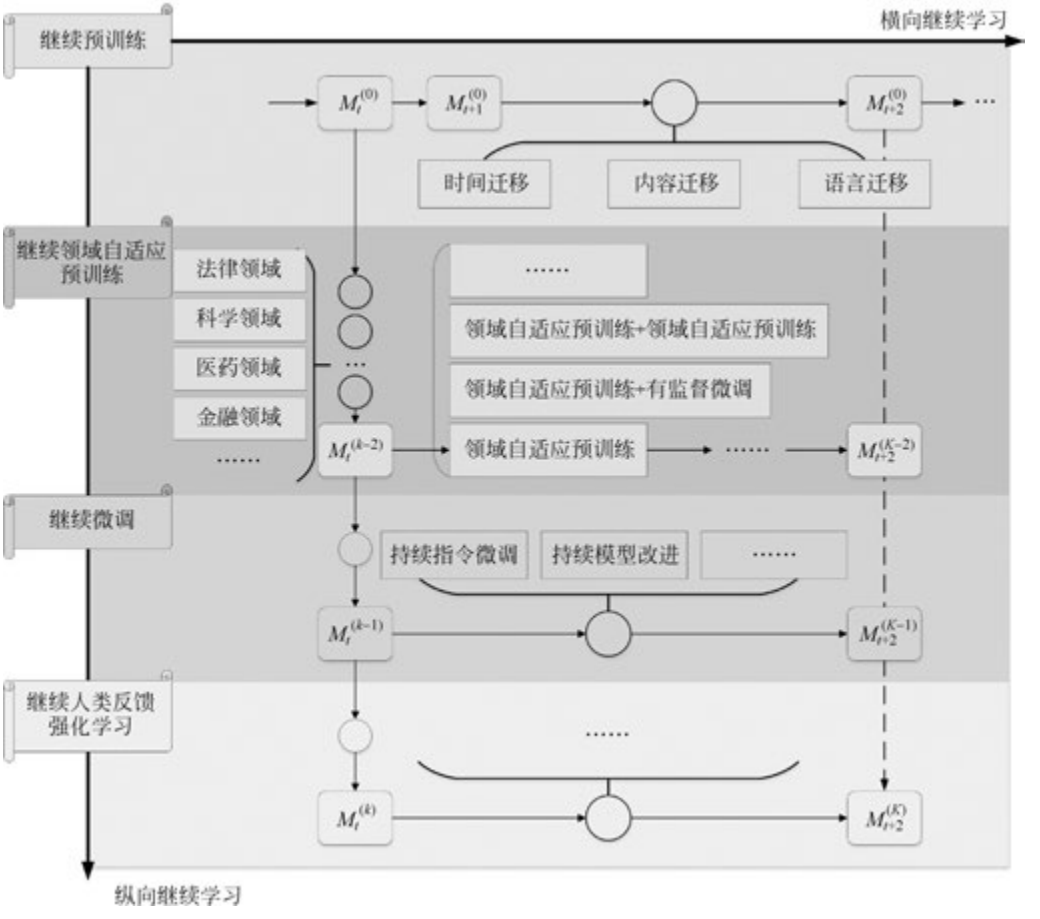

图 5-2　继续学习的总体框架图（参考 SHI 等人论文[54]）

纵向继续学习包括继续预训练、继续领域自适应预训练、继续微调及继续人类反馈强化学习，其中继续微调和继续人类反馈强化学习将在第 6 章和第 7 章进行讲解，纵向继续学习

的作用是防止大模型遗忘一般知识。横向继续学习是指在部署大模型之后，当一组新的数据可用时，模型将不断更新，横向继续学习的目标是防止在长任务序列中知识的平行遗忘。横向继续学习和纵向继续学习的区别如图 5-3 所示。

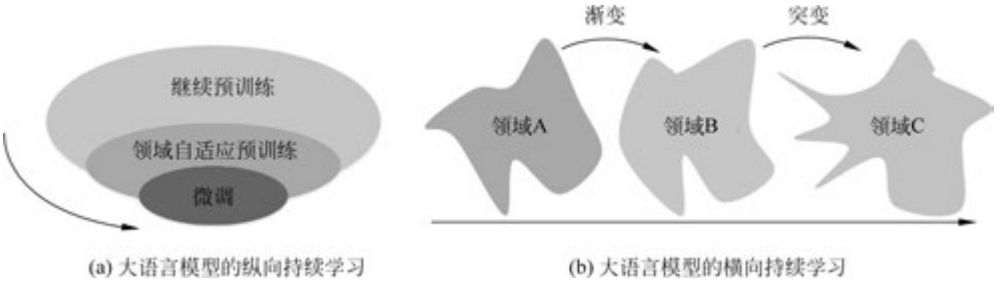

图 5-3 横向继续学习和纵向继续学习的区别示意图（参考 SHI 等人论文[54]）

注意：图 5-2 中对于继续预训练和继续领域自适应预训练进行了区分，其中继续预训练在通用数据集上进行持续预训练，为模型提供更加丰富的语言知识和泛化能力，包括时间迁移、内容迁移和语言迁移等横向继续预训练内容，同时以其为基础，在垂直领域构建了面向行业通用能力的纵向继续学习体系。

