5. DeepSpeed 初始化

设置 DeepSpeed 的配置参数后, 可以利用 DeepSpeed 进行模型训练的初始化, 初始化流程如下。

（1）确定运行的设备：首先，检查代码是否有指定的本地 GPU（通过 args.local_rank）。如果没有指定，则程序默认使用 CUDA 设备。否则，它会为进程设置指定的 GPU。

（2）初始化分布式后端：在分布式训练中，使用 deepspeed.init_distributed() 函数实现每个进程与其他进程的同步，初始化分布式环境。

（3）获取当前进程的全局排序：在分布式训练中，使用 torch.distributed.get_rank() 函数获得每个进程的唯一排序或 ID。

（4）设置 DeepSpeed 配置：根据用户参数（如是否进行 offload、使用哪个 Zero Stage 等）构建一个 DeepSpeed 配置字典，来决定训练设置。

（5）同步所有工作进程：使用 torch.distributed.barrier() 确保在进一步的初始化之前所有进程都已同步。

（6）DeepSpeed 初始化：这是最关键的一步。通过 deepspeed.initialize 函数，可以将模型、优化器、参数和先前构建的 DeepSpeed 配置传递给库，进行初始化。这个函数会返回一个已经根据 DeepSpeed 配置进行了优化的模型和优化器。

（7）梯度检查点：对于特别大的模型，梯度检查点是一种节省显存的技巧，即只在需要时计算模型的中间梯度。如果用户启用了这个选项，则会调用 model.gradient_checkpointing_enable() 方法来实现相关功能。

具体代码如下所示：

```
import deepspeed

if args.local_rank == -1:
    device = torch.device("cuda")
else:
    torch.cuda.set_device(args.local_rank)
    device = torch.device("cuda", args.local_rank)
    # 初始化分布式后端，它将负责同步节点/GPU
    torch.distributed.init_process_group(backend='nccl')
    deepspeed.init_distributed()

args.global_rank = torch.distributed.get_rank()

ds_config = get_train_ds_config(offload=args.offload,
                                stage=args.zero_stage,
                                enable_tensorboard=args.enable_tensorboard,
                                tb_path=args.tensorboard_path,
                                tb_name="step1_model")
ds_config[
    'train_micro_batch_size_per_gpu'] = args.per_device_train_batch_size
ds_config[
    'train_batch_size'] = args.per_device_train_batch_size * torch.distributed.get_world_size(
    ) * args.gradient_accumulation_steps

# 设置训练种子
set_random_seed(args.seed)

torch.distributed.barrier()

# 使用DeepSpeed对模型和优化器进行初始化
model, optimizer, _, lr_scheduler = deepspeed.initialize(
    model=model,
    optimizer=optimizer,
    args=args,
    config=ds_config,
    lr_scheduler=lr_scheduler,
    dist_init_required=True)

if args.gradient_checkpointing:
    model.gradient_checkpointing_enable()
```

6. 模型训练

借助 DeepSpeed 框架实现对模型的训练，训练步骤大致分为以下几个阶段。

（1）训练前的准备：使用 print_rank_0 函数输出当前的训练状态。该函数确保只有指定的进程（通常是主进程）会打印消息，避免了多进程环境下的重复输出。在开始训练之前，对模型进行一次评估，计算模型的困惑度。

（2）训练循环：每个周期的开始，都会打印当前周期和总周期数。在每次迭代中，数据批次先被移动到相应的 GPU 设备，接着模型对这个批次进行前向传播计算损失。使用 model.backward(loss) 计算梯度，并使用 model.step() 更新模型参数。对于主进程，还会使用 print_throughput 函数打印吞吐量，这有助于了解模型的训练速度和效率。

（3）保存模型：如果指定了输出目录，则模型的状态和配置将被保存。模型可以在不同的格式中保存，例如 HuggingFace 的模型格式或 DeepSpeed 的 Zero Stage 3 特定格式。save_hf_format 函数用于保存模型为 HuggingFace 格式，这意味着训练后的模型可以使用 HuggingFace 的 from_pretrained 方法直接加载。对于 Zero Stage 3，save_zero_three_model 函数负责保存，因为在这个阶段，每个 GPU 只保存了模型的一部分。

具体代码如下所示：

```
# 模型训练部分
print_rank_0("***** Running training *****", args.global_rank)
print_rank_0(
    f"***** Evaluating perplexity, \
    Epoch {0}/{args.num_train_epochs} *****",
    args.global_rank)
perplexity = evaluation(model, eval_dataloader)
print_rank_0(f"ppl: {perplexity}", args.global_rank)

for epoch in range(args.num_train_epochs):
    print_rank_0(
        f"Beginning of Epoch {epoch+1}/{args.num_train_epochs}, \
        Total Micro Batches {len(train_dataloader)}",
        args.global_rank)
    model.train()
    import time
    for step, batch in enumerate(train_dataloader):
        start = time.time()
        batch = to_device(batch, device)
        outputs = model(**batch, use_cache=False)
        loss = outputs.loss
        if args.print_loss:
            print(
                f"Epoch: {epoch}, Step: {step}, \
                Rank: {torch.distributed.get_rank()}, loss = {loss}"
            )
        model.backward(loss)
        model.step()
        end = time.time()
        if torch.distributed.get_rank() == 0:
            print_throughput(model.model, args, end - start,
                             args.global_rank)

if args.output_dir is not None:
print_rank_0('saving the final model ... ', args.global_rank)
model = convert_lora_to_linear_layer(model)

if args.global_rank == 0:
    save_hf_format(model, tokenizer, args)

if args.zero_stage == 3:
    # 对于Zero Stage 3，每块GPU只有模型的一部分，因此需要一个特殊的保存函数
    save_zero_three_model(model,
                          args.global_rank,
                          args.output_dir,
                          zero_stage=args.zero_stage)
```

5. 指令微调

指令微调又称**有监督微调**，是指在预训练大语言模型的基础上，通过使用有标注的自然语言形式的数据，对模型参数进行微调，使模型具备**指令遵循**（Instruction Following）能力，能够完成各类预先设计的任务，并可以在零样本情况下处理诸多下游任务。经过海量数据预训练后的语言模型虽然具备了大量的“知识”，但是由于其训练时的目标仅是进行下一个词的预测，因此不能够理解并遵循人类自然语言形式的指令。为了使模型具有理解并响应人类指令的能力，还需要使用指令数据对其进行调整。如何构造指令数据，如何高效低成本地进行指令微调训练，以及如何在语言模型基础上进一步扩大上下文等问题，是大语言模型在指令微调阶段的核心。

本章先介绍大语言模型指令微调训练方法，在此基础上介绍高效模型微调及模型上下文窗口扩展方法，最后介绍指令微调的代码实践。

5.1 指令微调训练

指令微调具体训练过程并不复杂，主要分为如下三个步骤：（1）针对每一项任务去明确地定义相应的自然语言形式的指令或者提示，这些指令或提示对任务目标以及输出要求进行清晰描述；（2）把训练数据调整成包含指令以及与之对应的响应的形式；（3）使用包含指令和响应的训练数据对预训练模型进行微调操作。从训练方式的角度来看，指令微调与预训练大体上较为相似，不过指令微调的目标函数往往只是针对输出部分来计算损失。

