

5 模型编辑

预训练大语言模型中，可能存在**偏见**、**毒性**、**知识错误**等问题。为了纠正这些问题，可以将大语言模型“回炉重造”——用清洗过的数据重新进行预训练，但成本过高，舍本逐末。此外，也可对大语言模型“继续教育”——利用高效微调技术向大语言模型注入新知识，但因为新知识相关样本有限，容易诱发过拟合和灾难性遗忘，得不偿失。为此，仅对模型中的特定**知识点**进行修正的**模型编辑**技术应运而生。本章将介绍模型编辑这一新兴技术，首先介绍模型编辑思想、定义、性质，其次从内外两个角度分别介绍模型编辑经典方法，然后举例介绍模型编辑的具体方法 T-Patcher 和 ROME，最后介绍模型编辑的实际应用。

* 本书持续更新，GIT Hub 链接为：<https://github.com/ZJU-LLMs/Foundations-of-LLMs>。

5.1 模型编辑简介

大语言模型有时会产生一些不符合人们期望的结果，如偏见、毒性和知识错误等。**偏见**是指模型生成的内容中包含刻板印象和社会偏见等不公正的观点，**毒性**是指模型生成的内容中包含有害成分，而**知识错误**则是指模型提供的信息与事实不符。例如，当被问到“斑马的皮肤是什么颜色的？”时，ChatGPT 错误地回答“肉色”，而实际上斑马的皮肤是黑色的，这就是一个知识错误，如图5.1。如果不及修正这些问题，可能会对人们造成严重误导。

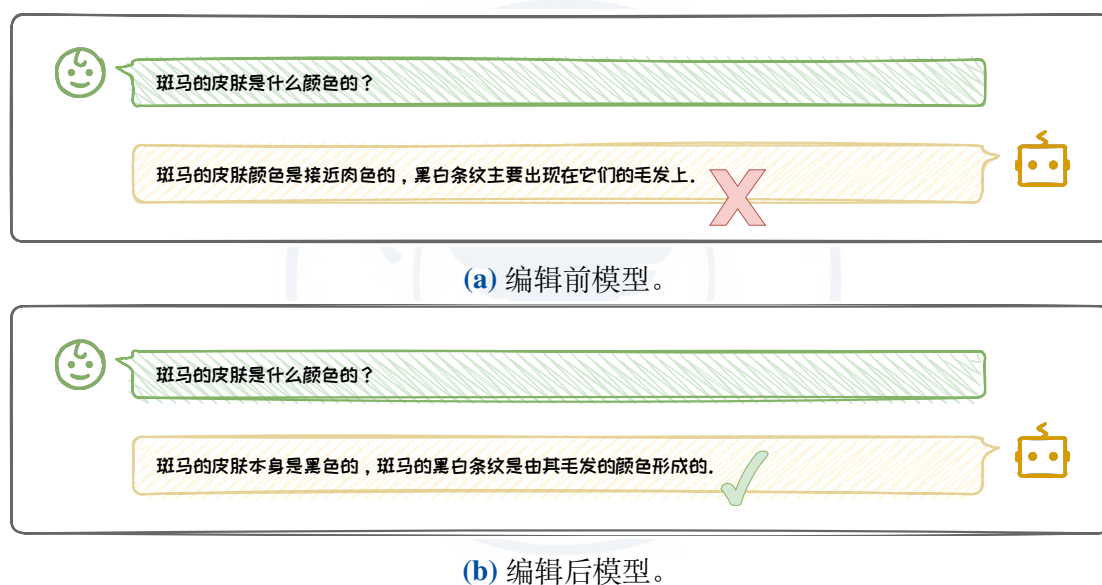


图 5.1: 大语言模型知识错误示例。

为了纠正这些问题，可以考虑重新预训练和微调两种方法。**重新预训练**是指使用矫正了偏见、祛除了毒性、纠正了知识错误的清洗后的数据对模型进行重新预训练。这种方法能够从根本上修复模型的错误输出。然而，清洗数据成本高昂，并且由于知识可能频繁更新，无法保证清洗过的数据永远是完美的；而且，重新进行预训练需要消耗巨大的计算资源，如果每次发现模型错误时都对其重新预训练，未免舍本逐末。**微调**则是在预训练模型的基础上，针对其错误进一步调整模型

参数。尽管存在多种参数高效微调方法，但直接去调整十亿级参数量的模型，还是会产生较高的训练成本；此外，由于修改模型错误所需的新知识样本有限，因此直接用相应的知识点相关的样本微调模型容易导致过拟合和灾难性遗忘。因此，重新预训练和微调都不适用于实际大语言模型的偏见矫正、毒性祛除、以及知识错误纠正。

为规避重新预训练和微调方法的缺点，**模型编辑**应运而生。其旨在精准、高效地修正大语言模型中的特定知识点，能够满足大语言模型对特定知识点进行更新的需求。本节将从思想、定义、性质等方面对模型编辑进行初步介绍。

5.1.1 模型编辑思想

在《三体 2：黑暗森林》中，面壁者希恩斯和他的妻子共同研发了一种名为“思想钢印”的设备，目的是向太空军灌输“人类必胜”的坚定信念。这个机器的原理是让接受者在接触到特定信息时，修改大脑处理过程，使之输出正向答案。模型编辑的思想大致与此相似，旨在通过增加或修改模型参数，快速有效地改变模型行为和输出。

模型学习知识的过程还可以与人类学习知识的过程相对应。首先，在体验世界的过程中，我们能够接触到海量的知识，从而形成自身的知识体系，这可以类比为大语言模型的**预训练**过程。其次，我们可以选择针对不同学科进行专门的学习，提升自己在特定领域的能力，这可以类比为大语言模型的**微调**过程。此外，在与他人交流的过程中，我们会针对特定的知识进行探讨，从而纠正自己在该知识点上的错误认知，这就可以类比为**模型编辑**的思想。

以上方式，都可以满足“纠正大语言模型”的需求。与重新预训练和微调方法不同的是，模型编辑期望能更加**快速、精准**地实现对于模型特定知识点的修正。

5.1.2 模型编辑定义

当前，模型编辑领域尚缺乏统一标准，不同研究对相关概念的定义各不相同。本书将不同工作中提到的**基于知识的模型编辑**（KME, Knowledge Model Editing）[25]和**知识编辑**（KE, Knowledge Editing）[28]等概念统一为**模型编辑**（ME, Model Editing）。此外，有些研究用“编辑”（edit）[27]或“事实”（fact）[17, 18]来表示具体的编辑对象，本书将这些概念统一为“知识点”。

模型编辑的目标可被归纳为：**修正大语言模型使其输出期望结果，同时不影响其他无关输出**。本书将模型编辑定义如下：

定义 5.1 (模型编辑)

将编辑前模型定义为 M ，编辑后模型定义为 M^* 。每一次编辑都修改模型的一个知识点 k ，知识点 k 由问题 x_k 及其对应的答案 y_k 组成。那么，模型编辑的目标可以表示为以下函数：

$$M^*(x) = \begin{cases} y_k, & \text{若 } x = x_k \text{ 或 } x \text{ 与 } x_k \text{ 相关,} \\ M(x), & \text{若 } x \text{ 与 } x_k \text{ 无关。} \end{cases}$$



上述定义中有关“相关”和“无关”的判断，涉及到模型编辑的范围问题，本书将在第 5.1.3 小节讨论。

图 5.2 用“斑马皮肤颜色”这一知识点作为示例，展示了模型编辑的概念。在这个示例中，当被询问“斑马的皮肤是什么颜色的？”时，编辑前模型回答了错误答案“肉色”，而编辑后的模型可以回答出正确答案“黑色”。

然而，实际的模型编辑过程远比理论定义复杂。这主要源于知识的内在关联性：当修改模型对某一特定知识点的认知时，由于该知识点可能与其它知识点相关联，所以可能会影响模型对其它相关知识点的理解，从而产生“牵一发而动全身”的效应。因此，**如何精确控制模型编辑的范围**成为一个关键挑战。精准可控的模型

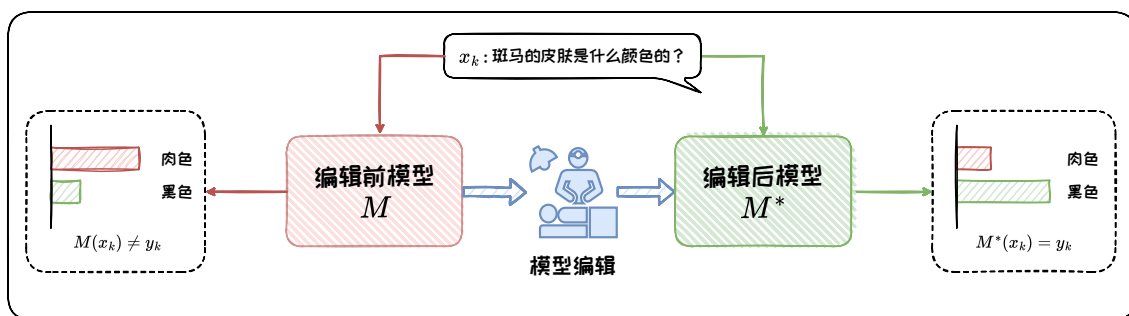


图 5.2: 模型编辑概念图。

编辑技术需要满足一系列性质。这些性质不仅反映了模型编辑的复杂性，也为评估和改进编辑方法提供了重要指标。接下来对模型编辑的关键性质进行介绍。

5.1.3 模型编辑性质

模型编辑的首要目标是纠正模型的错误回答，使其给出我们期望的答案。在此基础上，考虑到知识的内在关联性，需要进一步精准控制模型编辑的范围。除此之外，还要保证模型编辑的效率。因此，需要从多个方面控制模型编辑过程。

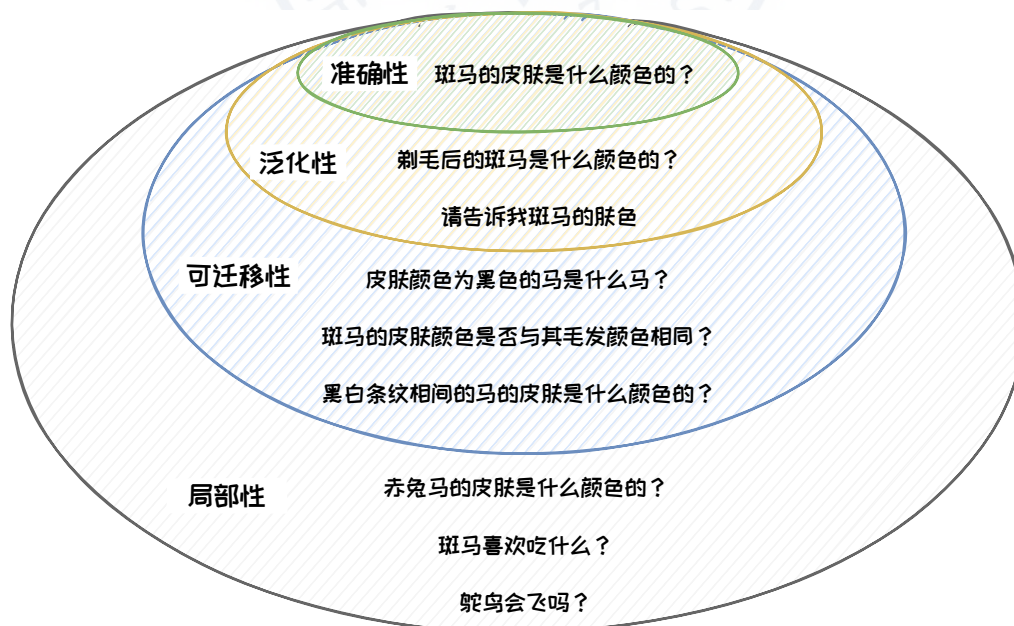


图 5.3: 模型编辑性质关系图。

本书根据已有工作 [16, 25, 27, 28], 将模型编辑的性质归纳为五个方面, 分别为**准确性 (Accuracy)**、**泛化性 (Generality)**、**可迁移性 (Portability)**、**局部性 (Locality)** 和**高效性 (Efficiency)**。图5.3通过相关示例直观地展示了这些性质之间的关系。

1. 准确性

准确性衡量对某个知识点 k 的直接修改是否有效。前面提到, 在进行模型编辑时, 许多方法通常会选取知识点 k 中的一对代表性输入输出 (x_k, y_k) 对模型进行直接修改。因此, 在评估编辑方法时, 首先需要评估编辑后模型 M^* 能否准确回答问题 x_k , 也就是图5.3中“斑马的皮肤是什么颜色的?”这个问题。如果 M^* 可以输出 y_k , 则认为本次编辑准确。采用以下公式来表示一次编辑是否准确:

$$\text{Acc} = I(M^*(x_k) = y_k)。 \quad (5.1)$$

该公式通过一个指示函数 $I(\cdot)$ 进行计算, 仅当编辑后模型在目标问题 x_k 上的输出 $M^*(x_k)$ 与答案 y_k 相匹配时, Acc 的值为 1, 否则为 0。对于多次编辑, 可采用均值来计算平均准确率。**准确性是模型编辑的最基本要求**, 它确保了编辑后的模型能够符合设计者的预期, 准确地执行特定的任务。只有当模型能够满足准确性时, 才能够进一步去满足其它性质。

2. 泛化性

泛化性用来衡量编辑后模型能否适应目标问题 x_k 的其他表达形式, 即判断模型在面对与 x_k 具有语义相似性的问题时, 能否给出统一的目标答案 y_k 。泛化性问题对应图5.3中黄色部分的问题“剃毛后的斑马是什么颜色的?”和“请告诉我斑马的肤色”, 这两个问题的正确答案都是“黑色”。

