ICS 35.040
CCS L 71

团　　体　　标　　准

T/AI 115.2—2024

信息技术　神经网络表示与模型压缩
第 2 部分：大规模预训练模型

Information technology—Neural network representation and model compression—Part 2: Large scale pre-training model

2024－12－30 发布　　2024－12－30 实施

中关村视听产业技术创新联盟　发布

目　　次

前　言

本文件按照GB/T 1.1—2020《标准化工作导则　第1部分：标准化文件的结构和起草规则》的规定起草。

本文件是T/AI 115《信息技术 神经网络表示与模型压缩》的第2部分。T/AI 115已经发布了以下部分：

——第1部分：卷积神经网络。

——第2部分：大规模预训练模型。

本文件由新一代人工智能产业技术创新战略联盟AI标准工作组提出。

本文件由中关村视听产业技术创新联盟归口。

本文件起草单位：北京大学、鹏城实验室、华为技术有限公司、北京百度网讯科技有限公司、厦门大学、杭州海康威视数字技术股份有限公司、中国电子技术标准化研究院、中国科学院自动化研究所、中科南京人工智能创新研究院、铁塔智联技术有限公司。

本文件起草人：田永鸿、杨帆、陈光耀、郑侠武、彭军、纪荣嵘、韩凯、胡晓光、燕肇一、张一帆、沈岗、曹刘娟、周奕毅、张玉鑫、马跃萧、吴宇航、谢展豪、倪铭坚、张翀、彭佩玺、马艳军、于佃海、陈秋良、陈泽裕、陈醒濠、唐业辉、王云鹤、蓝朝祥、杨绮明、郑传杨、张凯、彭博、李哲暘、谭文明、任烨、叶挺群、任文奇、冯仁光、周智强、王培松、程健、麻文军、杨雨泽、鲍薇、郑若琳、沈芷月、张伟民、赵海英、黄铁军、高文。

本文件的发布机构提请注意，声明符合本文件时，可能涉及到7.3.1.2与《一种基于可微量化训练的视觉Transformer压缩方法及系统》（专利号：202102951896）、7.2.1与《基于层间特征相似性网络稀疏化方法、装置、介质及设备》（专利号：202210842886.9）、7.4.4与《一种两阶段微调大语言模型代理的方法》（专利号：202410358176）、7.4.5与《一种提高大型语言模型适配多模态任务效率的方法》（专利号：202311290661.8）、7.3.1与《数据处理方法、装置、介质及电子设备》（专利号：202410154029.9）、7.3.2.3.4与《神经网络模型裁剪方法、装置、电子设备及存储介质》（专利号：202210615980.0）、7.2.5与《任务处理方法、装置、电子设备及存储介质》（专利号：202210709171.6）、7.3.1与《基于模型量化的任务处理方法、装置、设备及存储介质》（专利号：202211186183.1）、7.2与《一种特征提取的方法以及装置》（专利号：113065576A）、7.2与《一种注意力模型、特征提取方法及相关装置》（专利号：113627163A）、7.2与《一种数据处理方法及相关设备》（专利号：114897039A）相关的专利的使用。

本文件的发布机构对于该专利的真实性、有效性和范围无任何立场。

该专利持有人已向本文件的发布机构承诺，他愿意同任何申请人在合理且无歧视的条款和条件下，就专利授权许可进行谈判。该专利持有人的声明已在本文件的发布机构备案，相关信息可以通过以下联系方式获得：

专利持有人：北京大学、华为技术有限公司、厦门大学、杭州海康威视数字技术股份有限公司、中科南京人工智能创新研究院、中国科学院自动化研究所、百度在线网络技术（北京）有限公司

地址：北京市海淀区颐和园路5号、深圳市龙岗区坂田华为总部办公楼、福建省厦门市思明区思明南路422号、浙江省杭州市滨江区阡陌路555 号、江苏省南京市创研路266号麒麟人工智能产业园3号楼3楼、北京市海淀区中关村东路95号、北京市海淀区上地十街10号百度大厦

联 系 人：黄铁军

通讯地址：北京大学理科2号楼2641室

邮政编码：100871

电子邮件：tjhuang@pku.edu.cn

电话：+8610-62756172
传真：+8610-62751638
网址：http://www.aitisa.org.cn

请注意除上述专利外，本文件的某些内容仍可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

引　言

人工智能领域正在发生迅速的范式转变，深度影响了计算机视觉、自然语言处理、机器人、自动驾驶、智慧医疗等领域的发展。大规模预训练模型是人工智能技术体系的重要组成部分，是国民经济各行业应用人工智能的前提。该标准的目标在于提供大规模预训练模型可能涉及的表示和压缩技术方法参考，提升用户对模型的复用效果。使用时，对于大规模预训练模型的表示方法、传输方法应进行必要的支持，对于压缩技术可根据实际应用场景及技术构成做可选支持，具体的支持方法由后续标准进行补充。对于该标准规定的表示方法不要求平台原生支持，可以通过转换、工具包等形式进行支持，同时相关的定义可转化为与特定计算设备、框架匹配的形式和实现。T/AI 115旨在确立适用于不同种类神经网络的表示方法与模型压缩的规范，拟由三个部分组成：

——第1部分：卷积神经网络。目的在千确立适用千卷积神经网络的表示与模型压缩标准。

——第 2 部分：大规模预训练模型。目的在于确立适应多种推理平台和计算要求的大规模预训练模型的基本表示方法与加速压缩过程。

——第3部分：图神经网络。目的在于确立适应多种计算要求的高效图神经网络模型的基本表示方法与压缩加速过程。

信息技术　神经网络表示与模型压缩 第2部分：大规模预训练模型

1　范围

本文件规定了适应多种计算机要求的大规模预训练的表示、压缩表示和封装表示，以及其对应的压缩流程、适配流程、封装流程和模型传输与分发。

本文件适用于大规模预训练模型的研制、开发过程，以及在端云领域的高效应用。

注：对于本文件规定的表示与模型压缩方法不要求机器学习框架原生支持，可以通过转换、工具包等形式支持。

2　规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅日期对应的版本适用于本文件；不注日期的引用文件，其最新版本(包括所有的修改单)适用于本文件。

GB/T 42382.1-2023 信息技术 神经网络表示与模型压缩 第1部分：卷积神经网络

GB/T 5271.34-2006 信息技术 词汇 第34部分：人工智能 神经网络

3　术语和定义

GB/T 5271.34-2006 界定的以及下列术语和定义适用于本文件。

3.1

预训练模型 pre-trained model

一种深度学习模型，通过自监督或者无监督技术，在大量的训练数据上训练得到初始模型，可被迁移到目标相近的任务中进行使用的一种深度学习模型。

3.2

大规模预训练模型 large-scale pre-trained model

大模型 large-scale model

一种参数规模较大的预训练模型，通过自监督或者无监督技术从海量的通用数据中训练得到基础模型，并结合下游具体任务对其进行微调，最终被训练成具有逻辑推理和分析能力的人工智能模型。

3.3

转换器 transformer

一种基于多头注意力机制，包含残差连接、层归一化和全连接的、能并行处理序列数据的、序列到序列架构（Encoder-Decoder架构）的网络。

3.4

自注意力 self-attention

通过计算输入序列内部每个元素对其他所有元素的注意力权重，建立序列内部的不同位置间关系的机制。

3.5

多头注意力 multi-head attention
一种大规模预训练模型中利用多个参数量进行注意力计算机制。

3.6

层归一化 layer normalization
对一个中间层的所有神经元进行归一化。在Transformer结构中对一个位置的特征进行归一化。

3.7

前馈网络 feedforward network
神经网络中的推理模型。

3.8

量化 quantization
深度学习中用于模型压缩和加速的常见技术，通过降低模型参数的精度来减少存储和计算需求。

3.9

剪枝 pruning
深度学习中用于模型压缩和加速的常见技术，通过移除不重要的权重或神经元来简化模型。

4 缩略语

下列缩略语适用于本文件。
AI：人工智能（Artificial Intelligence）
BERT：基于双向Transformer的编码器表征（Bidirectional Encoder Representations from Transformers）
FFN：前馈网络（Feedforward Network）
GAN：生成对抗网络（Generative Adversarial Network）
GPT：生成式预训练Transformer（Generative Pre-trained Transformer）
HTTPS：超文本传输安全协议（Hyper-Text Transfer Protocol Secure）
LLM：大语言模型（Large Language Model）
MHA：多头注意力（Multi-Head Attention）
MLP：多层感知机（Multi-Layer Perception）
MSA：多头自注意力（Multi-Head Self-Attention）
RAG：检索增强生成（Retrieval-Augmented Generation）
REST：表述性状态传递（Representational State Transfer）
SGD：随机梯度下降（Stochastic Gradient Descent）
VAE：变分自编码器（Variational Auto-Encoder）
ViT：视觉Transformer（Vision Transformer）

5 概述

大规模预训练模型在表示、压缩与适配、传输与分发等环节相互关联，形成一个完整的生态系统。各个环节之间联系紧密，贯穿了从模型训练到应用的整个生存周期。各环节总体架构见图 1。

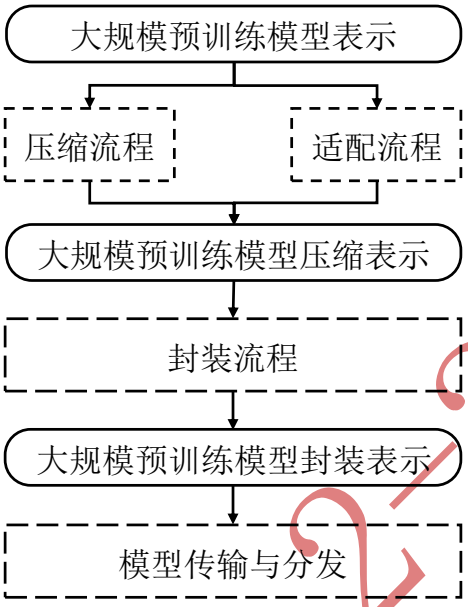

图1 大规模预训练模型表示与压缩总体架构

其中，大规模预训练模型表示的具体要求和应用流程见第 6 章，定义构建大规模预训练模型基本结构信息的基本语法和语义。大规模预训练模型压缩表示的具体要求和应用流程见第 7 章，用于需要资源受限设备以及专业场景任务的情况。大规模预训练模型封装表示的具体要求和应用流程见第 8 章，用于模型从不同端进行传输、更新。

6 大规模预训练模型表示

6.1 语法描述

6.1.1 通则

本部分定义大规模预训练模型表示的语法，从粗粒度到细粒度，即从模型结构定义、计算图定义、到节点定义，逐层嵌套，构建了整个大规模预训练模型的基本语法描述。该表示语法应在由特定计算系统（深度学习平台及相关软硬件）中完成，遵循以下原则。

a) 计算系统实现时，需要对语法要素按实际需要做出调整，包含但不限于：
   1) 关键字（参数）命名；
   2) 运算符命名；
   3) 数据类型。
b) 需要考虑必要层级的定义，包含：
   1) 模型结构定义；
   2) 计算图定义；
   3) 基本数据类型定义。

6.1.2 模型结构定义

模型结构表示神经网络模型的基本信息以及网络架构，描述模型结构的技术参数见表1。

表 1 模型结构定义

参数	类型	定义
version	int64	模型表示标准版本
contributors	ContributorsList	模型贡献者信息列表
framework_name	string	模型初始训练框架名称
framework_version	string	模型初始训练框架版本
model_name	string	模型名称
model_version	string	模型版本
doc_url	string	模型描述文档链接
graph	graph	模型具体计算图

注：来源GB/T 42382.1-2023

6.1.3 计算图定义

计算图定义见表2。

表 2 计算图定义

字段	类型	定义
operator_node	OperatorNode (repeated)	计算图操作节点
variable_node	VariableNode (repeated)	计算图变量节点
id	int64	计算图的唯一序号
parent_graph_id	int64	子计算图对应的父计算图的序号
forward_graph_id	int64	子计算图对应的前向计算图的序号
sub_graphs	Graph (repeated)	计算图包含的子图列表

计算图可能包含若干子计算图：

a） 大规模预训练模型表示中的条件语句、循环语句的内部操作节点和变量节点构成子计算图；

b） 大规模预训练模型中的若干操作结点可合并为一个新的操作结点，并交由第三方高效计算引擎（如 Nvidia TensorRT、intel Ngraph 等）执行，新的操作结点和对应的变量节点可用子计算图的形式表达。

计算图的parent_graph_id指向其父计算图，forward_graph_id指向其前向计算图。主计算图无对应的父计算图，其parent_graph_id表示为-1；若计算图无对应的前向计算图，其forward_graph_id表示为-1。

计算图包含操作节点和变量节点，操作节点对应网络中的运算操作，变量节点对应网络中的变量，变量包括但不限于网络参数和临时变量。操作节点接收一系列的变量节点作为输入，经过运算操作后输出一系列的变量节点。

6.1.4 操作节点定义

操作节点定义见表3。其中，运算操作域定义了操作节点的名称；input和output为操作节点的输入变量节点和输出变量节点；attribute为操作节点的属性。

表 3 操作节点定义

参数	类型	定义
name	string	节点名称
operator	string	节点操作名称
input	map<string, list(VariableNode)>	节点输入
output	map<string, list(VariableNode)>	节点输出
attribute	map<string, Attribute>	节点属性
doc_string	string	节点描述
definition	string	节点定义

6.2 语义描述

6.2.1 运算操作表示

6.2.1.1 通则

6.2.1.1.1 运算操作使用原则

大规模预训练模型所含的运算操作，由特定计算系统实现，应遵循以下使用原则。

1) 按实际需要，做出调整，包含但不限于：

    1) 关键字（参数）命名；
    2) 运算符命名；
    3) 数据类型支持范围。

2) 考虑运算涉及的要素，包含：

    1) 可支持的参数；
    2) 可支持的类型。

6.2.1.1.2 运算操作分类

大规模预训练模型的算子分为以下三类：

1) 大模型组网的通用算子，包括但不限于 embedding、layer_norm、linear、attention 算子，其定义见表 4 至表 7；

2) 神经网络模型的通用算子，包括但不限于基础数学类算子 reshape、concat，神经网络类算子 relu、softmax，详细内容可参考其他相关算子标准定义，不在本标准内重复定义；具体可参考标准包括 GB/T 42382.1-2023；

3) 大模型训练的通信算子，包括但不限于 reduce、allreduce、reduce_scatter、allgather、broadcast、send、recv 算子，定义如表 8 至表 14 所示。