1. 横向继续预训练：提升模型对时间推移和内容变化的适应性

1）语言迁移

（1）多语言能力：继续预训练可以帮助大模型学习并生成多种语言，从而适应多语言环境。这对于跨国公司、多语种社区和全球化服务场景至关重要，例如，Llama 系列的预训练模型可以学习多种语言，并通过继续预训练进一步强化其多语言能力。

（2）代码转换：程序语言通常包括非常多的类别，例如 Python、C++、C＃和 Java 等。预训练可以帮助模型理解和生成不同的编程语言，而继续预训练可以扩展模型支持的语言类型。这对于代码自动翻译、代码补全和跨平台开发至关重要。

2）内容迁移

（1）从单模态向多模态学习的模态扩展：继续预训练可以帮助模型学习多模态数据，例如文本、图像和视频。这可以提高模型在多模态场景下的表现力，例如图像描述、视频摘要和跨模态推理。

（2）从单领域到多领域的领域扩展：继续预训练可以帮助大模型学习特定领域的知识，例如法律、医学、金融、科学和代码。这可以提高模型在多个领域的表现力和适应性，例如，通过继续预训练，模型可以在学习医学知识的基础上，再进行金融知识学习，从而拓展模型在多个领域的专业性。

注意：这里的内容迁移很容易被误认为是纵向继续学习中的领域自适应预训练，但事实上两者有着本质的区别，领域自适应预训练的目标是让预训练模型在某个领域变得更专业，而横向继续学习则是希望大模型能够掌握更多的领域和方向。

3）时间迁移

（1）时序数据理解：继续预训练可以帮助大模型理解时间序列数据，例如新闻、社交媒体数据和经济数据。这可以提高模型在时间序列分析、趋势预测和事件检测等场景下的表现力。

（2）知识更新：继续预训练可以帮助模型不断地更新其知识储备，从而适应不断变化的世界。这对于新闻摘要、社交媒体分析和社会事件预测等场景至关重要。

2．纵向继续预训练：增强模型特定领域表现和专业性

领域自适应预训练（Domain-Adaptive Pre-training，DAP）利用特定领域的无标签数据，对通用预训练模型进行调整，使其更擅长理解和生成特定领域的语言。它位于纵向继续学习的中间阶段，介于继续预训练和继续微调之间。

1）DAP 的优势

（1）领域特定知识学习：通过 DAP，模型可以学习特定领域的词汇、语法、专业知识和领域特定模式，从而更好地理解和生成该领域的内容。

（2）提升领域任务表现：DAP 可以显著地提升模型在特定领域任务上的性能，例如文本分类、命名实体识别、关系抽取等。

（3）减少遗忘：DAP 可以有效地防止模型在特定领域学习过程中遗忘通用领域的知识，从而保持模型的泛化能力。

2）DAP 的挑战

（1）数据质量：DAP 的效果很大程度上取决于领域数据的数量、质量和多样性。低质量或缺乏代表性的数据可能会导致模型过拟合或泛化能力下降。

（2）数据异质性：不同领域的语言数据可能存在显著差异，这增加了 DAP 的难度，例如，法律文本和医学文本在词汇、语法和风格上都有很大差异。

（3）遗忘风险：虽然 DAP 可以减少遗忘，但仍然存在一定的风险。如果领域数据过多或模型调整过于剧烈，则可能会导致模型遗忘一些通用领域的知识。

3）DAP 的应用

DAP 可以应用于各种特定领域。

（1）法律：DAP 可以帮助模型更好地理解和生成法律文本，例如合同、判决书、法律文件等。这可以应用于法律咨询、法律文档自动摘要、法律文本分类等任务。

（2）医学：DAP 可以帮助模型更好地理解和生成医学文本，例如病历、研究报告、医学文章等。这可以应用于医学文本摘要、疾病诊断、药物推荐等任务。

（3）金融：DAP 可以帮助模型更好地理解和生成金融文本，例如股票行情、金融报告、金融新闻等。这可以应用于金融信息分析、股票预测、金融欺诈检测等任务。

（4）科学：DAP 可以帮助模型更好地理解和生成特定科学领域的文本，例如学术论文、科学报告等。这可以应用于科学文本摘要、科学知识图谱构建、科学研究辅助等任务。

（5）代码：DAP 可以帮助模型更好地理解和生成代码，例如代码补全、代码生成、代码错误检测等。这可以应用于软件开发、代码自动修复、代码质量控制等任务。

4）DAP 的未来研究方向

（1）更有效的数据利用方法：研究更有效的数据增强、数据筛选和数据重用方法，以提高 DAP 的效率和效果。

（2）领域自适应模型架构：研究更有效的领域自适应模型架构，例如多任务学习模型、模块化模型、可微分搜索模型等，以提高模型在特定领域的适应性。

（3）可解释的 DAP：研究可解释的 DAP 方法，以提高模型的可信度和透明度。

注意：不论是横向继续学习还是纵向继续学习，当涉及继续预训练时训练数据的格式、训练方法基本一致，最大的区别在于训练数据的内容分布和数据规模，因此在 5.2 节和 5.3 节针对行业和领域进行继续预训练时，我们统称为继续预训练，而不再区分领域自适应预训练。