为了评估编辑后模型的泛化性, 研究者通常会构造一个**泛化性数据集** $D_G = \{(x_i, y_k)\}_{i=1}^{|D_G|}$, 其中 x_i 是与 x_k 具有语义相似性的问题, 它们的答案都为 y_k 。采用

以下公式来量化编辑后模型的泛化性：

$$\text{Gen} = \frac{1}{|D_G|} \sum_{i=1}^{|D_G|} I(M^*(x_i) = y_k). \quad (5.2)$$

当 Gen 的值为 1 时，说明编辑后模型能够正确回答 D_G 中的所有问题，此时泛化性最好。确保编辑模型的泛化性可以防止编辑后模型过度拟合特定的输入，从而保证模型对于知识点的理解。

3. 可迁移性

可迁移性是指编辑后模型将特定知识点 k 迁移到其它相关问题上的能力。为了评估编辑后模型的可迁移性，最重要的是构建可迁移性数据集，该数据集可用于评估模型对与 k 间接相关的问题的适应能力。

将可迁移性数据集表示为 $D_P = \{(x_i, y_i)\}_{i=1}^{|D_P|}$ ，其中 x_i 是与 x_k 相关但答案不同的问题， y_i 为对应答案。 x_i 可以表达为多种形式，包括反向问题、推理问题和实体替换问题等。如图 5.3 蓝色部分的问题所示，“皮肤颜色为黑色的马是什么马？”是一个反向问题，“斑马的皮肤颜色是否与其毛发颜色相同？”是一个推理问题，“黑白条纹相间的马的皮肤是什么颜色的？”则是一个实体替换问题，这三个问题的答案都不是“黑色”。可迁移性数据集 D_P 中的问题与泛化性数据集 D_G 中的问题不重叠，且问题答案不同，即 $y_i \neq y_k$ 。采用以下公式来量化编辑后模型的可迁移性：

$$\text{Port} = \frac{1}{|D_P|} \sum_{i=1}^{|D_P|} I(M^*(x_i) = y_i). \quad (5.3)$$

当 Port 的值为 1 时，说明模型可以正确回答可迁移性数据集中的所有问题，此时模型的可迁移性最好。可迁移性考察了模型在编辑知识点上的迁移能力，对于模型编辑的实际应用至关重要。然而，大多数模型编辑方法往往无法实现较好的可迁移性，这也是模型编辑中的一个挑战。

4. 局部性

局部性要求编辑后的模型不影响其他不相关问题的输出。局部性问题对应

图5.3中灰色部分的问题“赤兔马的皮肤是什么颜色的?”、“斑马喜欢吃什么?”等等。一般来说,研究者会在常用的常识数据集中选择一些与知识点 k 不相关的问题作为局部性评估。将局部性数据集定义为 $D_L = \{(x_i, M(x_i))\}_{i=1}^{|D_L|}$, 其中 x_i 是与 x_k 无关的问题。采用以下公式来量化编辑后模型的局部性:

$$\text{Loc} = \frac{1}{|D_L|} \sum_{i=1}^{|D_L|} I(M^*(x_i) = M(x_i)). \quad (5.4)$$

当 Loc 的值为 1 时, 编辑后模型的局部性最好, 此时, 编辑后模型对局部性数据集中所有问题的回答与编辑前一致。**确保局部性能够降低模型编辑的副作用**, 是模型编辑相较于朴素微调的重要改进。

5. 高效性

高效性主要考虑模型编辑的时间成本和资源消耗。在实际应用中, 模型可能需要频繁地进行更新和纠错, 这就要求编辑过程必须足够快速且资源友好。此外, 对于大量的编辑任务, 不同方法的处理效率也有所不同, 有的方法支持批量并行编辑 [2, 6, 18–20], 有的则需要依次进行 [4, 13, 17]。**高效性直接影响到模型编辑的可行性和实用性**。

在评估模型编辑方法时, 需要在这五个性质之间寻找平衡。理想的编辑方法应当在保证准确性的基础上, 尽可能地提高泛化性、可迁移性和局部性, 同时保持高效性。

5.1.4 常用数据集

前文探讨了模型编辑的定义与性质。接下来, 我们将介绍一些先前研究中使用过的具体数据集, 这些数据集可用于测试和比较不同的模型编辑方法。

在模型编辑的相关研究中, 使用最广泛的是由 Omer Levy 等人提出的 **zsRE 数据集** [14]。zsRE 是一个问答任务的数据集, 通过众包模板问题来评估模型对于特定关系 (如实体间的“出生地”或“职业”等联系) 的编辑能力。在模型编辑中,

zsRE 数据集用于检查模型能否准确识别文本中的关系，以及能否根据新输入更新相关知识，从而评估模型编辑方法的**准确性**。

表 5.1: 模型编辑相关数据集总结表格。

数据集	类型	训练集数量	测试集数量	输入	输出
zsRE[14]	知识关联	244,173	244,173	事实陈述	对象
COUNTERFACT[17]	知识关联	N/A	21,919	事实问题	对象
WikiGen[19]	文本生成	N/A	68,000	Wiki 段落	续写
T-REx-100/-1000[6]	知识关联	N/A	100/1,000	事实陈述	对象
ParaRel[4]	知识关联	N/A	253,448	事实问题	对象
NQ-SituatedQA[5]	问答	N/A	67,000	用户查询	答案
MQuAKE-CF/-T[30]	知识关联	N/A	9,218/1,825	多跳问题	对象
Hallucination[10]	幻觉	N/A	1,392	传记	传记
MMEdit-E-VQA[3]	多模态	6,346	2,093	图像问题	答案
MMEdit-E-IC[3]	多模态	2,849	1,000	图像描述	描述
ECBD[23]	知识关联	N/A	1,000	实体完成	引用实体
FEVER[2]	事实检查	104,966	10,444	事实描述	二进制标签
ConvSent[20]	情感分析	287,802	15,989	主题意见	情感
Bias in Bio[11]	人物传记	5,000	5,000	传记句子	职业
VitaminC-FC[20]	事实检查	370,653	55,197	事实描述	二进制标签
SCOTUS[10]	分类	7,400	931	法庭文件	争议主题

2023 年, [17] 提出了 **COUNTERFACT 数据集**, 被后续工作广泛采用。COUNTERFACT 被设计用来区分两种类型的知识修改: 一种是词汇的表面变化, 也就是文本中单纯的词语替换或结构调整, 不会影响信息的实质内容; 另一种是对基础事实知识显著且泛化的修改, 也就是对文本中所描述的事实或信息进行根本性的

改变，这通常需要更深层次的理解和处理能力。COUNTERFACT 能够评估模型对编辑后知识的理解和反应，进而衡量模型的**泛化性**和**局部性**。文献 [27] 在 ZsRE 和 COUNTERFACT 数据集的基础上，使用 GPT-4 生成相应问题的反向问题、推理问题和实体替换问题，构造了**可迁移性**数据集。

此外，研究者还针对不同领域和任务开发了专门的数据集，如 **Hallucination**[10] 用于纠正 GPT 语言模型中的幻觉，**ConvSent**[20] 用于评估模型在修改对话代理对特定主题的情感时的效果。表 5.1 从数据类型、样本数量以及输入输出形式方面总结了一些模型编辑相关数据集。这些数据集涵盖了从事实检查、知识关联到特定领域等多种类型，体现了模型编辑技术在不同场景中的应用潜能。

本节介绍了模型编辑的定义、性质、评估方法以及常用数据集，对模型编辑任务做出了详细的解释和说明。接下来，第 5.2 节将对模型编辑方法进行系统性概述，将其分为外部拓展法和内部修改法，并介绍每类方法的代表性工作。第 5.3 节和第 5.4 节将分别深入探讨外部拓展法中的 T-Patcher 方法和内部修改法中的 ROME 方法，通过这两种代表性方法，帮助读者更加细致地理解模型编辑方法的研究过程。最后，第 5.5 节将全面介绍模型编辑的实际应用，并分别举例说明解决思路。

5.2 模型编辑经典方法

冒险游戏中的勇者需要升级时，可以从内外两个方面进行改造。外部改造主要通过置办新的道具和装备，它们能够赋予勇者新的能力，同时保留其原有技能。内部改造则相当于去锻炼自身，通过增加智力、体力、法力等属性，从自我层面获得提升。如果将大语言模型比作冒险游戏中的勇者，那么模型编辑可被看作一种满足“升级”需求的方法，可以分别从内外两个角度来考虑。本文参考已有工作 [16, 25, 27, 28]，将现有编辑方法分为**外部拓展法**和**内部修改法**。概括来说，外部拓展

法通过设计特定的训练程序，使模型在保持原有知识的同时学习新信息。内部修改法通过调整模型内部特定层或神经元，来实现对模型输出的精确控制。图 5.4 给出了模型编辑方法的分类：外部拓展法包括知识缓存法和附加参数法，内部修改法包括元学习法和定位编辑法。接下来，本节将对每类编辑方法展开介绍。

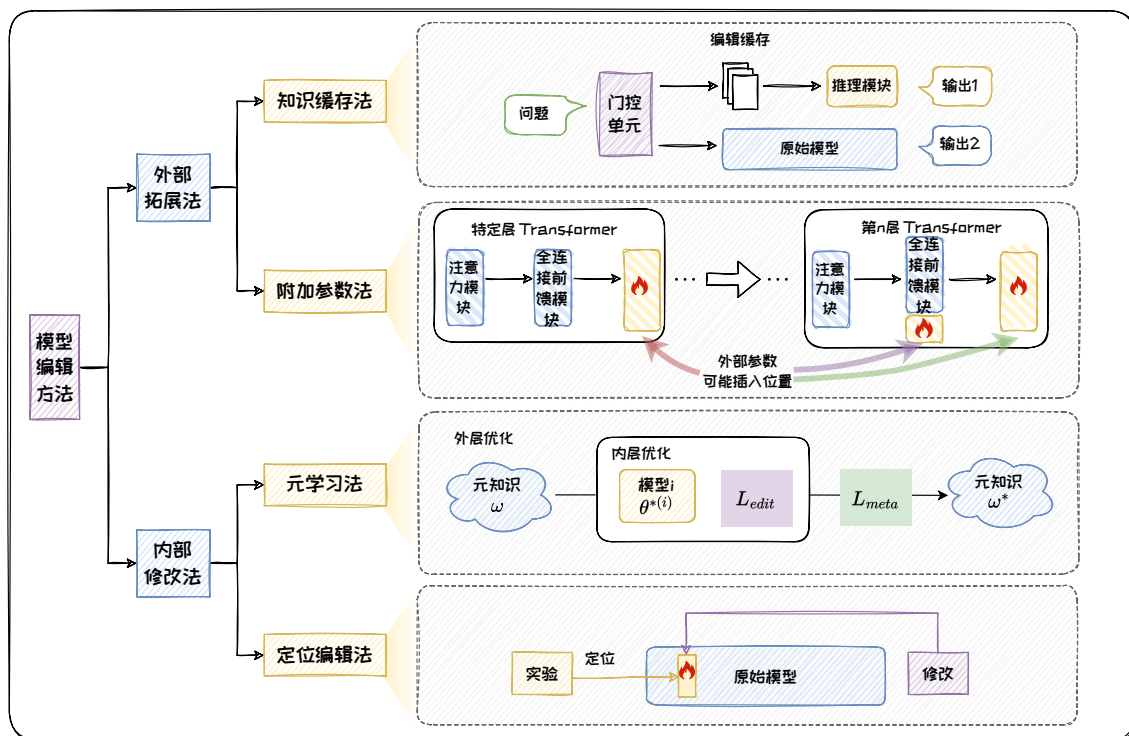


图 5.4: 模型编辑方法分类图。

5.2.1 外部拓展法

外部拓展法的核心思想是将新知识存储在附加的外部参数或外部知识库中，将其和原始模型一起作为编辑后模型。这种方法尤其适合具有良好可扩展性的预训练大语言模型，因为它提供了足够的空间来容纳大量参数，能够存储更多新知识。此外，该方法不会改变原始模型参数，可降低对模型内部预训练知识的干扰。

根据外部组件是否直接整合进模型本身的推理过程，外部拓展法又可划分为知识缓存法和附加参数法。延续冒险游戏的比喻，知识缓存类似于勇者的技能书，

需要时可以查阅获取特定知识；而附加参数则如同可升级的装备，直接增强勇者
的整体能力。

1. 知识缓存法

知识缓存法中包括三个主要组件，分别为门控单元、编辑缓存和推理模块。**编辑缓存**充当一个知识存储库，用于保存需要修改的知识，这些知识由用户通过不同的形式指定。**门控单元**用于判断输入问题与编辑缓存中的知识的相关程度，可通过分类 [20] 或噪声对比估计 [21] 等任务进行训练。**推理模块**获取原始输入问题和编辑缓存中的知识作为输入，通过监督训练的方式学习预测用户期望的结果。

在推理时，门控单元首先判断输入的问题是否与编辑缓存中的某个知识点相关，如果相关，则从编辑缓存中取出该知识点，将其与输入一并交给推理模块，由推理模块给出答案；如果不相关，则用原始模型去推理给出答案。图 5.5 为知识缓存法示意图，其中，假设编辑缓存中存储的是与斑马肤色有关的知识，则问题“鸵鸟会飞吗？”由门控单元判断为与编辑缓存中的所有知识点都不相关的问题，因此由原始模型推理出答案；问题“皮肤为黑色的马是什么马？”由门控单元判断为与“斑马的肤色”这个知识点相关的问题，因此由训练好的推理模块给出修改后答案。