6.2.1.2 基础运算操作定义

本条定义大规模预训练模型中常见的运算操作定义，具体定义见如下表格。

embedding 运算操作定义见表 4。

表 4 embedding 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
embedding	一个简单的查找表，用于存储固定字典和大小的嵌入	Input	X	输入张量	intTensor longTensor
			weight	词嵌入权重张量	Tensor
		Output	Y	输出映射张量	Tensor
		Attributes	input_size	可选，输入最大值	int
			embed_dim	可选，输出张量的维度	int

layer_norm 运算操作定义见表 5。

表 5 layer_norm 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
layer_norm	层归一化层操作	Input	X	输入张量	Tensor
			weight	缩放张量	Tensor
			bias	偏置张量	Tensor
		Output	Y	输出张量	Tensor
			mean	可选，输入的均值，其shape的前begin_norm_axis 维与 input_x 相同	Tensor
			variance	可选，输入的方差，shape同 mean 一致	Tensor
		Attributes	epsilon	为了数值稳定加在分母上的值	float
			begin_norm_axis	开始层归一化的维度	int
			begin_param_axis	可选，指定输入参数(weight, bias)需进行层归一化的开始维度	int

linear 运算操作定义见表 6。

表 6 linear 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
linear	对特征张量进行全连接层运算	Input	X	输入特征张量	Tensor
			weight	全连接层的权重张量	Tensor
			bias	可选，全连接层的偏置张量	Tensor
		Output	Y	输出特征张量	Tensor

attention 运算操作定义见表 7。

表 7 attention 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
attention	对特征张量进行自注意力运算	Input	Q	Q特征张量	Tensor
			K	K特征张量	Tensor
			V	V特征张量	Tensor
			attn_mask	可选，掩码张量	Tensor
		Output	Out	输出特征张量	Tensor
			softmax	可选，输出softmax值	Tensor
		Attribute	dropout	可选，dropout操作的概率	float
			return_softmax	可选，是否返回softmax	bool
			causal	可选，是否使用causal模式	bool

reduce 运算操作定义见表 8。

表 8 reduce 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce	规约进程组内的一个 tensor，随后将结果发送到一个进程	Input	X	输入张量	Tensor
		Output	Y	规约后的张量	Tensor
		Attributes	op	可选，规约操作类型	枚举类型，默认sum，可选sum，max，min，prod
			root_id	接收张量的进程	int
			group_id	进程组id	int

allreduce 运算操作定义见表 9。

表 9 allreduce 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
allreduce	规约进程组内的一个 tensor，随后将结果发送到每个进程	Input	X	输入张量	Tensor
		Output	Y	规约后的张量	Tensor
		Attributes	op	可选，规约操作类型	枚举类型，默认sum，可选sum，max，min，prod
			group_id	进程组id	int

reduce_scatter 运算操作定义见表 10。

表 10 reduce_scatter 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
reduce_scatter	进程组内每个节点将数据切分为若干维度，每一个节点对一个维度的所有数据进行reduce	Input	X	输入张量	Tensor
		Output	Y	规约后的张量	Tensor
		Attributes	op	可选，规约操作类型	枚举类型，默认sum，可选sum，max，min，prod
			nranks	可选，进程组中进程的数量	int
			group_id	进程组id	int

allgather 运算操作定义见表 11。

表 11 allgather 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
allgather	聚合进程组内的指定tensor，随后将聚合后的tensor 列表发送到每个进程	Input	X	输入张量	Tensor
		Output	Y	聚合后的张量	Tensor
		Attributes	nranks	可选，进程组中进程的数量	int
			group_id	进程组id	int

broadcast 运算操作定义见表 12。

表 12 broadcast 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
broadcast	将一个 tensor 广播到进程组的每个进程	Input	X	输入张量	Tensor
		Output	Y	广播后的张量	Tensor
		Attributes	root_id	执行广播的根节点	int
			group_id	进程组id	int

send 运算操作定义见表 13。

表 13 send 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
send	将一个 tensor 发送到进程组的指定进程	Input	X	输入张量	Tensor
		Output	/	/	/

		Attributes	peer	接收张量的节点	int
			group_id	进程组id	int

recv 运算操作定义见表 14。

表 14 recv 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
recv	接收某个进程发送的tensor	Input	/	/	/
		Output	Y	接收的张量	Tensor
		Attributes	peer	发送张量的节点	int
			group_id	进程组id	int

6.2.2 模块操作表示

6.2.2.1 概述

针对大规模预训练模型对不同场景的适配、加速和压缩，对大规模预训练模型的结构化模块做出的针对性的修改。本章节提供模块操作的基本定义以及其参考结构或计算流程。

6.2.2.2 模块操作定义

多头注意力机制 MHA 运算操作定义见表 15。

表 15 MHA 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
MHA	对特征张量进行多头注意力运算	Input	X	输入特征张量	Tensor
			attn_mask	可选，掩码张量	Tensor
		Output	Y	输出特征张量	Tensor
			softmax	可选，输出softmax值	Tensor
		Attribute	N_heads	可选，多头注意力的头个数	Int
			dropout	可选，dropout操作的概率	float
			return_softmax	可选，是否返回softmax	bool
			causal	可选，是否使用causal模式	bool

FFN的运算操作定义见表16。

表 16 FFN 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
FFN	对特征张量进行全连接网络运算	Input	X	输入特征张量	Tensor
		Output	Y	输出特征张量	Tensor
		Attribute	act	必选，非线性激活函数	string
			hidden_dim	必选，隐藏层特征维数	Int

嵌套式 Transformer 模块的操作定义见表 17，结构或计算流程见 7.2.1。

表 17 嵌套式 Transformer 模块操作定义

运算操作	描述	字段	关键字	定义	类型
嵌套式Transformer模块	对输入视觉句子和视觉单词特征进行特征变换	Input	Z	视觉句子特征	tensor
			Y	视觉单词特征	tensor
		Output	Z_new	更新后的视觉句子特征	tensor
			Y_new	更新后的视觉单词特征	tensor
		Attributes	n	视觉句子的数量	int
			m	视觉单词的数量	int
			dim	视觉句子特征维数	int

基于参数化跳连的 Transformer 模块的操作定义见表 18，结构或计算流程见 7.2.2。

表 18 基于参数化跳连的 Transformer 模块操作定义

运算操作	描述	字段	关键字	定义	类型
基于参数化跳连的Transformer模块	在标准Transformer模块上并联参数化跳连支路	Input	Z	输入特征	tensor
		Output	Z_new	输出特征	tensor
		Attributes	n	与MSA模块并联参数化跳连的数量	int
			m	与MLP模块并联参数化跳连的数量	int

Transformer统一归一化层模块操作定义模块的操作定义见表19，结构或计算流程见7.2.4。

表 19 Transformer 统一归一化层模块操作定义

运算操作	描述	字段	关键字	定义	类型
Transformer统一归一化模块	对输入进行归一化	Input	X	输入特征	tensor
		Output	Y	输出	tensor
		Attributes	σ^2	推理方差	tensor
			γ	仿射变换参数	tensor
			β	仿射变换参数	tensor
			M	窗口大小	Int
			$\hat{\sigma}_t^2$	训练历史方差	tensor
			$g_{\hat{\sigma}_t^2}$	训练历史方差梯度	tensor

Transformer 多模态融合模块操作定义模块的操作定义见表 20，结构或计算流程见 7.2.5。

表 20 Transformer 多模态融合模块操作定义

运算操作	描述	字段	关键字	定义	类型
Transformer多模态融合模块	对模态1和模态2根据特征重要性进行特征替换	Input	x1	模态1特征	tensor
			x2	模态2特征	tensor
			score1	模态1特征重要性得分	tensor
			score2	模态2特征重要性得分	tensor
		Output	x1_new	融合更新后的模态1特征	tensor
			x2_new	融合更新后的模态2特征	tensor
		Attributes	proj_1_to_2	模态1到模态2 token映射	function
			proj_2_to_1	模态2到模态1 token映射	function
			threshold	特征交换阈值	float

Adapter适配器操作定义见表21，结构或计算流程见7.4.2。

表 21 Adapter 适配器操作定义

运算操作	描述	字段	关键字	定义	数据类型
RepAdapter	对输入特征的进行线性缩放和偏移	Input	X	输入张量	Tensor
			weight	缩放张量	Tensor
			bias	偏置张量	Tensor
		Output	Y	输出张量	Tensor
Linear_Adapter	对输入的权重进行线性缩放和偏移操作	Input	X	输入张量	Tensor
			weight	缩放张量	Tensor
			bias	偏置张量	Tensor
		Output	Y	输出张量	Tensor

多模态大模型高效迁移操作定义见表 22，结构或计算流程见 7.4.3.1。

表 22 多模态大模型高效迁移操作定义

操作	描述	字段	关键字	定义	数据类型
高效迁移	对多模态预训练大模型进行高效迁移	Input	W	预训练权重张量	List of Tensor
			A	参数高效模块	List of Tensor
			R	动态路由模块	List of Tensor
			D	微调数据集	List of vectors
			T	迁移目标	Value
		Output	W_o	迁移之后的权重张量	List of vectors
			A_o	迁移之后的适配器张量	List of vectors
			R_o	迁移之后的动态路由张量	List of vectors

视觉大模型高效迁移操作定义见表 23，结构或计算流程见 7.4.4.1。

表 23 视觉大模型高效迁移操作定义

操作	描述	字段	关键字	定义	数据类型
高效迁移	对视觉大模型进行高效迁移	Input	W	预训练权重张量	List of vectors
			A	参数高效模块	List of vectors
			D	微调数据集	List of vectors
			T	迁移目标	Value
		Output	W_o	迁移之后的权重张量	List of vectors
			A_o	迁移之后的适配器张量	List of vectors

迁移压缩操作定义见表 24，结构或计算流程见 7.4.5.1。

表 24 迁移压缩操作定义

操作	描述	字段	关键字	定义	数据类型
迁移压缩	对预训练模型进行迁移压缩	Input	W	预训练权重张量	List of vectors
			A	参数高效模块	List of vectors
			D	微调数据集	List of vectors
			R	压缩目标	Value
		Output	W_o	压缩之后的权重张量	List of vectors
			A_o	压缩之后的适配器张量	List of vectors

6.2.3 多模态操作表示

6.2.3.1 概述

6.2.3.1.1 核心组件

多模态大模型是融合多个模态数据进行理解与生成的大模型。其模型结构的核心组件包括输入层、模态特定编码器、模态融合模块、模态对齐模块、解码层和输出层。

6.2.3.1.2 输入层

不同模态输入对应独特的输入处理模块，数据输入处理模块包括：

a) 文本数据：分词和嵌入处理；
b) 图像数据：进行像素值归一化和图像增强；
c) 音频数据：转换为频谱图或其他时频表示。

6.2.3.1.3 模态特定编码器

模态特定编码器包括但不限于：

a) 文本编码器：通常由词嵌入层、位置嵌入层、多层 Transformer 编码器组成，能够将文本数据转化为高维特征表示，包括 BERT、GPT、RoBERTa 等；
b) 图像编码器：通常包含卷积层（或自注意力层）和池化层，能够提取图像的多层次特征，包括 ResNet、Vision Transformer（ViT）等；

c) 音频编码器：通常通过卷积层或 Transformer 结构处理音频信号，提取有意义的音频特征，包括 Wav2Vec、Mel-Spectrogram、MFCC 等；
d) 视频编码器：通常通过处理视频数据中的时间和空间信息，从而提取出丰富的视频特征，包括 C3D、I3D 等。

6.2.3.1.4 模态融合模块

多模态数据融合策略包括：
a) 早期融合：在输入层或编码层进行融合，常用方法包括拼接、加权平均等；
b) 中期融合：在模型的中间层进行融合，通常利用多层交叉注意力机制（Cross-Attention）在不同模态间进行信息交换；
c) 晚期融合：在模型的高层或输出层进行融合，通常利用单独的决策模块对每个模态的数据进行处理，然后将结果合并。

6.2.3.1.5 模态对齐模块

模态对齐是指将不同模态的数据映射到相同的特征空间，对齐方法包括但不限于：
a) 共现分析：利用共现矩阵或共现图对不同模态的数据进行对齐；
b) 对比学习：通过构建正负样本对，最大化同一模态和不同模态特征之间的相似性；
c) 对齐变换：利用线性或非线性变换，将不同模态的特征对齐到相同的表示空间。

6.2.3.1.6 解码器和输出层

解码器和输出层包括但不限于：
a) 模态特定解码器：用于图像、文本的生成任务，其实用特定模态的解码器对文本或图像进行解码。例如，GPT 等对应文本生成器，GAN 或 VAE 等对图像生成器；
b) 联合解码器：用于生成多模态输出的任务（如视频描述生成），可以同时处理不同模态的输出；
c) 分类和回归模块：对于分类和回归任务，全连接层被添加在融合后的特征表示上以完成回归和预测任务。

6.2.3.2 图像模态算子

vision_embedding运算操作用于将图像数据嵌入到特征空间，具体定义见表25。

表 25 vision_embedding 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
vision_embedding	将图像数据嵌入到特征空间	Input	image	输入图像张量	Tensor
			resize_shape	可选，调整后的图像大小	Tuple
		Output	embedding	输出嵌入特征张量	Tensor
		Attributes	encoder_type	必选，编码器类型	string
			embed_dim	必选，嵌入维度	int
			dropout	可选，dropout概率	float

roi_pooling运算操作用于区域兴趣池化操作，提取图像中特定区域的特征。具体定义见表26。

表 26 roi_pooling 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
roi_pooling	区域兴趣池化操作，提取图像中特定区域的特征	Input	image	输入图像张量	Tensor
			regions	区域坐标张量	Tensor
			pool_size	池化大小	Tuple
		Output	pooled_features	输出池化特征张量	Tensor
		Attributes	spatial_scale	必选，空间缩放因子	float
			sampling_ratio	可选，采样比例	int
			aligned	可选，是否进行边界对齐	bool

