公开

博士研究生学位论文

人工智能运行环境中动态问题的研究

学 位 级 别：工　学　博　士
专　　业：软　件　工　程
研 究 方 向：人工智能运行环境
研　究　生：孙 茹 君
指 导 教 师：陈 左 宁

二〇一九年四月

A Dissertation Submitted in partial fulfillment of the requirement for the degree of Doctor of Philosophy in Software Engineering

Research on Dynamic Problems in Artificial Intelligence Runtime Environment

Candidate : SUN Rujun

Supervisor : CHEN Zuoning

April, 2019

摘　要

人工智能已经成为各领域实现跨越式提升的重要关键技术。面对日益增长的数据量，复杂多变的应用、算法和模型，以及不断涌现的新硬件和新系统，人工智能运行环境是人工智能应用能够高效执行的关键。由于人工智能应用执行的内在特征以及用户的目标实时调整，其运行环境面临诸多动态问题：数据复杂多样充满变化，使数据的处理过程具有不确定性；模型在不断更新和调整，执行过程充满变化；硬件资源日益丰富，资源组织和运行状态充满变化。

本文的研究工作致力于探索并描述人工智能领域尤其是其运行环境的动态新需求，探索动态人工智能运行环境的设计方法，服务适应高性能计算系统与人工智能应用领域相结合的运行环境设计，探索新的运行环境架构从而指导未来人工智能高性能计算系统运行环境的设计。

本文的主要研究内容包括以下三个方面：

构建了全面可量化分析的动态人工智能运行环境模型，发现了动态问题的共性特征和作用机理。该模型由多维数据特征模型、多层次模型表示、多层次资源模型组成，可描述执行全流程，比较不同领域，量化动态特征，易于调度分析，通用性、实时性强。就实际条件下的各要素进行了量化评估和动态性分析，并对人工智能运行环境的实际需求进行了初步估计。

在构建模型的基础上，设计了基于“需求感知、动态响应、协同执行”执行闭环的动态人工智能运行环境执行架构，该架构改进了已有人工智能运行环境基于“定义——执行”的单程执行机制，从原理上提出了动态问题的应对方式。

在动态人工智能运行环境的架构下，选择数据、模型、资源三个方面的典型问题，围绕近似和随动等核心动态响应策略，分别研究了 4 项关键技术：针对人工智能应用中数据关联程度较高、数据量较大的关系型数据，提出了基于邻域的局部近似方法，首次对超大规模的数据进行了分布式实验，计算开销相比于全局方法减少了 1 个数量级；针对实际应用需求动态可变和模型可调整、可稀疏化的特征，提出了基于动态指导的模型稀疏化方法，在保持模型基本功能的同时，实现稀疏实时量化可调；针对系统级动态资源，将人工智能数据预处理过程抽象为具有数据流关系的动态模块组合，设计了自适应的协同预处理调度策略，在复杂大规模系统上完成 PB 级数据处理，节约了数万超级计算机核组机时；针对芯片级动态资源，采用基本算子构建计算流图，设计了基于人工智能计算流图的动态自适应调度模型，可支持复杂可变硬件条件下复杂模型的实时动态调度。

关键词：人工智能；运行环境；动态问题；近似；随动

Abstract

Artificial intelligence has become an important technology in various fields. With increasing data volumes, varied applications, algorithms and models, and emerging hardware and systems, how to execute applications efficiently is significant. A well designed runtime environment is the key to efficient execution. However, AI applications have inherent dynamic characteristics, as well as the changing user demands. For example, the data is complex and varied, whose processing process is uncertain; the model is constantly updated and adjusted, whose execution process is full of changes; the hardware resources are increasingly rich, whose organizing and executing status are full of changes.

This work is devoted to exploring and describing dynamic demands in artificial intelligence runtime, exploring the design method of the environment, and guiding the design of future artificial intelligence runtime environment.

In this thesis, we focused on the dynamic problems in artificial intelligence runtime environment. The contributions are as follows.

Aiming at the dynamic problems in artificial intelligence runtime environment, we quantitatively analyze the data, model and resource characteristics. We propose multi-dimensional data feature model, multi-level artificial intelligence model representation, multi-level resource model, and perform these models to realistic AI and conduct a quantitative assessment. Based on the above models, the causes of the dynamic problems are analyzed, and the problems are abstracted.

We design a dynamic artificial intelligence runtime architecture based on the closed loop, which can be described as demand sensing, dynamic response and collaborative execution. The architecture describes the working mechanism of the dynamic artificial intelligence runtime environment and the design principle of each unit. We describe the target matrix in the closed loop, as well as the goal assurance.

With the core strategies of approximation and follow-up, the following four aspects are studied in the dynamic artificial intelligence runtime architecture. We propose a neighborhood-based approximation method for relational dataset, which has high data association and large amount of data. The method reduces the computational cost compared to the global method by an order of magnitude. We propose a dynamic-guidance-based model sparseization method to response to realistic application requirements and adjustable sparse features of models. It can perform real-time sparseization response while maintaining basic abality of a model. We propose a cooperative data preprocessing

strategy of AI applications which is abstracted by a dynamic module-flow and system-level dynamic resource model. It saves tens of thousands of machine-time of HPC computing resources. We propose a dynamic adaptive scheduling model which is based on computation-unit-flow and chip-level dynamic resources model. It can dynamicly response to requirements both from applications and systems.

Keywords: artificial intelligence; runtime environment; dynamic problems; approximate; servomechanism;

目 录

第 1 章　绪论

人工智能图像识别的准确程度超越了人类[1]，人工智能围棋[2] 和德州扑克[3] 在对弈中战胜了人类选手，人工智能在机器阅读理解测试也超越了人类[4]，人工智能领域在部分应用获得惊人的突破之后迎来了新一轮的发展，成果斐然，涌现出一大批新算法。

人工智能已成为多领域实现跨越式发展的重要技术手段，而人工智能的发展离不开大规模数据的积累和计算能力的提升。运行环境需要抽象计算资源与上层应用，为大规模的数据、复杂的算法模型和体系结构交流融合提供基础，并进行有效的管理调度，实现系统的高效运行。纵然一个特定的运行环境设计可以满足一种特定的人工智能应用，也会取得令人满意的结果，但若希望运行环境能够满足更多领域且不断发展的人工智能应用，就需要良好的运行环境设计。

“数据、算法、算力”成为众多人工智能研究者认可的人工智能驱动力，在本文中以数据、模型、资源作为主要表述，它们也是人工智能运行环境中重要的元素。人工智能运行环境中的各个要素并不是静态和固化的：数据复杂多样充满变化，使数据的处理过程具有不确定性；模型在不断更新和调整，执行过程充满变化；硬件资源日益丰富，资源组织和运行状态充满变化。这些因素都导致了人工智能运行环境具有动态特征。这些动态特征不仅与人工智能本身的内在特征有关，还与应用的执行模式有关；不仅发生在模型构建阶段，也发生在深度神经网络的推理、训练或者其他人工智能算法模型执行的各个阶段；不仅与人工智能的动态应用需求相关，还与包括软硬件在内的系统的实时运行状态相关。随着人工智能应用的发展，我们将会面对更加庞大的数据、更加复杂的模型或计算模式，以及更加庞大和复杂的硬件环境，这些变化将使人工智能的动态问题不可忽视。

随着动态问题的凸显，人工智能运行环境出现了诸多的不适应。目前人工智能运行环境没有在原生设计中考虑到动态问题，尤其是训练场景下，因此只能针对具体的应用对组件做动态化改造，没有全面的解决方案。人工智能的动态问题会不仅使应用本身在静态环境中执行效率低下，还有可能导致越来越多的新型动态模型和算法难以高效描述。随着人工智能应用的发展，人的作用也收到重视，“人在环路中”成为许多新方法的共同特征，人的动态指导更需要运行环境的动态能力。

1.1　研究挑战与机遇

人工智能应用数据规模越来越大，我们面临的问题规模更大，更复杂。例如常规的人工智能视觉训练数据达到了 GB 甚至 TB 级[5]，关系型数据达到了 PB 级[6]。大规模数据的全局处理需要极大的开销，因此近似估计成为重要需求。

人工智能算法主要以模型作为表现形式，其规模越来越大，模型中模块的种类和数

量越来越多。自然语言神经网络中的模型参数规模达到了十亿级[7]；图卷积网络[8] 中以点为模块的数量达到了数十万。随着数据的积累和模型的发展，未来的数据量和模型规模还可能继续增长。

复杂应用可能需要更复杂的计算模型，可能需要拆解成多种能力和技术来解决问题。例如目标检测需要融合图像识别和位置检测，文本摘要需要融合语言建模和精细化训练等。很多应用本身具有动态属性，其动态特征出现在预处理和模型执行等多个阶段。例如, 动态深度学习 (Dynamic Deep Learning) 中，网络可以根据输入来变形，也可以根据输入改变大小。神经模块网络 (Neural Module Network)[9] 由多个模块化的网络组成，需要根据输入数据实时决定构成网络的模块单元和组合方式。因此需要根据其动态特征作出适当的动态调度。即使在传统的人工智能模型训练过程中，为了达到计算速度、收敛速度、训练精度等多方面的平衡，可能会在流程中引入自适应的调整策略，例如动态调整批大小、学习率等，这些调整策略往往伴随着对资源需求的变化。而在更大规模的应用和资源的双重约束下，如何充分利用应用的执行特征以及资源的动态状态，进行适当的调整，是大规模人工智能应用高效执行需要重点解决的问题。

人工智能应用的执行需要更大的数据规模和计算需求、更大的计算资源或更长的运行时间。例如图像识别的训练可能会耗费 226 GPU 小时的时间[10]，自然语言处理的语料建模甚至会耗费 16 个 TPU 集群 4 天的时间[4]。训练是人工智能应用体系的重要组成部分，也占据了大量的资源。为了得到更好的模型，并在计算过程中充分利用硬件资源，训练技术的发展也呈现出动态性的趋势。如何利用资源的动态特征模型构建和执行提供基于资源的指导，并在目标约束条件下合理调整模型是值得研究的问题。随着问题规模的增加，运行人工智能应用的系统朝着更大规模，更高效能的方向发展。

复杂计算系统具有多层次、异构的特征，并朝着云端融合的方向发展。系统运行过程中的动态性不仅来自于应用本身状态的变化，也来自于用户需求、系统状态等变化。

计算能力的增长伴随着多层次计算能力的提升，从多核、多片，到多节点、多数据中心。代表高性能计算机最高能力的 Top500[11] 中，前 10 名的机器核心数均超过十万量级，半数达到百万量级，甚至神威太湖之光达到了千万量级。而在数据中心、尤其是云数据中心，这一量级变的更高。越来越多的应用，尤其是人工智能应用支持多计算单元的并行。其中最典型的例子是获得 2018 年戈登贝尔奖的两个人工智能应用：气象图片的识别[12] 和基因组关联研究[13]。

体系结构另一个发展趋势是异构。在摩尔定律已接近极限的条件下，“功率墙”[14] 也越来越严重。异构计算提供了一种较为高效的解决方法，在采用传统多核、众核的同时，加入一些加速的核，后者能够维持低功耗下较低频率的计算，对于峰值计算速度和整体计算速度的提升都发挥了重要的作用。多核众核芯片、FPGA、Intel MIC 等结构都在计算环境中成为了计算单元的重要组成部分或者传统 CPU 的重要补充。随着 GPU 的出现和广泛应用，越来越多的人工智能专用结构和专用芯片应运而生，计算环境的异构性也越来越强。仿生网络、脉动阵列处理器、模糊电路等一系列专用结构应用在了新型体系结构设计中，面向特定领域的人工智能芯片已呈爆发态势，例如 TPU 系列、Diannao

系列、NPU，以及各厂商、各研究机构即将推出的数不胜数的人工智能芯片。计算中心/数据中心广泛容纳多种体系结构，广泛拥有 CPU、GPU 等处理器，而集成度较高的超级计算机更是多采用集群或大规模并行处理 (MPP) 的结构。体系结构的异构性多层次地体现在核间、片上、节点上以及集群中。如何高效利用异构的体系结构，成为包括人工智能应用在内的所有应用的重要挑战。越来越多的计算在云环境上完成，人工智能也不例外，其训练部分几乎都需要依靠云计算。在动态可伸缩的云资源上，经过适应性调整的应用可以在满足基本计算要求的同时实现资源的集约和效率的提升。

在体系结构日益进步的今天，计算存储通信等各方面均有所发展，且不同应用对于各种计算资源的需求也有所差异。在应用与体系结构综合设计中，越来越强调资源均衡设计。而如何满足不同应用模式条件下的均衡设计，就需要动态能力。

解决人工智能运行环境中的动态问题，需要科学描述人工智能运行环境，表示其中的动态特征，并构建合理的运行环境。对于动态数据，全局数据分析的时间代价往往是不可接受的；对于动态模型，静态的模型执行方法难以满足动态的需求；对于动态资源，固定的调度方法难以提供最优调度策略，甚至有时难以满足基本需求。人工智能的类脑本质——近似和随动可以为动态问题提供应对策略。设计良好的运行环境能有效应对人工智能本身的动态特征，使新型人工智能系统的效益最大化。

1.2 研究内容和目标

本文研究人工智能运行环境中的动态问题，需要首先研究人工智能运行环境的组成和特点，分析其中有哪些动态问题，这些动态问题如何表示，进而能够量化地分析人工智能运行环境。

面对人工智能运行环境中的动态问题，需要研究什么样的运行方式更加适合，明确运行环境的运行机制。在现有的条件下研究所设计的运行环境中的关键技术，主要研究各要素中有哪些亟待解决的动态问题，该如何应对。

本文的研究工作致力于探索并描述人工智能领域尤其是其运行环境的动态新需求，探索动态人工智能运行环境的设计方法，服务适应高性能计算系统与人工智能应用领域相结合的运行环境设计，探索新的运行环境架构从而指导未来人工智能高性能计算系统运行环境的设计。

1.3 本文结构

本章讨论本文研究的背景、意义、内容和主要贡献。

第 2 章介绍了全文所讨论的各研究内容的背景和相关工作。

第 3 章主要分析人工智能运行环境的构成，其中有哪些动态问题，如何针对动态人工智能运行环境进行建模，分别就人工智能运行环境中的数据、模型、资源构造适宜描述动态特征的模型。并讨论了这些模型下如何描述人工智能应用的运行。

第 4 章从整体的角度讨论了动态人工智能运行环境的架构，并设计了基于“需求感

知、动态响应、协同执行”闭环的运行机制。就运行机制中各个流程的具体设计和要素进行了分析。

第 5 章从细节和具体应用如何实现的角度讨论了第四章所设计的动态人工智能运行环境架构中的关键技术。关键技术的选择涵盖了运行环境的各个要素——数据、模型、资源调度；以及应用执行中预处理、模型执行等各个阶段。

第 6 章对全文工作进行了总结，并对未来工作进行了展望。

1.4 主要贡献

1. 构建了全面可量化分析的动态人工智能运行环境模型，发现了动态问题的共性特征和作用机理。该运行环境模型包括“人工智能多维数据特征模型”、“人工智能多层次模型表示”和“人工智能多层次资源特征模型”。通过人工智能多维数据特征模型内多层次的维度、规模、时序性等参数，量化分析了人工智能领域不同类别的数据特征，指出了人工智能领域的数据动态性体现在高维稀疏数据不均衡分布引发的执行动态和数据的生成、转化操作引发的执行动态等特征。人工智能多层次模型表示在以基本算子组成的计算流图这种主要模型表示方法的基础之上，综合考虑人工智能模型在执行阶段的所有层次，包括外层执行方式、基本功能模块组合图、内层计算流图，量化分析了每个层次的参数、计算需求、执行方法、规模等特征，得出人工智能模型的动态性体现在不同层次执行时的方法、参数、控制流、数据流等特征变化的结论。人工智能多层次资源特征模型对系统、芯片两级资源进行了特征分析和针对人工智能应用的性能建模，指出资源动态体现在执行时的状态变化以及应用对资源的需求变化中。

2. 设计了基于“需求感知、动态响应、协同执行”闭环的动态人工智能运行环境架构，并在架构设计中针对需求感知全面分析了人工智能应用执行的多方面目标，形成目标矩阵；针对动态响应，分析了动态响应机制；针对协同执行分析了模型表示到硬件资源的映射方式。该架构改进了已有人工智能运行环境基于“定义——执行”的单程执行机制，从原理上提出了动态问题的应对方式。

3. 在动态人工智能运行环境的架构下，选择数据、模型、资源三个方面的典型问题，围绕近似和随动等核心动态响应策略，分别研究了以下关键技术：针对人工智能应用中数据关联程度较高、数据量较大的关系型数据，提出了基于邻域的局部近似方法，该方法计算开销相比于全局方法减少了 1 个数量级；针对实际应用需求动态可变和模型可调整、可稀疏化的特征，提出了基于动态指导的模型稀疏化方法，在保持模型基本功能的同时实现稀疏实时量化可调；针对系统级动态资源，将人工智能数据预处理过程抽象为具有数据流关系的动态模块组合，设计了自适应的协同预处理调度策略，在复杂大规模系统上与其他服务一起协同使用计算环境，56 小时完成 400TB 压缩数据的预处理，节约了数万超级计算机核组机时；针对芯片级动态资源，采用基本算子构建计算流图，设计了基于人工智能计算流图的动态自适应调度模型，可支持复杂可变硬件条件下复杂模型的实时动态调度。

第 2 章　相关工作

人工智能运行环境包括数据、模型和资源，本章将首先从这三个方面分别介绍与运行环境有关的研究和动态性的研究。人工智能运行环境中多层次的软件是整合数据、模型到资源的主要形式，本章将介绍包括库、框架等人工智能多层次的软件环境。针对人工智能运行环境中的动态问题，本文研究了近似计算、模型稀疏化、预处理、优化与调度 4 项关键技术，其相关工作将在本章介绍。

2.1　人工智能运行环境建模及动态性研究

2.1.1　数据研究

人工智能近期的爆发式发展在很大程度上依赖于大规模数据集的建立，百万、千万 (条) 乃至更大规模量级的数据出现在语音、文本、图像、视频等诸多领域，数据的规模几乎决定了该人工智能具体应用的计算等需求。基因装配 (Metagenome Assembly) 的数据规模达到了 2.6TB[15]，气象分析的数据规模达到了 3.5TB[12]，更有图数据的规模达到 136.9TB[6]。以上是经过处理的标注数据，原始数据的规模更大，对于这些数据进行预处理的难度也更大，例如上述图数据的原始规模达到了 PB 量级。