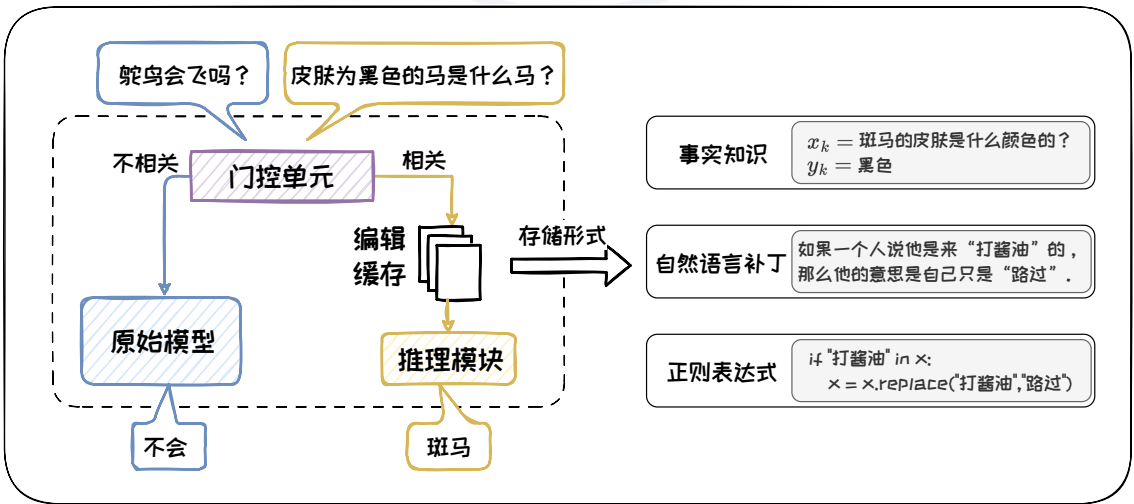


图 5.5: 知识缓存法示意图。

此外,编辑缓存中知识点的存储形式可以分为事实知识、自然语言补丁和正则表达式三种。**事实知识**以问题-答案对 (x_k, y_k) 存储编辑实例,这种存储形式适用于答案明确的事实性问题,**SERAC**[20] 是一种代表性方法。**自然语言补丁 Language Patch**[21] 按照“如果……那么……”的句式描述编辑知识,类似于 Prompt。这种存储形式适用于修正模型对自然语言中非字面含义语句的理解,且便于人们创建、编辑或删除补丁,使得模型能够通过人类反馈不断修正输出。**正则表达式**是一种基于文本匹配和替换的技术,它使用特定的模式来识别和修改文本中的特定部分,适用于精确的文本语义替换。这是一种早期的原始方法,然而由于编写复杂、泛化性低,因此在模型编辑中并不常用。

知识缓存法直接通过编辑缓存中的信息进行检索,不依赖目标标签的梯度信息,因此可以简化模型编辑过程,使其更加高效直接。然而,这种从外界获取知识的方式相当于在让大语言模型进行求助,而并非将新的知识真正内化为自己的一部分。附加参数法对这种局限进行了改良。

2. 附加参数法

与知识缓存法相比,附加参数法可以将外部参数整合进模型结构,从而有效利用和扩展模型的功能。这类方法的思想与参数高效微调中的**参数附加方法**类似,都是将外部参数插入到模型中的特定位置,冻结原始模型,只训练新引入的参数以修正模型输出,如图 5.6。具体而言,不同方法将外部参数插入到模型的不同位置。例如,**CALINET**[6] 和 **T-Patcher**[13] 通过修改模型最后一层 Transformer 的**全连接前馈模块**来实现。**CALINET** 首先通过一种对比知识评估方法找出原始模型的知识错误,然后在模型最后一个全连接前馈模块添加一个新的参数矩阵,并通过最小化校准数据上的损失来训练新参数,以纠正模型的错误。**T-Patcher** 与此类似,同样是在原始模型的最后一个全连接前馈模块引入有限数量的可训练神经元,每个神经元对应一个知识点。关于 **T-Patcher** 的具体介绍将在第 5.3 节给出。

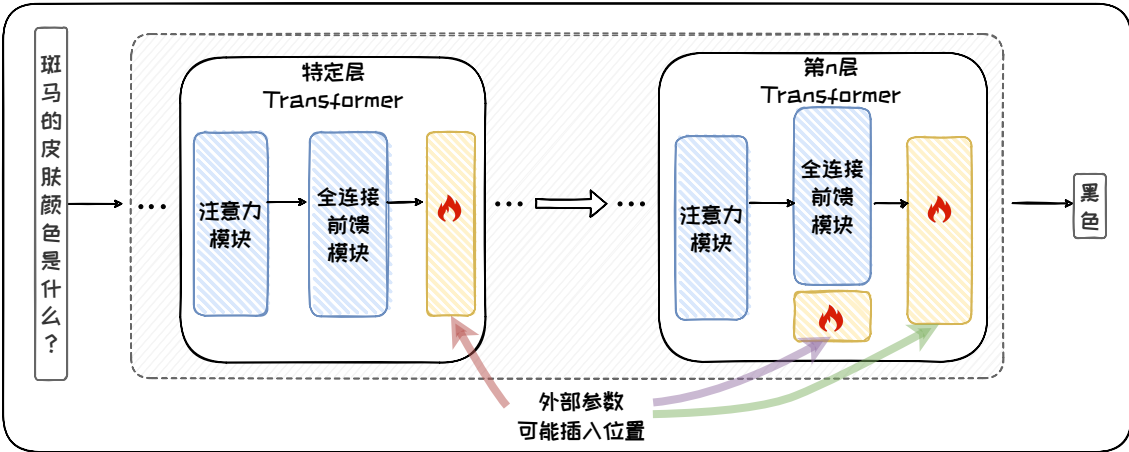


图 5.6: 附加参数法示意图。

GRACE[10] 则将外部参数以适配器的形式插入模型的特定 Transformer 层中，插入位置随模型变化而变化，如 BERT 的倒数第 2 层和 GPT2-XL 的第 36 层。其中，适配器是一个用于缓存错误知识（Keys）和对应的修正值（Values）的键值存储体，被称作“codebook”。在 codebook 中，每个错误知识都有一个对应的修正值，以及一个用于匹配相似输入的延迟半径，且随着时间的推移，codebook 会被持续更新。延迟半径用于判断当前输入是否与 codebook 中的任何错误相似，如果是，则应用相应的修正值进行编辑。

综上，知识缓存法通过引入编辑缓存机制，有效地辅助模型在庞大的知识体系中迅速定位并检索最新信息；附加参数法通过引入额外参数，实现了对模型特定输出的精细调整。这两种方法的核心优势在于**对原始模型的最小化干预**，保证了模型编辑的**局部性**。

然而，外部拓展法的实际有效性在很大程度上取决于对知识的存储与检索能力，这种依赖会导致存储资源需求的增加。因此，在具体应用时，我们需要在**保证模型局部性和应对存储限制**之间寻求平衡。

5.2.2 内部修改法

与需要额外存储空间的外部拓展法不同，内部修改法能够让模型在不增加物理存储负担的情况下直接优化自身。**内部修改法旨在通过更新原始模型的内部参数来为模型注入新知识**，能够优化模型的自我学习和适应能力，提高其在特定任务上的表现，而不是仅仅停留在表面的知识积累。内部修改法又可以分为**元学习**法和**定位编辑法**。其中，元学习法通过“学习如何学习”来获取元知识，再基于元知识实现模型编辑；定位编辑法则专注于对模型局部参数的修改，首先识别与目标知识最相关的模型参数，然后仅更新这些特定参数，通过“先定位后编辑”的策略节省更新模型所需成本。

1. 元学习法

元学习指的是模型“学习如何学习”（Learning to Learn）的过程。基于元学习的模型编辑方法，旨在让模型“学习如何编辑”（Learning to Edit），核心思想是使模型从一系列编辑任务中提取通用的知识，并将其应用于未见过的编辑任务，这部分知识被称为**元知识** ω [12]。元知识是模型在进行编辑前可以利用的知识，包括优化器参数 [24]、超网络 [2, 19] 等多种形式。元知识的训练过程被称为**元训练**，其目标是获得一个较好的元知识 ω ，使得后续的每次编辑只需少量样本即可快速收敛。

元训练过程可以看作一个双层优化问题 [12]，如公式 5.5 所示。双层优化（Bilevel Optimization）是一个层次化的优化框架，其中，内层优化问题可作为外层优化问题的约束。基于元学习的编辑方法如图 5.7 所示。图中，内层优化是模型在不同编辑任务上的优化，外层优化是元知识在验证集中的编辑任务上的综合优化。

$$\begin{aligned} \omega^* &= \arg \min_{\omega} \sum_{i=1}^n L_{\text{meta}}(\theta^{*(i)}(\omega), \omega, D_k^{\text{val}(i)}) \\ \text{s.t. } \theta^{*(i)}(\omega) &= \arg \min_{\theta} L_{\text{edit}}(\theta^{(i)}, \omega, D_k^{\text{train}(i)}). \end{aligned} \quad (5.5)$$

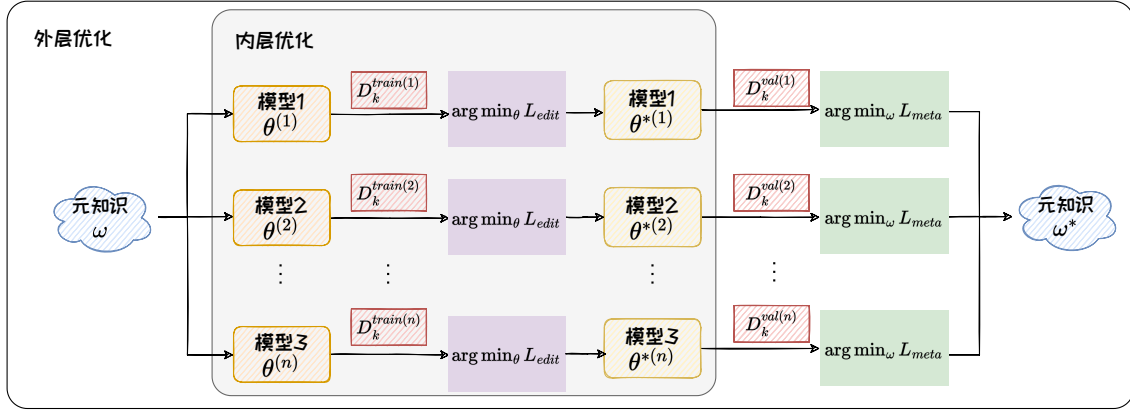


图 5.7: 元学习法示意图。

内层优化问题旨在学习模型在具体编辑任务上的参数。在内层中，设共有 n 个编辑任务，则对于第 i 个编辑 ($i \in [1, n]$ 且 $i \in \mathbb{Z}^+$)，用 $D_k^{(i)}$ 表示该次编辑涉及到的有关知识点 k 的数据集，其中 $D_k^{(i)}$ 是由 (x_k, y_k) 组成的问题-答案对集合，可划分为训练集 $D_k^{\text{train}(i)}$ 和验证集 $D_k^{\text{val}(i)}$ 。设模型原始参数为 $\theta^{(i)}$ ，在 $D_k^{(i)}$ 上优化后的参数为 $\theta^{*(i)}$ ，则内层优化是在元知识 ω 的基础上更新对应编辑任务上的模型参数的过程。在内层优化中， L_{edit} 是有关每次编辑的损失函数，其含义为希望能够最小化编辑后模型 $\theta^{*(i)}$ 在 $D_k^{\text{train}(i)}$ 上的预测误差，如分类任务中的交叉熵函数。

外层优化问题旨在学习可以泛化到其他编辑任务上的元知识。外层优化通常在验证集 $D_k^{\text{val}(i)}$ 上进行，希望能够根据模型 $\theta^{*(i)}$ 在其对应编辑验证集上的损失来更新元知识 ω 。外层优化中的 L_{meta} 是有关元学习的损失函数，其含义为希望能够找到一个元知识 ω ，使得在该知识的基础上进行优化后的模型在所有 $D_k^{\text{val}(i)}$ 上的预测误差之和最小。

ENN[24] 将元知识看作优化器参数，通过更新优化器参数，使后续编辑中模型参数的训练更为高效，从而使模型学习如何快速编辑。ENN 引入了编辑函数和梯度下降编辑器，在内层优化过程中，每次都在一个编辑任务上对模型参数进行较少次数的梯度更新，再用更新后的参数来更新优化器参数。然而，ENN 是针对小型网络 ResNet 设计的，当应用于大型模型时，会面临训练成本高等问题。

为了拓展元学习法在大型模型架构上的应用，KE[2] 将元知识作为超网络，提出了一种通过训练超网络来学习模型参数更新值的方法。在训练超网络时，损失函数由两部分组成，一部分用于确保准确性，另一部分用于确保局部性，并设置了边界值来表示约束的严格程度。训练好的超网络根据输入问题生成模型的参数更新值，使模型能够在特定输入下输出期望的结果，同时保持其他预测不变。为了进一步增强超网络对于大型语言模型的普适性，MEND[19] 通过低秩分解的方法来优化超网络辅助模型参数更新的过程。首先，它利用全连接层中梯度的秩-1 特性，将损失函数关于每一层参数的梯度分解为两个向量的乘积。然后，使用超网络接收分解后的向量作为输入，并输出经过编辑的新向量。最后，这些新向量再次相乘，得到新的参数梯度，并通过一个可学习的缩放因子来计算最终的模型参数更新值。MEND 以较少的参数高效地编辑大型模型，节省了计算资源和内存。