本节将从指令微调数据构造、数据评估和影响、指令微调训练策略以及开源指令微调数据等方面进行介绍。

5.1.1 指令微调数据

指令微调数据通常由文本对构成，包含“指令输入”与“答案输出”两个关键部分。“指令输入”，是指人们向模型提出的各类请求，包含定义精准、清晰的指令或者提示信息，其核心作用在于详细阐释任务的目标究竟是什么，以及明确规定输出需要满足的各项要求。指令涵盖的范畴极为广泛，包括问题回答、信息分类、内容总结、文本改写等。“答案输出”，则是期望模型依据所

接收的指令而回答内容，这些回答需要符合人们预先设定的期望。答案输出的内容，可以使用人工手段或借助自动化方法来构建。举例来说，倘若想要训练模型使其拥有问题回答能力，那么便可以按照如下的方式来构造数据：

```
User:
复旦大学有几个校区？

Assistant:
复旦大学共有四个校区，分别是邯郸校区、江湾校区、枫林校区和张江校区。

• 邯郸校区：是复旦大学的主校区，通称 “本部”，位于上海市杨浦区邯郸路 220 号。
• 江湾校区：位于上海市杨浦区淞沪路 2005 号，杨浦区新江湾城西北部，距邯郸校区约 3.5 公里。
• 枫林校区：俗称医学院校区，位于上海市徐汇区东安路 130 号，是复旦大学除药学院外各医学院系的所在地。
• 张江校区：位于上海市浦东新区张衡路 825 号，是由复旦大学出资，上海张江集团共同建设。
```

其中“复旦大学有几个校区?”是任务指令，“Assistant”后的文字是模型学习的目标。针对机器翻译、摘要生成、文章写作、代码生成、数学解题等几乎所有任务，都可以按照上述格式准备“输入-输出”数据。例如，针对文章写作任务，可以给出更清晰的指令要求以及与之匹配的输出，如下所示：

```
User:
请帮我写一篇介绍复旦大学历史沿革的短文，要求 500 字以内，包含复旦大学的创建历程、发展历程等。

Assistant:
复旦大学创建于 1905 年，其诞生与民族救亡图存紧密相连。彼时，马相伯深感 “自强之道，以作育人才为本；
↪ 求才之道，尤宜以设立学堂为先”，遂创立复旦公学，校名取自《尚书大传》中 “日月光华，旦复旦兮”，
↪ 寄寓复兴中华的深切期望。

初创的复旦公学，广纳贤才，汇聚各方有志师生，艰难起步却蓬勃发展。辛亥革命后，学校逐步扩充，
↪ 学科领域不断拓展，由单一文科迈向文理兼具，为后续腾飞筑牢根基。

进入 20 世纪中叶，全国院系调整，复旦大学迎来新契机。诸多高校相关院系并入，师资力量大增，学术资源汇聚，
↪ 理科优势凸显，数学、物理等学科成果斐然，逐步成长为国内顶尖学府。

改革开放后，复旦更是锐意进取。一方面与国际高校频繁交流合作，引进前沿知识理念；另一方面加大科研投入，
↪ 在生物医药、信息技术等前沿领域屡获突破，培养出大批栋梁之材，持续书写着辉煌篇章，
↪ 以深厚底蕴、卓越风姿屹立于世界高校之林，为国家发展、人类进步贡献源源不断的智慧力量。
```

如果期望模型具备理解多轮对话的能力，也可以基于上述方式将对话历史都做为指令，让模型学习最后一轮的输出结果。举例来说，针对闲聊任务可以构造如下数据：

```
User:
什么是 AI？

Assistant:
人工智能是工程和科学的分支，致力于构建具有思维的机器。

User:
你是什么语言编写的？

Assistant:
Python

User:
为什么使用 Python 呢？

Assistant:
对于大模型来说，Python 是主流的开发语言。像 TensorFlow 和 PyTorch 这样的框架都提供了 Python 接口。
```

通过这样的对话数据，可以把最后一轮“Assistant”回答前的所有数据当做“输入”，最后一轮“Assistant”回答做为“输出”，模型针对这样的“输入-输出”微调后，就能够具备理解上下文对话并给出符合期望输出的能力。

可以看到，指令微调数据由一系列文本对构成，其中每一对都涵盖了“指令输入”与“答案输出”两个关键部分。乍一看，指令微调数据构造并不复杂，但其实构建指令微调数据集是极具挑战性的任务，复杂性在诸多层面均有体现。在数据收集阶段，获取高质量指令数据集需耗费大量时间与资源，既要广泛招募参与者，精心规划有效的收集策略，还要全力保证收集到的数据兼具多样性与高质量。收集来的数据后续必经重写与筛选流程，研究人员常运用深度演化、广度演化策略以及主题多样性增强手段，而这些操作对专业知识储备和专业工具辅助的依赖程度极高。此外，数据标准化也影响指令微调效果的重要方面，只有保证数据集中指令及输入输出格式一致，模型才能精准理解、妥善处理数据。同时，数据集要具备广泛覆盖领域，需要将低资源领域与专业领域涵盖在内，以此提升模型通用性与特定领域性能。为契合各类用户不同需求以及多样化应用场景，构建支持多语言的指令数据集迫在眉睫。种种复杂性相互交织，使得指令微调数据集构建困难重重，迫切需要跨学科协同合作，探索创新方法。

5.1.2 数据构建方法

为了应对指令微调数据集构建中遇到的各种挑战，研究人员不断探索高效的数据构建方法。总体而言，指令微调数据集的构建方法可以分为四大类：手动构建、现有数据集转换、自动构建以及综合模式。本节将分别对这几种构建方法进行详细介绍。

1. 手动构建

手动构建指令的方法比较直观，可以在网上收集大量的问答数据，再人为加以筛选过滤，或者由标注者手动编写提示与相应的回答。虽然这是一个比较耗费人力的过程，但是手动构建指令微调数据集仍然具备诸多显著优势：1）高质量：专业的标注人员会对数据集进行处理与审核，这一过程有效剔除了杂质，使得数据达到更高的质量水准，为后续研究提供坚实可靠的基础；2）可解释：经过人工处理，数据的含义更加明晰，能与人类的认知模式紧密契合，研究者在使用过程中能够轻松理解数据所蕴含的意义，进而更好地挖掘其中价值；3）灵活可控：研究人员能够依据不同任务需求，灵活调整训练样本，使其精准适配多样化的研究场景，充分满足个性化的研究需要，极大地提升了数据集的实用性与适配性[106]。

通常有两种方法来构建手工生成数据集。第一种方法是通过公司员工、志愿者、标注平台人员等直接创建一组指令文本,包括指令和答案。标注过程需要遵循给定的要求和规则。例如,Databricks-dolly-15K[178] 是由数千名 Databricks 员工根据文献 [24] 中列出的指令类别创建的。一些指令允许标注员参考维基百科数据作为参考文本。OASST1[179] 则是通过全球众包生成的，有超过 13,500 名志愿者参与了标注过程。OL-CC[180] 也是众包和人工标注生成的开源中文指令数据集。在开放平台上，276 名志愿者分别扮演人类用户和 AI 助手的角色开展对话，并对构建的文本进行全方位的审核，包含 10,000 条“指令-回答”数据对和 1,600 人工指令数据。Aya Dataset[181] 是多语言指令微调数据集，由来自 119 个国家的 2,997 名贡献者使用 Aya 标注平台协作标注。包含超过 204,000 个数据，覆盖 65 种语言。贡献者参与三个任务：从头开始创建新示例（原始标注）、改进现有示例以提高质量和全面性（重新标注），以及对现有贡献的质量提供反馈（标注反馈），遵循发现-改进-核实（Find-Fix-Verify）范式。