5.2 继续预训练的技术实现

目前，已经有许多成熟的继续预训练框架，为开发者提供了完整的实现方案，例如，MedicalGPT 项目提供了一个专门用于训练医疗大模型的框架，涵盖了继续预训练、有监督微调及人类反馈强化学习等多种训练方式。另一个项目 LLaMA-Factory 则提供了统一高效的框架，适用于对 100 多个大语言模型进行精细调整。此外，LLaMA-Factory 还支持多种硬件环境和云服务平台，包括昇腾 NPU 等，通过特定的安装命令和兼容性设计确保在不同平台上的高效运行。这些框架不仅简化了继续预训练的流程，还显著地提升了模型的训练效率和性能。由于目前针对视觉大模型和多模态大模型的继续预训练资料相对较少，因此本章节将以大语言模型的继续预训练为例来进行讲解。

注意：MedicalGPT 和 LLaMA-Factory 两个项目均可以在 GitHub 上找到。访问地址如下。

MedicalGPT：https://github.com/shibing624/MedicalGPT。

LLaMA-Factory：https://github.com/hiyouga/LLaMA-Factory。

5.2.1 数据准备与模型选择

在数据标注的流程中，数据准备与模型选择是至关重要的初始步骤。数据准备涉及收集、整理和预处理用于标注的数据集，确保数据的质量和多样性，以便模型能够从中学习到有效的特征。模型选择则是指根据特定任务的需求和数据的特性，挑选合适的预训练模型作为基础。这一选择将直接影响后续标注的准确性和效率。合理的数据准备与恰当的模型选择为数据标注工作奠定了坚实的基础，是确保标注质量与模型性能提升的关键环节。

1. 训练数据集构造

大语言模型通过学习未被标记的文本进行预训练，从而学习语言的表征。通常，预训练数据集可以从互联网上获得，因为互联网上提供了大量的不同领域的文本信息，有助于提升

模型的泛化能力。继续预训练数据集形式较为简单，以 LLaMA-Factory 的继续预训练数据集为例，文本格式描述如下：

【示例 5-1】 继续预训练数据集 pretraining_dataset.json 的内容格式

```
[
      {"text": "document"},
      {"text": "document"}
]
```

在进行继续预训练时，只有 text 列中的内容（document）会用于模型学习，而不在这个格式内的文本数据则不会被训练。除了构建训练数据集外，LLaMA-Factory 还要求对于上述格式的数据集进行注册，具体的注册过程可以通过在 dataset_info.json 文件中写入数据集相关信息来完成，数据集注册信息如下：

【示例 5-2】 继续预训练数据集注册范式

```
"数据集名称": {
  "file_name": "data.json",
  "columns": {
    "prompt": "text"
  }
}
```

相比于 LLaMA-Factory 的数据集格式，MedicalGPT 的要求则更为简单，只需先将文本内容复制到一个 txt 文件中，然后放到指定目录下。

2. 模型选择：性能匹配与算力承载的双重考量

在基于继续预训练技术构建行业大模型的过程中，选择合适的模型对于个人用户或中小型企业而言，不仅是技术决策，更是关乎业务成效的关键步骤。这一选择需细致权衡两个基本维度：模型的性能是否可以精准对接业务需求，以及企业或个人自身的算力资源是否足以支撑模型的继续预训练。

注意：这两个维度是面向用户业务场景和算力资源的，但除此之外，还要考虑选择的项目（例如 MedicalGPT 和 LLaMA-Factory）是否支持想要训练的模型，例如 LLaMA-Factory 是否支持 Qwen2.5-7B 模型等，这部分内容可以在项目首页查询（网址为 https://github.com/hiyouga/LLaMA-Factory）。

首先，模型性能的评估应深入而具体，紧密围绕业务场景的实际需求展开。算法专家需要对候选模型的处理速度、准确性、泛化能力等核心指标进行细致分析，例如，在自然语言处理领域，应优先考虑那些在语言理解深度、语境处理能力等方面表现卓越的模型，而在计算机视觉任务中，则需重点关注模型在图像识别精度、实时处理能力等方面的表现。通过这样的精准评估，确保所选模型能够与特定业务场景实现最佳匹配，从而最大化地发挥其效能。