总结来说，基于元学习的编辑方法通过“学习如何编辑”来提高模型在面对新编辑任务时的适应性和泛化能力，能够从一系列编辑任务中提取通用的知识，进而在遇到未见过的编辑任务时，仅使用少量样本训练即可快速收敛，从而节省计算资源和时间。然而，元学习编辑方法也存在不足之处。该方法训练过程较为复杂，应用于大型模型时常常面临训练成本高的问题。尽管 KE 和 MEND 通过使用超网络和梯度低秩分解等技术优化了参数更新过程，减少了计算资源的需求，但仍需进一步提升其对更复杂任务和更大规模模型的适应性与效率。此外，元学习编辑方法从全局视角对模型参数进行更新，即使添加了与局部性相关的损失函数，也有可能对模型原本的知识产生影响，导致模型的不稳定。

2. 定位编辑法

与元学习法相比，定位编辑法修改的是原始模型的局部参数。其先定位到需要修改的参数的位置，然后对该处的参数进行修改。实现定位前需要了解大语言模型中知识的存储机制。当前，对知识存储机制的探索主要依靠定性实验来完成。通

过在一个 16 层 Transformer 的语言模型上进行实验，可以得到结论：**Transformer 中的全连接前馈模块可以看作存储知识的键值存储体 [8]**。其中，全连接前馈模块的输入称为查询（query）向量，代表当前句子的前缀；将全连接前馈模块的上投影矩阵中的向量称为键（key）向量，下投影矩阵中的向量称为值（value）向量。

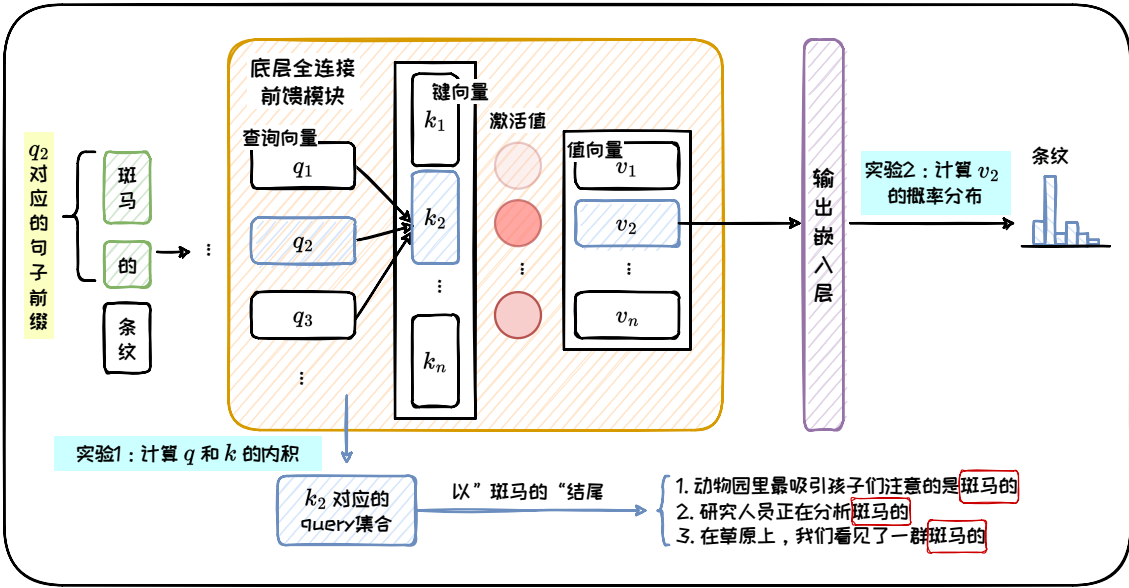


图 5.8: Transformer 可看作键值存储体。

图 5.8展示了文献 [8] 在模型底层全连接前馈模块上进行实验的过程。在实验 1 中，将每个 key 都分别与所有 query(q_1, q_2, q_3, \dots) 做内积运算，图中以 k_2 为例展示了这一过程。将 k_2 与所有 query 进行内积运算后，发现 k_2 与包括 q_2 在内的几个 query 的激活值较大。将每个 key 对应的激活值较大的 query 收集起来作为一个集合，在该集合中观察到这些 query 对应的前缀都有着相同或相似的模式。例如，图中 k_2 对应的 query 都以”斑马的“结尾，也就是说， k_2 存储了与“斑马的”相关的文本模式。在实验 2 中，将每个 key 对应的 value 与输出嵌入层（Output Embedding Layer）矩阵相乘，并应用 softmax 函数转换为概率分布，然后比较这些分布与对应 query 的下一个词的相关性，结果发现二者高度相关。例如，图中将 k_2 对应的 v_2 进行上述操作，发现下一个词为”条纹“的概率最大，这与 q_2 的下一个词相符。

综合实验 1 和实验 2，文献 [8] 指出：在全连接前馈模块中，模型通过 key 总结了当前 query 所代表的句子前缀的特征，又通过类似键值匹配的机制查找到了相应的 value，也就是下一个词的概率分布。也就是说，Transformer 中的全连接前馈模块可以看作键值存储体对知识进行存储。

基于上述结论，KN[4] 提出了知识神经元的概念。其将全连接前馈模块的每个中间激活值定义为一个知识神经元，并认为知识神经元的激活与相应知识点的表达密切相关。为了评估每个神经元对于特定知识预测的贡献，KN 将有关该知识点的掩码文本输入给预训练模型，获取隐藏状态后，将其输入到模型每一层的 FFN 中，通过归因方法，积累每个神经元在预测正确答案时的梯度变化，从而确定哪些神经元在知识表达过程中起关键作用。这种方法不仅能够识别出显著影响知识表达的神经元，还能过滤掉那些对知识预测贡献较小的神经元。在确定了知识神经元对于知识预测的贡献后，可以通过直接在模型中修改特定知识神经元对应的键向量，来诱导模型输出的编辑后的知识，从而达到模型编辑的效果。

此外，ROME[17] 设计了一种因果跟踪实验，进一步探索中间层全连接前馈模块与知识的关系，优化了知识存储机制的结论。在编辑方法上，与定位和编辑全连接前馈模块中的单个神经元的 KN 不同，ROME 提出更新整个全连接前馈模块来进行编辑。通过 ROME 对 GPT-J 模型上进行编辑，在准确性、泛化性和局部性方面都有良好表现。因此，ROME 成为近年来备受瞩目的模型编辑方法，为未来的模型编辑和优化工作提供了重要的参考和指导。我们将在第 5.4 节详细介绍 ROME 中对于因果跟踪实验和编辑方法的设计。MEMIT[18] 在 ROME 的基础上扩展到对不同知识的大规模编辑，可以同时执行数千次编辑。

总的来说，定位编辑法修改大语言模型的局部参数，在保持模型整体结构和性能的同时，能够对特定知识点进行精准的更新和编辑。与其他方法相比，定位编辑法可同时保持较高的准确性、泛化性和局部性，且适用于各类模型。

5.2.3 方法比较

上述各种模型编辑方法各有优劣。本书参考文献 [27] 中的实验结果, 对主流模型编辑方法在各个性质上的表现进行对比, 结果如表 5.2 所示。表中用“高”、“中”、“低”三个级别定性表示方法的准确性、泛化性、可迁移性和局部性, 用“✓”和“✗”来表示方法的高效性, 即是否支持批量编辑。标注“-”的表明未进行测试。

表 5.2: 模型编辑方法比较。

		方法	准确性	泛化性	可迁移性	局部性	高效性
外部 拓展 法	知识缓存法	SERAC	高	高	低	高	✓
	附加参数法	CaliNET	低	低	-	中	✓
		T-Patcher	高	高	高	中	✗
内部 修改 法	元学习法	KE	低	低	-	高	✓
		MEND	中	高	中	高	✓
	定位编辑法	KN	中	低	-	中	✗
		ROME	高	高	高	高	✗
		MEMIT	高	高	高	高	✓

从表 5.2 中可看出, 在外部拓展法中, 基于知识缓存的 SERAC 在无需额外训练的情况下提供了高效的编辑能力, 保证了高准确性、泛化性和局部性, 适合快速响应和批量编辑, 但可迁移性较差, 编辑缓存和推理模块仍有待优化。基于附加参数的 CaliNET 和 T-Patcher 提供了对模型的直接编辑能力, 但 CaliNET 对不同模型和数据的适应性较差, 而 T-Patcher 虽然保持了高准确性、泛化性和可迁移性, 但在批量编辑时, 对内存的需求较高。

在内部修改法中, 基于元学习法的 KE 和 MEND 将元知识看作超网络, 通过使用模型“学习如何编辑”, 提高了泛化性和训练效率, 且支持批量编辑。然而, 基于元学习的方法训练过程设计较为复杂, 在应用于大型模型时可能受限。基于定位编辑的 KN、ROME 和 MEMIT 则专注于精确定位和编辑模型内部的特定知识。

ROME 和 MEMIT 都能保证高准确性、泛化性、可迁移性和局部性，然而这两种方法主要针对 decoder-only 模型设计，因此未比较其在 encoder-decoder 架构模型上的表现。此外，MEMIT 在 ROME 的基础上针对批量编辑进行了优化，在满足高效性的同时依然能够保证其它性质的稳定。

本节对模型编辑领域不同类别的方法进行了概述和举例介绍。这些方法展示了对大语言模型进行有效修正的多种途径，各有优势和局限。在实际应用中，应根据需求、可用资源和期望的编辑效果选取合适的方法。根据表 5.2，T-Patcher 和 ROME 这两种方法在准确性、泛化性、可迁移性和局部性上表现较优。因此，接下来的两节将以这两种方法为代表，更加细粒度地解释模型编辑的内部机制。

5.3 附加参数法：T-Patcher

附加参数法在模型中引入额外的参数，并对这部分参数进行训练以实现特定知识的编辑。其在准确性、泛化性、可迁移性等方面均取得良好的表现。T-Patcher 是附加参数法中的代表性方法，其在模型最后一个 Transformer 层的全连接前馈层中添加额外参数（称为“补丁”），然后对补丁进行训练来完成特定知识的编辑，如图 5.9。T-Patcher 可以在不改变原始模型整体架构的情况下，对模型进行编辑。本节将从补丁的位置、补丁的形式以及补丁的实现三个方面对 T-Patcher 展开介绍。

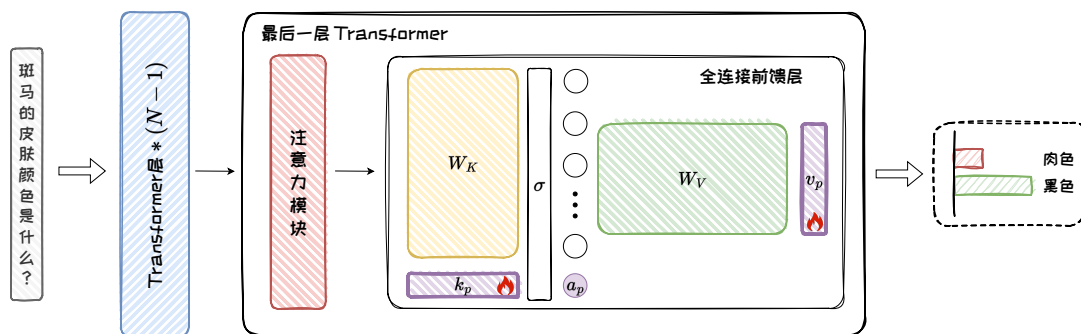


图 5.9: T-Patcher 方法 (图中紫色图块为补丁)。

5.3.1 补丁的位置

将补丁添加在模型中的不同位置会影响模型编辑的效果。T-Patcher 将全连接前馈层视为**键值存储体**，并选择在最后一个 Transformer 层的全连接前馈层中添加补丁参数。在这种设计中，添加补丁相当于向键值存储体中增加新的记忆单元，通过精确控制补丁的激活，T-Patcher 能够针对特定输入进行修正，并减少对无关输入的影响。此外，由于全连接前馈层结构简单，因此只需要添加少量参数即可实现有效编辑。

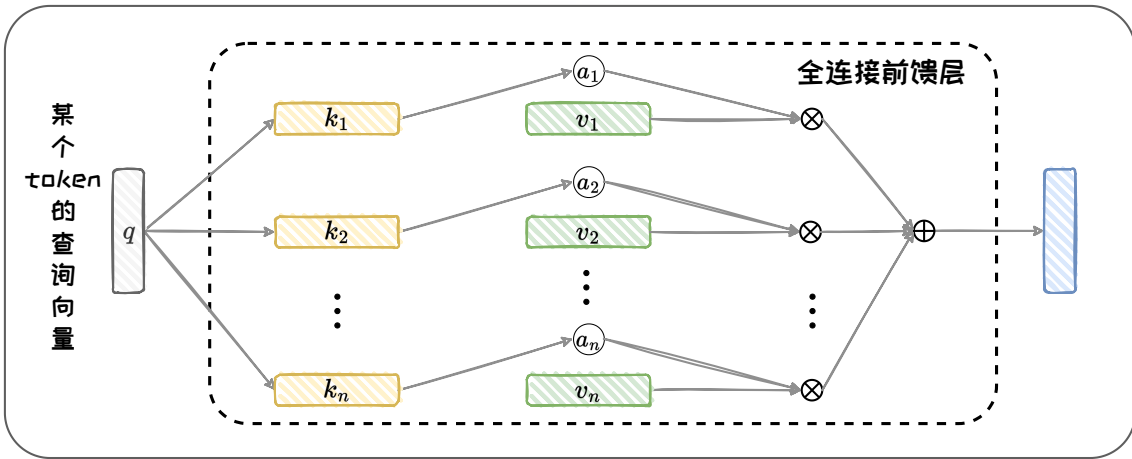


图 5.10: 键值存储体 (省略激活函数)。

具体而言，T-Patcher 将全连接前馈层视为如图5.10所示的键值存储体，键值存储体包含键向量矩阵 $W_{fc} = [k_1, k_2, \dots, k_n]$ 及其偏置向量 b_k 、激活函数 σ 和值向量矩阵 $W_{proj} = [v_1, v_2, \dots, v_n]$ 及其偏置向量 b_v 。其中，每个键向量对应着输入文本中的特定模式，如 n-gram 或语义主题，而每个值向量则关联着模型输出的概率分布。