数据集的类别和特征是由人工智能的应用种类所决定的，例如：针对关系挖掘领域，斯坦福大学提供了 SNAP[16] 关系数据集；针对图像识别领域，由斯坦福大学发起的 ImageNet[5] 数据集是最主要的应用数据；针对自然语言处理领域，Facebook 等提供了 SQuAD 问答数据集[17-18]。数据集的数量、类别越来越多，谷歌试图囊括所有的数据集，推出了数据集搜索工具[19]。

对于动态数据的研究，包括对领域动态数据的处理，例如针对时序图数据[20] 研究了分析方法与分析框架，针对视频这类动态数据[21] 研究了如何通过深度学习寻找同一个目标，Theriault[22] 研究了动作识别。此外还有针对动态数据的系统操作方式的研究，如数据库中的动态数据增删修改[23] 等。

目前对于人工智能应用所涉及的数据缺乏统一的描述和研究，与数据相关的零星研究多停留在具体应用子领域中，以提供数据集为主要方法。即使是数据集搜索工具，也只提供针对每个数据集的概要描述。人工智能的数据特征研究局限于应用领域的内部，例如特征提取、表示学习 (representation learning) 等技术，缺少全局性的描述。对于动态性的研究，没有融入数据建模中，也没有体系化地分析动态性原因。

2.1.2　模型研究

模型描述了基本的人工智能算法，是分析其计算模式的重要入手点。AlexNet[24] 是用以图像识别的深度卷积神经网络，包含卷积层、池化层、全连接层等，最初在 2 个 GPU

上完成计算。VGG[25] 在 AlexNet 的基础上，加深了卷积网络的层数。随后又出现了更深层的网络，如 MSRANet。GoogLeNet[26] 引入 inception 模块增强了卷积的功能，通过大量的小模块增加网络的宽度和深度，同时保持较低的计算量。ResNet[27] 用残差网络综合了加深的网络和增强的卷积模块。R-CNN(Region-based Convolutional Network)[28] 在图像识别的基础上增加了物体检测，解决定位和识别两个问题，整个流程包括位置块选取，训练卷积神经网络，训练 SVM 和训练回归参数。SPP Net[29] 改进了 R-CNN 计算量过大的缺点，只对原图计算一次卷积，采用空间金字塔池化 (SPP) 的方法，可将任意大小的图像转换成固定大小的特征向量来识别。Fast R-CNN[30] 综合了前两者的优点，并提高了训练的速度。Faster R-CNN[31] 将定位和识别融合成区域生成网络 (RPN) 层，提高了检测的速度。SSD[32]/YOLO[33] 等模型将目标检测整体看作是回归问题进行求解，直接一步得到目标位置及其类别，实现了目标的实时检测。LSTM(Long short-term memory) 单元是循环神经网络的一种重要结构单元，适合处理时序数据，通过几种“门”来控制神经元的状态，由 LSTM 或其变体构成的编码器解码器等模型在自然语言理解领域发挥重要的作用。

随着应用领域越来越广泛，人工智能应用本身越来越复杂，最初简单的卷积神经网络等单一模型逐渐显现出其缺点，真实场景下模型的构成已经不单单是一个简单的卷积神经网络或者类似规模的结构了。很多应用借用了典型模型的部分或者全部结构，根据场景和意图，搭建出更复杂的模型。一个典型的例子是生成对抗网络 (Generative Adversarial Networks, GAN)[34]，利用两个卷积神经网络相互配合训练，得到一个产生以假乱真数据的网络。在自然语言理解领域，模块的构建还依赖于语义的提取，是一个动态的过程。随着人工智能的发展，元学习、多模型协同、生成查询网络、图网络等一系列更有效的智能方法被提出。

人工智能的模型表示主要方式是计算流图，包括 prototxt(caffe[35]) 等计算与参数分离的方式，或 ONNX[36] 计算与参数结合的方式，计算流图也可以在框架中通过 API 搭建。计算流图的主要组成单元是卷积等基本算子，后者也是大部分人工智能框架的 API，计算流图描述了最终生成模型的结构。人工智能模型的训练还包括迭代过程，目前模型训练的迭代往往由人工智能框架内置实现，用户通过 API 调用几种迭代方式选项之一来进行训练。

已有的模型动态性研究多为动态模型构建，动态深度学习 (Dynamic Deep Learning) 中，网络可以根据输入来变形，也可以根据输入改变大小。例如图片问答 (VQA) 领域的动态模型 Neural Module Network[9] 由多个模块化的网络组成，需要根据输入数据实时决定构成网络的模块单元和组合方式。以自动机器学习为代表的模型建立和优化方法，泛化了机器学习的能力，使计算过程中出现了更多数量、更多样的模型，且随着任务的进行，模型会演化和进化，从而导致了模型的动态性。由多个人工智能模型组成的模型森林，在执行过程中不同部分也会出现动态的特征。

然而，计算流图本身并不能完全描述人工智能模型的执行过程，尤其是模型训练中的迭代过程，因此使用计算流图作为模型表示与拆分的依据具有局限性。关于动态性的

研究往往局限在应用本身或构建的模型本身，运行环境缺乏应用、资源和动态需求的协同设计。

2.1.3 硬件资源研究

人工智能应用无论在终端还是云端都有大量的运行需求。系统级的环境包括以 Summit、神威太湖之光为代表的超级计算机，以阿里云、亚马逊云服务 (AWS) 为代表的云计算环境，以及 CPU 服务器、GPU 服务器等小规模计算设备。从系统的角度来看，硬件资源主要包括计算资源、存储资源、网络资源等。

对于系统级环境的模型主要依赖于系统级性能评测指标，例如评价超级计算机计算性能的 Linpack[37] 性能指标，侧重评价访存和网络性能的 Graph500[38] 性能指标，评价系统对于人工智能支持能力的 Deep500[39] 性能指标。这些指标重在评价系统执行特定种类任务的能力，同一指标的对比可以在一定程度上反应系统之间的性能差别。

系统级资源动态性的研究包括亚马逊云环境中进行图片识别的动态训练[27]，可以在不同的训练阶段选择不同的计算资源量，同时保持原有的训练精度，这在总体上节约了训练资源。除此之外，云上的虚拟化构建也是一种动态的应对方法。

芯片是主要的计算单元，其蓬勃发展也是近年来人工智能取得突破式发展的重要保障。运行人工智能应用的芯片级资源包含以 CPU 为代表的通用计算芯片，以 GPU、神威系列为代表的领域通用计算芯片，以 TPU、Cambricon 和其他 ASIC 设计为代表的专用芯片，还有可重构硬件 FPGA 等。随着应用的发展和领域应用研究的深入，人工智能领域内部的专用硬件设计越来越多，例如针对自动驾驶的芯片 Nvidia TegraX1、Tesla FSD 等。

为了适应于不同的人工智能计算，处理器中出现了越来越多的特殊部件。例如：乘法、加法树，PE 组，主要代表是 Diannao 系列[40]，采用一系列的乘法单元、加法树、激活单元直接流水，组成适合神经网络计算的结构；张量部件，主要代表是 NVidia GPU 中的 Tensor Core，一拍可以完成 4*4 矩阵 A, B, C 的乘加操作 $A \times B + C$；脉动阵列，主要代表是 Google TPU[41]，由同构的 PE 组成二维矩阵乘法单元 (256*256 的脉动阵列)，相邻 PE 会以特定的步骤进行数据传输。数据首先经过恰当的排布，然后权值由上到下流动，数据由左到右流动，最后得到一定形式的卷积。

申威 26010(sw26010)[42] 是由我国自主研制的众核处理器，具有 260 个计算核心，这些计算核心组成了 4 个核组，每个核组包括 1 个主核与 64 个从核，从核以 8*8 的形式排列，支持行列寄存器通信。芯片频率为 1.45GHz，双精度浮点峰值计算性能约 3TFlops，片上拥有 32GB 内存，可支持核组分别使用内存或共享内存，每个从核核心支持 64KB 缓存。

芯片级资源上的动态性也有研究,例如寒武纪设计了针对稀疏化的硬件结构 Cambricon-X[43]，支持稠密计算方式转化为固定稀疏程度的计算。

目前针对硬件资源大都针对特定的系统和硬件结构，缺少整体上的模型和建模，缺少对计算、访存等资源细粒度的量化建模。对于资源动态特征，缺乏建模和全流程协同

设计。

2.2 人工智能运行环境框架研究

2.2.1 超大规模编程模型

超大规模编程模型是满足大规模应用在大规模系统上运行的重要工具，本节从编程模型发展的角度，讨论几种超大规模编程模型。Chapel[44] 是一个通用并行编程语言，最初为 Cray 超级计算机所设计，兼顾传统 HPC 语言的高效性和面向对象的编程语言的易学和面向对象性。X10 是 IBM 推出的编程模型，服务于多种系统，最初作为高效编程模型，现在发展成为生态系统。X10 编程模型基于异步划分的全局地址空间模型（Asynchronous Partitioned Global Address Space Model），该模型的全局地址空间划分为一个一个的槽，各线程异步执行。X10 的编程语言类似于 C/C++ 和 Fortran 的原生代码，还支持 JVM、Scala。Cilk 最初由 MIT 开发，是一个多线程的运行时系统，Intel 在此基础上开发了 Cilk Plus。该编程模型是 C/C++ 的扩展，在 icc 编译器上实现，通过 API（包括 cilk_spawn, cilk_sync, cilk_for 等）调用其功能。它支持基于任务偷取（work steal）的负载均衡，此外还支持并行描述、可扩展的内存管理（Intel TBB）、任务调度（调度器负责）等功能。Intel 并行编程工具中有两种模型：线程基本块 TBB 和向量基本块 ArBB。前者支持单节点多核编程，其底层组件包括任务调度器、线程局部存储、同步源语、内存分配、线程等，其高层应用领域包括支持一般的并行算法（与 C++ 标准库中算法范围一致）、支持并发容器（与 C++ 标准库中的容器范围一致）。后者主要支持向量化功能，与其他编程模型中的 SIMD 机制一致。统一并行 C(UPC) 是 2006 年提出的 C 标准下的并行编程模型，该模型将标准 C（ISO C99）扩展为一个分区全局地址空间（Partioned Global Address Space，PGAS）语言。该模型支持明确的并行执行模型、共享的地址空间、同步源语和内存一致性模型、并行基础库。加州大学伯克利分校和劳伦斯实验室开发了此编程模型的实现版。Charm++ 是加州大学伊利诺伊分校提出的大规模并行编程模型，基于 C++，采用面向对象的方式实现。其基本思想是任务的极细粒度分解。它包含通信组件、负载均衡策略和线程模型，其上的应用以科学计算为主（例如 NAMD），数量较少。在此基础上该学校提出了 AMPI 模型 (Adaptive MPI)，该模型基于 MPI 标准，添加了对处理器的虚拟化，多个处理器可以在一个实际的物理处理器上。OpenCL 和 cuda 都是面向异构处理器的编程模型，二者都是通过扩展 C 的 API 来实现。OpenCL 更通用，支持数据并行和任务并行；cuda 主要面向 NVidia GPU，支持数据并行。

综上，Chapel 的特点是用户友好；X10 的特点是形成了一个生态系统；Cilk Plus 和 Intel 并行编程工具的特点是面向特性结构进行了多领域的整合；UPC 虽构成了实际标准，但仍缺乏推广和应用领域；AMPI 和 Charm++ 的思想先进，但不愠不火；OpenCL 和 cuda 主要应用领域仍停留在 GPU，但 OpenCL 已经显现出对更广泛异构架构的支持，而鉴于 NVidia GPU 的广泛使用和人工智能应用对其的依赖，cuda 和其上的 cuDNN

等库也蓬勃发展。

2.2.2 人工智能编程模型和运行环境

人工智能领域主要使用了前述超大规模编程模型中基于基本数据结构并行化的函数库，例如 Intel TBB 和 ArBB 以及 OpenCL 和 cuda 等，侧重于基本数学库和专用计算库，具有专用体系结构的支持。更高层次的编程模型以计算流图[35] 为主，依赖于人工智能运行框架中通过 API 整合调用基本数据结构 (如张量) 和基本算子 (如卷积等基本计算单元) 实现。其具体的方式将在后文 2.3.4分析。

已有的人工智能运行环境包括数学库、人工智能专用计算库、人工智能框架等。以 BLAS 为代表的高性能数学库可以在通用硬件环境上提供部分人工智能计算的高效实现，以 cuDNN[45]、swDNN[46] 为代表的专用结构上的专用计算库有效利用了硬件特点提高了计算效率，为人工智能计算的抽象提供了基础。以 Halide[47] 为代表的领域专用语言，以及 prototxt、ONNX[36] 等人工智能模型表示方法提供了与硬件无关的抽象和调度的理论基础。以 TVM[48]、Glow[49] 为代表的人工智能编译环境，在编译优化技术之外还提供了面向特定硬件环境的调度优化接口。以 TensorFlow[50]、PyTorch[51]、Caffe[35] 为代表的人工智能框架，综合利用了从底层基础库到上层运行环境的支撑系统，涵盖了人工智能应用执行的全生命周期，提供了描述应用和模型的能力，提供了它们在不同体系结构上的运行机制与优化手段。此外，Chainer[52] 等一批新框架可以在每次执行时重新定义计算流图，探了索动态计算流图的执行。第 2.3.4(c) 节将作具体分析。

目前针对人工智能的研究更侧重于上层应用的拓展、新模型的构建以及专用硬件结构的设计；针对人工智能运行环境的研究也更注重模型的表示能力和执行方式，以单节点静态环境为主，对于动态问题研究较少，缺乏人工智能应用中“人在环路中”的设计。这些运行环境的执行架构多为“定义－执行”的单程结构，缺少对动态特征和动态需求的应对模式。

2.3 人工智能运行环境关键技术研究

本文研究了人工智能运行环境中的近似计算、模型稀疏化、预处理、优化与调度 4 项关键技术。它们的相关工作如下：

2.3.1 近似计算技术

数据规模的增长使计算过程中的全数据吞吐变得代价昂贵，采样和近似计算技术是分析超大规模数据的简单方法。对于一般的非关系数据，采样方法广泛应用。蒙特卡洛方法就是一种采样技术，按照一定概率分布获取样本。此外还有反向采样 (Inverse Sampling)、拒绝采样 (Rejective Sampling) 以及重要性采样 (Importance Sampling)。例如文章[53] 分析了神经网络计算对于重要的样本所花的计算时间非常有限，并提出了一种依据重要性对样本进行采样的方法。对于更复杂的关系型数据，其分布不能用简单的模型描述，所以简单的采样方法也难以近似。对于这类数据的度量往往采用中心性度量

的方法，而其分布的描述也基于“关系”和中心性度量值。

人工智能模型也是一种复杂的网络 (关系型数据)，其中心度度量有多种方法，这些方法可以分析不同方面的重要程度。中介中心度 (Betweenness Centrality) 基于最短路径，而距离中心度 (Closeness Centrality) 基于平均距离。因为中心度的计算是 NP 完全问题[54]，所以当数据规模扩大时，很多算法的可扩展性受到了挑战。即使算法可以扩展，由于数据量的增大，我们也很难在短时间内获得一个完整的中心性度量结果。随机游走算法给我们提供了一种简单的根据网络结构进行中心度度量的方法，一个典型的例子是 PageRank[55]，类似的还有 TrustRank 和 SimRank。TrustRank 通过添加信任种子并通过偏向的 PageRank 算法来对抗垃圾邮件。SimRank 可以通过关联关系添加相似性特征，可以应用在学术引用关系数据中。Gao[56] 分析了关联数据排序的方法，可以应用于一种关系构成的图、多种关系构成的关系系列、以及具有丰富元数据的关系数据。Welch[57] 引入了用于社交图谱排名的语义信息。然而，以上方法都需要获得全局数据，但全局数据会导致规模庞大的数据计算量巨大甚至不可接受。

针对以上问题，近年来出现了对于关系数据的近似计算。Pfeffer[58] 综合了中介中心度和距离中心度，提出了基于局部数据的 k-中心逼近方法。Riondato[59] 通过随机抽样及其分析提出了所有节点之间中心度的近似算法。Bressan[60] 证明了关系数据中的局部信息非常重要。因此，我们可以通过仅访问全局数据中的一小部分来求得中心度度量值。Bressan 在另一篇文章[61] 给出了近似 PageRank 的次线性复杂性证明。随后有一些实验性的研究进行了一些平行实验[62]。但是对于极大的数据量来说，“局部”区域仍然太大，且不恰当的数据分布可能仍然会使计算需要全局信息。

采样技术对于解决大规模数据的近似计算十分重要。为了获得近似均匀分布的采样数据，基于随机游走的采样是有效的方法[63]。这类算法包括基于重新加权的随机游走[64]，Metropolis-Hastings[65] 和最大度随机游走[66]。对于近似计算的研究较为分散，数据的采样与数据本身密切相关。但是大多数研究的重点都关注于采样和近似本身，对于不同类别的数据，尤其是关系型数据的研究较少。此外，采样和近似计算与后续分析的关系也需要进一步分析。

2.3.2 模型稀疏化技术

在模型动态性研究中，模型稀疏化是重要的研究方向，稀疏化会在一定程度上导致模型的动态性。最初模型稀疏化研究的原因是为了减少模型的参数，随着应用的复杂和目标的提高，越来越多的极深网络开始出现，例如最初针对图像识别设计的 VGG[25]、针对语音识别的多语言极深卷积网络[67]、针对人脸识别应用的 DeepID3 网络[68]、文字识别的序列化深度卷积网络 SVDCNN[69]。这些深层的网络参数量极大，计算量极大，对于应用设计和运行来说都是挑战。网络稀疏化是一种有效的模型简化方法：通过减少神经元之间的连接来减少参数、计算甚至通信。

目前研究模型稀疏化的多用于模型压缩，以期在推理时使用更小的模型。Thom[70] 研究了基于稀疏表示的快速分类、多层感知器的稀疏连接和在线编码器的稀疏化，与传

统的非稀疏模型相比，稀疏性可以作为正则化器，并且可以获得明显更好的分类结果；训练分类器的计算复杂度可降低大约一个数量级。针对专用领域的特定稀疏神经网络研究繁多：卷积神经网络的稀疏化可以通过在卷积层之前添加 bias 层对参数初步处理，使后续的卷积核参数降低 90%，且针对 ILSRVR2012 的实验证明对于精确度的影响少于 1%[71]；循环神经网络的稀疏化采用一般的预训练、剪枝、再训练的方法，百度的实验[72]可将网络参数量减少到 1/8 且几乎不影响精确度；语音识别和增强领域的深度神经网络的稀疏化，可以通过整流线性单元 (ReLU) 实现，在隐藏层之间建立线性连接，Xu[73]证明在对稀疏网络进行修剪和重新训练之后，可以在不降低性能的情况下大大减少计算和存储需求。在计算资源受限的 FPGA 上，通过 L1 正则化在卷积层预处理数据不仅可以提高系统的准确性，而且还可以进一步减少后续全连接层的计算量[74]。此外还有自动编码器的稀疏化[75-76]，循环神经网络的稀疏化[77] 等。