在对模型性能进行评估的过程中，用户可以参考几个开放平台的测评结果做出决策，例如司南 OpenCompass 大模型测试平台，如图 5-4 所示。

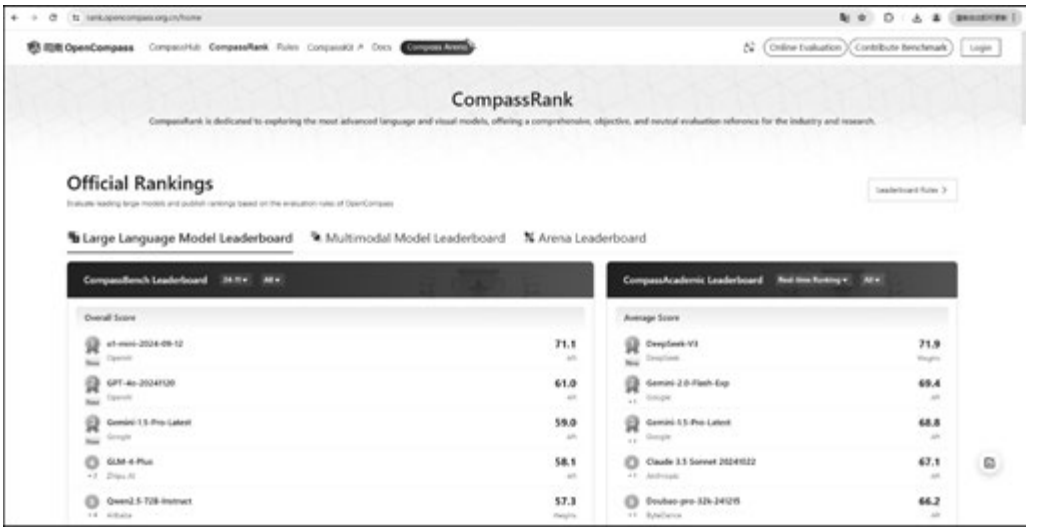

图 5-4　OpenCompass 大模型测试平台（网址为 https://rank.opencompass.org.cn/home）

OpenCompass 是一个开源开放的大模型评测平台，构建了包含学科、语言、知识、理解、推理五大维度的通用能力评测体系，支持超过 50 个评测数据集和 30 万道评测题目，支持零样本、小样本及思维链评测，是目前最全面的开源评测平台之一。OpenCompass 支持大部分主流 HuggingFaces 上的大语言模型和多模态模型的评测，只需几行简单配置，便可轻松开展模型评测。自发布以来，获得了企业界和学术界的大量关注，被阿里巴巴、腾讯和清华大学等多所企业与科研机构用于大语言模型和多模态模型的研发过程。除了 OpenCompass 外，还有像 lm-evaluation-harness、Open LLM leaderboard 和 FlagEval 等较为常用的测试平台，这部分内容已经在 4.4 节阐述过，感兴趣的读者可以自行查阅上述章节。用户可以通过这些平台给出的测评结果来确定心仪的模型。

在选择模型时，除了要考虑性能因素外，企业或个人用户自身服务器的承载能力同样不容忽视。如果选择非开源模型，则需要查看发布模型的官方是否提供了继续预训练的接口。按照目前的情况来看，暂时还没有哪家公司开放了继续预训练的 API，因此用户只能考虑选择开源大模型进行继续预训练。在选择开源模型时，必须对服务器的硬件配置进行全面而细致的评估，包括 CPU 和 GPU（或 NPU、TPU）的处理能力、内存和显存容量、存储速度等关键参数。这一评估旨在确保所选模型能够在现有服务器环境下稳定、高效地运行，避免因资源不足而导致继续预训练无法开展或模型性能下降等问题。通过合理的算力承载评估，可以保障业务的连续性和用户体验的顺畅，为模型的继续预训练提供基础。

注意：值得注意的是，关于大模型继续预训练所需的计算资源，笔者本人并未找到准确的资料。这可能是由于训练过程所需数据量过大，导致普通机构和个人无法进行复现测试，而使用较小的数据量进行预训练，效果又不明显。不过，我们也看到了一些相关报道，例如OpenAI的科学家Andrej Karpathy称，训练ChatGPT的模型（GPT-3.5）时需要10TB数据，6000个GPU训练12天才能完成一轮（这里只是一轮，而要获得一个较好结果，往往需要多次实验），而LLaMA在训练7B的模型时，需要83 000个A100（80GB显存）。也就是说，如果你有83 000块A100（80GB显存），则需要训练1000h（约42天），才能训练一个7B的模型。

5.2.2　训练环境准备

在当今软件开发与数据科学领域，环境管理工具对于确保项目的可重复性、一致性和高效性至关重要，例如，一个算法工程师往往需要同时开展多个工程项目，这些项目运行的环境和依赖包既可能相同也可能不同，即便是相同的依赖包或运行环境，其版本需求也可能存在差异。为了确保同一台服务器上能够同时独立且高效地运行这些环境，环境管理就显得尤为重要，而环境管理工具则是专门为了管理运行环境而开发的工具套件或容器化解决方案。以下将对3种广泛使用的环境管理工具Docker、Conda和Pip进行简单介绍。

1）Docker

（1）Docker简介：Docker是一个开源的容器化平台，它允许开发者将应用程序及其依赖项打包到一个可移植的容器中。这个容器可以在任何支持Docker的平台上运行，从而实现“一次构建，到处运行”。Docker的基本工作逻辑如图5-5所示。

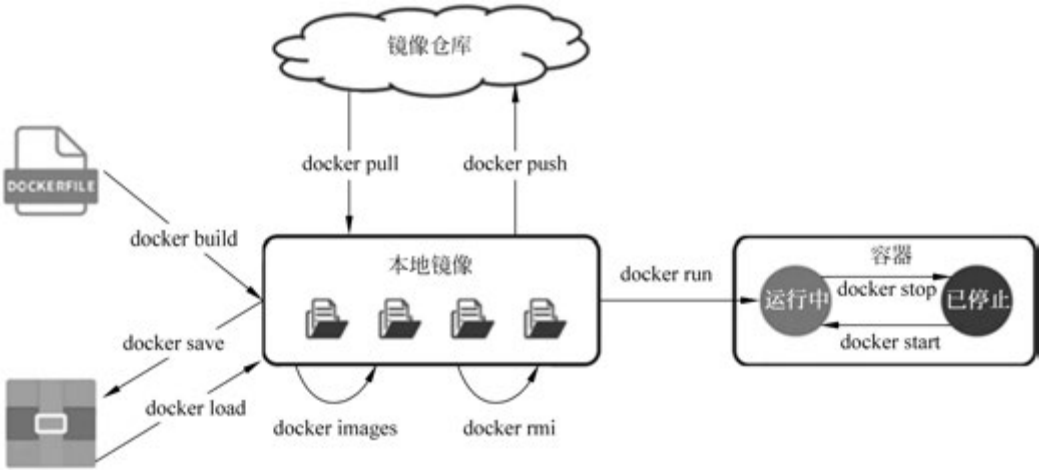

图5-5　Docker的基本工作逻辑

Docker的最小运行单元是容器（Container），而每个容器都是由镜像启动的。镜像的获取主要有3种途径：首先，可以从公共或私有的镜像仓库中直接拉取（Pull）所需的镜像；其