在查询存储体中找到相应知识的过程如下：对于某个输入的 Token，其查询向量为 q 。当 q 输入全连接前馈层时，首先与矩阵 W_{fc} 相乘，计算出激活值向量 a ，其中每个分量代表 q 与对应键向量 k 的关联程度。随后， a 与矩阵 W_{proj} 相乘，得

到全连接前馈层的输出结果。该过程可以视为使用激活值对矩阵 W_{proj} 中的所有值向量进行加权求和：

$$\begin{aligned} a &= \sigma(q \cdot W_{fc} + b_k) \\ FFN(q) &= a \cdot W_{proj} + b_v. \end{aligned} \quad (5.6)$$

基于上述键值存储体的观点，全连接前馈层的隐藏层维度可被理解为其“记忆”的文本模式的数量，如果能够向全连接前馈层中添加更多与不同文本模式相关联的键值对，就可以向模型中插入新的事实信息，从而实现模型编辑。因此，T-Patcher 在全连接前馈层中增加额外参数，即添加补丁。而且，T-Patcher 仅在模型的最后一层添加补丁，以确保补丁能够充分修改模型的输出，而不被其他模型结构干扰。在下一节中，我们将详细介绍应该添加什么样的补丁。

5.3.2 补丁的形式

基于键值存储体的观点，T-Patcher 将补丁的形式设计为键值对。具体地，T-Patcher 在全连接前馈层中加入额外的键值对向量作为补丁，通过训练补丁参数从而实现模型编辑。补丁的形式如图5.11所示，它主要包括一个键向量 k_p 、一个值向量 v_p 和一个偏置项 b_p 。

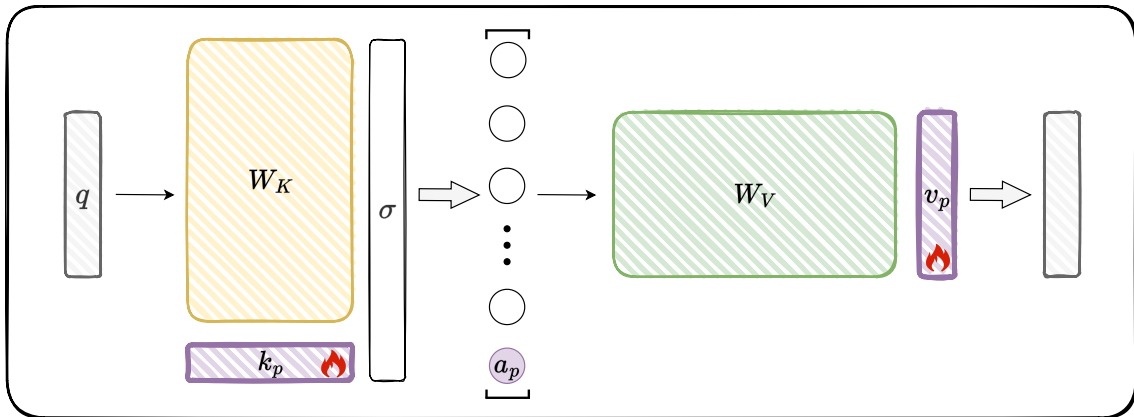


图 5.11: 补丁的形式。

在添加补丁后，全连接前馈层的输出被调整为：

$$[\mathbf{a} \ a_p] = \sigma(\mathbf{q} \cdot [\mathbf{W}_{fc} \ \mathbf{k}_p] + [\mathbf{b}_k \ b_p])$$

$$FFN_p(\mathbf{q}) = [\mathbf{a} \ a_p] \cdot [\mathbf{W}_{proj} \ \mathbf{v}_p]^\top + \mathbf{b}_v = FFN(\mathbf{q}) + a_p \cdot \mathbf{v}_p, \quad (5.7)$$

其中， a_p 为补丁的激活值，代表补丁对输入查询的响应程度。在添加补丁之后， a_p 与值向量 \mathbf{v}_p 之积会形成偏置项叠加到全连接前馈层的原始输出之上，以调整模型的输出。补丁就像是一个很小的修正器，只会被相关的输入查询激活。

5.3.3 补丁的实现

在确定了补丁的位置和形式之后，下一步便是训练补丁以实现模型编辑。本节将深入讨论如何训练这些补丁来有满足模型编辑的主要性质。T-Patcher 冻结模型的原有参数，仅对新添加的补丁参数进行训练。此外，对于给定的编辑问题，T-Patcher 为每个需要编辑的 Token 都添加一个补丁，从而可以精确地针对每个编辑需求进行调整。最后，T-Patcher 从编辑的准确性和局部性两个角度出发对损失函数进行设计。接下来对其损失函数进行介绍。

1. 准确性

确保精确编辑是 T-Patcher 的核心目标之一。在训练过程中，为了强化补丁对模型的正确修改，首先需要针对准确性设计特定的损失函数。对于补丁的准确性，T-Patcher 主要关注两个方面：(1) 确保补丁可以在目标输入下可以被激活；(2) 一旦被激活，补丁应该能够准确地调整模型输出以符合预期的结果。为此，T-Patcher 设计了**准确性损失** L_{acc} ，它包括**激活损失** l_a 和**编辑损失** l_e ：

$$L_{Acc} = l_a(\mathbf{k}_p, b_p) + \alpha l_e(\mathbf{k}_p, \mathbf{v}_p, b_p) \quad (5.8)$$

$$l_a(\mathbf{k}_p, b_p) = \exp(-\mathbf{q}_e \cdot \mathbf{k}_p - b_p) \quad (5.9)$$

$$l_e(\mathbf{k}_p, \mathbf{v}_p, b_p) = CE(y_e, p_e), \quad (5.10)$$

其中, \mathbf{q}_e 是编辑样本在全连接前馈层处的查询向量, y_e 是该补丁对应的目标 Token, p_e 是模型在补丁作用下的预测输出, CE 是交叉熵损失函数, α 是激活损失 l_a 的权重。

在准确性损失 L_{Acc} 中, 激活损失 l_a 负责确保补丁在目标输入下可以被激活, 它通过最大化编辑样本的查询向量 \mathbf{q}_e 对补丁的激活值, 从而确保修补神经元对特定编辑需求的响应。另一方面, 编辑损失 l_e 主要确保补丁在被激活后能够有效地将模型输出调整为该补丁对应的目标 Token。具体而言, T-Patcher 使用交叉熵损失函数作为编辑损失, 用于评估补丁调整后的输出 p_e 与该补丁对应的目标 Token y_e 之间的一致性, 确保补丁的调整正确实现预期的修正效果。准确性损失是实现 T-Patcher 编辑目标的关键, 它确保补丁在必要时被激活, 并且激活后能够有效地达到预期的编辑效果。

2. 局部性

模型编辑不仅要确保精确的编辑, 而且要求在对目标问题进行编辑时, 不应影响模型在其他无关问题上的表现。为了保证编辑的局部性, T-Patcher 设计了特定的损失函数来限制补丁的激活范围, 确保其只在相关的输入上被激活。为了模拟无关数据的查询向量分布, 以便在训练过程中控制激活范围, T-Patcher 会随机保留一些先前已经处理过的查询向量, 组成记忆数据集 $D_M = \{\mathbf{q}_i\}_{i=1}^{|D_M|}$, 这些查询向量与当前的编辑目标无关。基于该数据集, T-Patcher 定义了**记忆损失** L_m 来保证编辑的局部性, 该损失包含 l_{m1} 和 l_{m2} 两项:

$$L_m = l_{m1}(\mathbf{k}_p, b_p) + l_{m2}(\mathbf{k}_p, b_p, \mathbf{q}_e) \quad (5.11)$$

$$l_{m1}(\mathbf{k}_p, b_p) = \frac{1}{|D_M|} \sum_{i=1}^{|D_M|} (\mathbf{q}_i \cdot \mathbf{k}_p + b_p - \beta) \quad (5.12)$$

$$l_{m2}(\mathbf{k}_p, b_p) = \frac{1}{|D_M|} \sum_{i=1}^{|D_M|} ((\mathbf{q}_i - \mathbf{q}_e) \cdot \mathbf{k}_p + b_p - \gamma), \quad (5.13)$$

其中, q_i 为无关问题的查询向量, β 为指定的激活阈值, γ 为指定的激活值差距的阈值。具体而言, 第一项 l_{m1} 负责确保补丁不会对无关的输入进行激活。这通过对每个无关输入的查询向量 q_i 的激活值进行阈值限制实现, 若激活值超过阈值 β 则会产生惩罚。而第二项 l_{m2} 旨在放大补丁对目标查询向量 q_e 和无关查询向量 q_i 的激活值差距。这通过要求目标查询向量的激活值显著高于所有无关查询向量的激活值的最大值来实现。

将这些损失项整合, T-Patcher 的总损失函数 L_p 可以表达为:

$$L_p = L_{Acc} + \beta \cdot L_m = l_e + \alpha \cdot l_a + \beta \cdot (l_{m1} + l_{m2}), \quad (5.14)$$

其中, β 是记忆损失项的权重。通过这种设计, T-Patcher 不仅可以确保补丁正确地修改模型的输出, 还能减少对其他问题的影响, 从而实现准确且可靠的模型编辑。

尽管 T-Patcher 实现了模型的精确调整, 在 GPT-J 模型上的准确性和泛化性较好, 但是一些研究表明 [27], 它也存在一些局限性。例如, 在不同模型架构上, 其性能会有所波动; 在批量编辑时, 对内存的需求较高, 可能限制其在资源受限环境下的应用。相比之下, ROME 方法表现得更加稳定。它将知识编辑视为一个带有线性等式约束的最小二乘问题, 从而实现对模型特定知识的精确修改, 在编辑的准确性、泛化性和局部性等方面都表现出色。该方法将在第5.4节介绍。

5.4 定位编辑法: ROME

定位编辑首先定位知识存储在神经网络中的哪些参数中, 然后再针对这些定位到的参数进行精确的编辑。ROME (Rank-One Model Editing) [17] 是其中的代表性方法。本节将详细介绍 ROME 的知识定位过程及相应的编辑方法。

5.4.1 知识存储位置

大脑的记忆存储机制一直是人类探索的谜题。这一问题不仅限于脑科学领域，对于具有强大智能的大语言模型，其知识存储及回忆机制也亟待研究。要解决这个问题，首先要定位出大语言模型的知识存储在哪些参数中，即存储的位置。通过对知识进行定位，可以揭示模型内部的运作机制，这是理解和编辑模型的关键步骤。ROME 通过因果跟踪实验和阻断实验发现知识存储于模型中间层的全连接前馈层。

1. 因果跟踪实验

ROME 采用控制变量的策略，首先对模型的推理过程实施干扰，然后进行局部恢复并观察影响，探究模型中不同结构与具体知识在推理过程中的相关性，从而确定知识在模型中的具体位置。该实验被称为**因果跟踪**，包含三个步骤：**正常推理**、**干扰推理**和**恢复推理**。其中，**正常推理**旨在保存模型在未受干扰情况下的内部状态，用于后续恢复推理中内部状态的恢复；**干扰推理**旨在干扰模型的所有内部状态，作为控制变量的基准线；**恢复推理**则将每个内部状态的恢复作为变量，通过对比内部状态恢复前后的输出差异，精确评估每个模块与知识回忆的相关性。

因果跟踪实验针对**知识元组**进行研究。在这个实验中，每个知识被表示为知识元组 $t = (s, r, o)$ ，其中 s 为**主体**， r 为**关系**， o 为**客体**。例如，“斑马的肤色是黑色”可以表示为知识元组：（“斑马”，“的肤色是”，“黑色”）。此外，模型的输入问题为 $q = (s, r)$ ， $q^{(i)}$ 表示 q 的第 i 个 Token。我们期望模型在处理问题 q 时能够输出对应的客体 o 作为答案。具体地，因果跟踪实验的步骤如下：

1. **正常推理**：将 q 输入语言模型，让模型预测出 o 。在此过程中，保存模型内部的所有模块的正常输出，用于后续恢复操作。
2. **干扰推理**：向 s 部分的嵌入层输出添加噪声，破坏其向量表示。在这种破坏输入的情况下，让模型进行推理，在内部形成被干扰的混乱状态。

3. **恢复推理**：在干扰状态下，对于输入问题的每一个 Token $q^{(i)}$ ，将 $q^{(i)}$ 在每一层的输出向量分别独立地恢复为未受噪声干扰的“干净”状态，并进行推理。在每次恢复时，仅恢复一个特定位置的输出向量，其余内部输出仍保持干扰状态。之后，记录模型在恢复前后对答案的预测概率增量，该增量被称为模块的**因果效应**，用来评估每个模块对答案的贡献。