目前对于稀疏性的研究大多停留在应用层，除了专门应对稀疏的硬件外，对于体系结构与稀疏性的关系研究较少。且针对稀疏性的研究较为宽泛，大多停留在有无问题上，其量化分析较少、细粒度分析更少。因此，大多数稀疏性的研究所提供的动态方法策略停留在开关层面，几乎没有动态实时精确调度的能力。

2.3.3 预处理技术

人工智能应用与海量的数据密切相关。无论是训练过程中需要的大量带标签的输入数据，还是在计算过程中因势利导、实时产生的多样化数据，都需要经过适当的处理满足一定的标准规范后进入计算过程。如果没有高质量的数据，就难以得到高质量的训练结果。传统的预处理遵循一定的计算规则就可以完成，例如变形、重新排布等。但随着数据量的极大增加和应用的复杂多样化，预处理已难以简单融合进计算，更难以被忽略。面对海量数据，预处理过程的不确定性因素增加，预处理的任务也更加多元。预处理的过程强烈依赖于数据本身的特征，而这些特征大部分需要在预处理的计算过程中同步获得。

数据预处理作为全计算流程中的重要一环不可忽视。在一般的数据挖掘中，预处理涵盖的范围较广：数据集的基本统计和分析、数据的准备、缺失数据的补全、噪声数据的处理、数据的简化、特征的选择、数据的选择、数据的去中心化都是其重要组成部分[78]。在图像识别的训练过程中，对于图片数据的常见预处理过程包括归一化 (normalize) 和白化 (whiten)。归一化能够通过减少内部协变量的转换有效提升后续训练速度[79]。白化能够减少输入的冗余，使特征更相关[80]。其他方法还包括二值化、缩放等。在批量训练时，对于训练数据的乱序化 (shuffle) 能够加快收敛[81]。Pittman 等人通过对于一系列 DNN 同时训练时在 CPU 中预处理通信过程的优化，使其节约训练时间和能耗[82]。

数据预处理离不开计算环境的支撑。超大规模的计算需要更高性能的计算环境。在数据量日益增长的今天，大数据与高性能计算的关系日益被挖掘[83], 二者之间越来越耦合, 也互相借鉴。威斯康星大学 HTCondor 课题组提出了高吞吐计算 (High Throughput Computing) 概念[84]。高吞吐计算关注数据在长时间范围内“吞吐量”, 而非传统 HPC 关

注的短时间的“计算量 (计算能力)”。Kalmegh[85] 分析了 HPC 系统数据库海量关系型数据面临的挑战。俄亥俄州立大学的 Islam 提出了利用 key-value 存储, 将 HDFS 移植在 HPC 的常用存储系统 Lustre 上[86-87], 提高了 IO 效率。日本理化研究所的数据同化技术 (Data Assimilation) 能够将实时数据与 HPC 深度结合[88-89], 利用“京”超级计算机的高性能计算能力, 每 30 秒将雷达捕获的天气数据和数值模拟计算进行同步, 提供实时局部天气的预测。

然而，目前对于数据预处理的研究与计算过程的研究分离，预处理对计算资源的需求更是没有与全流程的需求和处理相结合。数据与处理的多样性与动态性需要对计算资源的动态利用。此外，大部分预处理操作具有吞吐量高、计算密度相对较低的特征。但很多计算系统并不是专为数据密集型应用设计, 尤其是人工智能系统，其后续的训练需要大量的计算资源进行高密度的计算。将预处理直接移植到运行系统中往往会遇到一系列问题。

2.3.4 优化与调度技术

(a) 执行优化技术

模型优化技术能有效提高执行时的效率和资源利用率。随着人工智能框架的成熟，这一技术广泛应用在人工智能软件环境中。模型优化有两个层次：体系结构无关的优化和体系结构相关的优化，通过由高层到低层的一系列中间表示实现。从模型到特定体系结构的指令需要一整套的中间表示。

体系结构无关的中间表示常在框架中被描述为高层次的中间表示 (High Level IR)，其优化主要是针对计算流图的优化，融合技术是其中的关键。人工智能中的融合 (fusion) 技术体现在多个方面。数据的融合需求包括人脸融合、特征融合、立体声融合、传感器融合等，数据的融合与应用强烈相关。模型融合是指综合多个模型的优势形成一个新的模型，例如多模型学习[90] 和异构模型融合[91]。模型融合也依赖于应用的约束。模型中的算子融合，将相同数据流前后两段计算合并，能够在执行过程中降低对于不同资源的需求，例如减少访存、简化计算等，从而提高执行效率。在运行时领域，算子融合与体系结构息息相关。TVM[48] 中利用操作融合技术，可以自动优化计算图，将连续的多个操作融合成一个，从而减少了中间结果缓存的开销。Glow[49] 的优化技术也在中间表示上完成。XLA[92] 将计算流图优化并转化成线性代数基本算子。

体系结构相关的中间表示在框架中被描述为低层次的中间表示 (Low Level IR)，包含对应体系结构的基础指令、内存分布、通信描述等一系列表示。体系结构相关的优化的主要研究思路是从基本算子的专有实现入手，针对每一种特定的硬件结构，以线性代数或者人工智能基本运算单元为粒度，优化实现。例如 BLAS、LAPACK、eigen 等都针对不同特定平台有所设计和优化。更典型的例子是针对 GPU 的基本算子库 cuDNN[45], 针对 Intel 处理器的基本数学库 MKL[93], 针对神威处理器的基本人工智能算子库 swDNN[94]。框架对于体系结构相关的优化往往依赖于底层优化库，也有针对硬件结构做代码生成的方式 (如 TVM[48] 针对 GPU 的 cuda 代码生成和针对 CPU 的 llvm 代码生成)。

高层优化依赖于人工智能模型的表示能力，以计算流图作为主要模型表示的方法仅仅能描述一个层次的计算，对于迭代等不同层次描述有限，更缺少时间域上动态描述的能力，因此综合的优化能力有限。低层优化依赖于特定体系结构上算子的具体实现，对于不断涌现的硬件结构缺少统一的应对方案。但已有自动优化 (auto tuning)[95-97] 技术为遍历更广泛的优化空间提供方案，其在人工智能领域的应用也越来越受到重视。

已有的表示方式需要层层下沉 (lowering) 直到一个具体的硬件上的指令，缺少全系统全硬件的协同优化。更多的情况下，基于中间表示的优化手段缺少分布式的支持，而后者只在部分框架中得到了部分实现。

(b) 模型并行化技术

随着数据和模型规模的增大和终端对于人工智能需求的增加，单节点的计算无论是时间开销还是内存开销已经难以满足很多应用的需求。除了体系结构的创新，并行化成为了重要的应对方法。

Tal Ben-Nun[98] 详细分析了深度神经网络的多种并行方式。按照计算环境总体上分为共享内存的并行方式和分布式并行方式。前者包括数据并行、模型并行、流水线并行和混合并行。后者包括基于模型一致性 (Model Consistency) 的分布式并行、基于参数分布 (Parameter Distribution) 的分布式并行和基于分布式训练 (Distributed Training) 的分布式并行。

混合并行可以融合数据、模型并行与流水线并行的优点。例如 2014 年实现的 AlexNet[10] 中，卷积层用数据并行，全连接层用模型并行。2017 年实现的 AMPNet[99] 中，中间表示和控制流用到了模型并行，而异步执行用到了流水线并行。2012 年实现的 DistBelief[100] 实现了多种并行方式的融合，能支持数十亿参数规模和数万 CPU 的环境。

利用参数服务器[101] 可以完成中心化的参数更新，而利用 AllReduce 通信可以完成非中心化的参数更新。多模型融合的计算天然适应于分布式的环境，其关键是利用通信机制保证参数的一致性。以模型结构搜索为主要操作的自动机器学习由于依赖性弱，可并行性较强。

目前的并行化实现多关注于一个模型、一个应用，且计算资源也相对固定。其计算资源往往局限于广泛使用的 CPU、GPU 等架构，对于新出现的硬件结构应对不足。框架的并行能力要落后于单一应用的并行能力。人工智能的分布式和并行研究大多关注并行本身，随着应用的复杂和动态需求的增加，并行和调度的结合越来越重要。

(c) 动态调度技术

人工智能的方法算法逐渐发展，具有动态性的技术越来越重要。集成学习 (Ensemble Learning) 技术使多种方法的成果能综合到一个任务中，提高学习效果，在分类、预测、函数近似等方面应用广泛。而多种方法的同时使用，在一定程度上引起了不均衡的问题，从而导致了动态选择和动态调度的需求。

除了模型层面的动态之外，数据层面的动态也不容忽视。早期的深度学习在数据投入模型之前经过了“预处理”的过程，适合批处理，而随着预处理和数据本身越来越复杂，难以规整化的数据增多，由不规则数据引发的动态性的计算开始出现且越来越重要。

人工智能应用执行动态性体现在运行和调度上。随着模型的复杂和应用目标的多元化，“编译后运行”的执行方式已经难以满足新型人工智能应用的计算过程，尤其是复杂应用的训练过程，“实时运行”成为了新趋势。而“实时运行”的实现就需要运行时系统的动态支持：整合应用的实时需求，并提供计算资源环境的实时状态。

一批动态的框架应运而生。Chainer[52] 是第一个灵活动态的框架，所有的层都用 python 实现，可以在训练时实时构建计算图，用户因此可以根据每次迭代或者每个样本的条件更改计算图，定义了 define by run 的新型框架运行方式。DyNet[102] 能够动态创建计算图，为每个样例 (或批样例) 创建完全不同的图，可以在训练中动态调整神经网络的拓扑结构。PyTorch[51] 借鉴了 Chainer 的思想。TensorFlow Fold[103] 提出了动态批处理的新技术在静态框架上封装动态接口。

针对特定体系结构组成的集群，任务的迁移与调度也是一个重要的研究方面。例如微软针对 GPU 集群设计的 Gandiva 深度学习调度系统[104] 就是一次有意义的尝试。

已有的动态支持主要关注“动态模型”，在表现形式上是人工智能框架支持动态计算图。动态计算图的应用场景主要局限在自然语言处理领域，因此，其构建和执行主要面向小规模计算图和单节点的计算环境。如何利用体系结构的特征，在保证动态性的同时高效执行，如何应对包括其他领域模型在内的更加复杂庞大的动态模型，都是值得关注的问题。

2.4 小结

本章介绍了人工智能运行环境各组成部分、人工智能运行环境框架、以及动态运行环境中关键技术的相关研究工作。人工智能运行环境中对于数据的研究分散于具体的应用领域，缺乏统一的描述和研究，没有将数据动态型引入数据建模中，或体系化分析动态原因；对模型的研究体现在不断涌现的新算法、新模型上，模型建模方式以计算流图为主，难以全面描述执行过程；对于资源的研究体现在针对人工智能领域的新结构上，缺少应用对于资源的细粒度量化建模；动态的数据、模型、资源缺乏协同设计。人工智能运行环境包括数学库、人工智能专用计算库、人工智能框架等，目前的运行环境框架侧重于上层应用的扩展、新模型的构建等，研究更注重模型的表示能力和执行能力，对动态问题研究较少。在几项关键技术的相关工作中，近似技术的主要实现方式是利用局部数据推测全局信息和使用采样技术；模型稀疏化技术主要用于推理时的模型压缩，缺乏对体系结构和稀疏性的关系研究，缺少量化研究，动态研究主要侧重于有无，缺乏精确调度能力；预处理过程一般与计算过程分离，缺乏全流程协同设计；优化技术主要包括以编译优化为主的计算流图优化和操作优化，包括数据并行与模型并行等多种方式的模型并行化，以及面向“动态模型”的动态调度技术，缺乏实时的动态能力和面向体系结构适应力的动态能力。

第 3 章　动态人工智能运行环境建模

研究人工智能运行环境，首先需要分析环境中的各个要素。运行一个典型的人工智能应用，需要有数据、资源，其次是应用需求。数据和资源都是直接要素，应用需求一般是得到具有描述数据本质特征并对未来数据进行判断的模型或者算法。人工智能模型不能凭空作为输出，它需要一定的约束和设置，我们能学习到的往往只是模型的参数或者在约束条件下的模型结构，因此模型本身也是运行环境的要素。

前文中已经介绍过数据、模型、资源三个方面的相关工作，本章将从这三个方面对动态人工智能运行环境建模。

3.1　动态数据

以深度神经网络为代表的人工智能模型的成功，很大程度上依赖于大规模数据集的建立。一个好的数据集对于模型构建和运行至关重要。随着 ImageNet 标注数据集出现，和基于此数据集的人工智能模型接近甚至超越人类的图片识别率，各应用领域数据集不断涌现，以这些数据作为输入的计算会有相应的特点。数据特征的分析能够描述计算、预测计算，从而指导计算设计。大多数人工智能应用和算法都采用了“数据流计算模型”，以数据作为计算模块之间的衔接，数据在整个计算的不同流程中会呈现不同的特征，这些特征往往与模型或算法的设计密切相关。人工智能领域的数据是否有不同于其他领域数据的特征？不同子领域的数据究竟有什么区别？这些数据特征对于计算又会有怎样的影响？

3.1.1　数据特征建模

我们提出*多维数据特征模型*来描述人工智能应用中的数据，如图 3.1，特征中维度属性需要多维、多层次的参数来描述。

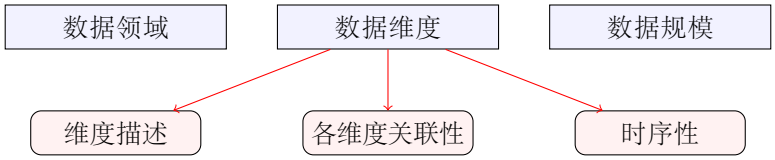


图 3.1　多维数据特征模型

公开数据集已经越来越丰富和多样，表 3.1中列举了一些典型的公开数据集，并按本节的特征模型进行了分析。典型数据集来自于不同的应用，有图像识别、语言分析、物体检测、自然语言理解、推荐和排序系统等。形式上有图片、音频、视频、语言文字等。随着人工智能应用领域的发展，数据集已经难以枚举完全，数据集与网络内容类似，

也成为复杂多样的互联网中一粟。未来可能会向领域化、专门应用的方向发展。若想一窥全貌，也许只能依赖数据集搜索。

(a) **维度与维度关联**

常规的数据集大多为低维数据。例如大部分公开数据集中的图像数据为 2 维，宇宙或空间的图像数据为 3 维，语音或部分文字数据为 1 维，关系型数据多为 2 维。

除了空间维度外，属性维度为数据添加了更多的信息，可能使数据在表示方法上增加 1 维。例如图像中的 RGB 三色维度将每一张图片由长乘宽 $w * h$ 扩展为长乘宽乘通道数 3, 即 $w * h * c$，宇宙空间数据可能有 16 个通道[105]。

然而在计算过程中，属性维度数据有多种处理方法。

1. 不同通道的数据分别进行计算，通道之间的数据几乎没有直接交互，例如 RGB3 通道的图片数据的不同通道在卷积神经网络 AlexNet 的第一层分别计算。
2. 属性维度的数据作为数据的附属品，不参与计算过程，只做最终呈现。例如一些条件下的名称属性。
3. 属性数据作为一个整体向量，直接参与计算过程。例如，用于推荐计算的商品购买信息、商品评价等。

此外，大批量的数据同时处理，会在计算的过程中形成“批 (batch)”这个维度。例如深度神经网络的训练过程中，输入数据 (图片) 可以以批的形式，一次计算 32 张 (一批)。时序特征也会形成维度 (如语音)，在很多数据中是首要维度。

时序性 时序性是一种特殊的维度，可以表现在数据中，也可以表现为单独的时序特征。数据的时序性可能来源于时间本身，也可能来源于先后顺序。视频、语音中的时序特征来源于时间本身。句子、文章中词语的先后顺序也可以作为时序特征的一种，成为人工智能模型的时序输入。还有气象、生物和医学等领域的间隔采样，也形成了时序性的数据。时序性的数据常用于趋势预测，通过历史数据来预测未来。时序性的数据的分析为机器学习领域带来了更多模型，例如 LSTM、GRU 等。

在数据特征中，维度之间可能会有关联关系，这就形成了*维度关联*，例如：关系型数据中 2 个维度有关联关系；多条目样本数据中样本数量与样本类别两个维度有关联关系。具有关联关系维度的规模与样本空间大小在一定程度上反映了数据的高维分布，部分数据集可能会有高维分布稀疏的特征。

(b) **数据领域类别与规模**

大规模数据集是近年来人工智能尤其是深度学习得以迅速发展的关键。数据集作为应用的基本输入，其规模在一定程度上反映了计算需求。附录 A中列举了人工智能领域多类别典型数据集的信息。

图像识别是人工智能领域具有代表性的应用，其主要目标是分类。因此，图像识别数据的主要模式是一系列的图片和类别标签，该领域的典型数据集如表 A.2。从中可以看出，标注的图片样本大都经过了预处理，规格远小于拍摄图片 (如 4000*3000 pixel)。用于训练的图片数据，需要标注类别，上述数据集中的类别最大在十万量级。若需要估

表 3.1　典型数据的多维数据特征模型

数据领域	典型数据集	维度描述	维度关联性	时序性	数据规模参数	实际规模
图像识别	imagenet[5]	样本数量 样本类别 图片 RGB 通道	数量 - 类别	-	15M 22K 224*224 3	1.2TB
图像识别	MegaFace[106]	样本数量 样本类别 图片 RGB 通道	数量 - 类别	-	4.7M 690K 2 维图片 3	910GB [1] 159GB [2]
语言建模	WikiText 103[107]	样本 (文章) 数量 短语 单词	数量 - 类别	序列	28K 总类别 103M 总类别 268K	181MB
语言问答	SQuAD 2.0[17-18]	样本 (问答) 数量 单词	无	序列	100M -	44MB
关系分析	Clueweb[108]	边数量 点 (src, dst)	边 - 点	-	42.57B 978.4M	500GB
关系分析	Sogouweb[6]	边数量 点 (src, dst)	边 - 点	-	12.25T 271.9B	400TB [3]
视频识别	HACS[109]	样本数量 样本类别 时序 图片 通道数	数量 - 类别	有	1.55M 200 2 秒 * 帧数 视频图片大小 3	15.4MB
位置检测	Open Images Google[110]	样本数量 图片 位置 类别 通道	位置 - 类别	-	1.9M 2 维图片 总位置 15.4M 总类别 600 3	561GB
决策游戏	AlphaGo Zero[2]	棋谱数量 出手数量 棋盘 落子与否	-	-	29M - 181*181 3	生成数据
决策游戏	冷扑大师[3]	样本数量 牌面信息	-	-	约 10^{160} 个决策点	生成数据