6.2.3.3 文本模态算子

ext_embedding运算操作用于将文本数据嵌入到特征空间。具体定义见表27。

表 27 text_embedding 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
text_embedding	将文本数据嵌入到特征空间	Input	text	输入文本张量	Tensor
			token_type	可选，令牌类型张量	Tensor
		Output	embedding	输出嵌入特征张量	Tensor
		Attributes	encoder_type	必选，编码器类型	string
			vocab_size	必选，词汇表大小	int
			embed_dim	必选，嵌入维度	int
			dropout	可选，dropout概率	float

positional_encoding运算操作用于为文本数据添加位置编码。具体运算操作定义见表28。

表 28 positional_encoding 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
Positional_encoding	为文本数据添加位置编码，帮助模型理解序列信息	Input	sequence	输入文本序列张量	Tensor
			max_length	最大序列长度	Int
		Output	encoded	输出位置编码特征张量	Tensor
		Attributes	embed_dim	必选，嵌入纬度	Tensor
			periodic_function	可选，周期函数类型	String

6.2.3.4 音频模态算子

mel_spectrogram运算操作将音频信号转换为梅尔频谱图。具体运算操作定义见表29。

表 29 mel_spectrogram 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
mel_spectrogram	将音频信号转换为梅尔频谱图	Input	audio_signal	输入音频信号张量	Tensor
			sample_rate	采样率	Int
			window_type	窗函数类型	String
			n_mels	频带数量	Int
		Output	spectrogram	输出梅尔频谱图张量	Tensor
		Attributes	hop_length	必选，跳跃长度	Int
			n_fft	必选，FFT窗口大小	Int

Mfcc运算操作提取音频信号的梅尔频率倒谱系数。具体运算操作定义见表30。

表 30 Mfcc 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
Mfcc	提取音频信号的梅尔频率倒谱系数	Input	audio_signal	输入音频信号张量	Tensor
			sample_rate	采样率	Int
		Output	mfcc_features	输出MFCC特征张量	Tensor
		Attributes	n_mfcc	必选，MFCC特征数量	Int
			n_mels	必选，梅尔滤波器组的数量	Int
			hop_length	可选，跳跃长度	Int
			n_fft	可选，FFT窗口大小	Int

6.2.3.5 多模态融合算子

cross_modal_attention 运算操作定义见表 31。

表 31 cross_modal_attention 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
cross_modal_attention	对不同模态数据进行交叉注意力计算	Input	modality_1	第一模态输入特征张量	Tensor
			modality_2	第二模态输入特征张量	Tensor
			attn_mask	可选，掩码张量	Tensor
		Output	attn_output	输出交叉注意力特征张量	Tensor
			attn_weights	可选，注意力权重张量	Tensor
		Attributes	num_heads	必选，注意力头数	int
			embed_dim	必选，嵌入维度	int
			key_dim	必选，键的维度	int
			value_dim	必选，值的维度	int
			dropout	可选，dropout概率	float

concat_fusion 运算操作定义见表 32。

表 32 concat_fusion 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
concat_fusion	将多模态特征拼接在一起，形成融合特征表示	Input	modality_1	第一模态输入特征张量	Tensor
			modality_2	第二模态输入特征张量	Tensor
		Output	fused_output	输出融合特征张量	Tensor
		Attributes	fusion_dim	必选，融合后特征维度	int
			dropout	可选，dropout概率	float

expert_selection 运算操作定义见表 33。

表 33 expert_selection 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
expert_selection	根据输入数据选择适合的专家模型	Input	input_data	输入数据张量	Tensor
		Output	selected_expert	输出选择的专家模型索引	int
		Attributes	num_experts	必选，专家模型的数量	int
			selection_method	必选，专家选择方法	string

gated_expert_routing 运算操作定义见表 34。

表 34 gated_expert_routing 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
gated_expert_routing	使用门控机制路由输入数据到适合的专家模型	Input	input_data	输入数据张量	Tensor
		Attributes	num_experts	必选，专家模型的数量	int
			gate_dim	必选，门控维度	int
			activation	可选，门控机制的激活函数	string
			dropout	可选，dropout概率	float

mixture_of_experts 运算操作定义见表 35。

表 35 mixture_of_experts 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
mixture_of_experts	综合多个专家模型的输出结果	Input	expert_outputs	各专家模型的输出结果张量	Tensor
		Output	final_output	最终综合的输出结果	Tensor
		Attributes	num_experts	必选，专家模型的数量	int
			combine_method	必选，结果综合方法	String
			dropout	可选，dropout概率	float

6.2.3.6 模型对齐和转换算子

modal_alignment运算操作定义见表36。

表 36 modal_alignment 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
modal_alignment	对齐不同模态的数据，确保它们在同一特征空间内	Input	modality_1	第一模态输入张量	Tensor
			modality_2	第二模态输入张量	Tensor
			align_method	对齐方法	string
		Output	aligned	输出对齐后的张量	Tensor
		Attributes	align_dim	对齐维度	int
			dropout	可选，dropout概率	float

6.2.3.7 模态特定的优化算子

modality_specific_optimizer运算操作定义见表37。

表 37 modality_specific_optimizer 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
modality_specific_optimizer	优化特定模态的参数	Input	X	输入特征张量	List of Tensor
			learning_rate	学习率	float32
			optimizer_method	优化方法（如Adam、SGD等）	string
		Output	Y	优化后的参数	List of Tensor
		Attributes	modality	必选，模态类型（如文本、图像、音频等）	string

modality balancing optimizer 运算操作定义见表 38。

表 38 modality balancing optimizer 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
modality balancing optimizer	平衡不同模态的损失和梯度，确保各模态的贡献均衡	Input	X	输入特征张量	List of Tensor
			balance_factor	平衡因子，用于调整各模态的权重	float32
		Output	Y	平衡后的参数	List of Tensor
		Attributes	modality	必选，模态类型（如文本、图像、音频等）	string

cross-modality regularization 运算操作定义见表 39。

表 39 cross-modality regularization 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
cross-modality regularization	正则化多模态之间的关系，防止过拟合和模态偏差	Input	X	输入特征张量	Tensor
			regularization_rate	正则化率	float32
			regularization_method	正则化方法（如L2、L1）	string
		Output	Y	正则化后的参数	Tensor
		Attributes	modality	必选，模态类型（如文本、图像、音频等）	string

7 大规模预训练模型压缩表示

7.1 概述

基于Transformer的预训练大模型应包含词向量、多个Transformer模块（每个Transformer模块包含多头注意力机制、全连接网络、正则化、跳跃连接）以及任务相关模块。这种模型推理阶段的存储瓶颈在词向量以及Transformer模块中的权重，计算瓶颈在多头注意力机制和全连接网络中的矩阵乘法。而存储和计算直接影响了推理的时延和功耗。为了减少推理过程中的存储和计算，许多方法对词向量，权重矩阵进行了压缩，或者对矩阵乘法进行了加速和优化。这些用于词向量和权重矩阵压缩，加速和优化矩阵乘法的方法包括但不限于：

a） 大模型结构优化：通过调整模型的结构来减少模型的复杂性和计算需求；

b） 大模型加速压缩：通过算法和硬件优化来加速模型的推理和训练，同时尽量减少对存储的需求，主要包括稀疏化、量化等；

c） 大模型迁移压缩：通过迁移学习和模型压缩结合的方式，减少模型在新任务上的复杂性和大小，具体包括多模态迁移和多任务迁移。

7.2 大模型结构优化

7.2.1 嵌套式结构

嵌套式结构利用一个外部Transformer和一个内部Transformer分别提取全局特征和局部特征，即使用一个外Transformer模块来对图像块之间的关系进行建模，用一个内Transformer模块来对子图像块之间的关系进行建模，从而构建丰富的视觉表征的技术方案。嵌套式结构既保留了图像块层面的信息提取，又做到了像素层面的信息提取，能够显著提升模型对局部结构的建模能力，进而提升模型的识别效果。

嵌套式结构将输入图像切块，每个图像块看作一个视觉句子Z，构成视觉句子序列，并将图像块进一步切分为子图像块，每个子图像块看作一个视觉单词Y。嵌套式视觉Transformer模块会对视觉单词Y和视觉句子Z进行联合处理，对于每一组m个视觉单词，使用一个内部Transformer模块来进行特征提取和关系建模，模块输出见式（1）和式（2）：

$$Y'^{i}_{l} = Y^{i}_{l-1} + MHA\left(LN\left(Y^{i}_{l-1}\right)\right) \quad (1)$$

$$Y_l^i = Y'^i_l + FFN\left(LN\left(Y'^i_l\right)\right) \quad (2)$$

式中：

Y——视觉单词；

Z——视觉句子。

对一组视觉单词进行处理之后，嵌套式结构将这组视觉单词向量拼接起来，线性变换层linear将拼接向量映射为一个大向量，加到视觉句子上对其特征进行补充，得到新的视觉句子，见式（3）：

$$Z_{l-1}^i = Z_{l-1}^i + linear\left(Vec\left(Y_l^i\right)\right) \quad (3)$$

对于n个视觉句子，嵌套式结构使用一个外部Transformer模块来进行特征提取和关系建模，见式（4）和式（5）：

$$Z'_l = Z_{l-1} + MHA\left(LN\left(Z_{l-1}\right)\right) \quad (4)$$

$$Z_l = Z'_l + FFN\left(LN\left(Z'_l\right)\right) \quad (5)$$

上述内部和外部2个Transformer模块共同组成嵌套式Transformer算子。通过串联若干个嵌套式Transformer算子，就构建了嵌套式视觉Transformer网络结构。

嵌套式Transformer模块的操作定义见表 17，伪代码描述见表40。

表 40 嵌套式 Transformer 模块伪代码描述

嵌套式Transformer模块	描述符
def transformer_in_transformer(Z, Y){	
Y = Y + MHA(Y)	内部多头自注意力
Y_new = Y + FFN(Y)	内部全连接网络
Y_reshape = Y_new.reshape(-1, n, dim)	
Z = Z + linear(Y_reshape)	
Z = Z + MHA(Z)	外部多头自注意力
Z_new = Z + FFN(Z)	外部全连接网络
return Z_new, Y_new	
}	

7.2.2 参数化跳跃连接

参数化跳跃连接通过引入恒等映射之外的带参数的投影来更好地丰富特征多样性和防止特征坍塌。参数化跳连和多头自注意力模块（MSA）模块是并行的，其表示形式见式（6）：

$$AugMSA(Z_l) = MSA(Z_l) + Z_l + \sum_{i=1}^{T} \mathcal{T}_{li}(Z_l; \Theta_{li}), l \in [1,2,\cdots,L] \quad (6)$$

式中：

$T_{li}(\cdot)$——在第l层中的第i条参数化的跳连Θ_{li}表示它的参数。

在一层中，共包含T条参数化跳连。除了原来的跳连，参数化跳连提供了更多的替代路径来绕过注意力机制。与恒等投影直接将输入块复制到相应的输出不同，参数化投影可以将输入特征转换为另一个特征空间。不同参数化跳连的权重矩阵不同，$T_{li}(\cdot)$对输入特征进行不同的变换，因此更多的参数化跳连并行可丰富特征空间。

参数化跳连$T_{li}(\cdot)$的一个简单实现形式见式（7）：

$$\mathcal{T}_{li}(Z_l;\Theta_{li}) = \sigma(Z_l\Theta_{li}), l \in [1,\cdots,L], i \in [1,2,\cdots,T] \quad \text{.....(7)}$$

式中：

σ —— 激活函数（例如GeLU）；

Θ_{li} —— 权重矩阵；

$T_{li}(Z_l;\Theta_{li})$ —— 独立地对不同图像块的特征作变换，保留其独特属性。

注：考虑到在自注意力网络中跳连同时存在于多头自注意力模块（MSA）和多层感知机模块（MLP），参数化跳连也可以类似地嵌入到多层感知机模块（MLP）中，见式（8）：

$$AugMLP(Z'_l) = MLP(Z'_l) + Z'_l + \sum_{i=1}^{T}\mathcal{T}_{li}(Z'_l;\Theta'_{li}), l \in [1,2,\cdots,L] \quad \text{.....(8)}$$

式中：

Z'_l —— 多层感知机模块（MLP）的输入。

7.2.3 基于循环矩阵的高效部署

直接部署7.2.1和7.2.2等方法需要引入矩阵乘法运算将消耗极大的计算代价。本文件使用分块循环矩阵来实现高效的部署。

首先，将原矩阵Θ划分成$b \times b$个子矩阵，见式（9）：

$$\Theta = \begin{bmatrix} C^{11} & C^{12} & \cdots & C^{1b} \\ C^{21} & C^{22} & \cdots & C^{2b} \\ \vdots & \vdots & \vdots & \vdots \\ C^{b1} & C^{b2} & \cdots & C^{bb} \end{bmatrix} \quad \text{.....(9)}$$

每个子矩阵C^{ij}都是循环矩阵，它可以通过循环一个向量中的元素得到，见式（10）：

$$C^{ij} = circ(\mathbf{c}^{ij}) = \begin{bmatrix} c_1^{ij} & c_{d'}^{ij} & \cdots & c_3^{ij} & c_2^{ij} \\ c_2^{ij} & c_1^{ij} & c_{d'}^{ij} & & c_3^{ij} \\ \vdots & c_2^{ij} & c_1^{ij} & \ddots & \vdots \\ c_{d'-1}^{ij} & & \ddots & \ddots & c_d^{ij} \\ c_{d'}^{ij} & c_{d'-1}^{ij} & \cdots & c_2^{ij} & c_1^{ij} \end{bmatrix} \quad \text{.....(10)}$$

循环矩阵通过傅里叶变换进行计算。在计算中，输入特征先被划分成多个小块特征，后进行做快速傅里叶变换。在频域内，特征和权重做逐点的相乘运算，再通过逆傅里叶变换将计算结果映射回空间域。循环矩阵输出如式（11）所示：

$$\mathcal{T}(Z)^i = \sigma\left(\sum_{j=1}^{b} Z^j C^{ij}\right) = \sigma\left(\sum_{j=1}^{b} \mathrm{IFFT}\left(\mathrm{FFT}(Z^j) \circ \mathrm{FFT}(\mathbf{c}^{ij})\right)\right) \quad \text{.....(11)}$$

式中：

FFT——快速傅里叶变换；

IFFT——快速傅里叶逆变换；

$T(Z) = [T(Z)^1; T(Z)^2; \cdots; T(Z)^b]$。

分别将参数化跳连并联到Transformer模型的多头自注意力模块（MSA）和多层感知机模块（MLP）将得到增强的多头自注意力模块和增强的多层感知机模块。交替堆叠这两个模块构成整个增强的Transformer模型。

基于参数化跳连的Transformer模块模块的操作定义见表18，伪代码见表41。

表 41　基于参数化跳连的 Transformer 模块伪代码描述

基于参数化跳连的Transformer模块	描述符
def aug_transformer(Z){	
Z = Z+ MHA(Z) +sum(augshortcut(Z),n)	与MSA模块并联参数化跳连
Z _new = Z+ FFN(Z) + sum(augshortcut(Z),m)	与MLP模块并联参数化跳连
return Z_new	
}	