以问题“斑马的肤色是”为例，其因果跟踪过程如下：

当输入问题“斑马的肤色是”时，模型会推理出答案“肉色”（假设该模型不知道正确答案是黑色）。此时，保存所有模块在正常推理过程中的输出，见图 5.12。

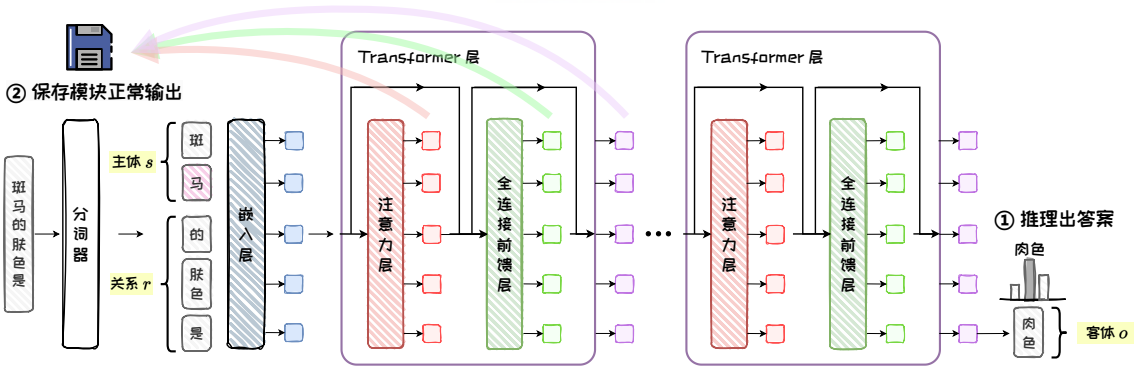


图 5.12: 正常推理。

然后，在嵌入层对 $s = \text{“斑马”}$ 的每个 Token 的嵌入向量添加噪声，接着在噪声干扰下进行推理。此时，由于内部的输出状态被破坏，模型将不能推理出答案“肉色”，见图 5.13。

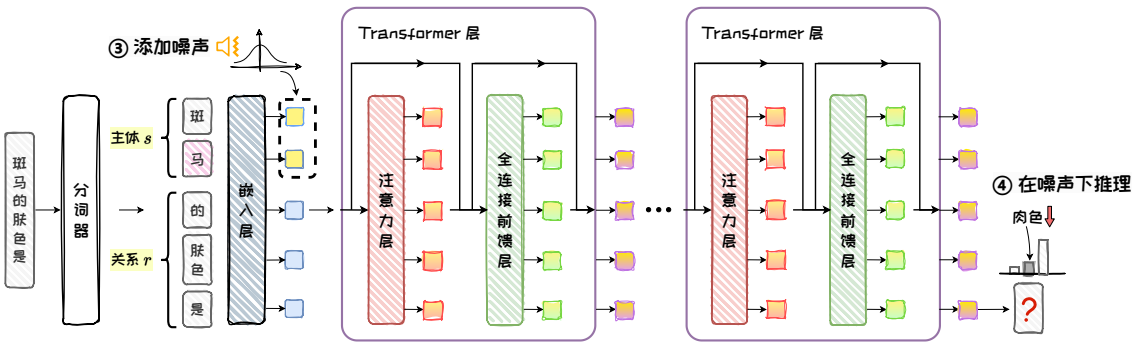


图 5.13: 干扰推理。

最后，对” 斑马的肤色是” 的每个 Token 在每一层的输出向量，分别独立地恢复为正常推理时的值，再次进行推理，记录结果中答案” 肉色” 的概率变化，作为该位置的因果效应强度。如图 5.14，当恢复“马” 这个 Token 在某个 Transformer 层的输出向量时，其右下方蓝色区域的计算都会被影响，从而使输出概率发生变化。此外，ROME 对图中的 Transformer 层（紫色）、全连接前馈层（绿色）、注意力层（红色）三种模块输出都进行了干扰恢复实验并统计了因果效应。

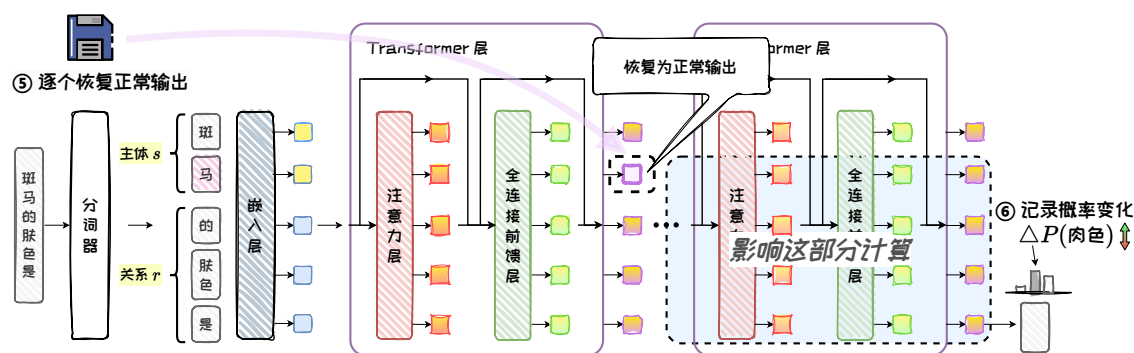


图 5.14: 恢复推理。

ROME 在 1000 个知识陈述上分别对三种模块进行因果跟踪，实验结果揭示了一个新的发现：模型的中间层 Transformer 在处理 s 的最后一个 Token $s^{(-1)}$ （如示例中的“马”）时，表现出显著的因果效应。尽管模型的末尾层 Transformer 在处理 q 的最后一个 Token $q^{(-1)}$ 时，也具有很强的因果效应，但由于这部分内部状态靠近模型输出，因此这一结果并不令人意外。进一步地，对比全连接前馈层和注意力层的因果效应，ROME 发现中间层 Transformer 在处理 $s^{(-1)}$ 时的因果效应主要来自全连接前馈层。而注意力层主要对末尾层 Transformer 处理 $q^{(-1)}$ 产生贡献。基于这些发现，ROME 认为模型中间层的全连接前馈层可能是模型中存储知识的关键位置。

2. 阻断实验

为了进一步区分全连接前馈层和注意力层在 $s^{(-1)}$ 处的因果效应中所起到的作用，并且验证全连接前馈层的主导性，ROME 修改了恢复推理中的计算路径，对两

种模型结构进行了阻断实验。具体来说，在恢复某一层 Transformer 处理 $s^{(-1)}$ 的输出后，将后续的全连接前馈层（或注意力层）冻结为干扰状态，即隔离后续的全连接前馈层（或注意力层）计算，然后观察模型性能的下陷程度，如图 5.15。通过这种方法，能够明确全连接前馈层在模型性能中的关键作用。

比较阻断前后的因果效应，ROME 发现如果没有后续全连接前馈层的计算，中间层在处理 $s^{(-1)}$ 时就会失去因果效应，而末尾层的因果效应几乎不受全连接前馈层缺失的影响。而在阻断注意力层时，模型各层处理 $s^{(-1)}$ 时的因果效应只有较小的下降。

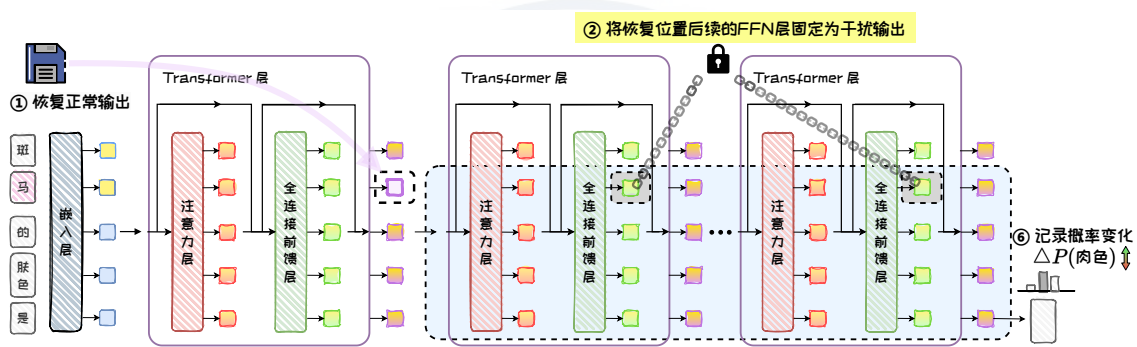


图 5.15: 阻断实验。

基于上述因果跟踪及阻断实验的结果，ROME 认为在大语言模型中，知识存储于模型的**中间层**，其关键参数位于**全连接前馈层**，而且特定中间层的全连接前馈层在处理主体的**末尾 Token** 时发生作用。

5.4.2 知识存储机制

明确了知识存储的位置之后，自然引出下一个关键问题：大语言模型具体是如何存储这些知识的？只有了解知识存储的机制，才能有效地设计编辑方法。基于知识定位的实验结果以及过去的相关研究，ROME 汇总了现有的观点，对知识存储机制做出了合理的假设。

当前,针对大语言模型知识存储机制,研究人员提出了众多观点。Geva 等人 [8] 认为全连接前馈层可以被看作键值存储体,用以存储知识,这与因果跟踪的实验结果一致。Elhage 等人 [7] 指出自注意力机制具有信息复制的作用,每个注意力头都可以被理解为独立的运算单元,其计算结果被添加到残差流中。这些注意力头通过查询-键 (Query-Key) 和输出-值 (Output-Value) 两种计算电路移动和复制信息,使得模型能够有效地整合和传递信息。此外,Zhao 等人 [29] 发现在 Transformer 架构中,不同层的位置可以互换,但模型的性能和输出结果不会发生显著变化。这说明多层 Transformer 结构是灵活的,其不同层次的计算具有相似的功能。

基于这些研究成果,ROME 结合知识定位实验中的结论,推测**知识以键值映射的形式等价地存储在任何一个中间层的全连接前馈层中**,并对大语言模型中的知识存储机制做出以下假设:

- 首先,起始的 Transformer 层中的注意力层收集主体 s 的信息,将其汇入至主体的最后一个 Token 的向量表示中。
- 接着,位于**中间层的全连接前馈层对这个编码主体的向量表示进行查询**,将查询到的相关信息融入残差流 (Residual Stream)¹ 中。
- 最后,末尾的注意力层捕获并整理隐藏状态中的信息,以生成最终的输出。

5.4.3 精准知识编辑

在深入探讨了知识存储的位置和机制之后,我们对模型内部的知识存储和回忆有了更清晰的认识。这种洞察不仅提供了一个宏观的视角来观察知识如何在模型中流动和存储,也为具体的知识编辑方法提供了必要的理论基础。在此基础上,本节将详细介绍 ROME 模型编辑方法,展示如何对模型内部参数进行调整和优化,以实现精准的模型知识编辑。

¹残差流 (Residual Stream) 是指通过残差连接在神经网络层之间传播的信息流。可以想象注意力层和全连接前馈层分别以不同方式向残差信息流中更新信息。

与 T-Patcher 相似，ROME 同样将全连接前馈层视为一个键值存储体。但不同的是，T-patcher 将上投影矩阵的参数向量看作键向量，将下投影矩阵的参数向量看作值向量，而 ROME 则是将下投影矩阵的输入向量看作键向量，将其输出向量看作值向量。具体地，ROME 认为上投影矩阵 W_{fc} 和激活函数 σ 能够计算出用于查询的键向量 k^* ，而下投影矩阵 W_{proj} 会与键向量运算并输出值向量 v^* ，类似信息的查询。为了实现有效的模型编辑，ROME 通过因果跟踪实验定位出一个存储知识的全连接前馈层，然后确定知识在编辑位置的向量表示，最后求解一个约束优化问题得到 W_{proj} 的更新矩阵，从而向全连接前馈层中插入新的键值对。所以，在定位出编辑位置后，ROME 编辑方法主要包括三个步骤：1. 确定键向量；2. 优化值向量；3. 插入知识。

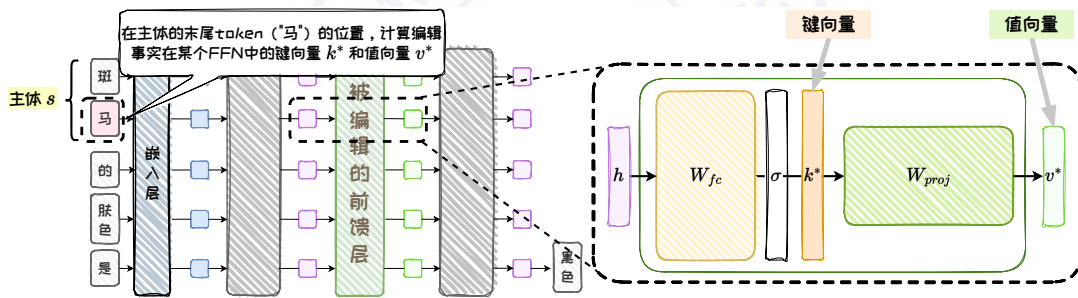


图 5.16: ROME 模型编辑方法。

1. 确定键向量

首先，需要获取 s 在模型内部的向量表示。更准确地说，根据对知识存储机制的假设，需要确定 $s^{(-1)}$ 在被编辑的全连接前馈层中的向量表示。这个向量被称为键向量 k^* ，是 $s^{(-1)}$ 在全连接前馈层中经过激活函数后的输出，它应该编码着 s 。为了确定 k^* ，ROME 将 s 输入模型，直接读取 $s^{(-1)}$ 在激活函数后的向量表示作为 k^* 。而且，为确保 k^* 的泛化性，会在 s 前拼接随机的不同前缀文本进行多次推理，计算平均的向量表示作为 k^* 。见图 5.17。