第二种方法是通过从网页上抓取人类生成的真实问答数据,并将其标准化为指令格式。InstructionWild v2[182] 中的所有指令都是从网上收集的，涵盖了社交聊天、代码相关问答等主题，大约包含 110,000 个指令。LCCC[183] 是一个中文对话数据集, 包含 LCCC-base 和 LCCC-large 两个版本。其中 LCCC-base 采用两阶段数据收集方案，首先挑选专注发布新闻的微博帐号作为高质量用户，再收集其微博帖子下方评论并把评论路径视为对话一部分；LCCC-large 则是从包括中国 Chatterbot 语料库、PTT 闲话语料库等多个开源存储库收集语料库，并与青云语料库、贴吧语料库一同清洗后处理成单轮对话数据集。

2. 现有数据集转换

收集和改进现有数据集也是一种用于构建指令微调数据集的方法，它涉及整合和修改多个开源数据集，最终将它们合并成一个新数据集用于大模型指令微调。文献 [106] 指出这种构建方式具有以下优点：（1）多样性和全面性，生成的数据集具有丰富的数据来源、多样化的任务类型和广泛的领域覆盖；（2）规模大，选择的数据集越多，规模越大；（3）节省时间，这种构建方式可以减少数据集构建所需的时间。这种数据集构造的主要是难点是质量与格式标准化。需要全面考量

源数据集的质量情况，同时还要对数据的格式进行标准化处理，这涉及多方面细致的工作以及对不同数据原有特点的把握等，操作起来较为复杂且容易出现遗漏等情况。此外，大部分已有数据集都是为传统自然语言处理任务准备，并没有包含多样性的提示词，如何构造大量多样性且语义相同的提示词也是需要解决的难点。目前已经很多指令微调数据集采用这种方式进行构建。

OIG（Open Instruction Generation）[184] 是一个大型指令微调数据集，由 LAION 社区成员创建，包含 30 个数据集和 4300 万条指令，包含使用来自多种数据源的数据增强创建的指令。它不仅涵盖标准数据集（如 Natural Questions 和 Natural Instructions），还涵盖与对话、总结、教育等相关的数据。Flan 2022[185] 数据集则是由五个部分组成，分别是 Flan 2021[186]、T0[16]、SUPER-NATURAL INSTRUCTIONS[187]）、CoT 数据集和对话数据集。它涵盖了多达 1836 个数据集。每个指令提供了四个不同的指令输入模板，包括零样本、少量样本、CoT 模板。Flan 2022 构建过程中还使用了任务混合和输入反转等技术。输入反转（Input Inversion）是指将原始输入中的某些元素或部分进行反转或重新排列，以生成新的输入，用于增强模型的泛化能力和鲁棒性。例如，在对话任务中，将对话历史中的上下文和响应进行反转，以测试模型在不同输入顺序下的表现。在代码生成任务中，可以将代码和问题进行反转，在链式推理任务（Chain-of-Thought，CoT）中，将查询、答案和解释进行反转。任务混合（Task Mixing）则将来自不同任务的示例混合在一起进行训练，其目标旨在增强模型的泛化能力和适应不同任务的能力。

文献 [188] 针对提升大语言模型在开放领域命名实体识别中的能力进行了研究。通过整合 54 个现有的中英文命名实体识别数据集，并经过两步规范化，构建了 B^2NERD 数据集。研究指出，整合多个现有数据集的主要挑战在于实体定义的不一致性和模糊性。例如，有些数据集会区分“时代广场”这样的地点和“巴黎”这样的地缘政治实体，而另一些数据集则将两者统一标注为“LOC”。如果直接使用未经处理的混合数据，大语言模型在训练中可能会与这些不一致的数据对齐，导致模型记住特定数据集的标注规则，并在推理时对常见实体类型产生混淆。此外，合并数据集还容易引入大量冗余数据。许多数据集对常见实体进行了过多标注，而对长尾实体的样本标注较少。这种缺乏多样性的情况可能使大语言模型出现过拟合现象，并进一步导致知识遗忘和泛化能力下降的问题。

为了解决数据集合并中的定义歧义以及数据冗余等问题，文献 [188] 提出了一种多数据集合并方法，如图5.1所示。该方法分为两个步骤，第一步是系统地标准化所有收集到的数据集中的实体定义。针对不同数据集中存在的不一致实体定义，方法通过基于模型的交叉验证和基于规则的筛选自动检测这些定义冲突。随后，根据特定原则为每种独特的实体类型分配明确且可区分的标签，以消除模糊性。在此阶段，构建了一个通用的实体分类体系，涵盖了常见实体类型，并为新的 NER 任务提供了标签命名的指导依据。第二步则通过采用一种基于类别和语义多样性的数据修剪策略来减少冗余。具体而言，均匀选择每种实体类型的样本，同时强调语义多样性，通过选择文本相似度较低的样本来确保数据的多样性。最终，在 54 个中英双语命名实体识别数据集中应用该方法，得到了 B^2NERD，这是一个包含 16 个主要领域、400 多种实体类型的高级命名实体识别数