7.2.4　归一化加速

Transformer网络目前普遍采用了层归一化（LayerNorm）算子，而在推理网络时，该算子不仅需要在线计算过程，还需要复杂的开方操作，影响Transformer网络的运行速度和内存需求。归一化加速方法利用数据的先验统计信息，作用于相邻的线性操作中，其计算方法为：

$$Y = \gamma\frac{X}{\sigma} + \beta \quad (12)$$

式中：

γ, σ, β —— 均为常数，可合并至相邻线性层的计算当中。

注：该方法训练过程分为前向过程和反向过程。

7.2.4.1　前向过程

给定一个网络的输入$X_t \in R^{B\times C}$，则可以通过当前方差以及历史方差得到更加准确的平滑方差：

$$\sigma_t^2 = \frac{1}{B}\sum_{i=1}^{B} X_t^2 \quad (13)$$

$$\hat{\sigma}_t^2 = \sqrt[M]{\prod_{i=0}^{M-1}\sigma_{t-i}^2} \quad (14)$$

计算输出值见式（15）和式（16）：

$$Z_t = \frac{X_t}{\sqrt{\hat{\sigma}_t^2+\epsilon}} \quad (15)$$

$$Y_t = \gamma Z_t + \beta \quad (16)$$

平滑后的方差被更新到推理用的方差中：

$$\sigma^2 = \alpha\sigma^2 + (1-\alpha)\hat{\sigma}_t^2 \quad (17)$$

7.2.4.2 反向过程

由于在前向过程中引入了对统计值的平滑，所以采用平滑策略近似梯度计算反向过程中的梯度代价。Z_t的梯度计算方法见式（18）：

$$\frac{\partial L}{\partial Z_t} = \gamma \frac{\partial L}{\partial Y_t} \quad (18)$$

平滑方差的梯度估计方法见式（19）和式（20）：

$$g_{\hat{\sigma}_t^2} = \frac{1}{B}\sum_{i=1}^{B} \frac{\partial L}{\partial Z_i} Z_i \quad (19)$$

$$\varphi_{\hat{\sigma}_t^2} = \alpha \varphi_{\hat{\sigma}_{t-1}^2} + (1-\alpha)\frac{1}{M}\sum_{i=0}^{M-1} g_{\hat{\sigma}_{t-i}^2} \quad (20)$$

输出的梯度见式（21）：

$$\frac{\partial L}{\partial X_t} = \frac{1}{\sqrt{\hat{\sigma}_t^2 + \epsilon}}\left(\frac{\partial L}{\partial Z_t} - Z_t \varphi_{\hat{\sigma}_t^2}\right) \quad (21)$$

Transformer统一归一化层模块操作定义模块的操作定义见表19，伪代码见表42。

表 42　Transformer 统一归一化层推理前向伪代码描述

Transformer统一归一化层推理前向模块	描述符
def unified_norm(x, σ^2, γ, β){	
X_norm = x/$\sqrt{\sigma^2}$	归一化
Y=γX_{norm} + β	仿射变换
Return Y	
}	

7.2.5 多模态融合模块

多模态融合模块利用不同模态之间的对齐特性，将不同模态的信息自适应地进行混合，从而提高模态融合效果。多模态融合模块利用全连接网络构建一个评分模块$s^l(e_m^l)$，对第1层的第m个模态的每个特征进行重要性评估：

$$\hat{e}_m^l = \mathrm{MSA}\left(\mathrm{LN}(e_m^l) \cdot s^l(e_m^l)\right) \quad (22)$$

$$e_m^{l+1} = \mathrm{MLP}\left(\mathrm{LN}(\hat{e}_m^l)\right) \quad (23)$$

对评分模块施加L1约束使其在训练中稀疏化，即引入损失函数$\sum_{l=1}^{L}|s^l(e_m^l)|$。

跨模态的交互通过评分模块的分值来引导：

$$e_m^l = e_m^l \odot \mathbb{I}_{s^l(e_m^l)\geq\theta} + \mathrm{Proj}_{m'}^{M}(e_m^l) \odot \mathbb{I}_{s^l(e_m^l)<\theta} \quad (24)$$

其中I为指示函数，$\theta = 0.01$是一个阈值，⊙表示按元素相乘，$\mathrm{Proj}_{m'}^{M}(e_m^l)$表示将第m模态的特征$e_m^l$投影至第$m'$模态得到对应特征。当一个模态特征被替换时，利用残差结构的位置编码保留其位置特性。

当输入模态多于两个时，为了防止模态间替换混乱，模态间的特征混合方案以随机产生的形式固定为一种分配方案$a_{m'}(m) \in \{0,1\}^N$，该模态间的特征混合表达式为：

$$e_m^l = e_m^l \odot \mathbb{I}_{s^l(e_m^l)\geq\theta} + \sum_{m'=1,m'\neq m}^{M} a_{m'}(m) \odot \mathrm{Proj}_{m'}^{M}(e_m^l) \odot \mathbb{I}_{s^l(e_m^l)<\theta} \quad (25)$$

Transformer多模态融合模块操作定义模块的操作定义见表20，伪代码见表43。

表 43 Transformer 多模态融合模块伪代码描述

Transformer多模态融合模块	描述符
def tokenfusion(x1, x2, score1, score2, threshold){	
x1_new = x1	
x2_new = x2	
index1 = np.argwhere(score1 < threshold)	模态1重要性低的token坐标
x1_new[index1] = x2[proj_1_to_2(index1)]	模态1低分值token用模态2替换
index2 = np.argwhere(score2 < threshold)	模态2重要性低的token坐标
x2_new[index2] = x1[proj_2_to_1(index2)]	模态2低分值token用模态1替换
return x1_new, x2_new	
}	

7.3 大模型加速压缩流程

7.3.1 量化

7.3.1.1 概述

大规模预训练模型的量化是提高计算效率和减少模型复杂度的关键手段。量化的核心在于减少模型中参数的表示精度，通过降低数值精度来减小存储需求和加速计算过程，同时尽量维持模型的性能不变。量化技术能与多种模型优化策略相结合，比如剪枝技术，可以在量化前对模型进行剪枝，进一步优化模型效率。在执行量化之后，模型的每个权重都会被转换成较低精度的格式，例如，仅使用较少的位来表示每个权重值。特别是对于那些已经通过剪枝操作精简过的模型，量化可以直接应用于这些被保留下来的权重，从而实现额外的效率提升。量化过程中，可以根据权重的重要性来决定其量化的程度，这与剪枝时选取保留权重的策略相似。

7.3.1.2 量化准则

7.3.1.2.1 大规模预训练模型仿射变换量化算法准则

仿射变换量化的算法可用于优化规模预训练模型的免训练量化，通过直接应用等效的仿射变换来优化模型；同时，扩展了优化的范围，包括权重的缩放、平移等变换，从而显著减少量化过程中的误差。该算法利用仿射变换的逆矩阵来保持量化前后模型输出的等效性，并引入基于Levy-Desplanques定理的渐进式掩码优化方法，以确保矩阵优化过程的稳定性。具体算法流程如下。

a） 对给定的大规模预训练模型，应用仿射变换矩阵优化权重分布。

1) 伪量化函数定义为：

$$Q(x) = \Delta * \left(\text{clamp}\left(\left\lfloor \frac{x}{\Delta} \right\rfloor + zp, 0, 2^n - 1\right) - zp\right) \quad (26)$$

式中：

Δ、zp、n 分别代表量化步长、量化零点和比特位。

2） 对于单个线性层，仿射量化（AffineQuant）优化方法为：

$$\arg\min_{A} \|XW - XA^{-1}Q(AW)\|_F^2 \quad (27)$$

式中：

A、W、X、$Q(\cdot)$ 分别表示仿射变换矩阵、权重矩阵、量化函数、激活值。

仿射量化将仿射变换矩阵 A 左乘以权重矩阵 W，以更好地将权重分布与量化函数Q（·）对齐。扩大优化空间可以减小转换权重中的量化误差，从而减少困惑度。同时将仿射变换矩阵 A 的逆函数右乘以激活值 X，以保持激活和权重之间矩阵乘法输出的不变性。

3) 在大型语言模型量化中，仿射量化的优化目标如下：

$$\arg\min_{A,\delta} \|f_i(X,W) - f_i\big((X-\delta)A^{-1}, Q(AW), b+\delta W\big)\|_F^2 \quad (28)$$

式中：

f_i —— 第 i 个 Transformer 块；

$(X-\delta)A^{-1}$ —— 等效变换后得到的激活权重和偏差；

$Q(AW)$ —— 等效变换后得到的激活权重和偏差；

$b+\delta W$ —— 等效变换后得到的激活权重和偏差；

AffineQuant 结合了放射变化和平移交换，并使用 Transformer 块量化前和量化后输出的均方误差为优化目标。

b） 使用逆仿射变换矩阵处理激活值，以保持输出的一致性。

若每个对角元素的绝对值大于相应行中其余元素的绝对值之和，则矩阵 A 被视为严格对角占优。数学表达为，

$$|\boldsymbol{a_{ii}}| > \sum_{i\neq j}|\boldsymbol{a_{ij}}| \quad (29)$$

在 Levy-Desplanques 定理中还规定了所有严格对角占优矩阵都是可逆的。

c） 使用渐进式掩码，以优化矩阵稳定性。

为了确保仿射变换矩阵在优化过程中严格保持对角元素占主导地位，在每个优化块开始时，通过渐进掩码方法，固定除了主对角线上的元素之外的所有元素，随着优化进行，逐渐解冻主对角线附近的元素，所有矩阵元素都变得可学习以进行优化：

$$GM_{ij} = \begin{cases} 1 & i = j \\ \alpha & 0 < |i-j| \leq \frac{e}{t} \times \text{ hidden size} \\ 0 & \text{otherwise} \end{cases} \quad (30)$$

式中：

GM_{ij} —— 掩码矩阵的第 i 行第 j 列元素；

T —— 目标 epoch；

e∈[1,t] —— 当前的 epoch；

hidden size —— 仿射变换矩阵的维数；

α —— 稳定因子。

在每个优化块开始时，冻结除了主对角线上的元素。在注意力模块中，在每个注意力头中一致

更新，因为它们不受严格对角占优矩阵的约束。相反，远离主对角线的元素最初被冻结，随着优化过程逐渐解冻，其按稳定因子 α 缩放。伪代码见表 44。

表 44 仿射变换量化伪代码描述

仿射变换量化	描述符
quantize(W,X,A, GM,group_size,scale,qmin,qmax){	
A_inverse = A.mul(GM).inverse();	
W = W.matmul(A.mul(GM)).t();	
X = A.mul(GM).inverse().t().matmul(X)	
If(group_size>0){	
dim1, dim2 = W.shape	
W = W.reshape(-1, group_size)	
}	
W_int = ((W / scale).round()-(W / scale)).detach()+(W / scale)	
W_int = W _int.clamp(qmin, qmax)	
W_dequant = W_int.mul(scale)	
W_dequant = W_dequant.reshape(dim1, dim2)	
}	

7.3.1.2.2 视觉 Transformer 量化补偿和级联优化

低比特无训练量化引入了通道补偿方案，在Transformer的add的相关层采用量化补偿系数对不同通道进行补偿。并且以Block为重构单元，对网络权重和激活值进行级联地联合优化。具体流程如下。

a) 通道补偿：

1) 对于 shortcut 的 add 层，使用逐通道的补偿系数 scale 和 bias，用于调整通道的分布。插入逐通道的 scale 和 bias 算子后，其前向过程替换为：$y = scale \cdot x + bias$；

2) 通道补偿可调整输出的分布。对 y 进行量化，可以避免较大的性能损失。对相连的层则需要减去 bias，除以 scale，以保持一致。该操作如果对应矩阵乘法或者其他线性操作，可以被合并到该线性操作里。

b) 级联重构优化：

1) 先对激活值和权重量化间隔做限制的联合优化，再重构权重。首先，选取一个基本单元，其内部第 i 个单元的权重和激活值的量化间隔分别s_w^i和s_a^i。以 MSE 为损失函数优化量化间隔，优化目标如下：

$$\min_{\delta_a,\delta_w} \|L(s_w+\delta_w, s_a+\delta_a) - L(s_w, s_a)\|_2 \quad (31)$$

$$s.t. |\delta_w| < \theta s_w, |\delta_a| < \theta s_a \quad (32)$$

式中：

s_w、s_a——离线统计得到的量化间隔。

2) 对基本单元的权重进行重构，使其在量化中自适应的向上或向下取整。优化的目标函数为：

$$\underset{\delta W}{\operatorname{argmin}}\|L(W+\delta W)-L(W)\| \quad \text{.....(33)}$$

级联重构量化的伪代码描述见表45。

表 45 级联重构量化伪代码描述

级联重构量化	描述符
quantize(W,X){	
For i=1 to E	第一次优化
Loss=‖L(s_w+δ_w,s_a+δ_a)-L(s_w,s_a)‖	
更新δ_w和δ_a	
End for	
For i=1 to E2	第二次优化
Loss = L(W+δ_W)-L(W)	
更新δ_W	
End for	
W_int = (W _f/(s_w+δ_w).round()+δ_W).clamp(min, max)	量化权重
X_int = (X/δ_a.round()).clamp(min, max)	量化激活
}	

7.3.2 剪枝

7.3.2.1 概述

大规模预训练模型的剪枝操作是一种降低模型复杂度、提高计算效率的重要策略。具体而言，剪枝的目的是辨识并移除模型中的冗余参数，这样能缩减储存需求和计算负担，同时保持原有的模型性能。剪枝方法具有较强的兼容性，例如，对于量化方法，可应用于剪枝后的模型从而达到进一步的优化。在剪枝操作后，可以获得对应的稀疏掩码表，量化操作对于掩码表中为1的对应权重（即未被剪枝掉的权重）进行操作。而结构化剪枝由于删除了部分结构，因此可以直接对精简后的模型进行量化。

7.3.2.2 结构感知剪枝准则

权重裁剪通过对Transformer中的冗余权重的裁剪，可以大大地降低网络所需的内存和计算量。以往的方法在裁剪Transformer时，仅考虑了单个结构的重要性。由于Transformer中不同模块有其具体的物理意义，考虑到Transformer中不同结构的相互重要性，并基于该重要性进行权重裁剪，可采用Hessian阵来表征这种相互重要性，并通过理论推导简化裁剪过程的计算复杂度。