K: kilo, M: million, B: billion, T: trillion

1：图片为原始大小，其中中包含人脸

2：图片为裁减至人脸大小

3：压缩数据大小，解压后在 PB 级

计自然界物体类别，可根据生物分类学[111-112] 中界、门、纲、目、科、属、种不同粒度

的数量来推断，其他类别也可以通过对应领域分类学中的类别数来推断。与规模最相关的参数为样本数量，受限于标注成本，目前已知的最大规模在千万到亿。此外，目前标注的类别远未达到分类学中的类别数量，且与图像识别的具体领域有关，例如人脸识别 (Labeled Faces, MegaFace) 只需要区分不同的人。图像识别数据集的典型大小为 GB 级别，最大可至 TB 级。

自然语言处理领域任务类别较多、数据种类较多。该领域的典型数据集如表 A.1。一般的理解任务基于文本，典型的数据类型包括以文本为表现形式的文章素材、以句子对为表现形式的翻译素材、以问答句对为表现形式的语言理解素材。由于初始的问答数据需要人工回答来产生，因此数量有限，典型的问答数据规模在 MB 量级。但是文本、图片等语料信息规模庞大。Common Crawl 几乎包含了互联网上的所有抓取信息，达到了 PB 数量级。但是原始信息难以直接利用，需要经过一系列的预处理构造成恰当的表示。1 Billion Word Model 是目前较大规模的语料库，只包含文本信息，此类数据的规模在 GB 级。

关系型数据 (关系分析) 的主要特征是数据之间具有关联关系，这种关系可以用图表示。除了一般意义的图数据、社交网络数据、网页链接数据之外，还包括商品推荐数据、文本索引数据等。该领域的典型数据集如表 A.3。该类数据的主要维度是关系，次要维度包括数据属性特征等。为了得到数据之间的关系，原始数据需要经过预处理，如排序编号等。表 A.3列举了典型的数据集。根据数据类别，数据集的典型规模从十亿条边到千万亿条边不等，仅包含图结构，数据的规模最大能到 TB 甚至 PB 量级。近来，为了将关系型数据转化为与深度神经网络输入类似的条目型数据，出现了一系列方法。如基于随机游走的 Node2Vec[113]，将图中点上基于全局的关系近似为点的独立特征，并用向量描述。此外，基于图的数据也逐渐作为图神经网络的基本输入[114-115]，后者是人工智能领域的又一重要应用。

人工智能应用领域的数据种类多样，远非图像、语音、关系型数据能完全涵盖。其他数据集如表 A.4。经过标注的大规模数据集典型数据条目为百万到千万，以图像识别数据集最为典型。领域专用数据集的数据条目数量较少，如北京理工大学车辆数据集 BIT-Vehicle 中只有近 1 万张图片，但其规模也在不断增长中。随着传感器的增加和数据的不断积累，各领域的数据都将越来越多。

除了收集数据之外，还有很多应用中的数据是自动生成的。例如自然语言理解中有部分问题可以自动生成；游戏类应用 (如围棋) 的对弈可以由机器互博产生 (AlphaGo Zero)；关系型数据也可以用 Kronecker 生成器[116] 通过矩阵运算生成。

(c) 多维数据特征模型的优势

多维数据特征将模型在以类别归纳数据的基础上，添加了维度关联和时序属性，能够较全面地描述数据维度的关联和时序特征。相比于特定领域的数据描述方式或单一数据描述，多维数据特征能够通过关联维度的数值比较实现跨领域分析，预测或比较可能带来的计算关联的强度。同时，对于数据量的分析可以不局限于样本数量而着眼于样本的多种维度特征，更有利于后续计算的分析，尤其是动态性分析。

表 3.2　多维数据特征模型与其他数据特征模型的比较

描述方式	领域内比较	领域间比较	关联度分析	关联度比较	时序比较
多维数据特征模型	✓	✓	✓	✓	✓
领域数据集合 (如 SNAP[16])	✓	×	✓	×	✓
单一数据描述 (如 ImageNet[5])	×	×	×	×	×

3.1.2　数据动态性分析

即使人工智能数据集越来越多，涵盖的领域越来越广泛，将数据输入模型仍然不是一个简单的流程。它不仅需要每条数据的规格与模型需求严丝合缝，还可能需要数据在整体上按照某些分布、某些顺序输入，以求训练或执行中的好处。

对数据的要求越高，数据达到要求的难度越大。于是就出现了一系列处理数据的需求，而这些需求在一定程度上导致了数据的动态性。本节将讨论人工智能数据 (主要指输入数据) 的动态特征。

(a) **与维度属性有关的动态特征**

同一维度属性特征对齐处理导致执行动态性　经过预处理的数据会出现稀疏特征从而导致动态性。数据的来源不一、格式多样，预处理包括映射和格式调整等操作。映射的典型对象是经过计数的数据，例如将类别映射到计数的数据编码。格式调整的典型对象是具有时序特征的语音数据、文本语法数据和语言数据，例如为了将数据对齐输入特定的模型结构，需要对长短不一的原始数据加 0 补齐，在句对翻译数据中模型往往设定一个较长的标准作为输入规模，没有达到输入规模标准的输入数据就需要补 0 对齐后再进入模型执行。

由数据时序性特征导致的执行动态性　执行阶段数据生成和演化会导致数据动态。动态生成广泛应用在具有规则描述的游戏或其他具有数据生成模块的应用中。例如自动游戏对弈、GAN 模型，给数据添加噪声或变换从而增加数据量等。对于演进数据，计算过程中也要实时考虑数据的变化过程。

与维度属性有关的动态性特征可以通过维度属性本身的描述来表示，例如某维度是经过格式调整的对齐数据、某维度是时序性参数等。

高维稀疏数据导致计算过程中的动态执行　许多视觉数据具有稀疏的特征，这些数据包括：立体深度 (Stereo Depth，例如重新投影到平面上的激光雷达数据)、点云 (Point Clouds)、笔画等。文本文档通常表示为高维稀疏向量 (典型的维度是几千，而典型的稀疏度是 90% 到 95%)[117]。天文稀疏数据[118]、医学数据 (包括视网膜数据[119]、光声层析成像数据[120]) 和药物研发数据 (基因/蛋白质、化合物、基因组患者、动物模型、干预等[121]) 都具有稀疏的特征。这些稀疏数据的共同特点是高维特征，由于数据**不均衡分布**导致的执行不均衡，在部分计算环境中还可能会遇到调度和调整的情况。结合视觉和时序两种高度维度特征，运动、动作数据也具有稀疏的特性[122]。

与维度关联属性有关的动态特征可以通过关联维度的规模与样本空间大小来反映，

它间接反映了数据的稀疏程度。例如 ImageNet 图像识别数据集中数量和类别二者有关联，具体参数是 (15M, 22K)；Sogouweb 关系分析数据集中边与点有关联，具体参数是 (12.25T, 271.9B)。可以看出，后者的关联程度远高于前者，样本空间也远大于前者。

(b) **与规模有关的动态特征**

数据规模巨大导致处理过程具有动态性 在数据规模巨大的时候，可能需要读取到数据的具体内容才能对其进行恰当的处理。这一过程难以在计算之前进行完整预测。很多数据在处理过程中才会确定是否需要剪切、补全或者进行其他处理；一些应用的处理过程需要在一步计算结束之后，根据计算结果决定下一步的计算内容；一些应用需要根据整体数据特征实时改变计算，如增删或排布数据等。此外，还有一些数据在没有进行基本的近似分析时不能得到它的特征，所以对于计算的估计也会出现动态性。

采样条件下执行过程具有动态性 在数据规模巨大的时候，部分计算过程可能会涉及采样，采样会使执行过程具有不确定的特征从而导致动态。例如随机采样出现在集成算法中，每个子模型都只处理采样所得的一部分数据；在迭代执行的某些阶段，可能会遇到只需要部分采样数据的情况。

与规模有关的动态特征可以通过规模本身的大小间接反映。

3.2 动态模型

人工智能模型尤其是深度神经网络在近年来取得了长足的发展。其种类繁多，应用广泛，并且仍在持续演进。无论是卷积神经网络、循环神经网络，还是元学习、多模型协同、生成查询网络、图网络等新型人工智能模型，在执行或模型训练阶段的共同需求和方法描述，是运行环境中需要着重研究的问题。

3.2.1 人工智能多层次模型表示建模

人工智能模型的执行是多层次的结构，基于基本模型描述的计算流图只是其中的一层。第一层由外层执行方式构成，第二层由基本功能模块的组合图构成，第三层由基本算子组成的内层计算流图构成。

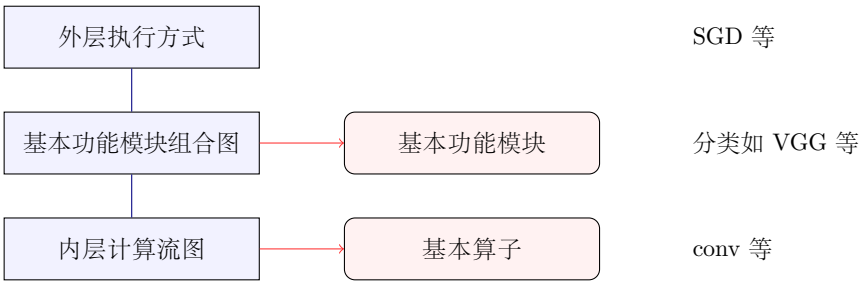


图 3.2 人工智能多层次模型表示

外层执行方式包括多种迭代计算方式 (SGD, Adagrad, Adadelta 等)。基本功能模块、内层计算流图、基本算子将在后文讨论。

在具体的人工智能应用的执行中，这些模型层次可能包含全部或只包含部分结构。例如卷积神经网络的训练过程主要由外层执行方式、内层计算流图两层构成；神经模块网络的执行过程由全部三层构成；推理过程只有后两层构成。

模型规模主要指模型中的参数规模，参数的规模往往依赖于输入数据的维度 (例如张量维度)，构成模型的不同算子参数规模也有不同。

表 3.3　典型模型规模

model	single input size	parameters	computations
AlexNet	227*227*3	60M	0.7G
VGG16	224*224*3	138M	15.3G
GoogleNet (Inception V3)	224*224*3	23.2M	5.7G
ResNet152	224*224*3	60M	11.3G
SqueezeNet1.0 [123]	224*224*3	1.3M	0.8G
Densenet-161	224*224*3	28.9M	7.8G
EncDec(lstm unit) [124-125]	70+87	709K	
NIC(GoogleNet + 1-layer lstm) [126]	image	默认 512 个 lstm 单元	
Bert [4]	text	340M	
GPT2 [7]	text	1542M	
Deep Jazz (2-layer lstm)	midi music(4096)	256 个 lstm 单元，4.5M	

K: kilo, M: million, G: billion

附录 B 收集了典型人工智能模型集合中的 251 个 (类) 模型。典型的 CNN 模型参数规模都在 M 级别，而以 LSTM 为主要构成单元的 RNN 模型参数规模与输入、输出数据规模强烈相关，从 K 到 M 不等。GAN 等由多个模块构成的模型，其参数规模需要累加每个模块的参数量和连接模块的结构参数。

在执行过程中，由于数据和参数都需要存入计算单元，内存需求 $m = (p + i) \times b$。其中 p 是模型参数量，i 是单样本数据的初始量和所有中间过程的数据量，b 是批次大小 (batch size)。例如 GoogleNet 中 $p = 23.2M$ (M: million)，$i = 9.8M$，设 $b = 32$。若采用单精度浮点，则不经过融合优化的内存需求至少为 4GB。

表 3.4　多层次模型表示与其他模型表示方法的比较

	迭代	模块	算子	动态性描述	调度粒度	新模型构建	执行	通用性	量化能力
多层次模型表示	✓	✓	✓	✓	可调	易	✓	✓	✓
计算流图	×	×	✓	×	固定	难	×	×	×

3.2.2　迭代计算细粒度特征建模

在以深度神经网络为代表的部分模型训练时，迭代的基础是采用反向传播算法进行基于梯度下降的参数更新。在计算中，反向依次对不同的基本算子构成的层进行参数更新。

迭代计算体现在“批次”与“轮次”两层迭代。内层的一次迭代包括一次正向计算与反向传播，迭代完成需要所有的数据按批次进入模型完成计算；外层一次迭代包括一次全数据的吞吐，迭代完成需要达到设定轮数值或达到收敛条件。

迭代计算要求计算过程按照迭代的顺序执行，以保证模型按照迭代的顺序进行参数更新，两个迭代步之间没有输入数据的关联约束。

(a) **批次需求**

随着人工智能应用训练数据的增长，一次性载入所有的训练数据变得几乎不可能。批处理 (batch) 成为一种有效的方法。当数据量足够时，批次化不会降低训练的效果，但批次大小 (batch size) 的大小对训练的不同目标有多方向的影响。

当 batch size 增大时，会增加内存需求，将小矩阵乘法转化为大矩阵乘法，并行效率更高；一次全数据集需要的迭代次数更少，但达到相同精度所花费的时间可能会增加。Keskar 等人[127] 讨论了 batch size 过大引发的问题。因此，需要权衡各因素寻找恰当的 batch size.

典型的 batch size 为 32(NVidia Kepler GPU, GoogLeNet)[128]。当多个运算单元同时计算时 batch size 可以成比例增加，例如 256 个 GPU 总共容纳 batch size 为 8192(Tesla P100 GPU, ResNet50)[129]。当需要更大的 batch size 以致于内存无法容纳时，可以拆分 batch 为多个 minibatch[130]。

在实际应用中，“批处理”技术可以有效利用向量化的部件，还可以将计算集合成为计算密度较高的操作。批处理时，多条数据可以同时进入模型进行运算，因此每个算子面对的数据维度会增加。例如，向量乘法经过批处理可以扩展为矩阵乘法。比向量化操作更广泛，除具有依赖关系的计算外，大部分计算都可以适应批处理。

(b) **轮次需求**

轮次 (epoch) 指处理完一遍全数据集的数据。使用了批次化的梯度下降算法需要更长时间来收敛，但现实应用中往往没有那么多的数据，因此通过将原始训练数据输入多遍，可以缓解这个问题。epoch 与 batch size 也具有相关性。典型的轮次需求为 90(ResNet50, ImageNet)[131],100(AlexNet, ImageNet)[132] 等。

3.2.3 迭代计算动态性分析

迭代计算的动态特征表现在多个方面。

迭代方法的变化导致动态性。随机梯度下降 (SGD) 中参数对于每个样本都更新一次；小批量梯度下降法 (Mini-batch GD) 中，参数对于每小批样本更新一次；Momentum 在更新参数时保留参数更新的方向；与前述方法用确定的学习率 (learning rate) 迭代不同，AdaGrad、RMSProp、Adam 可以在训练过程中通过不同的方法自适应调整学习率 (梯度平方和相关、梯度均值相关、梯度一阶／二阶距估计)。在训练过程中可以根据需求在线切换方法。

迭代参数的实时调整导致动态性。例如上述迭代方法中的批次大小、学习率等参数都可以在迭代执行的过程中 (不同的迭代轮数中) 进行调整。

迭代过程中加入数据转化或调整操作会导致执行动态性，典型的例子是数据混洗(shuffle)，其目的是减少数据的差异，以维持数据的一般性并防止过度拟合。常见的混洗场合有：在深度学习的每一轮混洗所有原始输入数据，或在一个批次中混洗数据。

3.2.4 基本模块细粒度特征建模

(a) 基本模块的构成

在人工智能快速发展的今天，基于功能划分的基本模块涵盖的范围更加广泛，除了基本的已有模型 (如深度神经网络中的 VGG) 外，还包括传统机器学习模型 (如支持向量机 SVG) 在内的一系列模块，其主要功能包括特征提取、分类、预测等。

特征提取 将原始数据的规模大大缩小但仍保持主要信息，其主要方法是降维技术，包括边缘检测、移动检测、主成分分析、自编码器等。

分类和识别 分类器可看作是一种特殊的特征提取器，识别可以通过在类别全集中的分类而实现。很多深度卷积神经网络最终的目标是得到一个分类器，当然，其类别数可能远远大于传统的分类器。训练好的深度卷积神经网络模型可以作为后续新模型的基本模块。

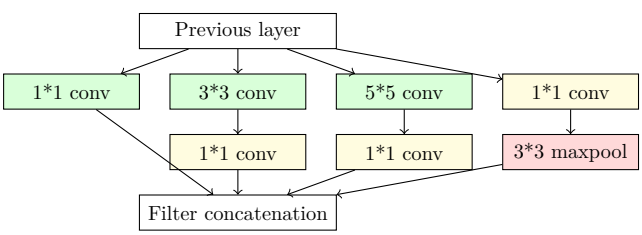
记忆 模块可以处理具有时间特征的数据序列。一般的记忆模块通过将输出接入模块内部，影响模块本身的参数，从而在未来仍能保持受前序输入的影响。

典型的基本模块包括 Inception, Residule, Shuffle, LSTM 等。Inception 模块 (如图 3.3(a)) 通过多个算子组合得到，其目的是在提供相当于 kernal size 较大的卷积的同时减少参数量。Residule 模块 (如图 3.3(b)) 通过在串行连接的深度网络之间添加跳跃连接形成。ShuffleNet[133] 中首次出现的 Shuffle 模块 (如图 3.3(c)) 是为了缓解多个 MobileNet 中卷积连续执行时至作用于局部数据的问题。通过混洗，可以在 shuffle 层将数据扩散。该模块引入了数据之间的依赖性，增加了访存的随机性或增加了通信量。LSTM 模块 (图 3.3(d)) 是循环神经网络的重要组成部分，可以处理具有时序特征的数据，有多种变体结构。