据集。该数据集精炼后包含约 5.2 万条数据，能够用于提升大语言模型在开放领域信息抽取任务中的表现，从而显著增强其能力。

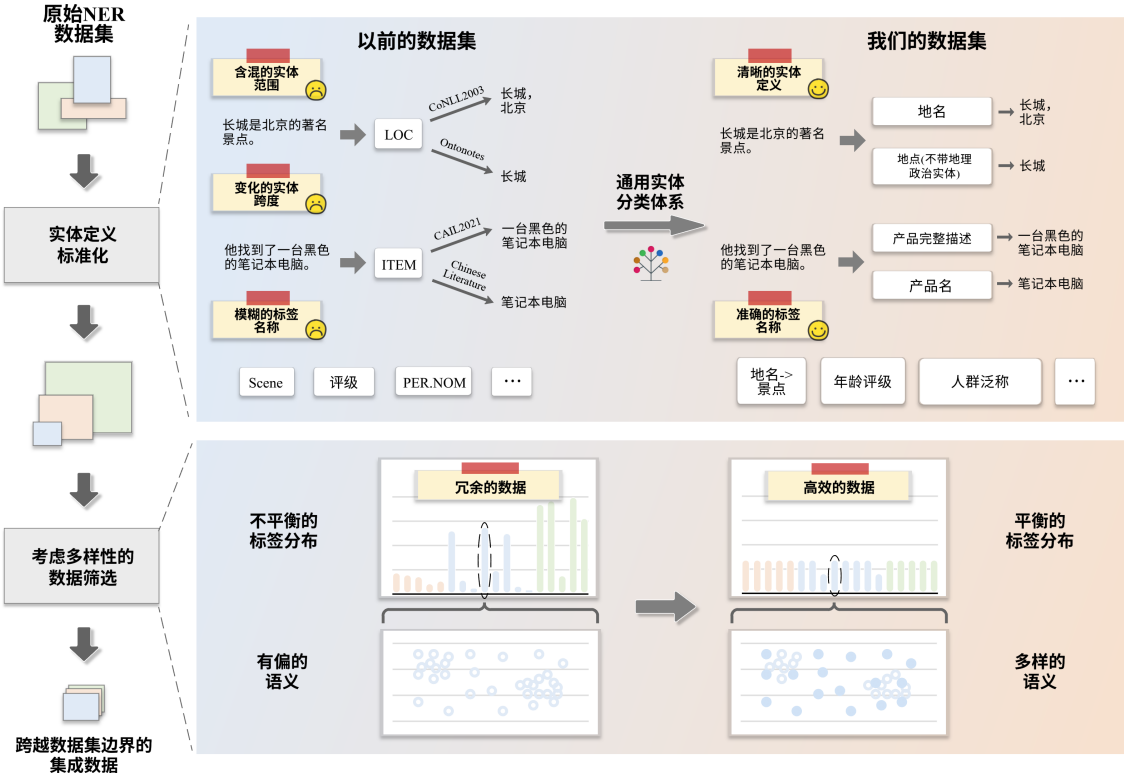


图 5.1 适用于大模型开放领域命名实体识别任务 B²NERD 数据集构建过程[188]

3. 自动构建指令

手动构建指令数据代价高昂，需要大量的人力投入。因此，一些研究尝试寻找更高效的替代方法。具有代表性的工作如 Self-Instruct[189]，利用大语言模型的生成能力自动构建指令。

Self-Instruct 数据生成是一个迭代过程。如图5.2 所示，它包含以下 4 个步骤。

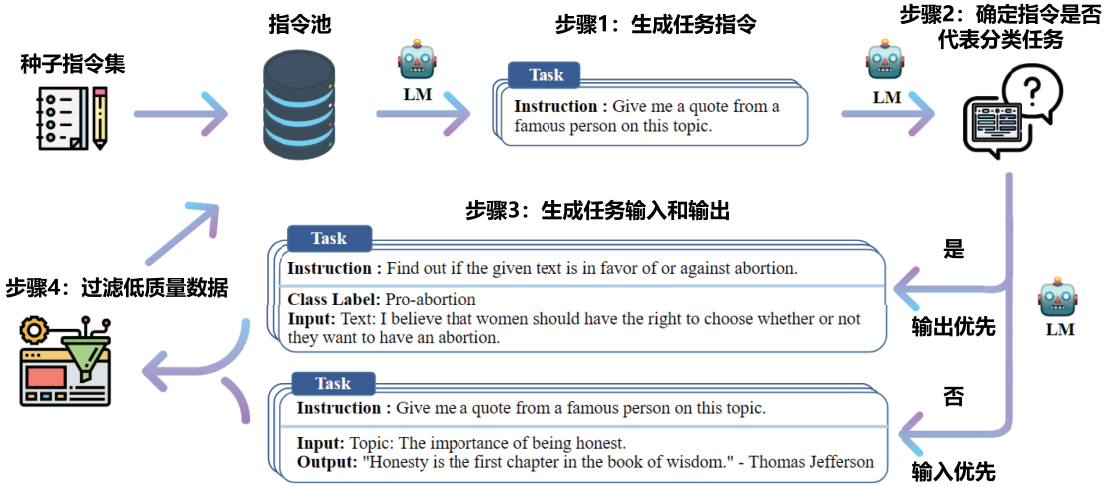


图 5.2 Self-Instruct 数据生成过程[189]

步骤 1：生成任务指令

手动构建一个包含 175 个任务的小型指令数据集，称为种子指令集，用于初始化指令池。然后让模型以自举（Bootstrapping）的方式，利用指令池生成新任务的指令：每次从指令池中采样 8 条任务指令（其中 6 条来自人工编写的种子指令，2 条是模型迭代生成的），将其拼接为上下文示例，引导预训练语言模型 GPT-3 生成更多的新任务的指令，直到模型自己停止生成，或达到模型长度限制，或是在单步中生成了过多示例（例如当出现了“Task 16”时）。本步骤所使用的提示如下所示：

```
Come up with a series of tasks:

Task 1: {instruction for existing task 1}
Task 2: {instruction for existing task 2}
Task 3: {instruction for existing task 3}
Task 4: {instruction for existing task 4}
Task 5: {instruction for existing task 5}
Task 6: {instruction for existing task 6}
Task 7: {instruction for existing task 7}
Task 8: {instruction for existing task 8}
Task 9:
```

步骤 2：确定指令是否代表分类任务

由于后续对于分类任务和非分类任务有两种不同的处理方法，因此需要在本步骤对指令是否为分类任务进行判断，同样是利用拼接几个上下文示例的方法让模型自动判断任务类型是否是

分类。

步骤 3：生成任务输入和输出

通过步骤 1，语言模型已经生成了面向新任务的指令，然而指令数据中还没有相应的输入和输出。本步骤将为此前生成的指令生成输入和输出，让指令数据变得完整。与之前的步骤相同，本步骤同样使用语境学习，使用来自其他任务的“指令”“输入”“输出”上下文示例做提示，预训练模型就可以为新任务生成输入–输出对。针对不同的任务类别，分别使用“输入优先”或“输出优先”方法：对于非分类任务，使用输入优先的方法，先根据任务产生输入，再根据任务指令和输入生成输出；而对于分类任务，为了避免模型过多地生成某些特定类别的输入（而忽略其他的类别），使用输出优先的方法，先产生所有可能的输出标签，再根据任务指令和输出，补充相应的输入。

“输入优先”提示模板如下所示：

```
Come up with examples for the following tasks. Try to generate multiple examples when possible. If
↪  the task doesn't require additional input, you can generate the output directly.

Task: Sort the given list ascendingly.
Example 1
List: [10, 92, 2, 5, -4, 92, 5, 101]
Output: [-4, 2, 5, 5, 10, 92, 92, 101]
Example 2
List: [9.99, 10, -5, -1000, 5e6, 999]
Output: [-1000, -5, 9.99, 10, 999, 5e6]


Task: Converting 85 F to Celsius.
Output: 85°F = 29.44°C


Task: {Instruction for the target task}
```

“输出优先”提示模板如下所示：

```
Given the classification task definition and the class labels, generate an input that corresponds
↪  to each of the class labels. If the task doesn't require input, just generate the correct
↪  class label.

Task: Classify the sentiment of the sentence into positive, negative, or mixed.
Class label: mixed
Sentence: I enjoy the flavor of the restaurant but their service is too slow.
Class label: Positive
Sentence: I had a great day today. The weather was beautiful and I spent time with friends.
Class label: Negative
Sentence: I was really disappointed by the latest superhero movie. I would not recommend it.

Task: Tell me if the following email is a promotion email or not.
Class label: Promotion
Email: Check out our amazing new sale! We've got discounts on all of your favorite products.
Class label: Not Promotion
Email: We hope you are doing well. Let us know if you need any help.

Task: {Instruction for the target task}
```