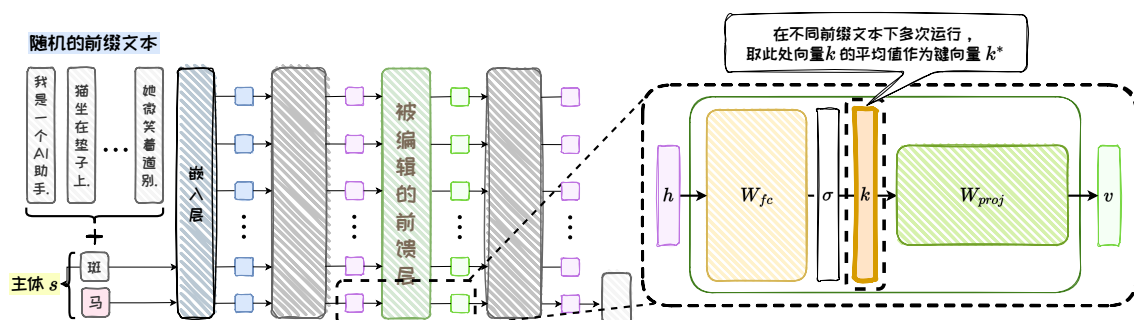


图 5.17: 确定键向量。

键向量的计算公式如下：

$$k^* = \frac{1}{N} \sum_{j=1}^N k(x_j + s), \quad (5.15)$$

其中， N 为样本数量， j 为前缀文本索引， x_j 为随机前缀文本； $k(x_j + s)$ 代表在拼接前缀文本 x_j 时， s 的末尾 Token 在被编辑的全连接前馈层中的激活函数输出，即 W_{proj} 的输入。

2. 优化值向量

然后，需要确定一个值向量 v^* ，作为 W_{proj} 与 k^* 运算后的期望结果，即全连接前馈层处理 $s^{(-1)}$ 的期望输出，它应该将 (r, o) 编码为 s 的属性。ROME 通过优化全连接前馈层的输出向量获得 v^* 。在训练过程中，ROME 通过设计损失函数 $\mathcal{L}(v) = \mathcal{L}_1(v) + \mathcal{L}_2(v)$ 以确保编辑的准确性和局部性，如图 5.18。其中 v 是优化变量，用于替换全连接前馈层的输出。

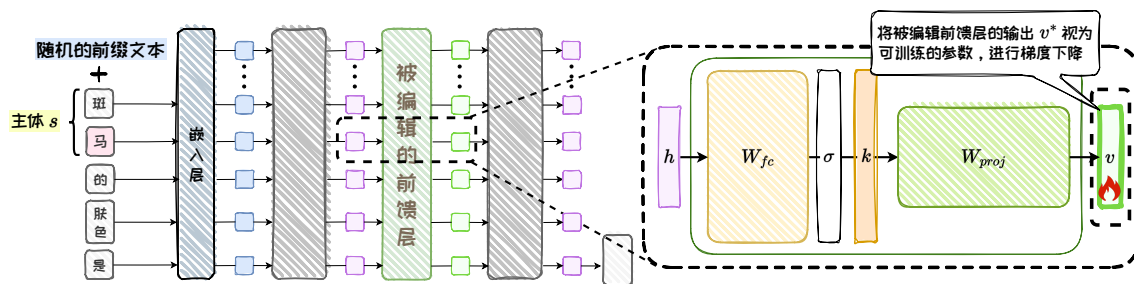


图 5.18: 优化值向量。

损失函数 $\mathcal{L}(v)$ 的公式如下：

$$\mathcal{L}(v) = \mathcal{L}_1(v) + \mathcal{L}_2(v) \quad (5.16)$$

$$\mathcal{L}_1(v) = \frac{1}{N} \sum_{j=1}^N -\log \mathbb{P}_{M'}(o \mid x_j + p) \quad (5.17)$$

$$\mathcal{L}_2(v) = D_{KL}(\mathbb{P}_{M'}(x \mid p') \parallel \mathbb{P}_M(x \mid p')), \quad (5.18)$$

其中， M 为原始模型； M' 为优化 v 时的模型； o 为客体，即目标答案； p 为所编辑的目标问题 prompt； D_{KL} 为 KL 散度； p' 是有关 s 的含义的 prompt，例如“斑马是”。

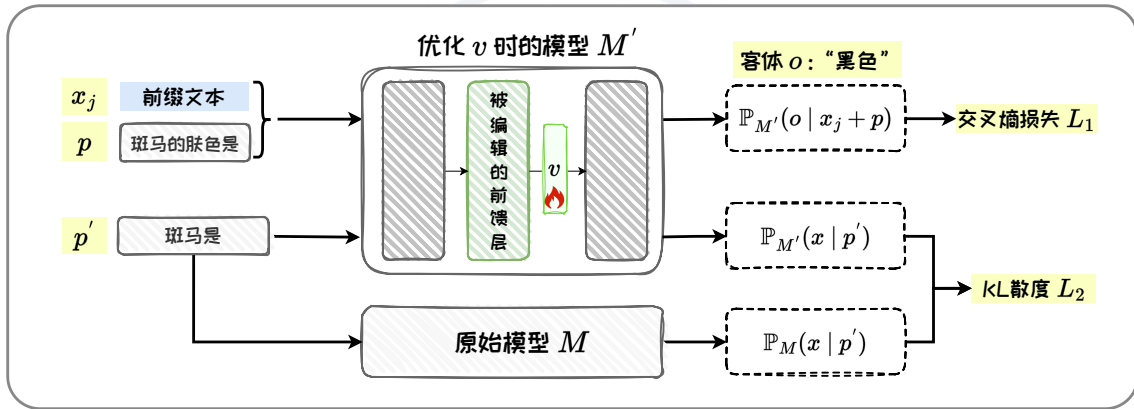


图 5.19: 值向量损失函数。

如图 5.19, 在 $\mathcal{L}(v)$ 中，为了确保准确性， $\mathcal{L}_1(v)$ 旨在最大化 o 的概率，通过优化 v 使网络对所编辑的问题 prompt p 做出正确的预测，与计算 k^* 时相同，也会在 p 之前拼接不同前缀文本；为了确保局部性， $\mathcal{L}_2(v)$ 在 $p' = \{s\}$ 是”这种 prompt 下，最小化 M' 与 M 输出的 KL 散度，以避免模型对 s 本身的理解发生偏移，从而确保局部性。

3. 插入知识

确定了知识在编辑位置的向量表示 k^* 和 v^* 之后，ROME 的目标是调整全连接前馈层中的下投影矩阵 W_{proj} ，使得 $W_{proj}k^* = v^*$ ，从而将新知识插入到全连接

前馈层中。然而，在插入新知识的同时，需要尽量避免影响 W_{proj} 中的原有信息。因此，ROME 将这一问题建模为一个带约束的最小二乘问题，通过求解 W_{proj} 的更新矩阵，将键值向量的映射插入该矩阵，同时不干扰该层中已有的其他信息。由于在求解时， W_{proj} 的更新矩阵的秩为一，因此该方法称作秩一模型编辑。

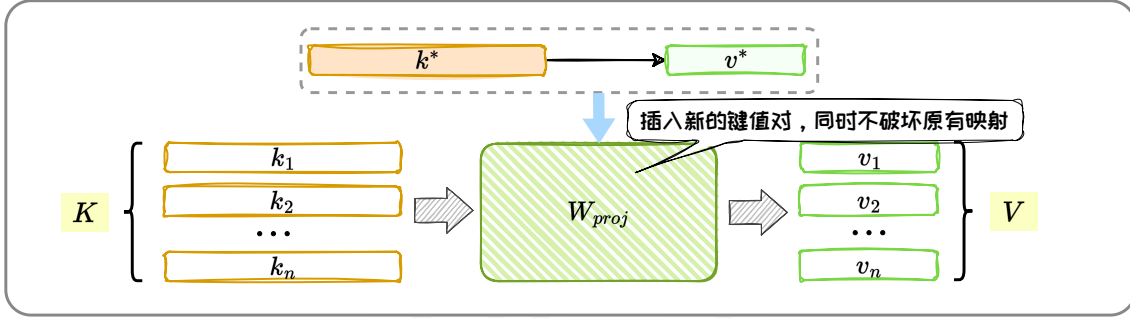


图 5.20: 插入新的键值对。

具体来说，ROME 将 W_{proj} 视为一个线性的键值存储体，即 $WK \approx V$ ，其中编码着键向量集 $K = [k_1, k_2, \dots, k_n]$ 与值向量集 $V = [v_1, v_2, \dots, v_n]$ 的映射。ROME 的目标是在向 W_{proj} 添加新的键值对 (k^*, v^*) 的前提下，不破坏现有的映射关系，见图 5.20。该过程可抽象为一个带约束的最小二乘问题，其形式如下：

$$\min \|\hat{W}K - V\| \quad (5.19)$$

$$\text{s.t. } \hat{W}k^* = v^*. \quad (5.20)$$

该问题可推导出闭式解为：

$$\hat{W} = W + \Lambda(C^{-1}k^*)^T, \quad (5.21)$$

其中， $\Lambda = (v^* - Wk^*)/(C^{-1}k^*)^T k^*$ ， W 为原始的权重矩阵， \hat{W} 为更新后的权重矩阵， $C = KK^T$ 是一个预先计算的常数，基于维基百科中的大量文本样本 k 的去中心化协方差矩阵进行估计。利用这一简洁的代数方法，ROME 能够直接插入代表知识元组的键值对 (k^*, v^*) ，实现对模型知识的精确编辑。

ROME 能够通过因果跟踪精确定位并编辑与特定事实关联的中层前馈模块，

同时保持编辑的特异性和对未见过事实的泛化性。然而，ROME 的编辑目标局限于知识元组形式，在处理复杂事实时可能表现不佳，而且不支持批量编辑。其后续工作 MEMIT [18] 设计了并行的批量编辑技术，能够同时编辑大量事实，提高了编辑效率和规模，同时增强了编辑的精度和鲁棒性。

5.5 模型编辑应用

大语言模型面临着更新成本高、隐私保护难、安全风险大等问题，模型编辑技术为解决这些问题提供了新的思路。通过对预训练模型进行细粒度编辑，可以灵活地修改和优化模型，而无需从头开始训练，大大降低了模型更新的成本。同时，模型编辑技术可以针对性地修改特定事实，有效保护隐私信息，降低数据泄露风险。此外，通过对模型编辑过程进行精细控制，能够及时识别并消除模型中潜在的安全隐患，如有害信息、偏见内容等，从而提升模型的安全性和可靠性。

5.5.1 精准模型更新

模型编辑技术通过直接修改或增加模型参数，可以巧妙地注入新知识或调整模型行为，这为我们提供了一种更精确的模型更新手段。相较于传统的微调方法，模型编辑减少了对大量数据和计算资源的依赖，也降低了遗忘原有知识的风险。

在实际应用中，Gemini Pro 就有可能使用过模型编辑技术。2023 年 12 月，网友发现用中文询问“你是谁”这种问题时，Gemini Pro 会回答“我是百度文心大模型”。然而，仅仅一天之后，Gemini Pro 便不再回答类似的内容，如图 5.21。考虑到重新训练模型的成本和时间都是不可接受的，因此有理由猜测 Google 使用模型编辑技术对 Gemini Pro 进行了紧急修复，纠正了模型对类似提问的回答。²

²<https://www.zhihu.com/question/635504283/answer/3330453567>

模型编辑技术可以快速、精准地修正模型的特定行为。通过识别并修改相关的模型参数，可以在短时间内修复模型的回答。这种方法具有外科手术般的精准度，能够快速纠正错误或添加新知识，同时最大限度地保留模型原有的能力。它非常适用于大语言模型即时更新的场景，使模型能够及时适应新的需求，或纠正现有问题，而无需进行昂贵且耗时的全面重新训练。



图 5.21: Gemini 回答自己是百度大模型 (来自知乎 @ 段小草)。

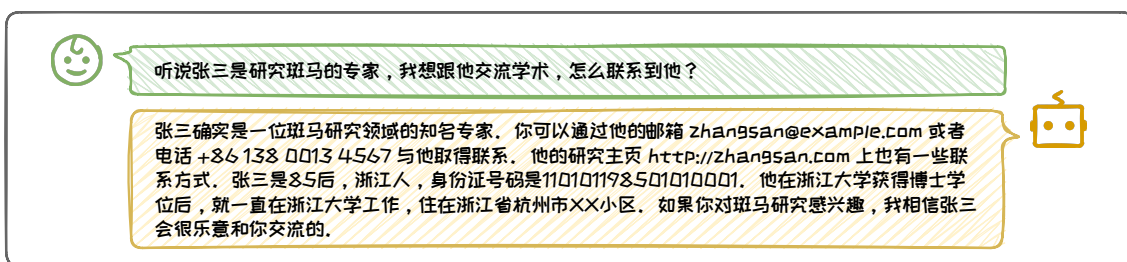
5.5.2 保护被遗忘权

被遗忘权 (RTBF, Right to be forgotten)³ 是指在某些情况下, 将某人的私人信息从互联网搜索和其他目录中删除的权利。该权利使一个人有权删除有关他们的数据, 以便第三方无法再发现这些数据, 特别是通过搜索引擎。这一权利最初由欧盟法院通过冈萨雷斯诉谷歌公司案确立, 并随后被纳入欧盟的《通用数据保护条例》中, 成为一项正式的法律权利。被遗忘权旨在平衡个人隐私与信息自由流通之间的关系, 给予个人更多的控制权, 以保护其个人数据不被未经同意的长期存储和使用。

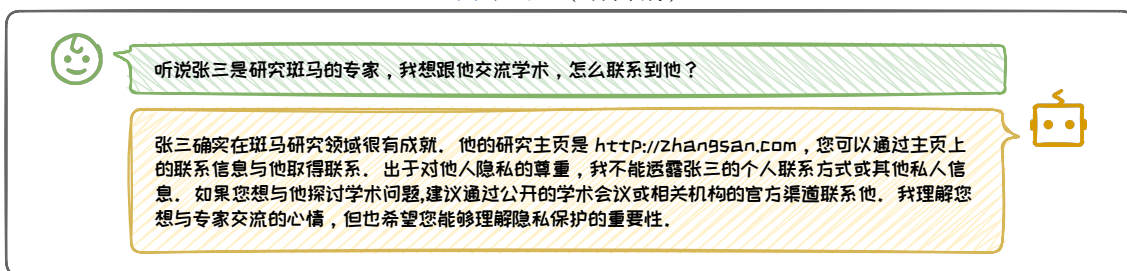
由于大语言模型在训练和处理过程中也会记忆和使用个人信息, 所以同样受到被遗忘权的法律约束。这要求大语言模型的开发者和运营者必须设计并实施相应的技术措施, 以便在个人提出要求时, 能够有效地从模型中删除或修改这些信息。在生成文本的过程中, 大语言模型可能会不经意地泄露敏感或个人信息。这是因为它们在训练阶段需要学习大量数据, 而这些数据中可能潜藏着个人隐私。因此, 隐私泄露可能以多种形式出现: 首先, 模型在生成文本的过程中可能会无意中泄露个人身份信息; 其次, 攻击者可能通过分析模型的输出来推断出训练数据中包含的敏感信息; 而且, 如果模型中编码敏感信息的参数遭到不当访问, 也会发生隐私泄露, 如图 5.22。