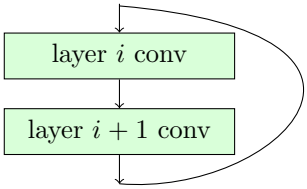
基本模块的尺度可以跨越多种规模，小可到一个基本算子 (如 ReLU)，大可到一个人工智能模型 (如 VGG)。模块的组合方式比以数据流为模式的基本算子更加多样，可以像数据流一样连接，可以像森林一样并列，也可以按照功能来嵌套。因此，模块间的交互方式除了数据流之外还包括可能包括控制逻辑，也可能不交互。

(b) 基本模块的组合方式

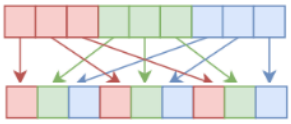
定义*基于模块的计算流图*：模块构成图中的点，有相互依赖关系的两个模块之间连接边。根据模块组合形式，边的构成也有多种形式：(1) 数据流连接：边权值为一个模块到另一个模块数据流中的数据量；(2) 控制流连接：边的方向是从控制模块到被控制模块，边权值为控制流需要的参数量；(3) 并列关系：模块之间没有边连接；(4) 嵌套关系：可以省略嵌套子模块，但将子模块的数据量等信息加入父模块的特征中；也可以分别将父模块与子模块作为两个独立的点，由子模块向父模块画一条边，边权值是子模块与父模块交互部分的数据量，如果子模块没有更小的组成模块，它将只有父模块一个邻居。



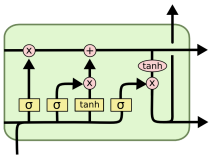
(a) GoogleNet 中的 Inception 模块



(b) Residule 模块

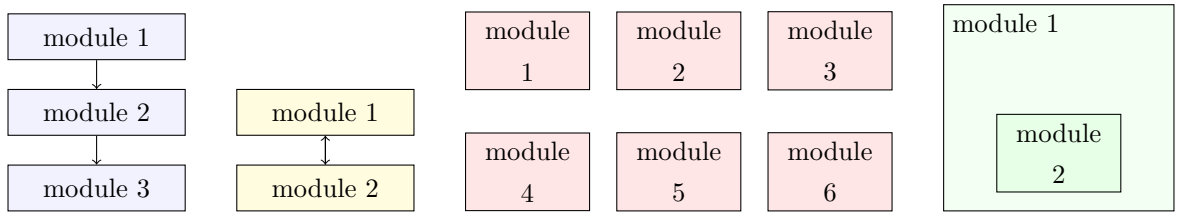


(c) Shuffle 模块



(d) LSTM 模块

图 3.3　典型基本模块



(a) 数据流连接　(b) 控制流连接　(c) 并列　(d) 嵌套

图 3.4　基本模块的主要组合形式

一个典型的基于模块的计算流图如图 3.5所示，图中刻画了生成对抗网络 (GAN) 的模块计算流图，该模型由生成器 generator 和判决器 discriminator 两个子模块构成。

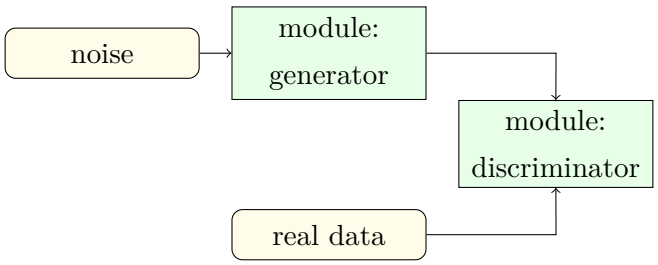


图 3.5　基于模块的计算流图 (GAN 模型)

(c) 基于模块计算流图的优势

模块相较于算子或算法，具有粒度适中、任务较完整的特点，适合作为基本的计算单元进行初步任务划分，基于模块的计算流图将为后续的计算调度提供重要的参考价值。本文所归纳的模块之间 4 种组合方式是未来复杂应用的基本构成单元，也是人工智

能领域中多个子领域交流融合产生新方法、新模型的主要表现方式。

3.2.5 基本模块动态特征分析

随着应用领域的拓展和应用类型的丰富，简单的静态模型已经难以满足复杂的人工智能应用，对于能够实时调整模型结构的需求越来越多，动态模型应运而生，支持动态图的框架随之出现[51-52,134-135]。其中的主要技术是对于动态计算图 (Dynamic Computation Graph) 的处理，如动态批次化 (Dynamic Batching)[135]，将引发动态性的不规则输入中类似的计算集中在一起，其接口封装在静态框架上，从而实现动态调用。

以动态算子为基本组成的模型仅仅是动态模型的一种，在更复杂的应用中，以模块为基本组成的动态模型更为常见。例如模块化的神经网络 (Neural Module Networks)[9,136]、基于控制器模块的动态网络[137]、基于中间过程数据的动态执行结构[138]等。这类动态模型的特征是**由输入或执行过程中的数据决定执行哪些模块**。模型的组成一般有两种方法：

1. 已有包含所有需要功能的模块池，由变化的输入生成实时模型 (基于模块组合的计算流图)。例如图 3.6举例说明了根据输入生成动态模块构成的模型：分析输入 input，假设由模块 input analysis 确定有 3 个功能需求，于是生成 3 个模块 count, image recognition, locate，并通过 answer combination 模块关联，最终得到输出。当然，根据不同的输入 input，可能会获得不同的功能需求，从而生成不同的模块。此类应用中，需要根据输入和应用要求合理地拆解多种需求、能力、技术，匹配恰当的功能模块来应对，并关联这些模块 (基本模块的主要组合形式)。
2. 已有一个较大规模的模型，该模型包含所有可能执行的模块和可能的连接关系，根据输入或者中间过程的数据确定执行哪些模块。例如集成学习中，由子模块通过并列的方式构成模型，在执行阶段通过中间过程数据决定执行哪些子模块。

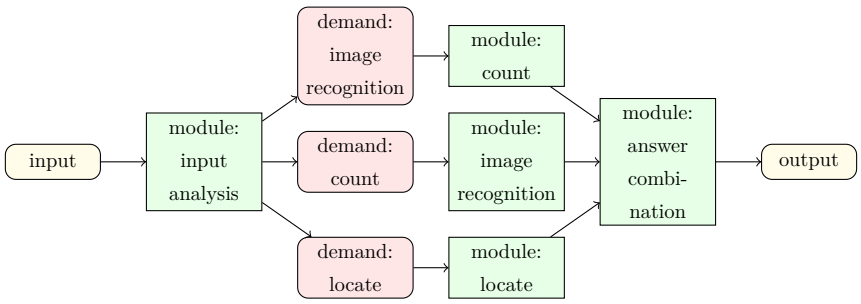


图 3.6 基于动态模块的模型构成举例

3.2.6 基本算子细粒度建模

人工智能模型构成元素多样、关系复杂。直接研究模型或模块的整体特征不能有效地指导运行环境设计。且模型发展迅速、难以计数，而其基本算子相对有限。将复杂人工智能模型的基本计算单元拆分，可以得到基本算子。

按照数据流关系拆分基本算子，可以将模型分为计算单元和数据流 (数据依赖) 关系。在神经网络中，神经元可以抽象成一个计算单元。计算单元和数据流关系组成了一张计算图。典型的基本算子包括卷积、池化、归一化等。人工智能的基本算子较为复杂，例如卷积的基本运算还可以拆分成乘法。在常见的人工智能运行环境 (如框架、库等) 中，基本算子是其 API 的重要组成部分，应用或模型中一系列计算能够成为基本算子的一般原则是其广泛适用于多种模型，且粒度相对恰当 (粒度大于一般的数学库中的 API)。在分析人工智能基本算子的同时对比分析其他领域的基本算子，例如科学计算中的算子——线性代数、排序等，可以为复杂的人工智能计算提供参考。

人工智能基本模型中的不同算子适用于不同的条件，此外一些算子由于计算较为复杂，逐渐被另一些计算较为简单、功能类似的算子**取代**，例如 ReLU 逐渐取代 Sigmoid。还有一些算子，为了综合其他效果 (例如添加相当于归一化的效果)，进行略微调整**演化**成了一类包含多种形态的算子，例如 LReLU、PReLU[139]、ELU[140]、SELU、RReLU[141] 等。

(a) **基本算子分类**

计算密度表示单位数据上的计算量，由计算方式和数据访问方式共同决定。分析不同基本算子的计算密度，可以研究其所组成的整个人工智能模型的计算特征。

如表 3.5，根据计算特征的不同，本文将基本算子分为 4 类：*稠密计算*、*按元素计算*、*遍历计算*、*数据排布*。此种类别划分，综合考虑了算子的计算和数据需求，可以适应当前绝大多数的算子，同时也能在一定程度上满足新型算子。

表 3.5　人工智能基本算子分类及计算密度

类别	稠密计算 dense computing	按元素计算 element-wise computing	遍历计算 traversal computing	数据排布 data arrangement
基本算子举例	conv, deconv	absval, bias, dropout, eltwise, exp, relu, power, scale, sigmoid, lstm, tanh, $\cdots$	batch norm, lrn, loss, pooling, $\cdots$	clip, concat, crop, flatten, reshape, slice, split, tile, $\cdots$
数据量	局部小范围数据 n^2+k^2	单元素	局部小范围数据或局部数据 n	整体或局部数据 $O(n)$
计算量	n^2k^2	$O(1)$	$O(n)$	无
计算密度	$O(n)$	$O(1)$	$O(1)$	
访存量	$O(n^2)$	单元素	$O(n)$	$O(1)$ 到 $O(n)$

稠密计算以小矩阵卷积为主要代表，其涉及的数据量一般为局部少量数据，如 3×3 或 5×5 的小矩阵卷积。稠密计算的计算密度较高，常见的计算密度数值为 $O(n)$。它的实际数值与单位计算单元的体系结构参数 (例如内存大小) 有关。

按元素计算是对单个元素 (数据) 的特定计算操作，其计算特征与元素本身的计算有关，与数据量和体系结构几乎无关。例如 exp 需要对于元素或者向量中的每个元素求 e 的幂，power 需要对于元素或者向量中的每个元素求幂，二者的计算密度与 exp 和 power 函数本身的计算复杂度和实现有关。

遍历计算涉及的数据也是局部数据，但范围可能比稠密计算涉及的范围要大，例如归一化操作 (batch norm) 涉及的数据较大 (如一个批次的输入数据)，池化操作 (pool) 涉及的范围较小 (如 2×2 的小矩阵)。遍历计算除了按照元素进行简单的基础操作之外，还需要附加一定的整体操作，例如比大小、求极值、求误差等。其基础操作往往简单，整体操作靠累加或者维护一个具体的值实现，主要的计算过程是遍历，其计算密度是常数。

数据排布是对数据的基本操作，包括添加外围 0 元素、拆分成块、合并等，涉及的数据范围可以是局部数据也可以是整体数据。其基本操作是访存和数据转移，计算密度低，也可以通过前后层 (模块) 整合来优化隐藏这部分开销。

(b) **人工智能基本算子与其他领域基本算子的关系**

表 3.6　数据分析领域基本算子的计算特征

类别	基本运算	数据量	空间需求	单节点计算	空间 (单节点)	计算 (单节点)	通信 (总)
线性代数	M*M	$O(n^2)$	$O(n^2)$	$O(n^3)$	$O(n^2)/p$	$O(n^3)/p$	$O(n^2)*p^{0.5}$
	M*V	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2/p+n)$	$O(n^2)/p$	$O(n)$
	$V_{1\times n}$*$V_{n\times 1}$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^2)/p$	$O(n^2)/p$	$O(n^2)$
	$V_{n\times 1}$*$V_{1\times n}$	$O(n)$	$O(n)$	$O(n)$	$O(n)/p$	$O(n)/p$	$O(p)$
图	关系穷举	$O(n)$	$O(n^n)$	$O(n^n)$	$O(n)/p$	$O(n^n)/p$	$O(n)$
	图遍历	$O(n+e)$	$O(n+e)$	$O(e)$	$O(e)$	$O(e)/p$	$O(cut(e))$
逻辑	逻辑操作	$O(n)$	$O(n)$	$O(n)$	$O(n)/p$	$O(n)/p$	$O(p)$
转换	k 维 FFT	$O(n^k)$	$O(n^k)$	$O(n^k \log n)$	$O(n^k)/p$	$O(n^k \log n)/p$	
集合	集合操作	$O(n)$	$O(n)$	$O(1)$ to $O(n)$	$O(n)/p$	$O(n)/p$	$O(1)$ to $O(p)$
排序	排序	$O(n)$	$O(n)$	$O(n \log n)$	$O(n)/p$	$O(n/p \log n \cdot \log p)$	$O(n)$

数据分析领域一些典型的基本算子如表 3.6，计算需求除以数据量可得到计算密度 (Computational Intensity)，常见的计算密度如表 3.7。

表 3.7　典型基本算子的计算密度

$O(1)$	$O(\log n)$	$O(n)$
按元素计算、遍历计算，如向量乘法	排序、转换计算、Stencil	矩阵乘法

按照算子分析的计算密度只能达到“数量级”程度的精度，可以粗略的看出矩阵乘法和由它构成的卷积计算计算密度高于一般的排序、转换计算，远高于按元素计算、遍历计算、向量计算。然而，如需得到计算密度的精确数值，还需要结合体系结构特征，如存储容量、数据精度等一系列参数。结合体系结构的具体参数，不同算子，尤其是计算密度依赖于数据量的算子，在不同的体系结构上会有不一样的特征，与缓存大小有关。

科学计算领域的基本算子约有 13 类[142]：有限状态机、组合逻辑、图遍历、结构网格、稠密矩阵计算、稀疏矩阵计算、谱方法、动态规划、N-body、MapReduce、回溯/分支与界、图模型、非结构网格。这 13 类计算在典型的高性能计算环境上都有涉及[143]，人工智能应用中也有部分涉及。数据分析领域的基本计算约分为 8 类[144]，除了表中类别之外还有采样计算 (如蒙特卡洛)、统计操作 (计数)。

在科学计算或数据分析领域，同一个应用涉及的算子较为单一，即使有多个算子，算子之间的组合方式也较为简单。但算子本身容纳的数据量巨大，例如 Linpack[37] 计算需要求解一个超大规模矩阵，分子动力学模拟[145] 需要求解超大规模牛顿运动方程、费恩曼路径积分等物理学描述。计算模式、数据访问模式与可分割性与算子本身有关，一般的，在规则条件下，可扩展性较强。

人工智能模型，其模型本身构成复杂，有可能规模巨大，基本算子可能只有几类，且一个单一的算子涉及的数据量有限，例如针对 ImageNet 图像进行识别的 AlexNet 网络，卷积单元可能只面对 224*224 的图像和 3*3 的卷积核，但模型本身有多层多个卷积、池化等基本算子。其计算模式、数据访问模式、可分割性不仅与算子有关，还与模型的构成、不同算子之间的依赖关系有关。很多模型的基本算子连接关系复杂，可分割能力较差，模型可扩展能力严重依赖于模型结构。(典型的模型结构如附录图 B.1) 例如 AlexNet、GoogleNet、VGG-19/34 模型中，算子关系较简单，成静态的数据流形式。residule 形式的 VGG、图神经网络 (Graph Neural Network)、EncDec、多任务模型 (MultiTask Network) 等模型构成复杂，算子间可能有条件控制流关系。

(c) 基本算子计算特征分析

本节将依次分析人工智能中各基本算子的计算特征，包括基本计算、数据访问模式、和其他技术对算子计算的影响。

稠密计算 稠密计算算子主要包括矩阵乘法和卷积计算。该类计算中，对于每一个数据的操作较为简单和规律，计算的复杂之处在于多个数据的组合。因此，需要关注数据量的需求和数据的访问模式。

矩阵乘法的主要操作是浮点乘法加法，计算密度约为 $O(n)$，常见容量的双精度硬件单元中 (例如拥有 96KB 缓存的计算部件)，计算密度约为 9.79ops/Byte。其数据访问模式为连续访存和等距跨步访存，后者可以通过数据重新排布的优化转换成连续访存。“全连接层”是典型的矩阵乘法算子。

卷积计算的主要操作是小矩阵乘法，主要操作仍然是乘法和加法。计算密度与卷积的张量维度有关，在常见的二维张量条件下，卷积的计算密度为 $O(n)$，具体数值可参考矩阵乘法。然而由于其计算对象是小矩阵乘法，常见的小矩阵规模是 1×1、3×3 或

5×5 等，在此规模内的访存是连续的或者等距跨步的。以小矩阵为基本单位，数据的访问是连续移动，移动的步长与模型参数“跨步距离”(stride) 有关。常见的跨步距离为 1，常见的跨步距离会小于小矩阵的边长。因此，以一个浮点数据为单位，数据的访问是“连续 + 跨步 + 折回”的，如图 3.7表示 28×28 的二维矩阵上进行卷积操作 (卷积核大小为 3*3，跨步为 1) 的数据访问模式 (来自于 Mnist 手写数字集识别训练应用)。数据

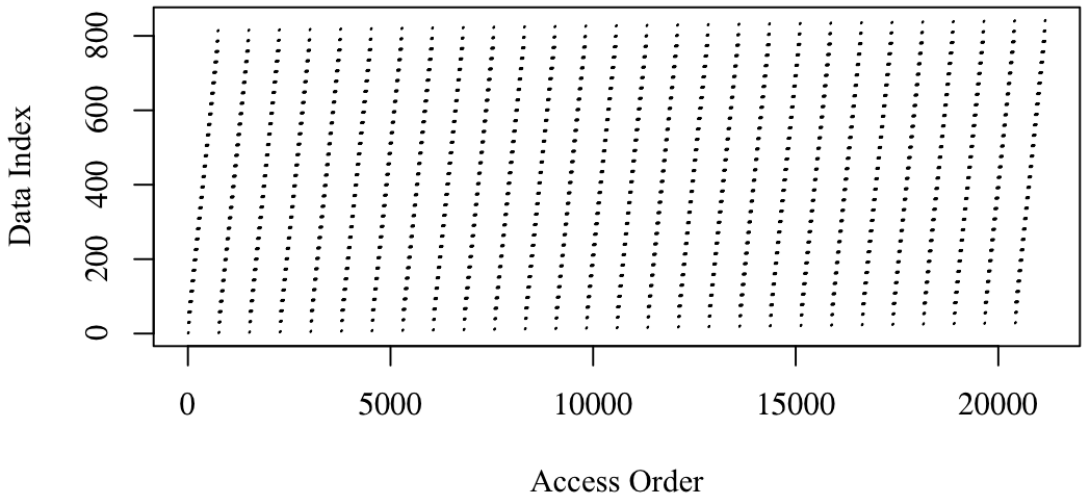


图 3.7　卷积计算中数据的访问模式

访问中的“折回”并不重复，因此，可能会出现额外的预取浪费。如果一个卷积操作的输入数据规模较小，数据访问可以一次获得所有数据，优化的空间较大。但若输入数据规模过大，例如从图中的 28×28 增加到 4000×3000，数据难以一次获得，针对数据访问模式进行优化可以大大降低数据读入次数。然而，卷积计算中，计算密度较高，瓶颈在于计算，只要数据供给能够填满硬件的计算，可以不必进行更优化的数据访问设计。