步骤 4：过滤低质量数据

为了保证数据的多样性，在将新生成的指令数据加入指令池之前，需要先衡量它和池中已有指令数据的相似度，只有当它和池中任何一条指令数据的 ROUGE-L 相似度都低于 0.7 时，才可能将其加入指令池。为保证数据的质量，还制定了一系列的启发式规则进行筛选：删除包含某些关键词（如“图片”）的指令数据、重复的指令数据、过长或过短的数据等。

使用 Self-Instruct 方法可以生成大量具有多样性的高质量数据。斯坦福的研究人员借助这种方法生成指令数据，在 LLaMA 模型上进行指令微调得到 Alpaca 模型，其在各项评估指标上都展现出了优异的性能。Alpaca 所使用的指令数据集的示例如下所示：

```
instruction:
Describe the structure of a typical cell.
input:
output:
A typical cell consists of a membrane, cytoplasm, nucleus, nucleolus, mitochondria, and various
↪  other organelles. The membrane encloses the other components and separates the cell from its
↪  environment. The cytoplasm is a semi-liquid layer within the cell's membrane. It contains the
↪  organelles and is the site of most biochemical reactions. The nucleus contains genetic
↪  material, while the mitochondria is responsible for energy production. Other organelles
↪  include endoplasmic reticulum and Golgi apparatus.

instruction:
Provide three possible solution to the following problem.
input:
How to reduce plastic waste?
output:
1. Reducing single-use plastic by using reusable grocery bags, food containers, and water bottles.
2. Recycling plastic waste and using the recycled material to create new products.
3. Educating people about the importance and benefits of reducing plastic waste.

instruction:
Paraphrase the following sentence and keep the same meaning.
input:
It is raining heavily outside.
output:
Rains are pouring down heavily outside.
```

5.1.3 指令微调数据评估与影响

指令微调数据的构造似乎并不困难，并且指令微调阶段所需要的训练数据量相对来说也比较少。根据 OpenAI 联合创始人 Andrej Karpathy 在微软 Build 2023 大会上对外公开的信息来看，构造通用大语言模型，在指令微调阶段也仅仅使用数万条数据。但是，指令微调数据直接影响指令微调的最终效果[42]，如何构造指令微调数据仍然有很多理论和实践问题亟待解决。接下来将从数据质量、数据多样性、数据对结果影响角度进行介绍。

1. 数据质量

指令数据的质量和多样性通常被认为是衡量指令数据的两个最重要的维度。文献 [190] 针对指令微调数据质量的影响进行了研究。由于指令微调数据包含输入和输出两个部分，因此在数据质量的度量中，文献 [190] 中将指令微调数据质量 $q(x_i)$ 分为两个部分: 指令质量 $q_I(x_i)$ 和回复质量 $q_R(x_i)$。指令微调数据质量可以形式化的表示为:

$$q(x_i) = f_q(q_I(x_{i<t}), q_R(x_{i\geqslant t})) \tag{5.1}$$

其中，f_q 是一个聚合函数，它显式或隐式地结合指令质量得分和响应质量得分。指令质量 q_I 可进一步细分为：1）清晰度 q_I^C，用于衡量任务理解的难易程度；2）准确性 q_I^A，用于衡量指令与预期任务的契合程度；3）明确性 q_I^E，用于衡量指令对输出约束（例如格式和样式）的明确界定程度。$q_I(x_i < t) = g_I(q_I^C(x_i < t), q_I^A(x_i < t), q_I^E(x_i < t))$，其中 g_I 也是聚合函数。同样的，对于回复的度量，其质量 q_R 可通过以下方式评估：1）正确性 q_R^C，用于衡量回复是否正确回答了指令；2）连贯性 q_R^H，用于衡量回复的逻辑一致性；3）相关性 q_R^P，用于衡量回复与指令的相关程度。最终的回复质量可判定为 $q_R(x_i \geqslant t) = g_R(q_R^C(x_i \geqslant t), q_R^H(x_i \geqslant t), q_R^P(x_i \geqslant t))$[①]，其中 g_R 同样为聚合函数。需要注意的是，上述所有提及的质量度量组件仅为示例，并不是所有关于指令微调数据质量的衡量都要有细粒度评价值。

对数据质量的评价可以从人工设计的指标、基于模型的指标、大模型评分以及人工评分等类型进行设计。具体来说：

（1）人工设计的指标通常依据词汇、句法以及样本间语义相似性等语言分析方面来评估数据质量。每个指标都是凭借对所研究语料库的语言、领域和任务的先验知识，以经验性的方式设计而成。DQI[191] 就是典型的人工设计指标，包含了词汇量、样本间的 N 元语法频率及关系、样本间语义文本相似度、样本内单词相似度、样本内语义文本相似度、每个标签的 N 元语法频率以及样本间语义文本相似度等指标。

（2）基于模型的指标利用训练过的模型来预测每个数据的质量。用于数据质量评判的模型可以与正在开发的语言模型有着相同或相似的架构，也可以采用完全不同的方式。困惑度（Perplexity）[192] 就是最常见的基于模型的评测指标。文献 [193] 就提出使用一个小的 GPT 类型的模型对数据进行过滤的方法。文献 [194] 则提出使用 RoBERTa 来对数据的一致性、相关性、合理性等方面进行评分。文献 [195] 使用 Qwen-1.8B 模型来过滤 UltraChat 数据集。

（3）基于大模型评分的方法则是使用已经开发出来的能力较强的模型对指令微调数据进行评判。文献 [196–199] 等都是使用 GPT-3.5 或者 GPT-4 对数据进行评价。

（4）人工评分则是采用人在环路（human-in-the-loop）的方法，直接使用人工对数据质量进行评判。OpenAssistant[200] 就是采用这种方式进行构建的，其实每个指令-响应对，标注人员要根据

① 原始文献 [190] 中，此处公式为 $q_R(x_i \geqslant t) = g_R(q_R^C(x_i < t), q_R^H(x_i < t), q_R^P(x_i < t))$，经核对后修改

三个维度对其进行分类：垃圾检测、指令遵循情况以及回答质量。回答质量评分又被细分为五个方面，包括质量、创造性、幽默性、礼貌性和无害性，并采用五点李克特量表进行打分。

文献 [190] 对各类数据质量评价方法的影响模型训练的效果进行了评测。通过对比不同数据质量评价方法，使用包括 LLaMA-7B、LLaMA2-7B、LLaMA2-13B 以及 Mistral-7B 等在内的模型进行训练，利用 ARC、HellaSwag、MMLU、AlpacaEval 等评测集合进行评价。从实验结果中可以看到，基于数据质量选择的方法即使在小规模数据情况下也能与使用全量训练的结果相匹配，并且优于从原始数据中随机选择子集的结果。比如在 Alpaca 数据集上，使用文献 [201] 提出的基于模型的 IFD 质量评价方法，仅选取 5% 的数据，就能够在 ARC、HellaSwag 以及 AlpacaEval 等评测集合上超过使用全量数据进行训练的结果。这可以反映出，指令数据的质量对于指令微调的效果有重要影响。