次，在联网环境下，可以通过编写 Dockerfile 并执行构建命令来生成自定义镜像；最后，如果已经有镜像压缩文件，则可以通过导入(Load)的方式获得镜像。这 3 种途径为用户提供了灵活的镜像获取方式，以满足不同场景下的需求。

(2) Docker 的核心组件：Docker 的核心组件包括 Docker Engine、Docker Hub 和 Docker Compose 等。Docker Engine 是 Docker 的核心运行环境，负责创建和管理容器。它提供了容器运行时环境、镜像管理和存储管理等一系列关键功能。Docker Engine 的架构由 Docker Daemon、Docker CLI 和 Docker REST API 三部分组成，其中 Docker Daemon 负责后台管理，Docker CLI 提供命令行接口，而 Docker REST API 则允许程序化地与 Docker Engine 交互。Docker Hub 则是一个公共的 Docker 镜像仓库，用户既可以从中拉取官方或第三方镜像，也可以上传自己的镜像。它不仅提供镜像存储和搜索功能，还支持镜像版本管理，方便用户跟踪和管理不同版本的镜像。此外，Docker Hub 还支持私有仓库，用于存储敏感或专有的镜像，确保用户的数据安全和隐私。Docker Compose 是一个用于定义和运行多容器 Docker 应用的工具。首先它通过 YAML 文件定义应用的服务、网络和卷等配置，然后可以一键启动所有服务，大大地简化了多容器应用的部署和管理过程。Docker Compose 特别适用于微服务架构和多容器应用的开发和部署场景，使开发者能够更好地管理和运行复杂的应用程序。

(3) Docker 的关键技术：Docker 的关键技术涵盖了容器技术、镜像管理、网络管理和容器编排等多个方面。在容器技术中，Docker 利用 Linux 内核的命名空间(Namespace)实现进程、网络、挂载点等的隔离，并通过控制组(Cgroup)限制容器的 CPU、内存等资源的使用，同时 Docker 镜像由多个只读层组成，每层代表一次文件系统更改。在镜像管理方面，通过 Dockerfile 定义镜像的构建过程，包括基础镜像、安装依赖、复制文件等，而镜像仓库如 Docker Hub、私有仓库或其他第三方仓库则用于存储和分发镜像，镜像标签则用于区分不同版本的镜像，便于管理和使用。在网络管理上，Docker 支持桥接模式、主机模式、无网络模式等多种网络模式，实现容器间网络隔离，确保安全性和独立性，并通过端口映射将容器内的端口映射到主机，实现外部访问。至于容器编排，Docker 提供了原生的集群管理工具 Docker Swarm，支持多主机容器编排；同时，虽然 Kubernetes 不是 Docker 的一部分，但与 Docker 紧密集成，提供了更强大的容器编排能力；此外，容器间通过服务发现实现相互通信。这些技术共同构成了 Docker 的强大功能体系，确保了 Docker 的高效、灵活和安全性。

(4) Docker 的应用场景如下。

微服务架构：Docker 容器是实现微服务架构的理想选择，每个微服务可以运行在独立的容器中。

持续集成与持续部署(CI/CD)：Docker 容器用于构建、测试和部署应用程序，实现自动化流程。

云原生应用开发：Docker 是云原生应用开发的基础，支持快速迭代和弹性伸缩。

开发环境一致性：确保开发、测试和生产环境的一致性，减少“在我的机器上可以运行”等问题。

应用迁移与部署：通过 Docker 镜像，应用程序可以被轻松地迁移到不同的主机或云平台。

（5）Docker 的优势与挑战：Docker 具有可移植性、高效性、一致性和隔离性等优势，允许容器在任何支持 Docker 的平台上运行，快速启动且资源占用少，确保应用程序在不同环境中可以一致运行，并提供强大的环境隔离能力，然而，它也存在挑战，包括对初学者而言的学习曲线、需要关注的安全配置和镜像漏洞问题，以及大规模部署时可能出现的存储管理复杂性。

（6）Docker 的安装和容器化运行。接下来，将以在 Ubuntu 24.04 服务器上安装 Docker 并运行容器化应用程序为例进行简要介绍，代码如下：

```
//chapter_5/Docker_Install_and_Container.sh

#安装 Docker
#更新服务器：打开终端并更新包列表
sudo apt update

#安装依赖包：安装所需的依赖包
sudo apt install apt-transport-https ca-certificates curl software-properties-
common -y

#添加 Docker 官方 GPG 密钥：将 Docker 的 GPG 密钥添加到您的服务器
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
-o /etc/apt/keyrings/docker.gpg

#添加 Docker 仓库：将 Docker 仓库添加到 APT 源
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/
docker.gpg] https://download.docker.com/linux/ubuntu$(lsb_release -cs) stable" |
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

#更新包列表：再次更新包列表以确保所有包都是最新的
sudo apt update

#安装 Docker Engine：安装 Docker Engine 及其 CLI 工具
sudo apt-get install docker-ce docker-ce-cli containerd.io -y

#管理 Docker 系统服务
#启动 Docker 服务：启动 Docker 服务并设置为开机自启
systemctl start docker
systemctl enable docker

#验证 Docker 安装：通过运行 hello-world 镜像验证 Docker 是否安装成功
docker run hello-world

#运行 Docker 容器
#拉取镜像：从 Docker Hub 拉取一个示例镜像，例如 Nginx
docker pull nginx:latest
```

```
#基于镜像创建容器,注意代码中的 ID_Images 是镜像的 ID
docker run --name nginx -itd --shm-size=1g --gpus all -v /mnt/d/nginx/:/nginx ID_
Images /bin/bash

#进入创建的 Docker 容器,注意代码中的 ID_Container 是镜像的 ID
docker exec -it ID_Container /bin/bash
```