按元素计算　按元素计算时，计算只与当前数据有关，因此对计算特征的分析依赖于对具体算子计算操作的分析。

如表 3.8，基本算子的计算可以简单至一次加法；也可以复杂至如 tanh 函数所需，优化的近似算法也需要多次浮点乘法加法。虽然整体上计算复杂度是 $O(1)$，但仍需考虑基本算子实际需要的精确计算量。按元素计算的数据访问方式与基本的计算无关，与算法实现和输入数据有关，一般为顺序访问。“向量化”一列在实际应用中，也具有指导意义。一些具有 SIMD 部件的环境与加速函数库协同支持的函数如表中“有”所示。当向量化后数据之间的计算含有判断条件导致不完全相同时，向量化有可能无法实现，或者需要间接实现。当向量化后数据之间有依赖关系时，向量化无法实现，例如 lstm 单元。

遍历计算　遍历计算的对象数据是一批数据或者向量 (张量) 数据。除了每个数据的基本运算之外，可能有整体的累加等计算。此类基本算子一般在迭代计算求梯度等场景下出现。

如表 3.9，遍历类算子整体的累加运算范围多样。例如 batch norm[146] 操作的累加范围是一个 batch，lrn(Local Response Normalization) 范围是局部，而 loss 的范围是一

表 3.8　按元素计算的基本算子主要操作和计算量

基本算子	输入	输出	主要操作	向量化	计算量:(乘法; 加法), 其他
absval	x	$\|x\|$	求绝对值	有	(0; 1)
bias	x	$x+a$	加	有	(0; 1)
eltwise	x, y	$f(x,y)$	自定义计算,一般为加或乘法	有	自定义，一般为 1
exp	x	e^x	-	有	$\log x$ 可优化至 $O(1)$
power	a, b	a^b	-	有	$\log b$ 可优化至 $O(1)$
sigmoid	x	$\frac{1}{1+e^{-x}}$	-	无	
lstm	x_t, c_{t-1}, h_{t-1}	c_t, h_t	lstm 逻辑门组合，例如图 B.1(d)	无	(7;6),sigmoid 3 次,tanH 2 次
drop out	x	x/α 或 0	求呈 bernoulli 分布的随机数，乘法	无	多次加法，至少一次乘法，$O(1)$
tanh	x	$\frac{e^x-e^{-x}}{e^x+e^{-x}}$	-	无	$3 + 2\log x$，可优化至 $O(1)$
relu	x	$\max\{0, x\}$	-	无	(0;1)
scale	x	ax	-	有	(1;0)

表 3.9　遍历计算基本算子的主要操作和计算密度

基本算子	输入	输出	主要操作	范围	计算密度
batch norm	$(x_1, \cdots, x_n)$	$\gamma_i \frac{x_i - \mathrm{E[x_i]}}{\sqrt{\mathrm{Var[x_i]}}} + \beta_i$ 等	期望、方差	batch 全局	$O(1)$，至少 3 次乘法
lrn	$(x_1, \cdots, x_n)$	$\frac{x_i}{(k+\frac{\alpha}{n}\sum_i {x_i}^2)^\beta}$ 等	期望，乘加	batch 内局部	$O(1)$，至少 3 次乘法
loss	$\mathrm{x^{(t)}}$, $\mathrm{x^{(t+1)}}$	$\frac{1}{2n}\sum_{i=1}^{n} \|\|{x_i}^{(t+1)} - {x_i}^{(t)}\|\|_2^2$	乘、加	layer 全局	$O(1)$，至少 1 次乘法
pool	$(x_1, \cdots, x_n)$	$\max\{x_1, \cdots, x_n\}$	加	局部	1 次加法

个 layer。数据需求范围不同的算子在实际运行中数据访问次数可能不同，这取决于范围内数据量与局部存储容量的关系。当需求的范围过大，且算子涉及到更新每一个张量内数据时，可能需要整体数据的 2 次轮循访问。一般而言，此类算子的访问模式为连续访存，也可以通过快速方法求累加结果。

数据排布　数据排布这类算子的主要操作是数据访问，其数据访问模式为连续访存。常见的操作为数据复制。

从表 3.10中可以看出，基本层的反向传播的基本计算仍然局限在本文分析的人工智能基本算子范围内，且与对应算子的正向计算类似。按元素计算和遍历计算的反向传播取决于计算的具体操作，计算内容和计算量与正向相当。在实现中，数据排布类操作同样可以通过计算的实现优化。

表 3.10　神经网络基本层的反向传播基本计算和计算密度

基本层	反向传播基本计算	反向传播依赖的数据	计算密度
全连接 (fully connect)	$V_{n\times 1} \times V_{1\times n}$, 激活函数	后层误差，本层权值参数	$O(1)$
卷积 (conv)	矩阵转置，卷积	后层误差，本层卷积核	$O(n)$
池化 (pool)	矩阵扩展 (上采样)	后层误差，本层位置 (maxpool)	数据排布

3.2.7　基本算子动态特征分析

(a) 基本算子的动态稀疏性分析

稀疏性是很多科学计算得以高效执行的重要保证，而能否利用好更高的稀疏程度与应用执行时的动态能力密切相关。前文分析了 4 类基本算子，除数据排布之外，其余的算子都涉及到计算过程。稀疏性也在计算过程中体现。本节将从两个方面分析稀疏性：计算所导致的 (输出) 数据稀疏性和算子参数稀疏性。如图 3.8所示。

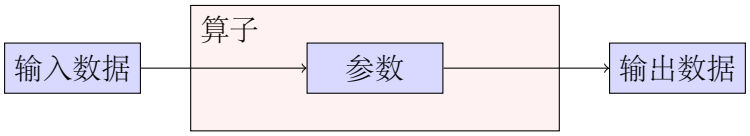


图 3.8　算子执行中的数据

表 3.11列举了稠密、按元素计算、遍历计算 3 类基本算子的参数特征、参数可稀疏化能力、输出数据特征、输出数据稀疏性。只有参数较多的算子 (例如 conv, bias, lstm, batch norm) 才有参数稀疏化的意义。dropout, ReLU 计算本身可以使输出数据稀疏化。对于输入数据为 0 的情况，按元素计算的所有基本算子都能保持输出为 0，稠密计算和遍历计算由于涉及到相邻数据，还需要考虑邻居数据的影响。

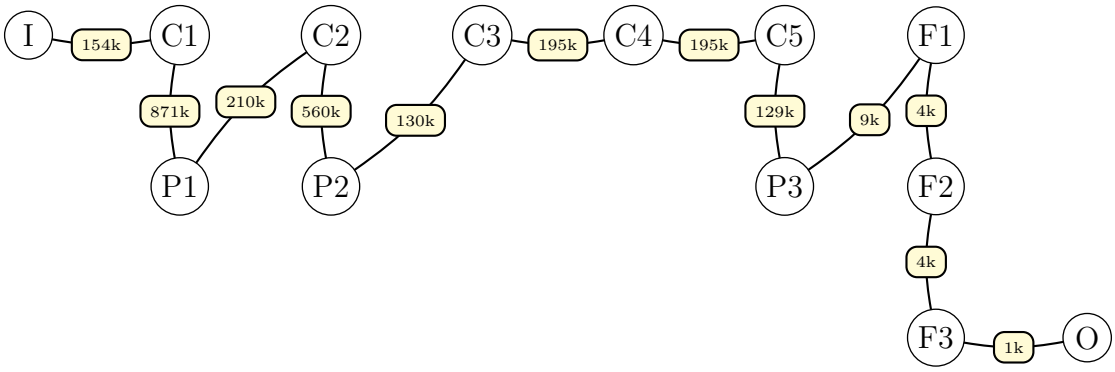
表 3.11　基本算子的稀疏性和可稀疏化分析

	稠密	按元素计算											遍历计算			
基本算子	conv	absval	bias	dropout	eltwise	exp	relu	power	scale	sigmoid	lstm	tanh	batch norm	lrn	loss	pooling
参数	有	无	少	无	无	无	无	无	少	无	有	无	有	无	无	无
稀疏化	可	–	否	-	-	-	-	-	-	-	可	-	否	–	–	–
输出数据	有	有	有	有	有	有	有	有	有	有	有	有	多	少	少	少
稀疏性	无	无	无	有	无	无	有	无	无	无	无	无	无	-	-	-

(b) 算子依赖关系动态性分析

基本算子所组成的数据流构成常规意义下的人工智能模型，算子之间的依赖关系是模型特征的重要方面。

定义“算子依赖图”：算子构成图中的点，有相互依赖关系的两个算子之间连接边，且边权值为一个算子到另一个算子的输出 (输入) 数据量。在算子依赖图中，正向传播时边的方向是前向，而反向传播时边的方向是后向。边权值的规模完全相同，正向传播的是数据，反向传播的是误差。



I: input unit, C: convolution unit, P: pooling unit, F: fully connected unit, O: output unit

图 3.9　AlexNet 的算子依赖图

图 3.9举例说明了 AlexNet 针对 ImageNet 输入构成的算子依赖图 (batch size = 1)。连接各算子点的边上数字表示从一个算子到另一个算子所需要传输的数据量。

在执行过程中，**算子状态和依赖关系动态变化会导致执行的动态性和拆分的动态性**。算子依赖图能够有效指导计算流图在算子之间的拆分，将模型拆分中的部分需求转化为图划分的具体问题。算子依赖图可通过图划分拆分。由算子之间依赖关系和算子内部依赖关系两种参数可构成全面的算子依赖图供拆分使用。由于依赖关系存在于点和边两个维度，此处的划分应该同时在点或者边上完成。模型的拆分可以转化成图划分问题，具体而言是边切割 (在图的边上切割，模型拆分) 和点分裂 (切割点，同时处理依赖关系，参数拆分或输入数据拆分) 的问题。

算子内部依赖关系会导致数据访问的动态性。除了算子之间的依赖关系，算子内部的计算也可能会出现依赖关系。例如卷积运算数据矩阵按行拆分时，同一行可能既属于上方计算，又属于下方计算；矩阵乘法拆分成分块乘法时，也需要考虑数据在同行/同列多次使用的问题。算子内部的数据种类较多，不同种类的数据耦合力度不同。典型的数据种类如表 3.12. 其中，属于数量维度的数据易拆分，且拆分时没有数据依赖关系；属于类别维度的数据在拆分时有可能产生依赖关系。

表 3.12　算子内部的数据种类举例

	类别维度	数量维度	综合
	输入数据	批数量	输入数据
	计算单元参数	计算单元个数	参数量
e.g.	卷积核参数	卷积核个数	卷积参数量

算子内部的参数量可以作为算子依赖图中点的权值参数，指导计算流图在算子内部的拆分。

(c) **算子动态稀疏性分析与算子依赖图的优势**

算子级的建模从基本算子分析和其所构成算子的依赖图两个层面进行了论述。对于不同算子计算特征和动态稀疏性的分析能够有效指导算子的执行、调度、改进和计算单元的设计；具有量化依赖关系的计算流图，与普通的模型结构图相比，更细致地描述了算子组合方式与依赖程度，能够有效服务于计算流图的拆分和调度。

3.3 动态资源

数据和模型的执行，离不开硬件环境。在计算环境的高速发展下，越来越多的新结构出现在新型人工智能硬件中，越来越多的新硬件融入系统环境，给计算系统带来了丰富的多样性。

3.3.1 多层次资源特征建模

人工智能运行环境中的体系结构是多层次多种类的。最宏观的系统层面，大到云计算系统、超级计算机系统，小到服务器系统，都是运行人工智能应用的重要环境。例如，涉及云计算的企业纷纷将云计算调整为智能云计算[147]，人工智能应用在云计算环境中占比越来越高；谷歌对于商业市场调研[148] 显示，超过一半的受访企业正在使用统计机器学习，机器人过程自动化或自然语言处理和生成，34% 正在使用深度学习。2018 年 11 月超级计算机系统 Top500[11]，前 10 名中的 5 个系统都提供了针对人工智能的专用硬件设计，其中多个系统为 2018 年戈登贝尔奖和提名奖的超大规模人工智能应用提供了运行环境[149]。微观层面，各类芯片，尤其是专门针对人工智能应用设计的硬件结构，为人工智能应用的运行提供了有力的支持。有以通用计算为代表的 CPU，有以领域通用计算为代表的 GPU，有以可编程逻辑为代表的 FPGA，还有以领域专用计算为代表的 TPU、Diannao 等。在运行环境中，多层次的体系结构需要多层次的资源管理与分配。由于人工智能应用具有领域专有的特征，因此可以针对性地进行更优化的调度。

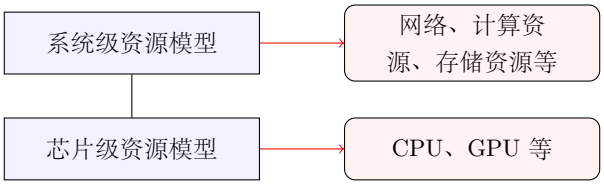


图 3.10　多层次资源特征模型

为了应对复杂环境下的动态特征，需要对动态资源进行建模，以指导系统的控制、应用的调度以及用户的需求反馈。本节构建*多层次资源特征模型*如图 3.10。

针对动态资源的建模不仅要考虑体系结构参数，还要考虑资源结构的多样性。多样性不仅包括不同种类的硬件单元，还包括同一种类下数量丰富的单元。体系结构是多层次的，可以对其每一层分别描述，也可以用后续调度的最小单位进行描述。

(a) **资源模型函数**

将系统中所有的硬件单元的关键参数集合，并恰当地描述，能为后续基于体系结构的动态调度提供依据。为此设计资源模型函数

$$f(\bigcup_{i}\{m_{ca}, m_{bw}, \bigcup_{j}(c), b_{bw}\}, b_{str}) \tag{3.1}$$

其中包含 3 层结构：第一层网络结构 b_{str}；第二层包括系统级的计算、存储、通信单元 (i)，相关参数包含容量和带宽等 (m_{ca}, m_{bw}, b_{bw})；第三层是芯片级资源，包括计算单元内部的部件 (j)，相关参数包括部件功能等 (c)。

<table>
<tr><td colspan="5">网络结构</td></tr>
<tr><td colspan="3">计算单元
$c_1, c_2, \cdots, c_{n_1}$</td><td>存储单元
$m_1, m_2, \cdots, m_{n_2}$</td><td>通信单元
$n_1, n_2, \cdots, n_{n_3}$</td><td>交互单元
$i_1, i_2, \cdots, i_{n_4}$</td><td>……</td></tr>
<tr><td>部件 1</td><td>部件 2</td><td>……</td><td></td><td></td><td></td><td></td></tr>
</table>

图 3.11　资源模型函数中的三层结构

(b) **动态资源实时状态函数**

为了得到实时状态函数，一方面需要将上述资源模型函数中固定的结构参数修改为系统实时状态参数，例如内存容量参数修改为可用内存容量 $(m_{current})$。另一方面需要添加一些描述，例如计算部件是否被占用 (c_{oc})。式 3.1可以修改为：

$$f(\bigcup_{i}\{m_{current}, m_{bw}, \bigcup_{j}(c, c_{oc}), b_{bw}\}, b_{str}) \tag{3.2}$$

(c) **多层次资源特征模型的优势**

多层次资源特征模型与一般的模型相比兼顾了系统级和芯片级的多层特征，描述了计算、访存、通信多角度的特征，相比于 Linpack、SPEC、Deep500 等性能指标，具有

表 3.13　多层次资源特征模型与其他模型或指标对比

	系统级	芯片级	计算	访存	通信	量化分析	实时性能
多层次资源特征模型	✓	✓	✓	✓	✓	✓	✓
Roofline 性能模型[150]	×	✓	✓	✓	×	✓	×
Linpack 性能指标[37]	✓	×	✓	×	×	×	×
SPEC 性能指标[151]	×	✓	✓	✓	✓	×	×
Deep500 性能指标[39]	✓	×	✓	✓	✓	×	×

对资源的量化能力。然而，多层次资源特征模型中一些具体参数仍需要通过具体的性能评测方式获得。

3.3.2 系统级资源特征建模

一个完整的人工智能应用的运行需要系统级别的环境。在系统层面上，计算、存储、IO 等硬件资源的相互协同，是应用高效运行的关键，而运行时系统，是软件环境应对动态问题的重要手段。

云计算系统、超级计算机系统、服务器系统都是人工智能应用的运行环境。而随着应用尤其是深度学习领域对计算的需求越来越大，人工智能越来越朝着与高性能计算深度融合的方向发展。表 3.14是几个重要的超级计算机中的人工智能设计，从中可以看出，硬件支持主要以 GPU 的使用为代表，而几乎所有的系统都有针对人工智能的软件环境。以下将从多角度分析以超级计算机为代表的系统级环境。

表 3.14　典型超级计算机中的人工智能设计

人工智能相关支持	系统
加速器 (以 GPU 为主)	Summit(V100), Sierra(V100), Piz Daint(P100), ABCI(V100), Titan(K20)
其他众核结构	Sunway Taihulight(sw26010), Tianhe(Matrix-2000), Trinity(Xeon Phi)
软件系统语言环境 (基础库)	IBM Power 系列 (OpenBlas, Bazel, NCCL 等), Sunway 系列 (swBLAS, swDNN 等)
软件系统应用平台 (框架、平台)	IBM Power 系列 (Caffe,TensorFlow,MXNet,Chainer 等) Sunway 系列 (swCaffe 等)
软件系统容器支持	ABCI

(a) 计算资源

主流的计算系统架构主要包括集群和 MPP 等，主要的计算资源包括 CPU、GPU 和一系列定制芯片等。主要芯片的参数在表 C.1。

表 3.15　典型系统的计算部件

计算部件组合	系统
CPU + GPU	Summit, Sierra, Piz Daint, ABCI, Titan
CPU + MIC(intel Xeon Phi)	Trinity
众核芯片 (sw26010)	Sunway Taihulight
CPU + FPGA	微软云人工智能环境
CPU + GPU + FPGA	百度 XPU