2. 数据多样性

数据集的多样性通常认为是开发偏差更小、泛化能力更强的大语言模型的关键。针对指令数据多样性问题，文献 [190] 提出，多样性可以从两个维度来进行衡量，一个是每个样本的个体多样性（例如：词汇和语义丰富度），另外一个是整个数据集的总体多样性（例如：所覆盖的嵌入空间的体积）。在子集选择过程中，偏向于那些任务和领域属于长尾分布中少数类别的数据点。这种采样理念旨在保持或近似原始嵌入簇的范围。数据多样性评价函数 $q(x_i)$ 可以用形式化表示为：

$$q(x_i) = f_d(q_L(x_i), q_S(x_i)) \tag{5.2}$$

其中，q_L 描述词汇多样性，q_S 则描述语义多样性。通常情况下，q_L 往往会考察 n 元语法、符号、单词以及序列的多样性。与之互补的是，q_S 强调语义多样性，即所选数据点的各种表示形式应在嵌入空间中实现最大化的多样性。可以依次或联合考虑词汇和语义多样性，以去除指令数据集中的任何重复内容。

文献 [190] 将数据多样性的评价分为人工设计的指标、基于模型的指标、基于几何的核心集采样（Geometry-based Coreset Sampling）、基于双层优化的核心集采样（Bilevel Optimization-based Coreset Sampling）等类型，具体来说：

（1）人工设计的指标可以从数据集的构成、来源、领域、主题、标注者、词汇、语义等层面定义。类型-词元比率（Type-token Ratio，TTR）用来反映输入 x_i 中不同词元的比率。基于此，可以进一步构造 MTTRSS[202]、MSTTR[202]、MATTR[203] 等方法。此外，文献 [204–206] 则使用 N-Gram 方法来评价文本的多样性。还可以使用 BERT 与 K-近邻（K-Nearest Neighbor，KNN）相结合的方法在语义层面评价数据的多样性。使用 BERT 对句子进行语义向量表示，使用 KNN 对数据集进行聚类，进而评价数据多样性情况[207, 208]。

（2）基于模型的指标与衡量模型质量的模型很类似，也是通过目标语言模型或代理语言模型来计算相关指数。数据集 S 的多样性可以直观地定义为其中每个数据 x_i 的稀有性度量之和。因

此，可以使用熵（Entropy）相关的方法来估计这种稀有性。样本越不常见、种类越丰富，数据集的多样性就越高。在此基础上，Rényi Entropy[209]、Simpson's Index (SI) [210, 211]、Vendi Score (VS)[212] 等方法也都相继提出。文献 [213] 则提出了使用**开放式标注**（Open-Ended Tagging）方法来评价模型多样性的方法。使用 GPT-4 等模型，对数据集中的每个数据进行类型标注，但是并不指定类型集合。根据模型输出的类型标签来过滤低频数据和重复类型数据。

（3）基于几何的核心集采样（Geometry-based Coreset Sampling），与显式计算多样性相关指标不同，文献 [214] 等开始研究引入核心集采样方法来选择指令数据集，从而系统地考虑数据集多样性问题。具体来说，核心集采样旨在找到最具有信息量和多样性的子集，该子集能够最好地代表整个数据集，因此在对子集进行训练的语言模型上，可以实现与整个数据集上相当甚至更好的性能。这种思想所采用的直觉是，在嵌入空间中，相似的样本往往具有相似的属性，且多样性较低。因此，通过控制子集中任意两个样本之间的最小距离，可以有效地抑制冗余信息。具体来说，可以通过解决最小最大设施定位（Facility Location，FL）问题[215]，即在给定预算大小 b 下从完整集 S 中选择子集 S_b，使得 $S\backslash S_b$ 中的样本与 S_b 中最近样本之间的最大距离最小化：

$$\min_{S_b \subset S, |S_b|=b} \max_{x_i \in S \backslash S_b} \min_{x_j \in S_b} d(g(x_i), g(x_j)) \tag{5.3}$$

该问题的求解是 NP 难问题，文献 [216] 提出的 K-Center Greedy 算法，文献 [217] 提出的 Herding Greedy 算法都可以用求解近似解。除此之外，还有 DEITA[218] 结合数据质量和多样性的算法陆续提出。

（4）基于双层优化的核心集采样（Bilevel Optimization-based Coreset Sampling）则是将核心集采样问题转换为了双层优化（Bilevel Optimization）问题，它包含两个循环：1）外循环用于优化从 S 中选择子集的硬掩码或软权重；2) 内循环用于优化在 S_b 上的模型参数 θ。可以将带有自监督语言建模损失的双层优化问题，按照如下方法形式化表示：

$$S_b = \arg \min_{S'_b \subset S, |S_b|=b} \sum_{x_i \in S'_b, \theta=\theta^*} NLL_i^{A|Q} \tag{5.4}$$

$$s.t. \theta^* = \arg\min_{\theta} \sum_{x_i \in S'_b} NLL_i^{A|Q} \tag{5.5}$$

$$NLL_i = \frac{1}{|x_i|} \sum_{j=1}^{|x_i|} -\log P(x_{i(j)} | x_{i(<j)}; \theta) \tag{5.6}$$

其中 NLL_i 表示针对每个数据 x_i 的负对数似然（Negative Log Likelihood），可以使用较小的模型进行学习，比如 MPT 125M[219] 等。

3. 数据对结果影响

大语言模型经过指令微调，可以完成多种类型的任务。指令微调数据对于模型结果有着重要的影响。本节分别以通用和问题任务为例，讨论指令微调数据与模型效果之间的关系。

针对通用任务，文献 [42] 提出了“表层对齐假设”（Superficial Alignment Hypothesis）。该假设指出，模型所具备的知识与能力，绝大部分是在预训练阶段积累和形成的，而指令微调的关键作用在于，引导模型掌握在与用户互动过程中应当运用何种格式的子分布。如果这一假设是正确的，进一步推导可得，人们可以用相当少的示例集便能对预训练语言模型实现充分且有效的微调。[220]。

为此，LIMA[42] 专门收集了一个数据集，该数据集涵盖了 1000 个提示以及与之对应的回复。在这个数据集中，输出（也就是回复）部分在风格方面是相互对齐的，不过输入（即提示）却呈现出多样化的特点。具体来说，LIMA 所期望获取的输出内容，是那种带有帮助性的、符合人工智能助手风格的内容。为了收集到这样的示例，研究人员从多个来源采样收集指令数据，包括高质量网络问答社区、Super-Natural Instructions[221] 指令集，以及大量的标注者手动编写的提示与回答。网络问答社区包含多个子版块，涵盖了不同的主题。Super-Natural Instructions 指令集也包含了多种多样的生成式任务。由于标注者各自编写的提示与回答具有天然的多样性，因此指令数据的多样性得到了很好的保障。

除此之外，LIMA 研究人员做了大量的工作来保证指令数据的质量。首先，指令数据来源的可靠已经在一定程度上保证了它的质量。其次，LIMA 额外制定了一些规则进一步提高其质量。例如，对社区指令数据采样时选择排名靠前的优质回答，将所有的回答统一成 AI 助手的风格，删除过长或过短的回答，删除以第一人称开头的回答，删除包含链接的回答，标注者精心手动编写回答等等。