裁剪可视为对网络权重的扰动，若初始权重为w，裁剪后变成了w+Δw，那么衡量这种扰动对网络产生影响的损失函数L：

$$\Delta \mathrm{L} = \mathrm{L}(\mathrm{w}+\Delta \mathrm{w})-\mathrm{L}(\mathrm{w}) = \Delta \mathrm{w}^{T} g_w + \frac{1}{2}\Delta w^{T} H_w \Delta w + \mathrm{O}(\|\Delta w\|^{3}) \quad \text{.....(34)}$$

在给定计算量的限制时，裁剪的优化目标为最大程度地降低权重扰动给网络带来的损失增长：

$$\min \Delta L \quad s.t. \quad FLOPs(w + \Delta w) \leq C \quad (35)$$

式中：

FLOPs —— 模型的计算量大小；

C —— 给定的计算量限制。

注：对于ΔL，因为损失函数往往是二次或二次以下的函数，可直接忽略其中最右边第三项高阶项。

对于Taylor一阶项$\Delta w^T g_w$，本方案可以具体表示为各个权重扰动影响的累加，见式（36）：

$$\Delta w^T g_w = \sum_{\{i \in S\}} (w_i - 0)\, g_{w_i} \quad (36)$$

式中：

S——被裁剪的权重集合。

Taylor二阶项$\Delta w^T H_w \Delta w$表示各个权重之间的交互关系，能够帮助识别需要裁剪的权重，表示为：

$$\Delta w^T H_w \Delta w = \sum_{(i,j \in S)} w_i H_{ij} w_j \quad (37)$$

注：由于Transformer参数量庞大，完整求出Hessian需要庞大的内存占用和计算量，在很多设备上无法实现，会阻碍算法的应用，可考虑一种近似方法，既保留这种交互关系，又能快速计算得到。

本方案将Transformer中的权重分成三个部分，分别是head部分、残差连接层、FFN中内部相连的层。对于三个部分，式（37）两两展开，既包含单独模块的关系，也包含两两模块之间的相关关系，最终优化目标可以表示为：

$$\min F(\alpha, \beta, \gamma) = \sum_{i \in S_P} \Delta w^T g_w + \lambda_{ij}(\alpha^2 + \beta^2 + \gamma^2 + 2\alpha\beta + 2\alpha\gamma + 2\beta\gamma) \quad (38)$$

其中，λ_{ij}的表示为：

$$\lambda_{ij} = N_i N_j \overline{H_{ij}} \quad (39)$$

式中：

N_i —— 表示权重参数数量；

H_{ij} —— 表示第i个模块和第j个模块的海森阵；

α, β, γ —— 为三个模块的裁剪比例。可采用进化算法对问题进行求解。

在每个模块内部，则采用Fisher信息矩阵进行裁剪。其裁剪准则为式（40）：

$$I = \frac{1}{N} \sum_{n=1}^{N} \left(w_i \frac{\partial L}{\partial w_i} \right)^2 \quad (40)$$

结构感知剪枝的伪代码描述见表46。

表 46 结构感知剪枝伪代码描述

结构感知剪枝伪代码	描述符
pruning(W，C，E，Q){	
For i =1 to E	
For j =1 to Q	
If C(q) < C:	
continue	
End if	
For p_t in q	
Prune according to I	
End for	

Compute f according to F(q)	
End for	
Keep top-k q	
Generate new q by Crossover and Mutation	

表 46 结构感知剪枝伪代码描述（续）

结构感知剪枝伪代码	描述符
End for	
Return new pruned w	
}	

7.4 大模型迁移压缩流程

7.4.1 概述

迁移学习指的是将一个在大规模数据集上预先训练好的模型（如图像识别、自然语言处理模型）应用到相关但不同的任务上，通过微调(fine-tuning)少量参数或增加少量任务特定层，以实现对新任务快速且高效的学习和适应。这个过程减少了训练参数，降低了训练成本和时间，保持或提高了模型在新任务上的表现。

大模型迁移压缩方法是一种参数高效的在大模型向下游任务进行迁移学习的同时进行模型压缩的方法。该方法针对大规模预训练模型迁移学习方法与大模型压缩方法在适配下游任务阶段存在效率低下的问题，通过插入特定结构并在迁移时对其进行稀疏训练，实现参数高效的大模型迁移与压缩。

7.4.2 基础操作

高效迁移学习与迁移压缩包含一系列适配器（Adapter）模块，Adapter适配器操作定义见表 21。

7.4.3 多模态大模型高效迁移

7.4.3.1 概述

多模态大模型高效迁移方法采用混合模态适配模块，以自动根据单模态指令和多模态指令进行权重切换，使得模型可以同时处理单模态和多模态任务。混合模态适配模块包含路由函数，可以根据输入模态选择最佳适配器路径，降低大型语言模型适配至多模态领域的成本。

多模态大模型高效迁移操作定义见表 22，伪代码见表 47。

表 47 多模态大模型高效迁移伪代码描述

高效迁移	描述符
Efficient transfer(W,A,R,D,T){	
// Initialize	
A ← initialize(W) ; // initialize adapter	
R ← initialize(W) ; // initialize router	
s ← 0 ; // initialize ratio s	
// Efficiennt finetuning	
While s ≤ T:	
s ← Update(s)	
For (X,$\hat{Y}$) in sample(D):	

表 47 多模态大模型高效迁移伪代码描述（续）

高效迁移	描述符
Y=model.forward(W,A,R,d)	
L=loss_function(Y, ,$\hat{Y}$)	
Backward(L,A)	
Backward(L,R)	
Return W,A,R as W_o, A_o, R_o	

7.4.3.2 具体方法

混合模态适配模块引入模态标记来指示输入模态，见式（41）：

$$t_m = mE_m \quad (41)$$

式中：

$E_m \in \mathbb{R}^{2\times c}$ —— 模态嵌入；

$m \in \mathbb{R}^2$ —— 输入模态的独热编码。

基于模态标记t_m，混合模态适配器可以动态调整输入特征$Z \in \mathbb{R}^{n\times c}$的适配，在实践中，Z可以是单模态或多模态的特征。因此，多模态适配器定义见式（42）：

$$Z' = Z + s \cdot \text{router}\left(f_{a_1}(Z), f_{a_2}(Z); f_\omega(t_m)\right) \quad (42)$$

式中：

f_{a_1}、f_{a_2} —— 表示 RepAdapters；

s —— 调节梯度尺度的超参数；

router —— 决定两个适配器路由路径的路由函数。

为进一步节省模型的训练参数量，两个适配器的下采样投影矩阵参数共享。路由函数router的定义见式（43）和式（44）：

$$\text{router}\left(f_{a_1}(Z), f_{a_2}(Z)\right) = \widehat{\omega_0} \cdot f_{a_1}(Z) + \widehat{\omega_1} \cdot f_{a_2}(Z) \quad (43)$$

$$\hat{\omega} = f_\omega(t_m) = \text{softmax}\left(\frac{t_m W_m + b_m}{\tau}\right) \quad (44)$$

其中，$W_m \in \mathbb{R}^{c\times 2}$和$b_m \in \mathbb{R}^2$分别是权重矩阵和偏置，τ是softmax函数的温度，$\hat{\omega}$表示路由权重。混合模态训练旨在冻结大型图像编码器和大型语言模型的参数，仅微调插入适配器的参数，以实现整个多模态大型语言模型的端到端联合优化。端到端的优化目标可表示为式（45）：

$$\text{argmin}\left(\mathcal{L}(f_\emptyset(I,T),R;\theta_a)\right) \quad (45)$$

式中：

R —— 表示正确应答标签值；

$\mathcal{L}(\cdot)$ —— 正确应答标签值和目标损失函数；

$f_\emptyset$ —— 大型语言模型；

θ_a —— 适配器的参数；

$I \in \mathbb{R}^{h\times w\times 3}$ —— 输入图像；

$T \in \mathbb{R}^l$ —— 文字说明。

在训练期间，构建一个从纯文本指令和文本图像指令随机采样的小型训练批次。整体训练目标$\mathcal{L}$可定义为：

$$\mathcal{L} = \sum_{i=1}^{m}\sum_{s=1}^{S+1}\log p\left(R_s^i|I^i,T^i,R_{0:s-1}^i;\theta_a\right)+\sum_{j=1}^{n}\sum_{s=1}^{S+1}\log p\left(R_s^j|T^j,R_{0:s-1}^j;\theta_a\right) \quad (46)$$

式中：

M —— 文本图像指令样本在一个小型训练批次中的数量；

n —— 纯文本指令样本在一个小型训练批次中的数量。

7.4.4 视觉大模型高效迁移

7.4.4.1 概述

视觉大模型的高效迁移适用于优化视觉大模型的空间成本和迁移效率，在给定一个预训练的视觉大模型的情况下，利用现有的结构重参数方法，把训练好的轻量级适配器通过矩阵乘法合并到原来的网络中，使推理过程中的模型结构保持不变。

视觉大模型高效迁移操作定义见表 23，伪代码见表 48。

表 48 视觉大模型高效迁移伪代码描述

高效迁移	描述符
Efficient transfer(W,A,R,D,T){	
// Initialize	
A ← initialize(W) ; // initialize adapter	
s ← 0 ; // initialize ratio s	
// Efficiennt finetuning	
While s ≤ T:	
s ← Update(s)	
For (X,$\hat{Y}$) in sample(D):	
Y=model.forward(W,A,d)	
L=loss_function(Y, ,$\hat{Y}$)	
Backward(L,A)	
A ← reparameterize(A,s) ; // reparameterize adapter	
Return W,A as W_o, A_o	

7.4.4.2 具体方法

视觉大模型高效迁移具体方法包括如下步骤。

a） 首先，设置重参数适配器的结构，其表达式由如下式（47）表示：

$$f(X;\theta) = s\cdot GLinear(XW_d + b_d) + X \quad (47)$$

式中：

X —— 输入特征；

$W_d \in R^{d\times c}$ —— 投影权值

$b_d \in R^c$ —— 偏置；

S —— 调节梯度标量的超参数；

$GLinear(\cdot)$ —— group-wise 变换，其表达见式（48）：

$$GLinear(X) = \left[X'_{g^0}W_{g^0}, \ldots, X'_{g^k}W_{g^k}\right] + b \quad \ldots\ldots(48)$$

式中：

$X'_i \in R^{n\times\frac{c}{k}}$ —— 对$X \in R^{n\times c}$特征的切分；

k —— group 的数量；

$W_i \in R^{\frac{c}{k}\times\frac{d}{k}}$ —— 投影权重；

$b \in R^d$ —— 偏置。

b) 其次，部署重参数适配器，把适配器部署到视觉 Transformer 的结构中，分别应用到多头注意力模块（MHA）和全连接层（FFN）的输入，其表达见式（49）和式（50）：

$$X'_l = MHA\left(f(LN(X_{l-1});\theta)\right) + X_{l-1} \quad \ldots\ldots(49)$$

$$X_l = FFN\left(f(LN(X'_l);\theta)\right) + X'_l \quad \ldots\ldots(50)$$

式中：

X_{l-1} —— 前一层的输入特征；

X_l —— 后一层的输入特征；

$f(\cdot)$ —— 适配器函数；

θ —— 适配器参数；

$LN(\cdot)$ —— 正则化函数。

c) 然后，融合训练后的重参数适配器，把训练后的适配器进行结构重参数化并融合到原模型中，其步骤如下：

1) 重新构建 group-wise 变换的定义，表达式为：

$$Linear(X) = XW_{sparse} + b \quad \ldots\ldots(51)$$

式中：

$W_{sparse} \in R^{c\times d}$是由原来的 group-wise 变换矩阵序列$W_i$合并及填充零元素得到的稀疏矩阵。

2) 继续简化适配器的公式定义，使其变换成一个简单的线性层，再通过矩阵相乘与模型中的投影矩阵融合，表达式：

$$f(X;\theta) = XW_{ada} + b_{ada} \quad \ldots\ldots(52)$$

式中：

$W_I \in R^{d\times d}$表示单位矩阵。

3) 重新描述结构重参数化后的视觉 Transformer，以全连接层为例，其模块的公式定义为：

$$FFN(f(X;\theta)) = \&\sigma(XW_{rep} + b_{rep})W_2 + b_2 \quad (53)$$

式中：

$W_{rep} = W_{ada}W_1$ —— 表示全连接层重参数化后的投影矩阵；

$b_{rep} = b_{ada}W_1 + b_1$ —— 表示投影偏置。

4) 与 3）类似，多头注意力模块也被重参数化，其公式定义见式（54）和式（55）：

$$FScore^i(X) = softmax\left(\frac{(XW^i_{Q_{rep}}+b_{Q_{rep}})(XW^i_{K_{rep}}+b_{K_{rep}})^T}{\sqrt{d_k}}\right) \quad (54)$$

$$Attn^i(f(X;\theta)) = Score^i(X)\left(XW^i_{V_{rep}} + b_{V_{rep}}\right) \quad (55)$$

式中：

$W^i_{Q_{rep}}$，$W^i_{K_{rep}}$，$W^i_{V_{rep}}$ —— 分别表示重参数化权重；

$b_{Q_{rep}}$，$b_{K_{rep}}$，$b_{V_{rep}}$ —— 分别表示重参数化偏置。

7.4.5 大模型迁移压缩

7.4.5.1 概述

迁移压缩方法主要分为模块设计、损失设计和流程设计三部分；

a） 模块设计可复用高效迁移方法的适配器模块并进行针对性改进。可代理原预训练权重进行压缩是指仅需对该模块的参数或结构进行压缩，即可实现对应预训练权重的压缩；

b） 损失设计往往与压缩方式和模块设计相耦合，即针对模块结构与权重进行损失设计，以通过指定的压缩方式，达到压缩目标；

c） 流程设计上，总体流程为压缩感知的高效迁移学习流程，并针对不同的压缩方式进行具体的设计，如剪枝可为迭代剪枝，量化为量化感知的参数高效微调。

迁移压缩操作定义见表24，伪代码见表49。

表 49 迁移压缩伪代码描述

迁移压缩	描述符
Transfer_compression(W,A,D,R){	
// Initialize	
A ← initialize(W) ; // initialize adapter	
s ← 0 ; // initialize ratio s	
// Pre-Finetuning (not necessary)	
For (X,$\hat{Y}$) in D:	
Y=model.forward(W,A,d)	
L=loss_function(Y, ,$\hat{Y}$)	
Backward(L,A)	
// Compression-aware Finetuning	