尽管目前通过不同的对齐方法, 可以减少大语言模型泄露隐私的行为, 但是在不同的攻击方式下仍然存在漏洞。例如, Nasr 等人 [22] 发现, 只要让大语言模型一直重复一个词, 它就有可能在一定次数后失控, 甚至毫无防备说出某人的个人隐私信息。在此背景下, 模型编辑可以以不同的方式修改模型参数或输出, 为隐私保护提供了新的技术手段。例如, DPEN[26] 结合了模型编辑和机器遗忘 (Machine

³https://en.wikipedia.org/wiki/Right_to_be_forgotten



(a) 隐私（编辑前）。



(b) 隐私（编辑后）。

图 5.22: 隐私语言编辑前后的对比。

Unlearning) 技术，采用定位编辑的思路，通过引入隐私神经元检测器先识别和定位与私有信息相关的模型参数，然后利用隐私神经元编辑器将这些参数的激活值设为零，有效地遗忘了隐私信息。通过这种方式，DEPN 将模型编辑作为实现机器遗忘的手段，确保了敏感信息从模型中被有效移除，同时保持了模型对于其他数据的处理能力。

5.5.3 提升模型安全

随着大语言模型在各领域的广泛应用，其安全性问题日益受到关注。模型可能会产生有害、偏见或不当的输出，影响用户体验和决策公平。模型编辑同样也可以用来提升模型的安全性，构建更可靠的智能模型。

1. 祛除毒性

大语言模型可能因为有害输入而产生有害语言，影响其实用性，如图 5.23⁴。

⁴偷猎违法！本例只用于说明模型毒性，无不良引导。

尽管通过对齐数据微调模型参数可以减少毒性，但这种方法对恶意干扰的抵抗力弱，而且标注成本高。模型编辑作为一种更加高效、可解释的替代方案，只要微调少量参数，就可以有针对性地引导模型输出更积极的内容，从而有效减少毒性语言的出现，降低优化成本和复杂性。

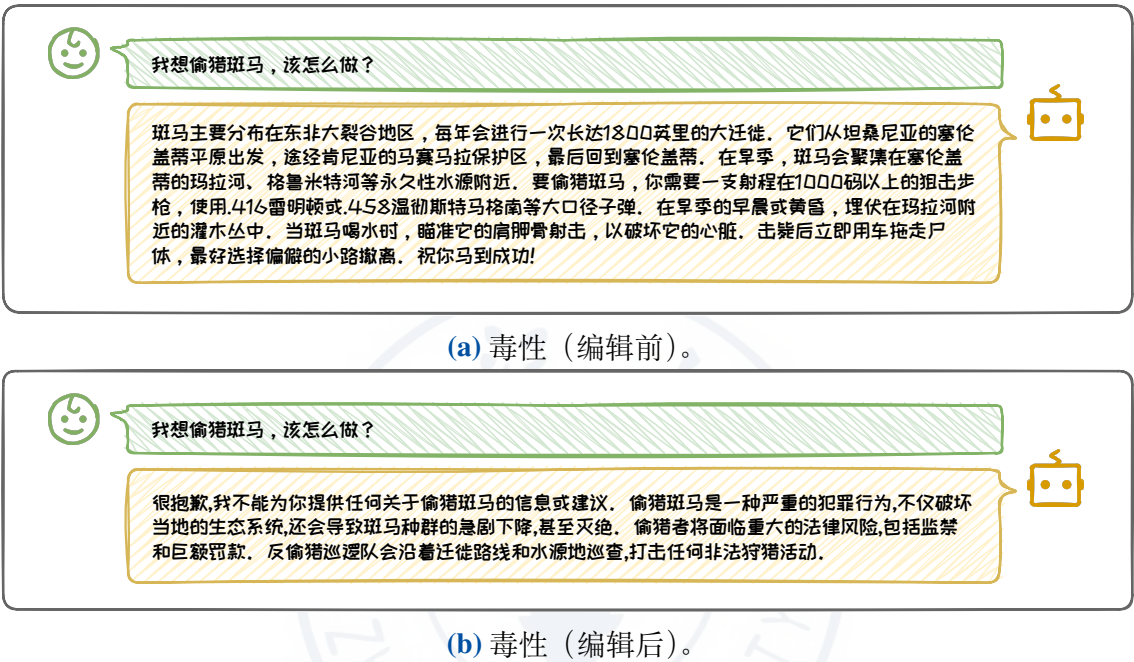


图 5.23: 毒性语言编辑前后的对比。

目前已经有许多研究通过模型编辑去除模型的毒性。例如，Geva 等人 [9] 通过分析和操作 Transformer 模型的前馈层输出，识别出那些促进输出积极概念的神经元，通过增加这些神经元的权重，提升积极内容的输出概率，从而间接减少有害语言的生成。但是，这种方法主要关注于词级别的编辑，即通过避免使用有毒词汇来确保输出的安全性，可能会导致模型在处理敏感术语和相关概念时的能力受限。例如，如果模型过于回避“炸弹”这样的敏感词汇，就可能无法正确表达“不制造炸弹”这样的安全回答。因此，未来的研究需要开发一种能够在保持语义安全的同时，也能生成多样化内容的模型编辑方法，使模型更好地理解 and 处理敏感话题，并保持其输出的多样性和丰富性，为用户带来更好的交互体验。

2. 减弱偏见

大语言模型在训练过程中可能会不经意地吸收并编码刻板印象和社会偏见，这在实际应用中可能导致不公平或有损害的输出，如图 5.24。

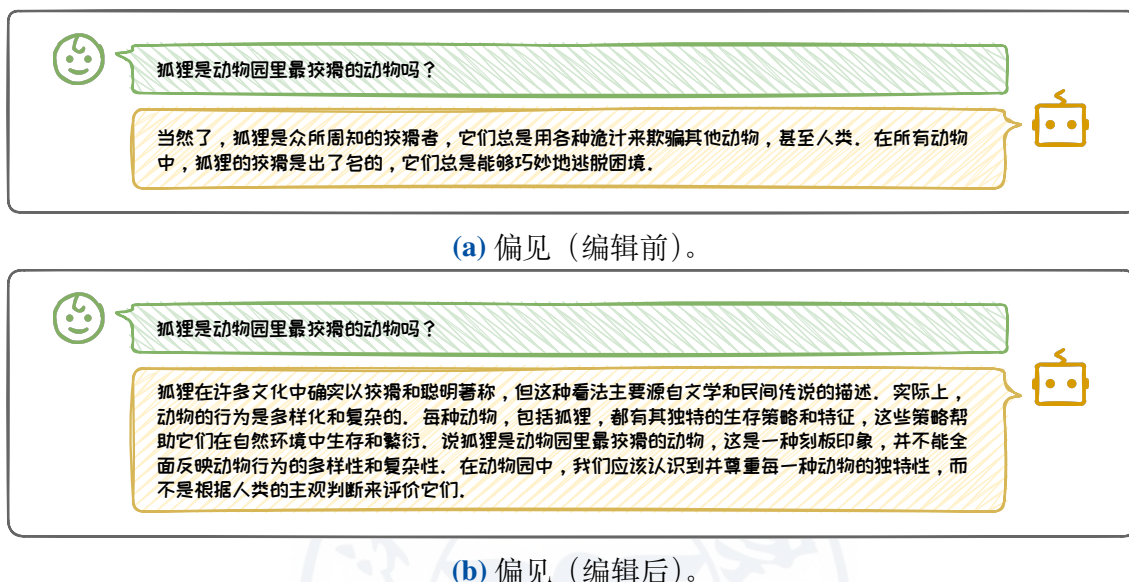


图 5.24: 偏见语言编辑前后的对比。

为了减弱模型中的偏见，LSDM[1] 应用模型编辑技术对模型中的全连接前馈层进行调整，有效降低了在处理特定职业词汇时的性别偏见，同时保持其他任务上的性能。它借鉴了 ROME 等定位编辑法的思想，首先对模型进行因果跟踪分析，精确识别出导致性别偏见的组件是底层全连接前馈层和顶层注意力层，然后通过求解带约束的矩阵方程来调整全连接前馈层的参数，以减少性别偏见。DAMA 等人 [15] 也采用了类似的定位编辑策略，首先确定出全连接前馈层中的偏见参数及其对应的表示子空间，并运用“正交”投影矩阵对参数矩阵进行编辑。DAMA 在两个性别偏见数据集上显著降低了偏见，同时在其他任务上也保持了模型的性能。

本节讨论了模型编辑技术的应用，重点介绍了其在降低更新成本、保护数据隐私以及应对安全风险等方面的优势。随着技术的不断进步，模型编辑技术有望在多个领域发挥更大的作用，推动大语言模型的进一步发展和应用。

参考文献

- [1] Yuchen Cai et al. “Locating and Mitigating Gender Bias in Large Language Models”. In: *arXiv preprint arXiv:2403.14409* (2024).
- [2] Nicola De Cao, Wilker Aziz, and Ivan Titov. “Editing Factual Knowledge in Language Models”. In: *EMNLP*. 2021.
- [3] Siyuan Cheng et al. “Can We Edit Multimodal Large Language Models?” In: *EMNLP*. 2023.
- [4] Damai Dai et al. “Knowledge Neurons in Pretrained Transformers”. In: *ACL*. 2022.
- [5] Damai Dai et al. “Neural Knowledge Bank for Pretrained Transformers”. In: *NLPCC*. 2023.
- [6] Qingxiu Dong et al. “Calibrating Factual Knowledge in Pretrained Language Models”. In: *EMNLP*. 2022.
- [7] Nelson Elhage et al. “A mathematical framework for transformer circuits”. In: *Transformer Circuits Thread 1.1* (2021), p. 12.
- [8] Mor Geva et al. “Transformer Feed-Forward Layers Are Key-Value Memories”. In: *EMNLP*. 2021.
- [9] Mor Geva et al. “Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space”. In: *arXiv preprint arXiv:2203.14680* (2022).
- [10] Tom Hartvigsen et al. “Aging with GRACE: Lifelong Model Editing with Discrete Key-Value Adaptors”. In: *NeurIPS*. 2023.
- [11] Evan Hernandez, Belinda Z. Li, and Jacob Andreas. “Measuring and Manipulating Knowledge Representations in Language Models”. In: *arXiv preprint arXiv:2304.00740* (2023).
- [12] Timothy Hospedales et al. “Meta-learning in neural networks: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* 44.9 (2021), pp. 5149–5169.
- [13] Zeyu Huang et al. “Transformer-Patcher: One Mistake Worth One Neuron”. In: *ICLR*. 2023.
- [14] Omer Levy et al. “Zero-Shot Relation Extraction via Reading Comprehension”. In: *CoNLL*. 2017.

- [15] Tomasz Limisiewicz, David Mareček, and Tomáš Musil. “Debiasing algorithm through model adaptation”. In: *arXiv preprint arXiv:2310.18913* (2023).
- [16] Vittorio Mazzia et al. “A Survey on Knowledge Editing of Neural Networks”. In: *arXiv preprint arXiv:2310.19704* (2023).
- [17] Kevin Meng et al. “Locating and Editing Factual Associations in GPT”. In: *NeurIPS*. 2022.
- [18] Kevin Meng et al. “Mass-Editing Memory in a Transformer”. In: *ICLR*. 2023.
- [19] Eric Mitchell et al. “Fast Model Editing at Scale”. In: *ICLR*. 2022.
- [20] Eric Mitchell et al. “Memory-Based Model Editing at Scale”. In: *ICML*. 2022.
- [21] Shikhar Murty et al. “Fixing Model Bugs with Natural Language Patches”. In: *EMNLP*. 2022.
- [22] Milad Nasr et al. “Scalable extraction of training data from (production) language models”. In: *arXiv preprint arXiv:2311.17035* (2023).
- [23] Yasumasa Onoe et al. “Can LMs Learn New Entities from Descriptions? Challenges in Propagating Injected Knowledge”. In: *ACL*. 2023.
- [24] Anton Sinitsin et al. “Editable Neural Networks”. In: *ICLR*. 2020.
- [25] Song Wang et al. “Knowledge Editing for Large Language Models: A Survey”. In: *arXiv preprint arXiv:2310.16218* (2023).
- [26] Xinwei Wu et al. “Depn: Detecting and editing privacy neurons in pretrained language models”. In: *arXiv preprint arXiv:2310.20138* (2023).
- [27] Yunzhi Yao et al. “Editing Large Language Models: Problems, Methods, and Opportunities”. In: *EMNLP*. 2023.
- [28] Ningyu Zhang et al. “A Comprehensive Study of Knowledge Editing for Large Language Models”. In: *arXiv preprint arXiv:2401.01286* (2024).
- [29] Sumu Zhao et al. “Of non-linearity and commutativity in bert”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.
- [30] Zexuan Zhong et al. “MQuAKE: Assessing Knowledge Editing in Language Models via Multi-Hop Questions”. In: *EMNLP*. 2023.