LLaMA 65B 模型使用 LIMA 数据进行训练后的结果如图5.3所示。Alpaca 65B[222] 同样也是基于 LLaMa 65B[34] 进行指令微调，但是它使用了 52,000 条指令微调数据。从实验结果上，可以看到使用 LIMA 仅使用 1000 条这样的指令数据，就可以媲美甚至超过指令数据是其几十倍的同等参数规模的其他模型。说明指令数据的质量和多样性是影响指令微调过程的关键因素。

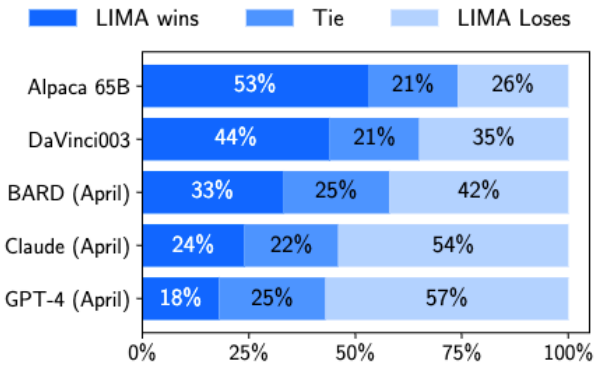


图 5.3 LLaMA 65B 模型使用 LIMA[42] 训练效果对比

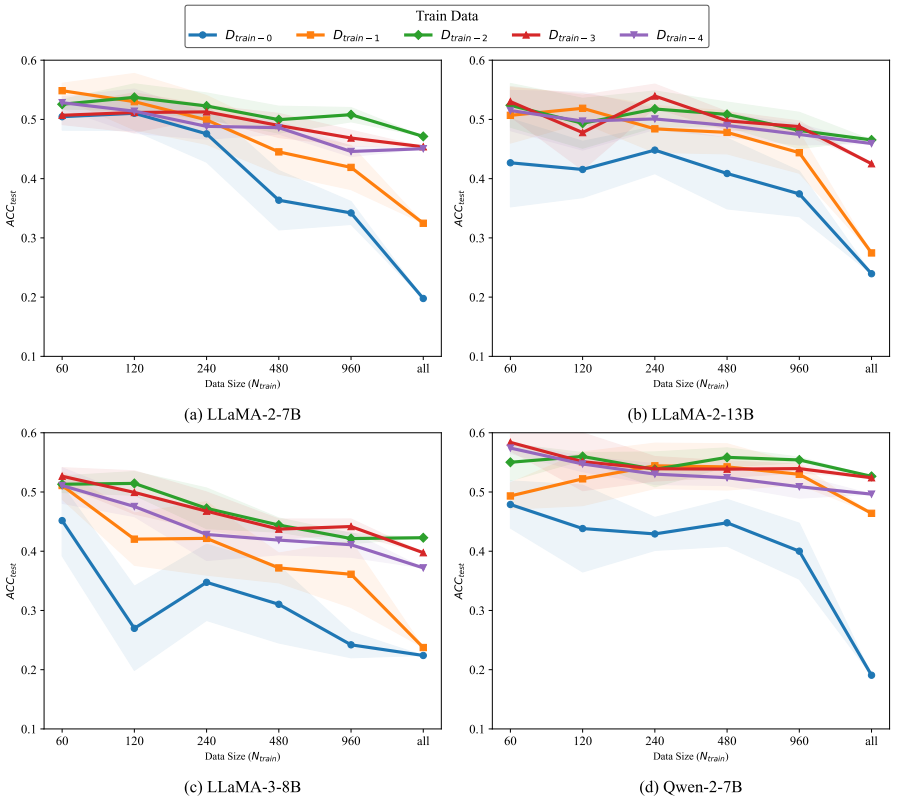
文献 [190] 研究也表明，在模型构建过程中，数据工程起着至关重要的作用，可以通过提升数据集的多样性，显著增强模型的泛化能力。训练数据多样性的提升，可以从多个方面着手，例如使用来自不同源头、具备不同特征且呈现不同分布的数据。此外，实验结果也说明，在数据选择环节，多样性有着不可忽视的作用。对比随机选择、均匀选择这两种常见方式，具备多样性的数据选择策略展现出明显优势。此外，相较于单纯聚焦于挑选高质量数据，若能将数据质量与多样性标准有机结合，模型也可以达到更好的效果[223]。

在问答任务方面，大语言模型的预训练依托于多样化的语料库来开展，这些语料库包含了多种类型的内容，并且涵盖了丰富的世界知识。大语言模型在预训练完成后，大量的知识被编码进了模型的参数之中。而通过监督微调的方式，就能够把这些已经编码进参数的知识有效地应用于问答任务里。然而，针对大语言模型的问答任务能力提升，存在着三个亟待解决的关键问题：（1）指令微调阶段，究竟需要多少数据量，才能使大语言模型掌握问答任务？（2）不同的指令微调数据集，会对大语言模型在问答任务上的表现产生怎样的影响？（3）不同的大语言模型在指令微调阶段，对于数据的需求方面存在着怎样的差异呢？

针对上述问题，文献 [224] 给出了详细的分析。研究人员使用了 ENTITYQUESTIONS[225]，这是一个包含维基百科上 24 个不同话题知识的问答数据集。选择了其中 12 个与地点相关的原始训练集作为训练数据，将它们对应的测试集作为测试集，并将剩余 12 个话题的测试集作为领域外测试集。通过设计的多模板补全机制，能够可靠地评估大语言模型对不同知识的记忆程度。利用该机制，根据其知识记忆水平将训练和测试集均进行了 5 个级别的划分。

文献 [224] 中将训练数据划分为六个不同的数据量级别，从 60 个样本到完整数据集不等，并通过从 12 个话题中均匀抽样来构建训练集。实验结果表明，仅需 60 个训练样本的指令微调，就足以使大语言模型高效执行问答任务，并展现出强大的泛化能力。如图5.4所示。无论基础模型或记忆水平如何，大语言模型在使用较少训练样本时的表现优于使用 960 个或全部样本。增加训练

数据并未带来显著的性能提升，反而可能损害模型表现。



(a) LLaMA-2-7B (b) LLaMA-2-13B

(c) LLaMA-3-8B (d) Qwen-2-7B

图 5.4 大语言模型指令微调问答任务数据量分析

此外，上述结果也显示，使用不同记忆层次的数据进行微调，会导致模型在知识激活上有显著而规律性的差异。大语言模型在回答预训练中记忆较好的知识时表现得更准确。如果使用大量在预训练模型中没有准确记忆的数据进行指令微调，会使得模型问答能力快速大幅度下降。如图5.4所示，在 LLaMA-2-7B 模型上，使用 960 条在预训练模型中没有准确记忆的数据进行微调，问答准确率就会下降到 30% 左右。LLaMA-2-13B、LLaMA-3-8B 以及 Qwen-2-7B 都存在非常类似的问题。这说明在指令微调中谨慎选择数据非常重要。同时，由于不同模型在预训练完成后，其知识记忆情况不同，这也导致需要针对不同模型构造不同的训练数据。这又进一步增大了指令微调阶段数据构造的难度。

5.1.4 指令微调训练策略

尽管从整体流程来看，指令微调的步骤并不繁杂，其训练代码甚至与预训练阶段的代码大体相同，然而，指令微调在模型获取各类关键能力的进程中却发挥着不可或缺的作用。此外，开源