(b) 存储资源

超级计算机的文件系统主要是 Lustre[152]、GPFS[153]、PVFS[154] 等分布式全局共享文件系统。实际系统往往对这些文件系统做了适应性的修改，使其更适合超级计算机高可扩展的分布式环境。在数据分析领域还有更广泛使用的 Hadoop 分布式文件系统

表 3.16　典型超级计算机中的文件系统

文件系统	典型系统	特点
Lustre	Piz Daint	高可扩展，大文件性能更好，一般写性能优于读性能
GPFS	Summit, ABCI	分存储层、服务器层和客户端层，伸缩性好
PVFS	-	元数据管理集中、可扩展性受限，缺乏容错机制
H^2FS[155]	Tianhe 2A	将本地存储和共享存储在一个动态单一命名空间中协同工作
BeeGFS[156]	ABCI	有效缓解现有 HPC 环境中的浪涌 IO 模式，闪存介质保障系统在任何时候的性能要求

(HDFS)[157]。

除了文件系统之外，存储容量也是资源要素，例如 Summit 的内存容量达到了 2.8PB，神威太湖之光的内存容量也达到了 PB 级[11]，其外存容量更大。

(c) **网络资源**

网络资源中含有软件定义的网络设备，将网络设备的控制面与数据面分离，实现网络流量的灵活控制。网络设备与系统中的其他资源相连组成了不同的网络结构。典型的网络结构包括胖树、环网、蜻蜓等。图 3.17列举了超级计算机中的典型网络结构。

表 3.17　典型超级计算机中的网络结构

结构	典型系统	结构特点
胖树/裁减胖树	Summit, Sierra, Sunway, Tianhe2	超节点内通信有优势，需要重新布局数据以优化通信
环网	Sequia(5D), Mira(5D), K(6D), Titan(3D)	适合网格状通信、适合全局消息分散的通信
蜻蜓	Piz Daint, Cori, Trinity	组内结构可多样，需要自适应路由支持

除了网络结构之外，网络参数还包括带宽、时延等。

(d) **软件资源**

除了硬件环境以外，软件环境也是应用得以运行的重要基础，前文已对基本的软件环境有所举例。从库，到框架，再到应用平台，在未来人工智能与高性能计算系统的结合下，软件环境也会出现一定的变化，多层次高效整合是其重要的方向。

3.3.3　系统级资源动态性分析

在与体系结构密切相关的系统级环境中运行人工智能应用，环境的动态性是重要特征。

首先是**应用的多样与动态性**，系统中可以同时运行多个应用，应用可重叠、可变化，在不同时间对于系统级环境的不同方面需求可能会不同，会影响资源的状态。例如云环境承载了多种人工智能应用，这些应用计算需求及计算特征可能完全不同。

其次是**环境资源的可伸缩性**。计算资源、存储资源、网络资源都因应用的复杂多样和应用需求的变化而有所变化，也会因其自身的调度、控制、优化、环境状态和生命周期产生动态的特征。例如，在不同的计算阶段，系统中网络需求有高有低，存储开销有大有小，计算负载有重有轻。

最后是**用户需求动态性**。应用的动态和资源的动态可能会导致用户对于正在执行的应用提出新的需求，这又将进一步影响整个系统环境的状态。例如，在人工智能训练的不同阶段，用户可能会根据模型执行的情况实时提出新的资源需求。

3.3.4 芯片级资源特征建模

对计算、存储等资源的一般性分析，以及实际应用的真实运行情况都是分析运行环境的重要内容。典型的芯片和其具体参数如附录 C表 C.1。

(a) **性能模型**

采用 Roofline 性能模型[150] 可以描述芯片的性能。横轴表示计算密度，纵轴表示计算能力，斜率表示带宽。Roofline 能粗略地反映出一个芯片对于特定计算密度应用的最高性能。

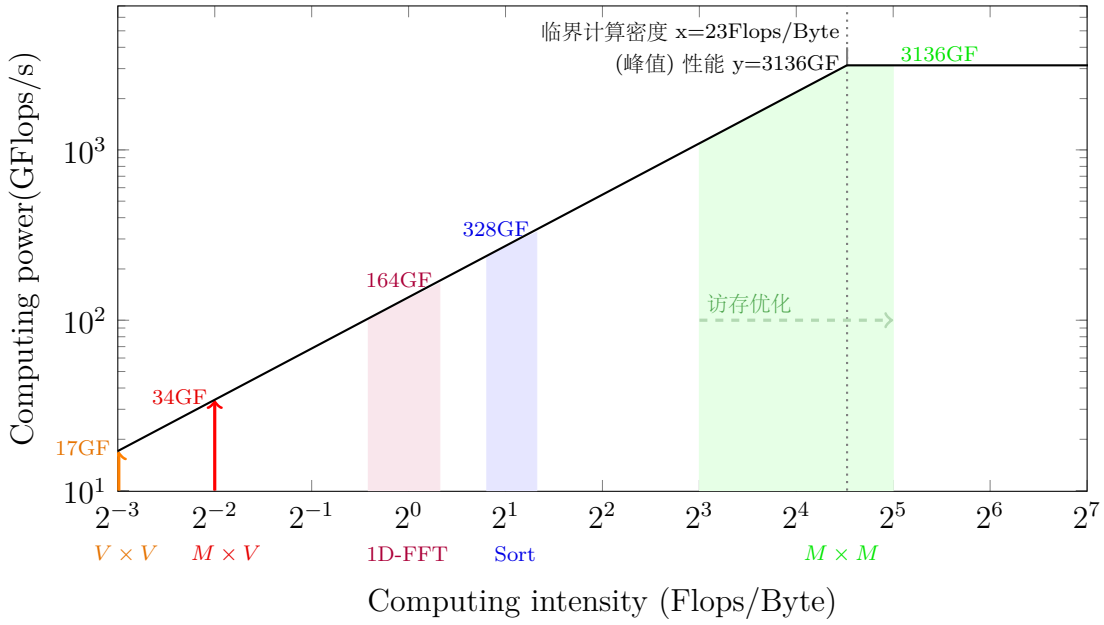
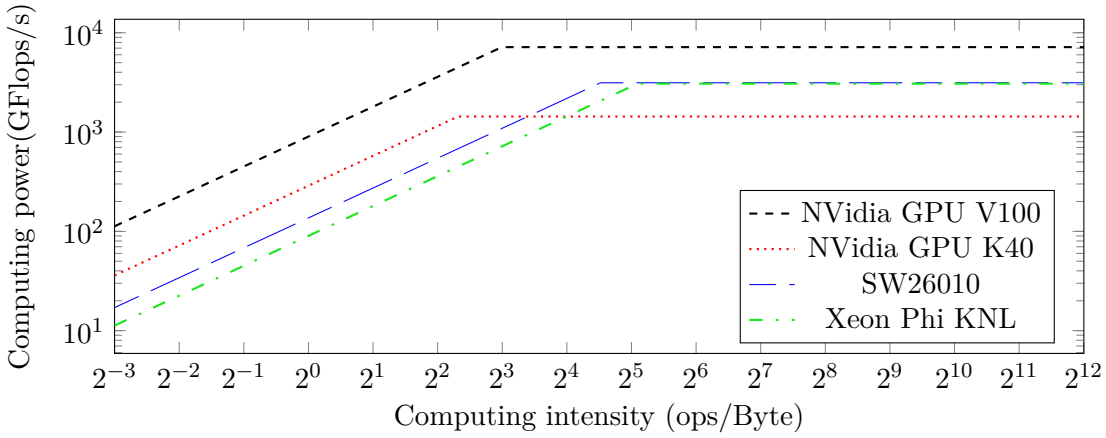


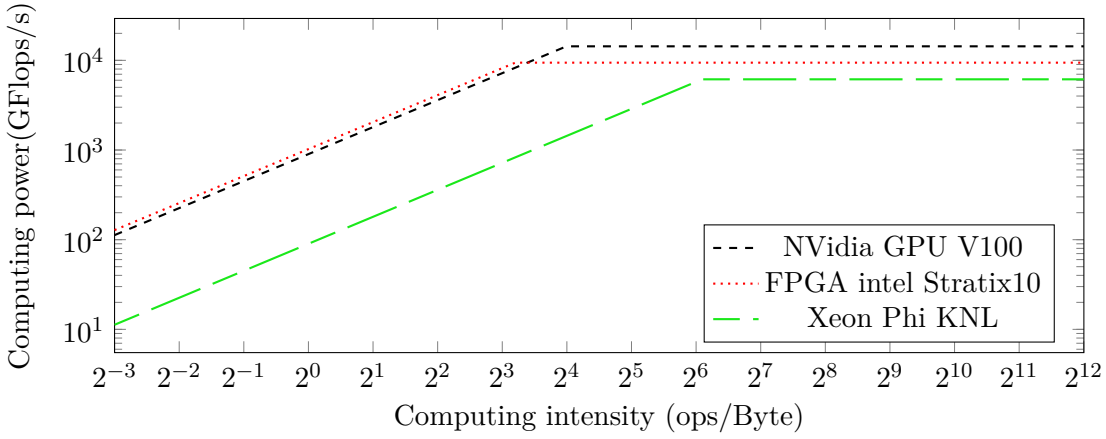
图 3.12　sw26010 芯片的 Roofline 模型

图 3.12表示 sw26010 芯片的 Roofline 性能模型，典型的计算操作在其上可以实现的性能已标注 (V*V, M*V, 1D-FFT 等)。各操作的理论性能为箭头所指位置 (Roofline) 线上，在实际运行中，常因为离散访存、控制流操作开销等原因而无法达到理论性能。图 3.13中，(a)-(c) 分别刻画了图中所示芯片双精度、单精度、半精度的性能模型。基本算子的计算密度可以参考图 3.12。

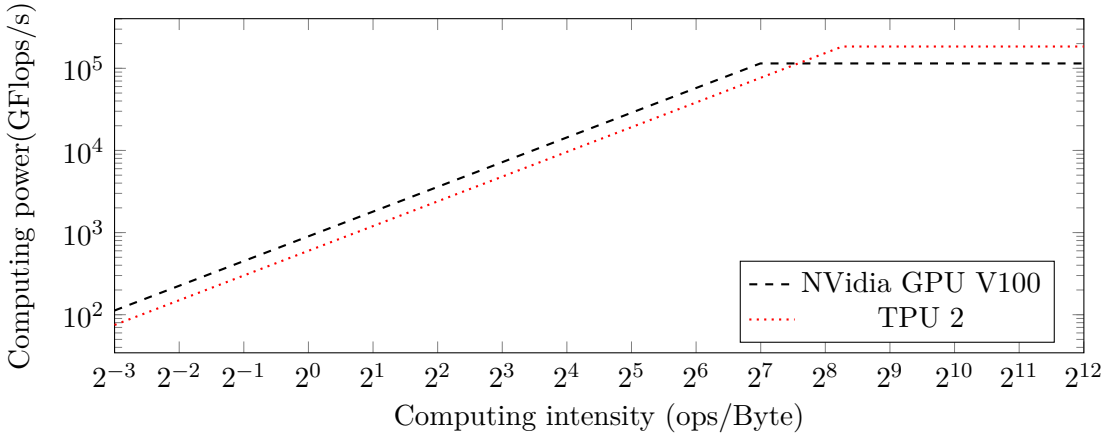
典型算子的计算密度已在第 3.2.6节中有所分析。然而，计算能力和计算密度所代表的参数只能反应人工智能环境的部分特征，在实际运行中组成部件、应用访存模式等



(a) 双精度 (DP)



(b) 单精度 (SP)



(c) 半精度 (HP)

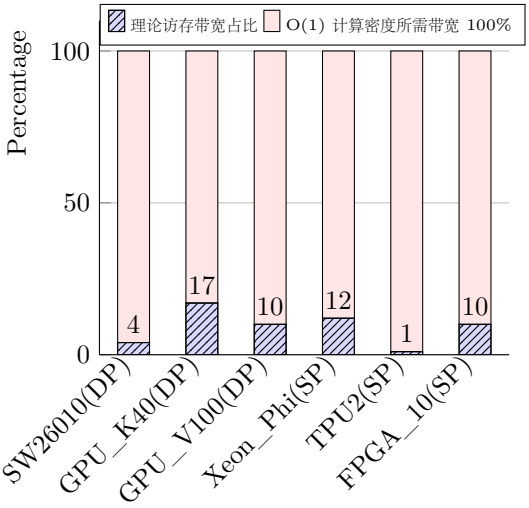
图 3.13　典型芯片的 Roofline

一系列因素都对执行情况有影响。因此，实际使用的性能模型需要兼顾理论性能分析与实际运行情况。

访存带宽是访存性能的重要方面。图 3.14是典型芯片的访存带宽与应用需求带宽，

芯片 (精度)	O(n) 计算密度所需带宽 GB/s	理论访存带宽 GB/s	O(1) 计算密度所需带宽 GB/s
SW26010(DP)	101	136	3072
GPU K40(DP)	207	288	1433.6
GPU V100(DP)	784	900	7680
Xeon Phi KNL(SP)	16	400	3072
TPU2(SP)	-	2400	184320
FPGA Stratix 10(SP)	19.7	1024	9420.8

(a) 典型芯片的访存带宽



(b) 理论访存带宽与数据密集型应用带宽需求

图 3.14　典型芯片访存带宽与应用需求带宽

从 (a) 中可以看出，所有的芯片都可以满足计算密度为 $O(n)$ 的应用 (如卷积、矩阵乘) 的访存带宽需求；而对于数据密集型的操作 (计算密度为 $O(1)$，以计算密度等于 1 时为例子)，芯片访存带宽往往不足。然而在实际应用中，计算密度为 $O(1)$ 的操作其具体计算密度往往大于 1，对于带宽的需求可能会降低数倍。但这些芯片表现出 5x 到 77x 的差距（子图 (b)），仍显对此类操作的带宽支持不足。

(b) **特殊结构**

NVidia V100 GPU 中的张量计算部件、寒武纪系列中的乘法加法树、TPU 中的脉动阵列等结构是处理器中的特殊结构。这些特殊结构针对特殊的计算设计，对于相应的计算十分契合且高效。对其性能建模可以参照 Roofline 模型，也可以利用实际运行多种具体算子的性能作为模型。由于这些部件并不是普适的结构，即使在人工智能领域也有许多计算不适合在其上运行。

3.3.5　芯片级资源动态性分析

芯片级环境中，动态性体现在综合应用中算子的组成多样，不仅有卷积等关键计算，也有其他不可或缺的算子。多个**同种算子如何组织**，**不同算子如何切换、如何流水**，需要动态地控制。即使一些芯片的特殊结构内部有数据流水的设计，但**数据如何组织**成恰当的形式以恰当的顺序进入这些结构也需要动态控制。此外，人工智能领域的一些计算体现出**稀疏**的特征，如何利用稀疏特征使计算更高效也需要动态设计。

指令动态　由于指令本身是动态的，在计算环境中能运行指令的最小单位即最小的动态粒度。例如，在 NVIDIA Tesla GPU 中能运行指令的最小粒度是 warp，每个 warp 中的所有 thread 执行同一条指令[158]。在 sw26010 中能运行指令的最小粒度是一个从核，每个从核有一个线程。

部件动态 FPGA 可重构硬件是典型的部件动态，可以编程决定硬件逻辑，根据应用做改变。此外，未来的芯片设计中可能会出现可伸缩性的结构，例如通过网络或总线组织不同的结构的启用与否和数量多少[159]。

3.4 小结

人工智能运行环境包含数据、模型和资源三个要素，本章分别对三要素进行了建模，用*多维数据特征模型*、*多层次模型表示*和*多层次资源特征模型*进行运行环境描述。

多维数据特征模型包括数据类别、维度、维度关联、规模参数等属性，该模型能够量化分析数据关联程度和时序性，从而分析数据的动态特征。在该模型下，我们分析了不同类别的典型数据集：对于面向分类的任务，一般的数据规模在百万至千万条数据，文本数据的典型量级为 MB，图片视频典型量级为 GB 到 TB；对于面向数据分析的任务，关系型数据典型量级为 GB 到 TB。不同属性中数据有不同层面的动态特征：维度属性中有预处理对齐引发的稀疏或者时序数据的动态；维度关联属性中有高维稀疏数据导致的动态；规模属性中有大规模数据处理过程和采样的动态等。数据的动态特征可以通过特征模型中属性的参数反映。

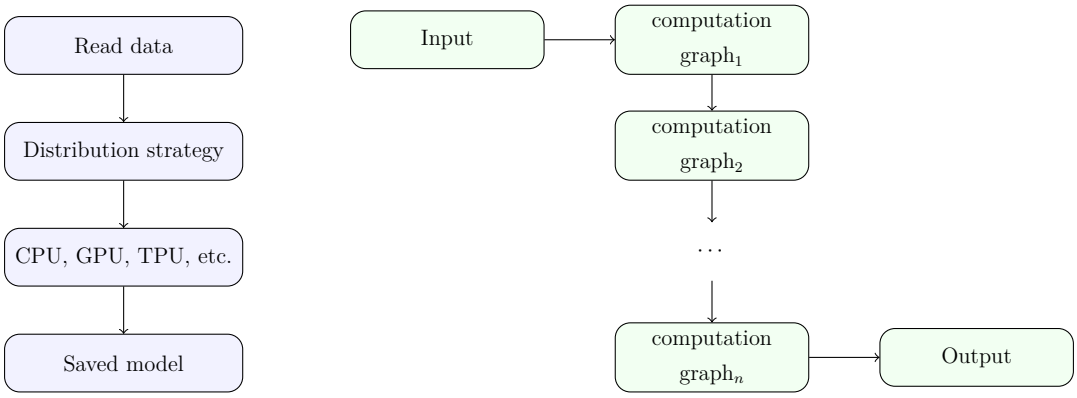
多层次模型表示由上至下 (或者由外到内) 包括外层执行方式、基本功能模块组合图和基本算子组成的计算流图。该模型更适应于复杂人工智能应用的计算过程，同时兼顾对动态特征的细粒度分析能力。在该模型下，我们分析了各层的组织形式和主要计算方式，分析了计算需求和动态特征。外层执行方式的动态特征表现在迭代方法变化、迭代参数调整，以及迭代过程中的数据转化操作；基本功能模块组合图中的动态特征表现在模块池条件下的动态模块选择 (由输入实时决定) 和超大模型条件下的模块动态连接 (由输入和中间过程决定)；基本算子组成的计算流图的动态特征表现在算子内部数据/参数稀疏性引发的执行动态性、算子内部数据依赖关系导致的数据访问的动态性、算子之间依赖关系动态变化导致的执行动态性和拆分动态性，以及算子演化带来的动态性。