表 49 迁移压缩伪代码描述（续）

While s ≤ R:	
s ← Update(s)	
A ← compress(A,s) ; // compress adapter	
迁移压缩	描述符
or (X,$\hat{Y}$) in sample(D):	
Y=model.forward(W,A,d)	
L=loss_function(Y, ,$\hat{Y}$)	
Backward(L,A)	
// Post-Finetuning (not necessary)	
For (X,$\hat{Y}$) in D:	
Y=model.forward(W,A,d)	
L=loss_function(Y, ,$\hat{Y}$)	
Backward(L,A)	
Return W,A as W_o, A_o	

7.4.5.2 具体方法

以迁移压缩中的迁移剪枝为例介绍具体方法。

a） 线性适配器（Linear_adapter）。其作用于各类线性操作的输出（包括多头注意力层中 QKV 特征向量和前馈层的投影输出）。线性适配器的公式见式（56）。

$$\mathbf{Y} = \vec{\gamma} \odot \mathbf{X} + \vec{\beta} \quad (56)$$

式中：

X —— 表示来自前一层（W_q、W_k、W_v、W_1、W_2）的输出；

$\vec{\gamma}$ —— 表示缩放因子；

$\vec{\beta}$ —— 表示偏移因子；

$\odot$ —— 表示 Hadamard 积。

通过广播操作，使得缩放和偏移参数能够应用于X 的适当维度。在重新参数化后，通过将缩放向量的特定元素设置为零，在重新参数化后，可以有效地剪枝掉权重矩阵 W 中的相应向量。适配器可以重新参数化为如下的权重矩阵，见式（57）。

$$W' = \mathrm{diag}(\vec{\gamma}) \cdot W, \quad B' = \mathrm{diag}(\vec{\gamma}) \cdot B + \vec{\beta} \quad (57)$$

式中：

W′ 和 B′ 表示适应后的权重和偏移矩阵；

b） 基于结构稀疏感知优化技术和重建 Loss 值，可以在迭代剪枝过程中逐步增强模型的稀疏性的同时保持性能。针对适配器参数的缩放问题，为了提升适配器权重的稀疏性，稀疏损失 L_{sparse} 使用 L1 范数的定义式为：

$$L_{sparse} = \sum_{i=1}^{n} |\vec{\gamma}_i|_1 \quad (58)$$

为了关注非零的缩放参数（即未被剪枝的参数），可使用 L1 惩罚函数来近似梯度。缩放参数的梯度表达式为：

$$\nabla_i f(X) = \nabla_i L(X) + \lambda, \mathrm{sign}(X_i) \quad (59)$$

在剪枝前使用相同的适配器对一个参考模型进行微调。在迭代循环中，通过重建损失从而利用参考模型的信息。重构损失等于每个稀疏训练适配器层的输出与相应参考适配器层输出之间的欧氏距离，其表达式为：

$$L_{rec} = \left| X_i - X_i^{ref} \right|_2 \quad (60)$$

式中：

x_i —— 稀疏训练适配器的输出；

x_i^{ref} —— 参考适配器的输出表

因此，对于输入 x，综合的损失函数通过分类损失 L_{cls}、重构损失 L_{rec} 和稀疏损失 L_{sparse}被定义为：

$$L = L_{cls} + \lambda_{rec} \cdot L_{rec} + \lambda_{sparse} \cdot L_{sparse} \quad (61)$$

式中：

λ_{rec} ， λ_{sparse} —— 调节重构和稀疏损失成分重要性的超参数。

c） 在剪枝模块集成之后，为在提高模型的稀疏率的同时进行高效的参数微调，每个迭代的剪枝阶段都会增加预定的剪枝率，并在接下来的 T 个周期内进行重新训练，训练完成后进入下一个剪枝周期。

注：该步骤会持续进行，直到模型达到预定的剪枝目标率。

7.4.6 大模型微调

7.4.6.1 Prompt 微调

Prompt微调是一种通过修改模型输入的方式来实现微调的方法，不需要对模型的参数进行大规模更新。通过在输入数据中添加任务特定的模态提示来引导模型生成正确的输出。Prompt微调的核心思想是利用自然语言中的提示（prompts）来引导模型进行任务。对于分类任务，通过在输入文本前后添加特定的提示语句，使模型能够根据任务需求进行分类。例如，在情感分析任务中，可以在输入句子后添加“这句话的情感是：”这样的提示语句，来引导模型输出情感类别。

Prompt微调的具体操作包括如下内容。

a) 提示语选择：设计合适的提示语句，以确保模型能够理解并执行任务。提示语可以是简单的词语或短语，也可以是完整的句子。

b) 输入处理：将提示语与输入数据结合，形成新的输入序列。对于生成任务，可以在输入文本中嵌入提示语句，引导模型生成与任务相关的文本。

c) 模型调整：在输入层或编码层进行调整，使模型能够处理带有提示语的输入数据。这种调整通常包括修改输入格式、更新词嵌入等。

Prompt微调运算操作定义见表50。

表 50 Prompt 微调运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
Prompt微调		Input	X	输入特征张量	Tensor

	在输入数据中添加任务特定提示语句		prompt	提示语句	String
		Output	Y	输出特征张量	Tensor
		Attributes	task_type	必选，任务类型（分类/生成）	String

7.4.6.2 LoRA 微调

LoRA（Low-Rank Adaptation）微调是一种通过对模型参数进行低秩适配的微调方法，能够在保持模型性能的同时，显著减少微调所需的计算资源和存储空间。LoRA微调的核心思想是通过低秩矩阵分解来调整模型参数。在传统的微调方法中，所有模型参数都会参与调整，而LoRA通过将高维参数矩阵分解为低秩矩阵，减少了参数数量，从而降低了计算复杂度。

LoRA 微调的具体操作包括如下内容。

a) 参数选择：将模型的参数矩阵分解为两个低秩矩阵，以减少参数数量。

b) 低秩适配：在微调过程中，仅调整这两个低秩矩阵A和B，而不改变原始参数矩阵W。这种方式能够保持模型的原始性能，同时快速适应新任务。

c) 融合更新：在每次反向传播中，将低秩矩阵A和B的梯度更新融合到原始参数矩阵中，以提高模型的适应性。

LoRA 微调运算操作定义见表 51。

表 51 LoRA 微调运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
LoRA微调	通过低秩适配微调模型参数	Input	X	输入特征张量	Tensor
			lora_params	低秩适配参数	List of Tensor
		Output	Y	输出特征张量	Tensor
		Attributes	rank	必选，低秩适配的秩	Int
			task_type	必选，任务类型（分类/生成/多任务）	String

7.4.6.3 RAG 模型增强

7.4.6.3.1 概述

RAG (Retrieval-Augmented Generation) 是一种结合了检索和生成的技术，其被用于自然语言处理任务中。在处理长文档和需要丰富背景知识的任务时，RAG将文档检索与生成模型结合，以提高生成任务的性能。RAG 模型主要由两部分组成，检索器（Retriever）被用于检索大文档集合中的文档，生成器（Generator）被用于生成基于检索文档的输出文本。

RAG 的工作流程如下：

a) 查询输入（Query Input）：接受用户输入的查询；

b) 文档检索（Document Retrieval）：使用检索器从文档集合中找到与查询相关的文档；

c) 检索结果整合（Result Integration）：将检索到的文档与查询一起输入生成器；

d) 输出生成（Output Generation）：生成器基于输入生成最终的输出文本。

7.4.6.3.2 RAG 操作定义

Document Retrieval运算操作定义见表52。

表 52 Document Retrieval 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
document_retrieval	从文档集合中检索相关文档	Input	query	输入查询	string
			document_collection	文档集合	list[string]
		Output	retrieved_documents	检索到的文档列表	list[string]
		Attributes	retrieval_method	检索方法（如TF-IDF、BM25等）	string
			top_k	返回前K个相关文档	int

Document Encoding运算操作定义见表53。

表 53 Document Encoding 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
Document Encoding	对文档进行编码，转换为向量表示	Input	document	输入文档	string
		Output	document_vector	文档向量表示	Tensor
		Attributes	encoding_method	编码方法（如BERT等）	string

Query Encoding运算操作定义见表54。

表 54 Query Encoding 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
Query Encoding	对查询进行编码，转换为向量表示	Input	query	输入查询	string
		Output	query_vector	查询向量表示	Tensor
		Attributes	encoding_method	编码方法（如BERT等）	string

Similarity Computation运算操作定义见表55。

表 55 Similarity Computation 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
Similarity Computation	计算查询与文档之间的相似度	Input	query_vector	查询向量	Tensor
			document_vectors	文档向量集合	List of Tensor
		Output	similarity_scores	相似度分数	List of float
		Attributes	similarity_metric	相似度度量方法（如余弦相似度等）	string

Output Generation运算操作定义见表56。

表 56 Output Generation 运算操作定义

运算操作	描述	字段	关键字	定义	数据类型
Output Generation	基于检索到的文档生成输出文本	Input	query	输入查询	string
			retrieved_documents	检索到的文档列表	List of string
		Output	generated_output	生成的输出文本	string
		Attributes	generation_method	生成方法（如GPT、T5等）	string
			max_length	生成文本的最大长度	int

8 大规模预训练模型封装表示

8.1 概述

相比于传统百万量级参数量的模型，大规模预训练模型参数量往往超过10亿量级，因此对模型封装与传输有着更高的要求。本章介绍了模型封装的表示方法，模型结构定义规范与数据结构，模型封装 、加密流程，模型解封装、解密流程以及其他方面的内容。通过标准化封装与分发规范，便于模型从不同端进行传输、更新，从而进一步提高标准的使用范围与应用场景。

8.2 模型封装表示

模型封装表示结构图如图 1模型封装表示结构图所示。

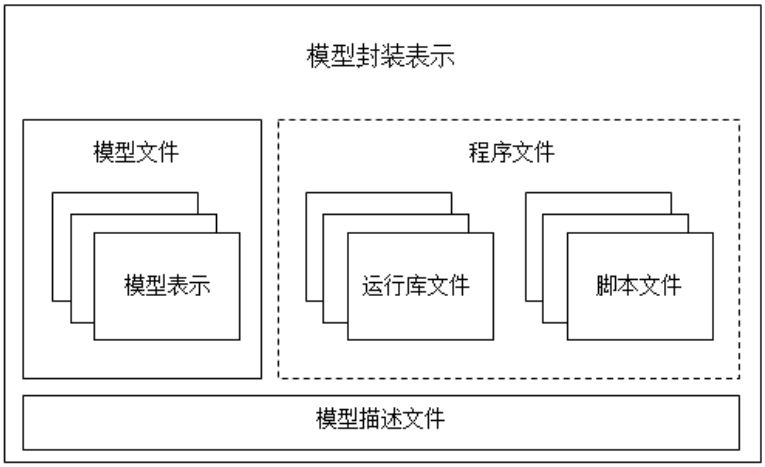

图 1 模型封装表示结构图

其中包括：

a) 模型封装表示：模型中的排布方式，封装信息，数据结构等说明定义；
b) 程序文件：第一部分为运行库，包括运行框架，运行环境所需要的程序文件；第二部分为运行脚本，包括与模型相关，输入数据获得运行结果的接口调用、模型推理程序脚本；
c) 模型描述文件：包括模型信息，模型属性以及其他的相关描述。

8.2.1 模型封装表示格式定义

封装模型文件应遵循以下的结构文件目录结构：

a) Model（模型目录）：存储封装表示的模型数据；

b) Program（模型程序）：存储算法、运行库、脚本文件等；

c) Meta-info（描述信息）：存储模型的属性信息、技术信息等。

整体目录结构见表57。

表 57 模型与文件路径结构

目录名	必要性	文件/子目录		描述
Model	必要	Model files（模型文件）		AI 模型文件
Program	可选	Library（库文件）		运行模型所需的库文件
		Script（脚本文件）		运行脚本文件，控制启动、停止、检查状态等
Meta-info	必要	按照模型标识符增加子目录	Management info	模型属性与管理信息文件
			Technical info	模型技术信息描述文件

8.2.2 模型数据结构定义

封装模型数据结构包括文件头信息（file header），模型头信息（model header），以及模型紧凑表达的数据信息（model data）。每个封装模型文件有一个单独的文件头，并且可以包含多个模型文件。模型头与对应的信息依次排列在后，具体信息见表58。

表 58 头信息与数据信息的排列

File Header	Model Header	Model Data	Model Header	Model Data

文件头的数据结构定义见表59。

表 59 文件头数据结构

字段	数据类型	值	定义
Start_code	unsigned int	0x5352434D	文件头起始码（SRCM）
Magic_number	unsigned int	0x47D02F93	文件头魔术数
Version	unsigned int	自定义	文件版本
Model_number	unsigned int	自定义	文件包含模型数量

模型头的定义见表60。

表 60 模型头数据结构

字段	数据类型	值	定义
Start_code	unsigned int	0x486F4D52	模型头起始码（HoMR）
Identifier	unsigned int	自定义	模型头标识符。当模型太大或有分段传输需求时，一个模型可以分为多个标识符相同的模型。
Check_sum	unsigned int	自定义	32 位 MD5 哈希校验和。
Residual updating identifier	unsigned int	自定义	如果本字段不为全0，则表明本模型使用残差更新。目标模型应标识符应与本字段相同。
Data size	unsigned int	自定义	模型的大小。

8.2.3 模型程序文件定义

模型程序文件包括两部分：即运行库（Library）和运行脚本（Scripts）。运行库主要包含了算法所依赖的运行环境（environment），根据场景、模型自编译的算法库文件（core），以及用于将统一模型表示转换为不同平台使用的转换器（converter）。模型程序文件定义见表61。

表 61 模型程序文件结构

	目录名	必要性	文件/子目录	描述
Program	Library	可选	environment	运行模型所需要的环境依赖库，如PyTorch, MXNet, numpy, pandas, Scikit-learn等
			core	一些模型所需要的预编译库，如数据预处理、可视化的库
			converter	将标准的模型表示转换为用于不同设备平台运行的工具
	Scripts	可选	Run.py	运行模型脚本
			Stop.py	停止模型脚本
			Status.py	检查模型状态脚本
			Convert.py	转换模型表达脚本
			[Else]	其他脚本文件

8.2.4 模型描述文件定义

模型描述文件用于描述模型的属性信息和技术信息等。其包括了属性信息文件（例如managementinfo.json）和技术信息文件（例如technicalinfo.json）。属性信息文件主要存储属性信息，其若干参数应按照表62。