模型内既存在仅完成预训练环节的模型，例如：Llama-3.1-70B、Qwen2.5-72B 等；也有经过指令微调的模型，例如：Llama-3.1-70B-Instruct、Qwen2.5-72B-Instruct 等。当着眼于特定场景下的多个任务效果提升需求时，一系列亟待解决的问题随之浮现：基于预训练模型进行训练还是基于经过指令微调的模型进行继续训练？所有任务融合在一起训练还是每个任务依次进行训练？不同的数据组成比例会对模型性能造成何种影响？这些训练策略如何影响模型性能的问题，文献 [226] 开展了较为系统探究工作。

为了简化研究难度，文献 [226] 中仅使用数学推理、代码生成和通用能力三大类任务研究数据量、数据组成比例、模型规模和指令微调训练策略等因素之间的关系。使用了三个基准评测，分别是用于数学推理的 GSM8K[227]、用于编程的 HumanEval[100] 和用于通用人类对齐的 MT-Bench[196]。在基础大模型方面使用了 LLaMA 7B 到 33B 不同参数规模进行了分析。探索了如图5.5中所示的四种不同的指令微调策略：多任务学习、顺序训练、混合顺序训练和双阶段混合微调。在指令微调训练数据方面，文献 [226] 分别使用了 GSM8K RFT[228]、Code Alpaca[229] 和 ShareGPT[41] 分别用于数据、编程和通用任务训练。

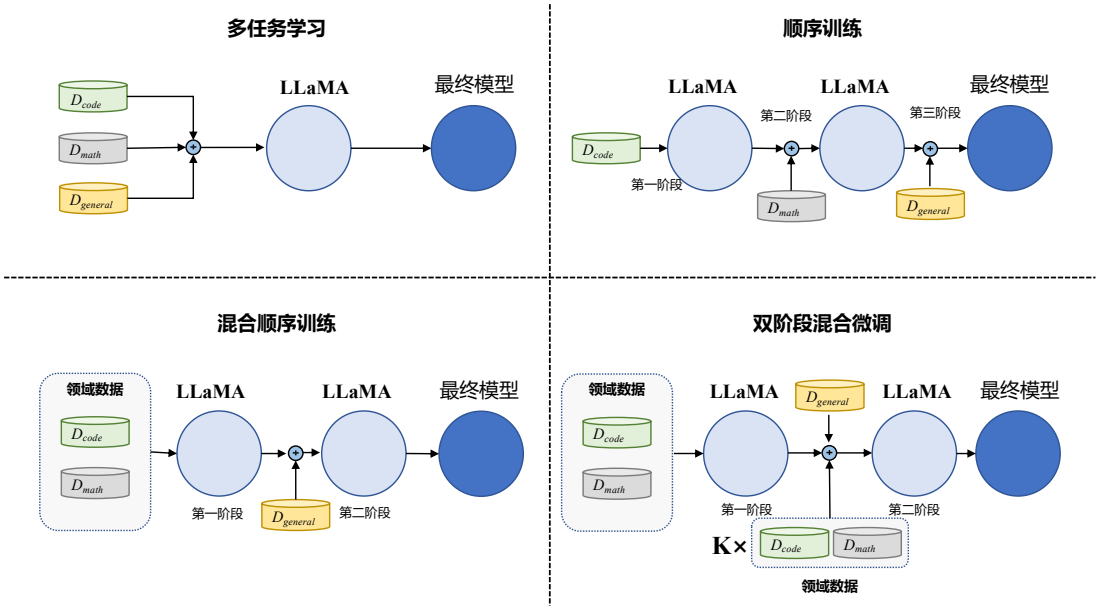


图 5.5 大语言模型指令微调训练策略[226]

如图5.5中所示，四种不同的指令微调策略的方式如下：（1）多任务学习：直接混合不同的指令微调数据源进行指令微调。如果我们将每个数据源视为不同的任务，那么这可以被视为多任务学习；（2）顺序训练：按顺序对每个数据集进行指令微调。按顺序对编程、数学推理和通用能力数据集进行训练。由于通用能力对于类人对齐最重要，将 ShareGPT 作为最后一个数据集；（3）混

合顺序训练：首先在领域数据集（代码、数学）上应用多任务学习，然后在通用能力数据集上进行指令微调；（4）双阶段混合微调：首先在领域数据集（代码、数学）上应用多任务学习，然后使用少量领域数据混合全量通用数据再进行指令微调。实验结果如表所示。

表 5.1 不同指令微调策略准确率对比[226]

方法	LLaMA - 7B			LLaMA - 33B		
	GSM8K	HumanEval	MT-Bench	GSM8K	HumanEval	MT-Bench
仅通用数据	11.10%	10.42%	5.88%	26.06%	24.30%	6.63%
仅数学数据	49.10%	6.71%	2.53%	57.91%	15.5%	3.18%
仅编程数据	4.51%	18.40%	4.30%	6.06%	26.82%	4.18%
多任务学习	**47.53**%	14.63%	5.76%	**56.69**%	18.9%	6.07%
顺序训练	31.39%	15.85%	5.72%	47.27%	24.80%	**6.73**%
混合顺序	32.60%	15.24%	6.02%	44.24%	24.4%	6.43%
双阶段混合	41.92%	**17.68**%	**6.08**%	56.36%	**25.00**%	**6.73**%

表5.1给出了不同训练策略下数学推理、代码生成和通用任务性能。从结果中可以看到，多任务学习在这些策略中保持了领域任务的能力，但对通用能力的损害最大。顺序训练和混合顺序训练虽然保持了通用能力，但损失了太多领域任务能力。从这些结果中，可以看到多阶段训练的一个固有缺点是灾难性遗忘先验知识。双阶段混合训练，这里所采用的策略是在最后阶段融合了 1/256 的领域数据和全量的通用数据，LLaMA-7B 数学推理准确率从 32.6% 上升到 41.92%，代码生成准确率从 15.24% 上升到 17.68%，相对于混合顺序和顺序训练策略都有显著改进。在最后的微调阶段混合领域任务数据对灾难性遗忘有显著缓解效果。

文献 [226] 研究还发现：（1）较大的模型通常在相同的数据量下表现出更优的性能，但是不同的任务随着模型参数增加而效果增长的速度完全不同；（2）数学推理和代码生成任务的效果随着训练数据量的增加而持续改进，而通用能力在大约在达到 1000 个样本后趋于平稳；（3）在数据有限的情况下，混合各类训练数据在一起可以在一定程度上增强所有任务效果，但在训练数据较为丰富时训练数据的混合则可能导致性能冲突；（4）指令微调数据量对效果的影响大于组成比例对效果的影响。详细的实验结果和分析可以参考文献 [226]。

5.1.5 开源指令数据集

指令数据集对于指令微调非常重要，无论手工还是自动构建都需要花费一定的时间和成本。目前已经有一些开源指令数据集，本节将选择一些常用的指令数据集进行介绍。如果按照类型来划分，指令微调数据集可以分为两大类：通用指令微调数据集（General Instruction Fine-tuning Datasets）和特定领域指令微调数据集（Domain-specific Instruction Fine-tuning Datasets）。通用指令微调数据集涵盖了各种跨领域指令，旨在提高模型在通用任务上的效果以及指令遵循能力效果。特定领域