注意：虽然 Docker_Install_and_Container 包含了一系列安装 Docker 镜像和容器化的命令，但是它并非是一个完整的 bash 运行文件，因此不能直接运行安装，用户需要根据自己的场景来选择使用哪些命令。

2）Conda

（1）Conda 简介：Conda 是一个开源的包和环境管理工具，被广泛地应用于 Python、R、Lua、Scala、Java、JavaScript、C/C++、FORTRAN 等语言的软件包管理。它允许用户轻松地安装多种版本的软件包及其依赖关系，并能够在其上构建和管理虚拟环境。

（2）Conda、Anaconda 和 Miniconda。

Conda 是一个开源的跨平台包管理器和环境管理系统，主要用于 Python 和 R 等数据科学与机器学习相关的编程语言环境。它是由 Anaconda 分发版提供的，但也可以独立安装。

Miniconda 是一个免费的用于安装 Conda 的精简版安装程序。它是 Anaconda 的一个轻量级引导版本，仅包括 Conda、Python、它们所依赖的包及一些其他有用的包（如 Pip、zlib 等）。

Anaconda 是一个开源的数据科学和机器学习平台，包含强大的包管理器 Conda，提供了一个统一的集成环境，用于安装、管理和运行 Python 或 R 编程语言的包和库。

（3）Conda 的安装及虚拟环境构建：在安装 Conda 之前，需要先下载 Anaconda 或 Miniconda。当然，也可以直接从 GitHub 上基于源码安装 Conda，但该方法难度较大，本书不会过多讨论。此外，对于 Windows 系统或带有图形化界面的 Linux 系统，还可以通过手动下载安装包逐步完成安装。

注意：Conda 可以通过 Anaconda 和 Miniconda 的方式快速安装。Anaconda 是一个包含了许多常用库的集合版本，Miniconda 是精简版本（只包含 Conda、Pip、zlib、Python 及它们所需的包），剩余的需要通过 conda install command 命令自行安装。Anaconda 和 Miniconda 的手动下载链接如下。

Anaconda 官网：https://www.anaconda.com/download。

Miniconda 官网：https://conda.io/miniconda.html。

接下来本书将重点讲解如何通过命令行的方式完成安装，并创建第 1 个虚拟环境。这里以 Anaconda 为例进行讲解，代码如下：

//chapter_5/Conda_Install_and_Virtual_Environment.sh

```
#下载 Anaconda 安装包。注意:如果提示颁发的证书已经过期,则需要在后面加上 #--no-
#check-certificate 参数
wget -c https://repo.anaconda.com/archive/Anaconda3-2024.02-1-Linux-x86_64.sh

#执行安装。执行下载好的安装程序 Anaconda3-2024.02-1-Linux-x86_64.sh。注意:由于
#版本不同,所以执行命令中的文件名会不一样,根据实际的文件名修改以下命令
bash Anaconda3-2024.02-1-Linux-x86_64.sh
#执行以上命令后,按 Enter 键进入安装流程,首先会显示授权协议信息,查看完授权协议信
#息后,需要输入 yes 接受授权协议才可以继续安装。接受授权协议安装后,默认安装当前用户目
#录下面,也可以修改成自己的目录

#检查 Conda 是否安装成功。运行如下命令,如果能够返回版本号,则表明 Conda 安装成功
conda --version
#安装完 Conda 之后,可能报没有 Conda,只需运行如下命令
source ~/.bashrc

#配置镜像源。首先查看已有的镜像源
conda config --show channels

#如果已有镜像源不够,则可以添加如下镜像源
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/
pkgs/free/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/
pkgs/main/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/
cloud/conda-forge/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/
cloud/bioconda/
conda config --add channels https://mirrors.bfsu.edu.cn/anaconda/cloud/
bioconda/
conda config --add channels https://mirrors.bfsu.edu.cn/anaconda/cloud/conda-
forge/
conda config --add channels https://mirrors.bfsu.edu.cn/anaconda/pkgs/free/
conda config --add channels https://mirrors.bfsu.edu.cn/anaconda/pkgs/main/

#目前国内提供 Conda 镜像的大学有以下几个
#清华大学: https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/
#北京外国语大学: https://mirrors.bfsu.edu.cn/help/anaconda/
#南京邮电大学: https://mirrors.njupt.edu.cn/
#南京大学: http://mirrors.nju.edu.cn/
#重庆邮电大学: http://mirror.cqupt.edu.cn/
#上海交通大学: https://mirror.sjtu.edu.cn/
#哈尔滨工业大学: http://mirrors.hit.edu.cn/#/home

#创建新环境。创建一个名为 env_name 的 Python 环境,并指定 Python 版本
conda create --name env_name python=3.8
```

```
#激活创建的新环境
conda activate env_name

#退出当前激活的 Conda 环境
conda deactivate

#克隆环境
conda create --name new_env_name --clone old_env_name

#删除环境
conda remove -n env_name --all

#查看所有环境
conda info --env
#或
conda env list

#包管理命令。注意,以下步骤一般需要在上述创建了虚拟环境后执行
#查看所有包:显示 Conda 所有包,如果是在激活的环境下,则会显示当前环境的所有包
conda list
#安装一个新的包
conda install package_name

#更新包
conda update package_name

#在 Conda 仓库中搜索指定的包
conda search package_name
#卸载 package_name 包
conda remove package_name
```