表 62 属性信息文件

字段名	必要性	类型	描述
model_name	必要	String	模型的名称，如 ResNet101
model_size	必要	Dict	描述模型的规模，包括模型参数量，单位可选MB/GB；模型计算量FLOPs，单位可选MFLOP/GFLOPS。如{“params”: “10MB”, “FLOPSs”: “1GFLOPs”}
model_task	可选	String	描述模型任务，可参考表 64
model_metrics	可选	Dict	模型评价指标，如 precision, recall, F1-score,accuracy
model_license	可选	String	模型开源协议，如 GPL-2.0, MIT 等
model_date	可选	String	模型生产日期，如2024/3/20
algorithm_doc	可选	String	算法说明文档
scripts_description	可选	String	脚本描述文档

技术信息文件主要存储技术信息，其若干参数应按照表 63。

表 63 技术信息文件

字段名	必要性	类型	描述
model_version	必要	unsigned int	模型版本
data_type	必要	String	模型数据类型，如FP32、FP16、BF16等
model_requirement	必要	String	模型所需的硬件支持,如CPU/GPU,内存的需求.如: “GPU:typeA, typeB, GPUmem:8G,CPU:typeC,Mem:8G”
model_env	必要	String	运行模型的系统、程序等基础环境,如”Ubuntu18.04-PYTHON3.7-PyTorch1.7-Cuda9.0”.
model_inputs	必要	List	模型输入，详见表 65
model_outputs	必要	List	模型输出
model_config	可选	Dict	模型运行时的配置文件,用于修改 input and output path, threshold, output format setting 等设置
model_framework	可选	String	模型运行框架，如pytorch
mode_resolution_type	可选	unsigned int	模型输入分辨率类型,例如 0:固定分辨率 ,1: 动态分辨率等.
converter_confg	可选	Dict	模型转换器输入配置文件，详见表 66
PTM_info	可选	Dict	预训练大模型结构信息，详见表 67

表 64 模型任务

字段名	必要性	类型	描述
image_classification	可选	String	图片分类任务
image_segmentation			图片分割任务
image_style_transfer			图片风格迁移任务
ocr_recognition			OCR字符识别任务
object_detection			物体检测任务
translation			翻译任务
text_to_image			图像生成任务
other			其他任务

表 65 模型输入描述

字段名	必要性	类型	描述
input_type	必要	String	输入的数据类型（模态）如 image, text, video, speech, 等.
input_name	可选	String	输入变量名
input_requirement	可选	String	输入数据格式要求，如类型PNG, CSV and PNG, 输入大小限制，输入尺寸限制
model_mean	可选	List	模型输入数据的均值
model_scale	可选	List	模型输入数据的方差

表 66 转换器配置描述

字段名	必要性	类型	描述
multi-batch_type	可选	String	板端推理多batch类型 ,如 static-batch,dynamic-batch等
input_shape	可选	List	输入分辨率列表,例如: [[1,1,112,112]]
resize_parameters	可选	Dict	输入数据预处理参数,例如 resize_height,resize_wide等
output_node	可选	Dict	指定网络输出节点的名称
model-optimization-setting	可选	Dict	模型转换过程的优化参数配置，例如 bn fusion,yolo split, weight compress等.

表 67 预训练大模型信息表

字段名	必要性	类型	描述
architecture	可选	String	模型基础架构，如“llama”，“Qwen”,“mpt”，“falcon”
attention	可选	String	所使用的attention模块描述，如“FlashAttention”、“SparseAttention”、“MQA”等，多种技术可一并写入
pe	可选	String	位置编码方式，如Relative、Learned、RoPE等
max_input_length	可选	Unsigned int	最长输入长度
max_output_length	可选	Unsigned int	最长输出长度
blocks	可选	Unsigned int	模型block层数
embedding_length	可选	Unsigned int	隐藏状态数
expert_count	可选	Unsigned int	MoE模型专家数
expert_used_count	可选	Unsigned int	推理时MoE模型所使用专家数

8.2.5 模型封装表示的加解密

模型封装表示加密可以有效防止未授权的第三方获取、攻击、篡改模型。

模型封装表示的加解密应符合GB/T 42382.1-2023 中第11章模型保护的内容。

8.2.6 模型增量更新

模型增量更新（残差更新）是一个在边端和云端之间需要更新模型的过程。新模型在云端生成，然后分发到边端，用于模型传播以促进更好的边端部署应用。在边端生成深度学习模型还需要进一步传输和融合模型。对于相同的任务或相似的任务的多模型场景，通常这些场景都包含多个级别的相同/相似任务，这些任务的模型通常共享相同的架构。对于云端和边端共享部分架构的情况，只需对相同部分架构的权重增量进行残差更新。其他部分需要独立传输。对于这些相似的模型，不同模型之间的权重存在高度相关性。相比于原始权重，权重的残差部分分布更加集中，多样性更少，更方便进行低比特量化。因此，可以利用模型残差更新技术来消除信息传输之间的信息冗余。

更新和传输的过程包括以下六个步骤：

a) 训练基础模型权重A并将模型从云端传输到边端；

b) 根据实际需要重新训练和微调基础模型权重A，以获得目标模型权重B；

c) 计算基础模型权重A和目标模型权重B之间的差异，以获得模型残差权重Δ=B-A；

d) 对模型残差应用残差量化，以获得量化模型残差权重Q(Δ)；

e) 将量化模型残差权重Q(Δ)从云端传输到边端；

f) 边端将残差部分传输得到更新模型B=A+ Q(Δ)。

流程图如图3所示。

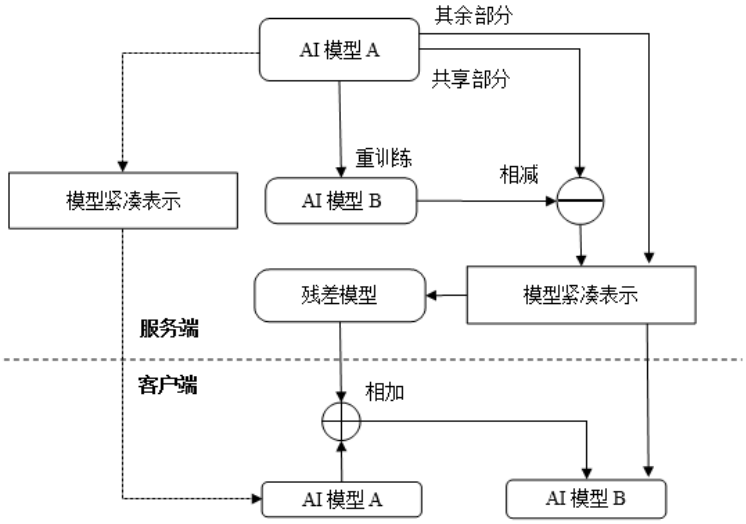

图 3 模型增量传输流程图

8.3 模型封装传输

8.3.1 模型封装分发流程

模型从服务端到客户端的分发流程应符合如下内容。

a) 分发流程的输入包括：

1) 模型表示/紧凑表示的文件；

2) 程序文件，依赖的运行库文件（可选）；

3) 模型描述文件。

b) 分发流程的输出包括用于传输的封装表示文件，具体流程如下：

1) 准备所需的输入文件；

2) 当文件需要残差更新时：

3) 使用残差更新流程；

4) 当模型需要加密时：

5) 使用加密流程；

6) 对模型封装文件头、模型头信息，并按照要求排列数据；

7) 组织其余文件；

8) 生成封装表示模型；

9) 将封装表示模型分发至不同设备。

完整的流程如图4所示。在模型的分发与传输中，需要基于相关的传输协议。当数据在从服务端传输到客户端（如从设备A传输到设备B）时，需要遵循上述的数据格式规范。对于控制和数据的接口，遵循表述性状态传递（Representational State Transfer，REST）的模式，并使用HTTPS的传输协议。

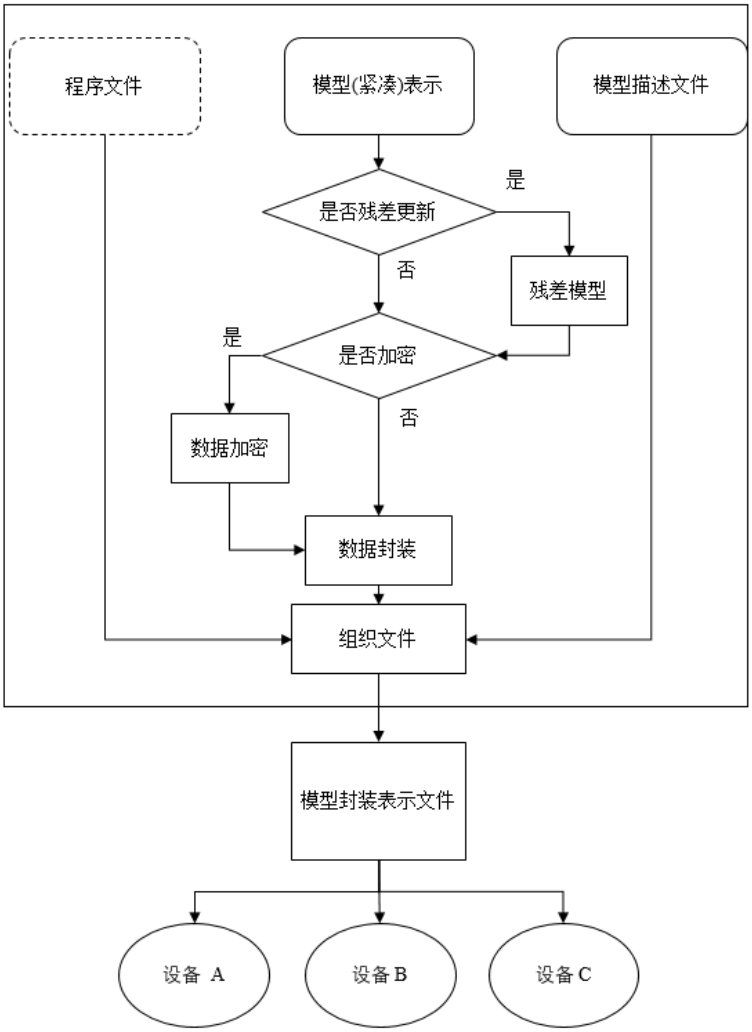

图 4 模型封装流程图

8.3.2 模型解封装流程

一旦客户端接收到完整的封装表示模型，就可以执行相应的解封装流程。流程的输入包括从服务端得到的模型封装表示。流程的输出包括模型文件，程序文件以及描述文件。设备端模型的解封装流程如下。

a) 准备输入文件；
——如果需要解密：使用相应的解密方法；
——如果是紧凑表示的模型：使用解压缩流程；
——如果需要进行模型转换：使用模型转换器进行模型转换。
b) 完成模型的解封装。

在端侧的模型，通过加载运行库和模型权重，做好运行推理的准备。当数据输入按照指定格式传入，经过前向计算，即可得到相应的模型输出。完整流程如图5所示。

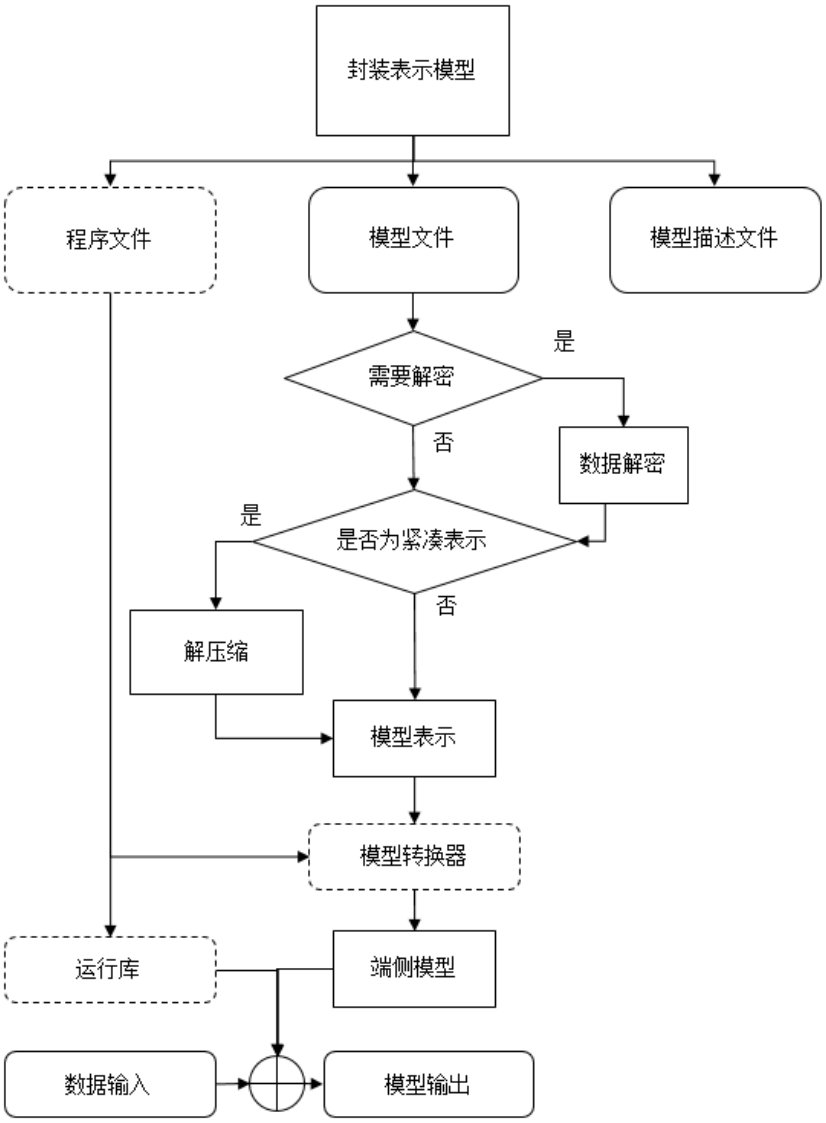

图 5 模型解封装流程

附 录 A
（资料性）
大规模预训练模型技术参考架构

A.1 原生预训练模型框架

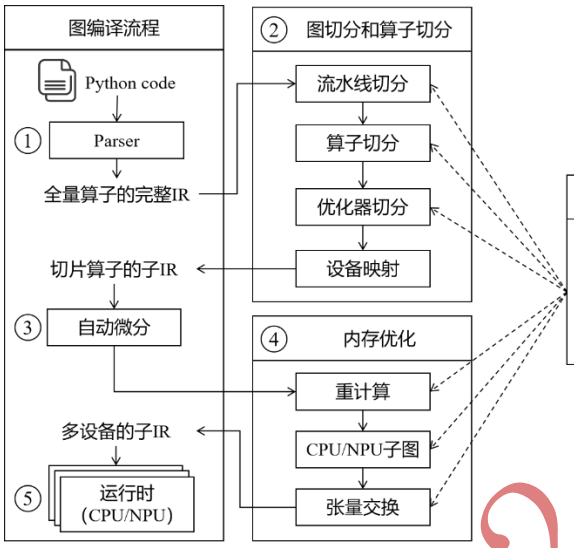

图 A.1 并行训练的图编译流程

框架在图编译阶段整合先进的大模型优化技术，为训练大模型提供更加完善的解决方案，如图A.1所示，神经网络模型可抽象为计算图，神经网络模型的并行训练可以抽象为把计算图切分到设备组成的拓扑图上。为实现通用性，需设计统一的中间层表示（IR, Intermediate Representation）承载多种并行策略及内存优化技术，进而在神经网络图编译期间将并行策略反映到针对IR的操作上。最终，框架使用和单机相同的模型代码，只需少量标注，就能实现并行训练。