注意：虽然 Conda_Install_and_Virtual_Environment 包含了一系列安装 Conda 和虚拟环境的命令，但是它并非一个完整的 bash 运行文件，因此不能直接运行安装，用户需要根据自己的场景来选择使用哪些命令。

3）Pip

（1）Pip 简介：Pip 是 Python 官方认证的包管理工具，专门用于管理 Python 包。它通常用于从 Python Package Index（PyPI）安装 Python 包，由 Python Packaging Authority（PyPA）管理和支持。Pip 可以在任何环境中安装 Python 包，但仅限于 Python 包的管理。Pip 的主要功能包括安装、升级和删除 Python 包。它管理的是 wheel 或源码格式的包，可能在安装过程中需要编译器。尽管 Pip 本身不直接支持虚拟环境的管理，但它常与 virtualenv 或 venv 等工具结合使用，以实现不同项目间环境的隔离。在依赖管理方面，Pip 的依赖性检查主要针对 Python 包，对于外部依赖的跟踪和管理能力较弱。这意味着，如果需要用到很多依赖于外部库的 Python 包（如 NumPy、SciPy 和 Matplotlib），则 Pip 可能无

法完全满足需求。

注意：Pip 无法单独创建虚拟环境，需要与 virtualenv 或 venv 结合使用来创建虚拟环境并管理依赖包。

(2) Pip 与 Conda 的区别：Pip 和 Conda 在包管理、环境管理和依赖管理等方面存在显著差异。Conda 是一个与语言无关的跨平台包和环境管理器，能够管理各种类型和语言的包及依赖，而 Pip 则是专门用于管理 Python 包的工具，主要从 Python Package Index (PyPI)安装包。Conda 只能在 Conda 环境中安装包，但支持多语言和多类型包的安装；Pip 则可在任何环境中安装 Python 包。在依赖管理方面，Conda 能够跟踪和管理非 Python 依赖，如 LAPACK 或 OpenSSL，而 Pip 的依赖管理主要限于 Python 包。此外，Conda 自带虚拟环境管理功能，能创建完全独立的虚拟环境并支持不同版本的 Python，而 Pip 需借助 virtualenv 或 venv 等工具，并且虚拟环境中的 Python 版本受限于已存在的 Python 可执行文件。Conda 可以管理二进制文件，无须编译器，而 Pip 可以管理 wheel 或源码，可能需要编译器。Conda 创建的环境真正独立，置于可执行路径中，而 Pip 与 virtualenv 结合使用时，环境基于符号链接，可能会导致非 Python 依赖的脆弱性，但相对 Conda 而言，在构建 Python 语言的算法时，Pip 的安装方式更加轻量化和简单。总体而言，Conda 提供了更广泛和灵活的包和环境管理能力，而 Pip 则在 Python 包管理方面更为专一和简单。Pip 和 Conda 的主要区别如表 5-1 所示。

表 5-1 Pip 和 Conda 的主要区别

	Pip	Conda
管理	wheel 或源码	二进制
需要编译器	Yes	No
语言	Python	Any
虚拟环境	通过 virtualenv 或 venv 等支持	支持
依赖性检查	屏幕提示用户选择	Yes
包来源	PyPI	Anaconda repo 和 Cloud

(3) Pip 的安装及虚拟环境构建。

创建 Python 虚拟环境的方法有很多种，本书以 venv 和 virtualenv 为例，介绍搭建 Python 虚拟环境的方法。Python 从 3.3 版本开始，自带了 venv 虚拟环境。它的很多操作和 virtualenv 相似，但是两者的运行机制不同。因为它仅支持 Python 3.3 及以后版本，所以如果要在 Python 2 上使用虚拟环境，则仍然需要使用 virtualenv，因此这里先来介绍 virtualenv，代码如下：

```
//chapter_5/Pip_Virtual_Install_and_Package_Management.sh

#安装 pip
#在 Ubuntu 中，安装 pip 非常简单。首先，打开终端，然后执行以下命令以安装 pip
```

```
sudo apt update
sudo apt install python3-pip
#安装完成后,可以使用以下命令检查 pip 是否安装成功
pip3 --version
#如果安装成功,则会看到类似于下面的输出
pip 20.0.2 from /usr/lib/python3/dist-packages/pip (python 3.8)

#安装 virualenv。由于 virualenv 不是官方自带的包,所以需要先使用 pip 安装
pip install virtualenv
#然后利用 virualenv 创建虚拟环境。假设是为了一个名为 my_project 的工程项目搭建一个
#虚拟环境,则可以将这个虚拟环境创建到这个工程项目的文件夹中,操作如下
cd my_project
virtualenv my_env
#上述的 my_env 就是创建的虚拟环境名称。另外,如果存在多个 Python 解释器,则可以选择指定
#一个 Python 解释器(例如 Python 3.7,注意系统中必须首先安装 Python 3.7),如果没有指定,
#则由系统默认的解释器来搭建
virtualenv -p /usr/bin/python3.7 my_env

#虚拟环境创建后,需要激活才能使用,操作如下
source my_env/bin/activate
#退出当前虚拟环境的操作如下
Deactivate

#在激活虚拟环境的情况下,可以通过如下命令安装依赖包
pip install package_name
```