A.2 预训练模型开发框架规范

A.2.1 流水线并行

简单地将模型切分到多设备上并不会带来性能的提升，因为模型的线性结构到时同一时刻只有一台设备在工作，而其它设备在等待，造成了资源的浪费。为了提升效率，流水线并行进一步将小批次(mini-batch)切分成更细粒度的微批次(micro-batch)，在微批次中采用流水线式的执行序，从而达到提升效率的目的，如图A.2所示。将小批次切分成4个微批次，4个微批次在4个组上执行形成流水线。微批次的梯度汇聚后用来更新参数，其中每台设备只存有并更新对应组的参数。其中白色序号代表微批次的索引。

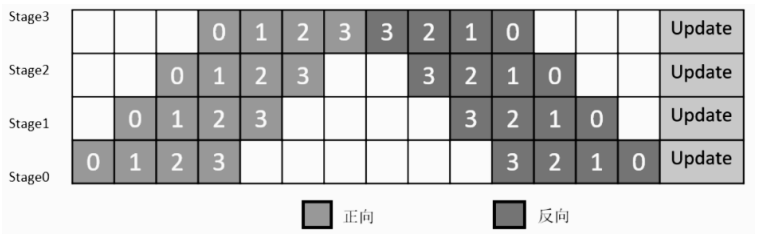

图 A.2 带流水线并行执行时间线示意图

框架应支持流水线并行，并对执行顺序进行了调整，来达到更优的内存管理。如图A.3所示，在第0 MicroBatch的正向执行完后立即执行其反向，以使第0 MicroBatch中间结果的内存得以更早地（相较于图A.2）释放，进而确保内存使用的峰值比图A.2的方式更低。

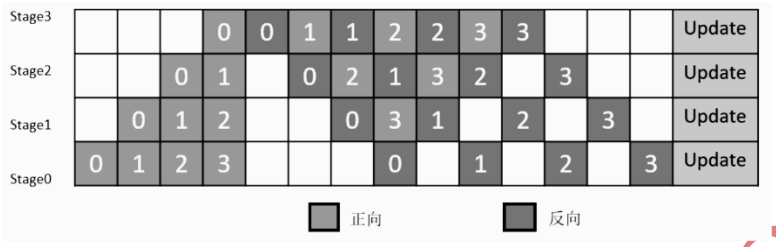

图 A.3 流水线并行执行时间线示意图

A.2.2 多专家并行

对不具备线性结构的模型，可用多专家并行或并行子图的切分方式。为了增大模型参数量，保持训练的计算需求基本不变，使用MoE (Mixture of Expert)，即将Transformer中Encoder的FeedForward Network (FFN)替换成MoE结构。如图A.4中的虚线框所示，经由Gate计算后，训练数据会独立地分发到一个或多个FFN中，而后FFN多个在计算完成后汇聚成最后结果。这里，FFN没有依赖关系，属于并行子图。

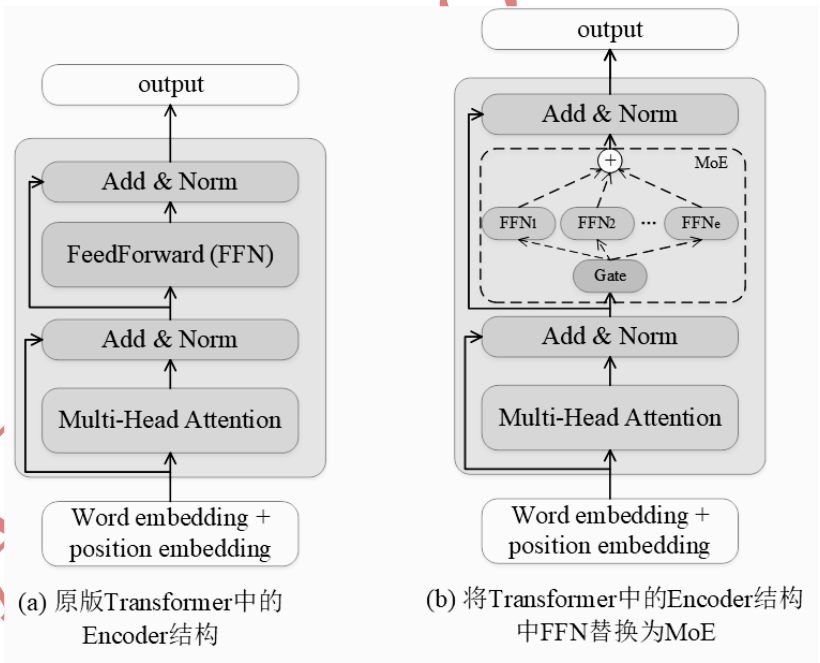

图 A.4 Transformer 中 Encoder 扩展为 MoE 结构

在该结构中，利用 Gate 函数将多个 FFN 分配到每台设备上，使每台设备获得一个或多个 FFN。最终，每台设备负责的参数量得以有效降低。若将训练数据经 Gate 计算后路由到本设备负责的 FFN，则直接传递给本设备的 FFN；若选择路由到其他设备的 FFN，则通过 AllToAll 通信，将训练数据发送至目标设备。同样地，经过 FFN 计算后，数据需要再次通过 AllToAll 通信，将相应结果路由回原设备。图A.5中展示了每个设备只有一个FFN的并行执行流程。

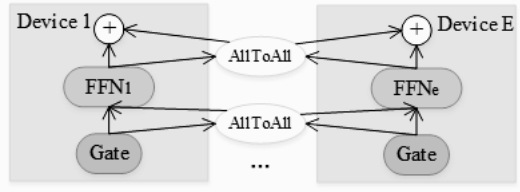

图 A.5 MoE 结构的一种并行方式

A.2.3 张量并行

张量并行中，有两个关键的建模，分布式张量分布（Tensor Layout）和张量重排布（Tensor Redistribution）。张量分布表示张量切分后，张量切片在集群分布情况。张量重排布表示两种张量分布之间的转换。模型切分流程如下图A.6所示，首先对每个算子的输入张量按策略进行切分，生成算子输入的张量分布，然后根据算子的数学定义，推导出输出张量分布；然后再检查前一个算子输出张量分布和下一个算子的输入张量分布，如果两种不同，则会插入张量重排布。

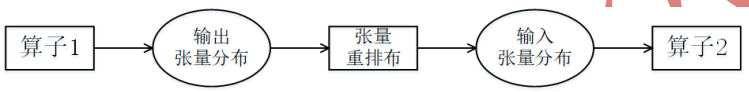

图 A.6 重排算子会转换张量分布

张量分布表示一个完整张量如何切分到集群。张量可以各维度切分到集群，也可以在集群上复制。张量重排布是不同张量分布之间的转换，张量在集群上从一种分布转成另外一种分布。在算子切分中，每个算子都是独立建模，前一个算子输出的张量分布和下一个算子输入的张量分布若不同，编译阶段会自动插入重排布来调整输入输出关系。重排操作借助于框架本身分布式通信原语，由“集合通信+split+concat”的算子组合。

A.2.4 优化器切分

优化器实现并行运算有两种方法，参数分组(Weights Grouping)和参数切分(Weights Sharding)。其中参数分组是将优化器内的参数及梯度做层间划分，训练流程如图A.7所示。将参数和梯度分组放到不同卡上更新，再通过通信广播操作在设备间共享更新后的权值。该方案的内存和性能收益取决于参数比例最大的group。当参数均匀划分时，理论上的正收益是N-1/N的优化器运行时间和动态内存，以及N-1/N的优化器状态参数内存大小，其中N表示设备数。而引入的负收益是共享网络权重时带来的通信时间。

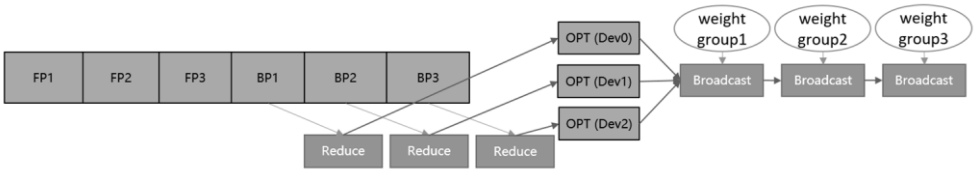

图 A.7 参数分组训练流程示意图

另一种实现方式参数切分是对参数做层内划分，即对每一个参数及梯度根据设备号取其对应切片，各自更新后再调用通信聚合操作在设备间共享参数。这种方案的优点是天然支持负载均衡，即每张卡上参数量和计算量一致，缺点是对参数形状有整除设备数要求。该方案的理论收益与参数分组一致，为了扩大优势，框架做了如下几点改进。如果对网络中的权重也做切分，可以进一步减少静态内存，但这就需要将迭代末尾的共享权重操作移动到下一轮迭代的正向启动前执行，保证进入正反向运算的依旧是

原始张量形状。此外，优化器并行运算带来的主要负收益是共享权重的通信时间，如果将其减少或隐藏，就可以带来性能上的提升。通信跨迭代执行可以通过对通信算子适当分组融合，将通信操作与正向网络交叠执行，从而尽可能隐藏通信耗时。通信耗时还与通信量有关，对于涉及混合精度的网络，使用fp16、bf16等通信，通信量相比fp32将减少一半。综合上述特点，参数切分的实现方案如图A.8所示。

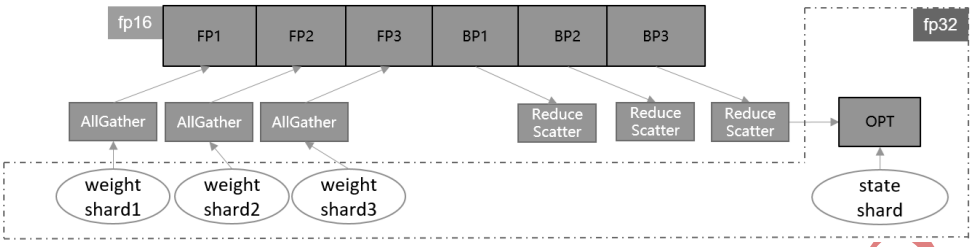

图 A.8 参数切分训练流程示意图

在实际网络训练的测试验证中，参数切分带来的内存收益是显著的。对于Adaptive Moment estimation (Adam)和Layer-wise Adaptive Moments optimizer for Batching training (LAMB)优化器训练网络，优化器自身的参数量和计算量不容忽视。经过参数分组，网络中的权重参数和优化器中的两份状态参数都减少了N-1/N倍，节省了静态内存空间。

A.2.5 重计算

框架应能根据正向图计算流程来自动推导出反向图，正向图和反向图一起构成了完整的计算图。在计算某些反向算子时，需要用到某些正向算子的计算结果，导致这些正向算子的计算结果，需要驻留在内存中直到这些反向算子计算完，它们所占的内存才会被其他算子复用。而这些正向算子的计算结果，长时间驻留在内存中，会推高计算的内存占用峰值，在大规模网络模型中尤为显著。

为了降低内存峰值，重计算技术可以不保存正向激活层的计算结果，让该内存可以被复用，然后在计算反向部分时，重新计算出正向激活层的结果。框架应提供了重计算的能力。

重计算功能具体实现为根据用户指定的需要做重计算的正向算子，复制出一份相同的算子，输出到反向算子上，并删除原正向算子与反向算子间的连边关系。另外，我们需要保证复制出来的算子，在计算相应的反向部分时才开始被计算，所以需要插入控制依赖去保证算子执行顺序。如下图所示：

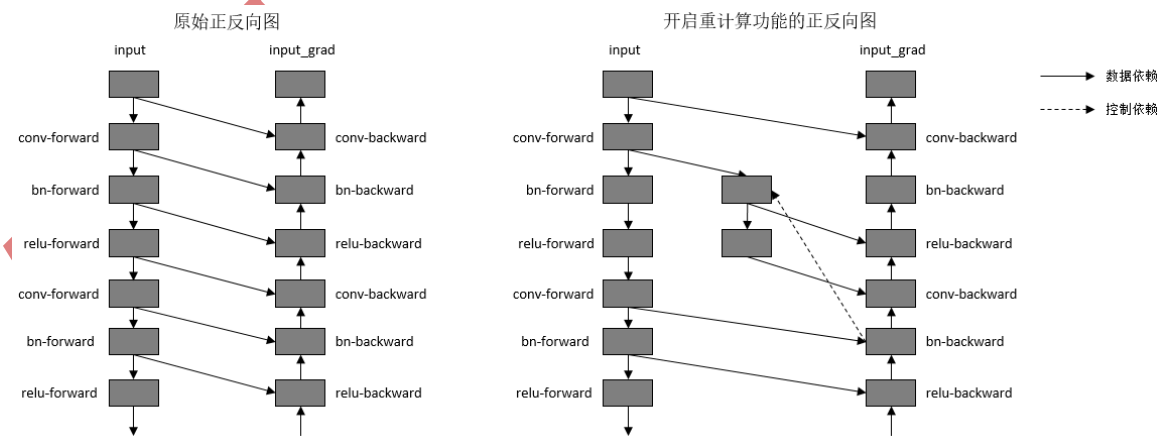

图 A.9 开启重计算功能前后的正反向示意图