除了 virtualenv,用户还可以使用 venv 来创建虚拟环境。venv 与 virtualenv 不同,由于它是 Python 3.3 版本及以后自带的虚拟环境创建和管理工具,因此不需要像 virtualenv 那样另外安装包。以下是使用 venv 创建虚拟环境的方法,代码如下:

```
//chapter_5/Pip_Venv_Install_and_Package_Management.sh

#安装 pip
#在 Ubuntu 中,安装 pip 非常简单。首先,打开终端,然后执行以下命令以安装 pip
sudo apt update
sudo apt install python3-pip
#安装完成后,可以使用以下命令检查 pip 是否安装成功
pip3 --version
#如果安装成功,则会看到类似于下面的输出
pip 20.0.2 from /usr/lib/python3/dist-packages/pip (python 3.8)

#创建 Python 虚拟环境。venv 创建 Python 虚拟环境的指令如下
python3 -m venv my_env

#虚拟环境创建后,需要激活才能使用,操作如下
source my_env/bin/activate

#在激活虚拟环境的情况下,可以通过如下命令安装依赖包
pip install package_name
```

5.2.3 启动第 1 个继续预训练程序

接下来，本节将以 LLaMA-Factory 的继续预训练为例，展示如何构建 Docker 环境并启动一个继续预训练进程，代码如下：

```
//chapter_5/Pretraining_bash_LLaMA_Factory.sh

#从 GitHub 克隆 LLaMA-Factory 的源代码，并安装依赖包
git clone --depth 1 https://github.com/hiyouga/LLaMA-Factory.git
cd LLaMA-Factory
pip install -e ".[torch,metrics]"

#使用以下命令运行 Llama3-8B-Instruct 模型的 LoRA 继续预训练
llamafactory-cli train examples/train_lora/llama3_lora_pretrain.yaml

#完成后可以参考如下有监督微调方法(sft)修改 yaml 文件，实现继续预训练的推理和模型合并
llamafactory-cli chat examples/inference/llama3_lora_sft.yaml
llamafactory-cli export examples/merge_lora/llama3_lora_sft.yaml

#上述安装过程是直接安装在当前环境下，如果用户想要构建一个 Docker 环境并进行安装，则可以
#参考下列代码。注意：以下 Docker 镜像和容器构建针对的是 Cuda 环境，如果针对的是 Ascend
#环境，则要切换到对应的文件夹下
cd docker/docker-cuda/
docker compose up -d
docker compose exec llamafactory bash
#容器创建好后，可以再执行上述运行代码执行继续预训练
```

注意：上述代码需要配合 LLaMA-Factory 工程一并使用。

5.3 本章小结

本章详细地介绍了继续预训练的概念、背景、意义、作用、优势及技术实现。继续预训练是一种先进的持续学习策略，通过不断引入新数据对经过初步预训练的模型进行知识更新和扩展，使其能够捕捉到最新的发展趋势，有效地应对数据时效性带来的知识老化问题。

在继续预训练中，模型不再从随机状态开始学习，而是基于已有的预训练成果，进一步吸收新的信息和知识。这种训练方式允许模型在保持原有泛化能力的同时，不断适应新的语言环境和使用场景，实现知识的持续积累和进化。

继续预训练的背景与意义主要体现在以下几个方面。

(1) 解决知识老化问题：通过持续引入新数据，继续预训练确保模型能够跟上时代的发展，避免因数据过时而导致性能下降。

(2) 增强模型适应性：通过吸收不同风格、主题或领域的文本信息，模型能够更好地适

应多样化的语言环境和使用场景。

（3）提高泛化能力：通过知识蒸馏等策略，新模型不仅能学习新知识，还能继承旧模型的泛化能力，实现稳定性和连续性的平衡。

（4）推动自然语言处理应用：在新闻摘要、社交媒体分析等领域，继续预训练能够确保模型输出与时代脉搏紧密相连，提高应用的准确性和有效性。

本章还介绍了继续预训练的技术实现，包括数据准备、模型选择、训练环境准备及启动第 1 个继续预训练程序。在数据准备方面，需要构建训练数据集并进行注册；在模型选择方面，需要权衡模型的性能和算力承载能力；在训练环境准备方面，介绍了 Docker、Conda 和 Pip 这 3 种环境管理工具的使用方法；最后，通过 LLaMA-Factory 的继续预训练示例，展示了如何构建 Docker 环境并启动一个继续预训练进程。

总体而言，继续预训练作为一种有效的持续学习策略，在自然语言处理领域具有重要的理论和实践意义。它不仅推动了预训练模型的发展，还为各类应用场景提供了更加灵活、强大的语言理解能力。通过本章的学习，读者可以更好地理解和掌握继续预训练的相关知识和技术实现。