多层次资源特征模型包括系统级资源模型 (以网络、计算、存储为代表) 和芯片级资源模型 (以 CPU、GPU 为代表)。我们对两层模型中各个属性的参数进行了形式化的量化表示，资源的动态特征由动态资源实时状态函数表示。该模型具有量化表示能力和实时特征属性，适应于动态资源的建模。具体而言，系统级资源动态性体现在系统中运行应用的多样与动态、环境资源的可伸缩和用户需求的动态性；芯片级资源的动态性体现在多个同种算子的组织、不同算子的切换/流水、数据的组织、利用计算特征进行的优化等操作，以及芯片部件的状态对执行过程实时动态影响。

动态人工智能运行环境建模将对后续阶段应用的执行提供理论支撑，对后续环境设计提供参考依据。

第 4 章　动态人工智能运行环境架构设计

目前的人工智能运行环境架构 (训练部分) 有两种设计，先定义后执行 (Define-and-Run) 和边定义边执行 (Define-by-Run)(图 4.1)。前者的主要执行方式为用户通过 API 定义全部计算过程，人工智能运行环境随即解释并将计算部署在适当的硬件资源中完成计算过程，之后获得运行结果。典型例子如 Caffe、TensorFlow、BLAS、cuDNN 等人工智能框架或数学库、人工智能专用计算库。后者的执行过程与前者类似，区别在于用户定义部分执行过程，运行环境随即执行，用户得到该执行结果之后再定义新的执行过程，用户每次的定义主要以计算流图的形式提供给运行环境，一次定义的粒度一般为一步计算。支持该模式的典型人工智能运行环境的例子如 DyNet、Chainer 等动态计算流构建的人工智能框架。



(a) 先定义后执行　　(b) 边定义边执行

图 4.1　目前人工智能运行环境架构设计

无论是先定义后执行还是边定义边执行，其执行架构是“定义－执行”的单程结构，缺少反馈。即使动态计算流构建的人工智能框架融入了计算流的动态构建，可以在一定程度上增加应用执行的动态性，但由于其动态粒度由“定义”全权控制，一旦“定义”其执行过程仍然是静态的、确定的，缺乏执行反馈尤其是资源状态反馈。

人工智能运行环境中的动态特征体现在数据、模型、资源各要素上。本章研究与执行相关的动态特征，与之相对应的与输入有关的动态特征。因此，本章的研究重点是运行环境的运行机制。

面对动态的人工智能数据、模型、资源，运行环境需要拥有动态的能力以应对各要素在运行中的变化。本章将构建基于“需求感知、动态响应、协同执行”闭环的动态人工智能运行环境架构。

4.1 动态人工智能运行环境架构

本节设计基于“需求感知、动态响应、协同执行”闭环的动态人工智能运行环境架构。

数据、模型和资源共同构成了人工智能的运行环境，其抽象模型已经在前文构建。将数据按照模型需求的方式合理地安排在资源中执行是运行环境需要提供的能力。图 4.2描述了动态人工智能运行环境的组成和运行机制。

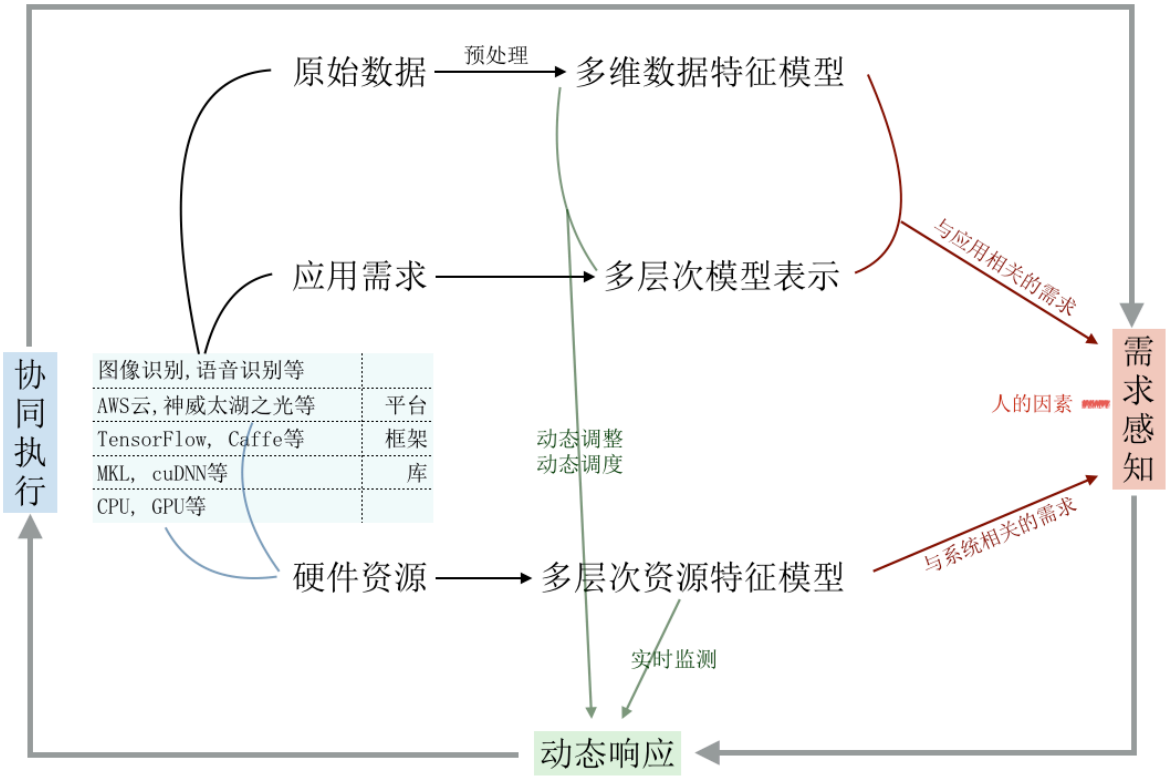


图 4.2　动态人工智能运行环境

原始数据是用户为目标应用提供的要素数据。可以来自标准数据集，也可以来自用户自定义的数据来源。值得注意的是，初始的原始数据仅仅是达到应用需求的参考输入数据之一，无论是预处理还是数据再生成，都是应用需求的重要方面。在大多数情况下，原始数据都需要经过处理才能输入模型执行。例如，数据需要统一格式；需要根据原始数据微调修改产生部分生成数据用以补充输入数据；数据需要拆分和合理排布，以适应后续执行等。当数据量巨大或后续计算需求较大时，对原始数据的基本分析也将为后续计算提供指导意义。例如，数据的整体结构、热点分布等。经过预处理的原始数据可以用前文所述的多维数据特征模型描述。

应用需求是人工智能计算环境的出发点。人工智能模型的构建，执行的目标和约束条件都由此产生，还需要根据原始数据设计。用前文提出的多层次模型表示可以在动态环境中描述应用需求。执行的目标在初始阶段一般转化成为最外层的执行要求，在执行过程中，此目标可以动态调整。与其他应用不同，人工智能应用的执行需要更全面地与用户交互。因此除了初始阶段的模型构建之外，在执行的全流程中，执行的目标和约束

条件可以由用户实时调整，也可以由用户设置策略自动调整。多层次模型表示可以通过现有的人工智能框架描述，也可以通过模型表示标准语言定义。但在执行过程中，为了保证动态，需要调整该表示中的各个元素。

硬件资源是人工智能应用的基本执行环境，提供基本计算功能。典型的组成部分包括计算结构、存储结构、互连通信结构等，可以通过前文所述的多层次资源特征模型描述。在执行过程中，硬件环境中的各个组成单元的状态会实时变化，这些变化可以通过系统性能监测装置获得，或通过执行监测结果计算而得。用户对于执行的硬件环境的运行要求也会出现变化，这些需求可以通过用户接口获得并结合硬件环境的具体情况作出恰当的安排。

表 4.1　动态人工智能运行环境变化

数据	样例数据规模变化，数据顺序变化，数据采样，数据格式、维度变化等
模型	模型结构变化 (有控制流条件的模型，算子、模块动态组合)，模型中基本组成单元变化 (算子调整)，模型参数变化等
资源	系统级变化 (增、删计算、存储、网络资源等)、芯片级变化 (调整频率，调整执行单元等)

在执行过程中，用户需求、资源状态、执行状态都可能出现新的变化，如表 4.1。这些变化体现了应用执行层面和系统运行层面的需求，需要运行环境响应和调整。

为此，我们设计动态的执行过程，并构成执行闭环，如图 4.2最外圈。执行闭环由需求感知、动态响应和协同执行三个要素构成，在人工智能运行环境中，三个要素互为补充、不断循环，是动态运行环境的运行机制。后文将分别分析这三个要素。

4.2　需求感知：目标矩阵

如图 4.2右侧，需求感知来源于用户指导 (“人的因素”) 和对执行状态的实时监测。而这些监测结果可以转化为来自于应用执行状态和系统运行状态中的多个方面的动态需求，如表 4.2。对于动态需求的实时获取，不仅可以指导人工智能应用的合理执行和资源的合理使用，还能对未来的需求作出恰当的预测，以应对可能出现的变化。

表 4.2　人工智能领域的动态需求

动态需求	需求描述	获取方式
应用执行状态	执行目标动态变化 (例如训练精度实时调整，步长实时调整)	框架内设置，预先设置动态条件
系统运行状态	内存、计算、通信、网络等参数和实时状态	已知参数或运行时通过系统接口动态获取

无论是应用的设计、执行还是动态调度，都依赖于不同层次的目标。本节将讨论人工智能领域的目标，为此构建目标矩阵，该矩阵包括两个维度：应用层面和系统层面。

表 4.3　分类问题目标矩阵示例

系统层面 \ 应用层面	准确率	召回率	f-measure	……
执行时间				
能耗				
资源利用率				
……				

在动态人工智能运行环境框架中，需要关注目标矩阵的变化而不是目标矩阵本身。动态调度的一般目标是满足资源与应用需求的匹配，此外还需要实时应对用户策略的调整。下文将举例描述不同层面的主要目标，其中很多目标已经形成了人工智能或相关领域内的性能评测标准。

4.2.1　应用层面的目标

人工智能领域的应用复杂多样，应用层面的目标综合了数据和模型的因素，可以被分解为多个子问题的目标，其动态性体现在子问题目标中阈值的调整。例如图像识别领域针对一系列标注的样本图片进行类别学习，并最终得到针对未知图片的类别描述这一应用。在训练过程中采用深度卷积神经网络，通过梯度下降的方法训练参数，是回归问题；而模型的优劣判断需要分析其对于训练集和测试集的应对能力，是分类问题；模型的调整还可能涉及到图模型等。

(a) **回归问题**

回归问题的目标包括：绝对误差 (Absolute Error) 、相对误差 (Relative Error, Relative to Magnitude)、平方误差 (Squared Error)、均方根误差 (Root Mean Square Error, RMSE)、和方差/误差平方和 (Sum of Squares due to Error, SSE)、平均绝对误差 (Mean Absolute Error, MAE)、平均偏差 (Mean Bias Error, MBE) 等。除了能应对最基本的回归问题之外，在以深度神经网络为代表的迭代训练中，这些参数指标是应用下一步如何运行、如何调整的重要指导。

(b) **分类问题**

分类问题的目标包括误差矩阵或者基于它计算而得的 sensitivity, specificity, overall accuracy 等。具体参数有准确率、召回率等。

面向特定问题的目标　结合场景的特定问题中，目标常根据以下几个方面的接受程度设置不同的权重综合考虑。

- 拒绝一个样本不会产生开销：最小开销 (minimal costs)
- 接受一个样本会得到回报：回报 (gain)

表 4.4　分类问题的目标

TP	true positive
TN	true negative
FP	false positive
FN	false negative

目标参数	计算方法
准确率 (precision, p)	TP/(TP+FP)
召回率 (recall, r)	TP/(TP+FN)
f-measure	2pr/(p + r)

- 接受一个样本不会产生开销：大损失 (big loss)
- 拒绝一个样本会产生开销：损失 (loss)

例如：垃圾邮件过滤（接受坏邮件比过滤掉好邮件更合理）；医学诊断（未识别出疾病比误诊健康的人更槽）。

(c) **聚类问题**

聚类问题的目标主要是判断相似程度是否达到设定阈值。根据条件的不同相似程度有多种参数。

当条件是不知道所属类别信息时，目标参数包括：调整后的排序指数 (Adjusted Rank Index)，用来测量两个类的相似度；基于相互信息的分数 (Mutual Information based scores)，用来测量两个类别在忽略置换变换条件下的相似程度；均匀性 (Homogeneity)，例如设置每类所含固定元素数量等。

当条件是知道所属类别信息 (ground truth 条件) 时，目标参数包括：完整性 completeness(每个已知类别的元素都能被聚类到自己所属的类别)、FM 得分 (Fowlkes-Mallows scores)、准确率召回率的几何平均值等。常见应用为验证类别，聚出新标准下的类别。

(d) **图模型**

图模型的目标一般为模型的合理程度，其评价标准为所得模型是否符合需求，在需要构造模型的场景下应用广泛。例如 ROC(Receiver Operating Characteristic) 曲线，其横坐标是 false positive rate (1- specificity)，纵坐标是 true positive rate (sensitivity)，可以用来确定所需的阈值。又如 LIFT 指标，衡量模型针对随机选择的目标的增强反应（相对于总样本空间），LIFT = target response / average response 。

应用层面的目标与多层次模型表示

应用层面的目标在人工智能运行环境中可以抽象为多层次模型表示中的动态参数，这些参数由数据、模型和具体目标共同决定。例如分类问题的准确率目标与着某一特定模型下的迭代次数有关。

4.2.2　系统层面的目标

系统运行层面的目标围绕着感知应用和感知系统两方面产生，描述一个具体问题在特定的环境下如何运行。其动态性体现在由执行状态实时变化导致的目标需求实时调整。

(a) **应用执行方面**

在应用执行方面，系统层面的目标包括执行时间、执行效率、执行的实时调整能力等。*执行时间与执行效率目标*希望能在最短时间内得到执行结果。相关的性能评测指标有单位时间内的吞吐量，固定数据集下固定应用算法的执行时间等。不限资源的条件下，可以寻找现有工艺下的最佳硬件和最优硬件数量。在资源约束条件下，可以寻找现有条件下的最快执行方式，利用一切优化手段对应用进行调整，例如将输入转化为更优化的计算流图以求更适应系统，在条件允许的情况下减少计算量以求更短的时间开销等。*动态需求下的实时反馈目标*注重应用执行过程中的动态性和调整能力，是多种性能的综合的、实时的需求。

(b) **系统资源方面**

在资源方面，系统层面的目标包括资源的有效利用、均衡开销、功耗等。*资源有效利用目标*注重系统中计算、存储、网络等资源的充分使用，或至少其中一方面达到峰值限制。*均衡开销目标*常见于分布式环境下，例如多个计算单元上的负载均衡、通信均衡且优化等。*能耗目标*常用于端侧，除了更高能效比的硬件之外，还需要更高效的计算流程或更精简的计算需求。

在实际运行中，资源层面的目标不仅需要考虑单一应用在资源上的动态执行，还需要考虑当前应用与其他应用的*协同运行*，资源的有效利用、均衡开销和功耗等目标都需围绕多应用协同而设计。

4.3 动态响应

面对来自多方的动态需求，需要运行环境提供动态响应的机制。在本框架中，动态响应机制由调度系统提供，如图 4.2中下侧绿色部分动态调整与调度。

4.3.1 响应时机

为了满足动态的需求，需要将动态调度融入执行过程中来响应模型调整。

心跳式调度 经过固定周期评价需求情况，调度调整。

抢占式调度 当有新的调度需求出现时，立即执行新调度。需要有监控环境以监测随时出现的调度需求。

不同的调度方式开销不同，如果需要支撑抢占式调度的监控环境响应越及时，其开销越大。

4.3.2 响应流程

包括近似和随动在内，有多种技术可以应对动态需求，但具体选择什么样的技术需要根据需求等因素做动态评估。设计评估流程如图 4.3。

首先需要确定实时的资源情况，并根据动态需求的具体目标计算出人工智能运行环境模型中的实际需求，例如对数据和模型执行的调整等；其次需要准备满足基本需求条件的技术集合，针对其中的每一项技术，评估该技术对需求目标的影响；最后需要根据

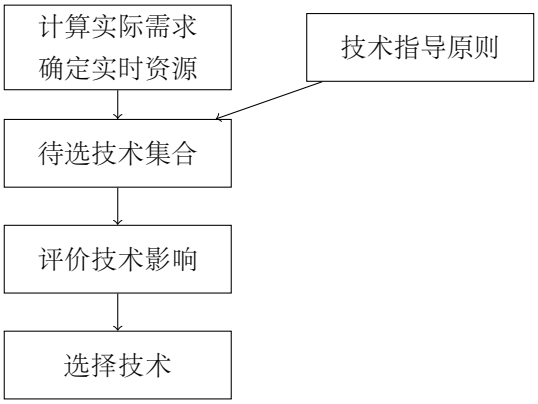


图 4.3　动态响应技术评估流程图

评估结果选择合适的技术或者技术集合。

4.3.3　响应要素

(a) **计算需求指标**

根据动态需求，需要准确计算或评估应用实时的计算需求和数据需求。计算需求反映为对硬件计算单元的需求，数据需求反映为存储容量、访存带宽、通信带宽和模式等需求。

(b) **确定软硬件资源**

需要评价已有的软硬件资源，判断实时目标的可实现性。

硬件资源的评价可通过前一章建立的多层次资源特征模型实现：通过实时监测系统获得硬件资源不同部分的当前参数信息。除此之外，还需针对应用需求从以下几方面讨论：*可容纳性*表示模型和数据组成的可执行单元能否容纳在一个计算单元之内，主要与计算单元的内存大小有关。*可计算性*表示上述可执行单元所需的计算时间、资源能够被所分配的计算单元满足，主要与计算单元的计算能力和计算部件有关。*可通信性*表示上述可执行单元与外部其他单元的互联能否被所分配的计算单元满足，主要与网络结构、带宽以及该节点所处位置有关。*计算通信比*表示上述可执行单元在单位数据内的计算需求和通信开销的比值。当需求的计算通信比大于实际硬件的计算能力与通信能力的比值时，应用是计算受限的；当需求的计算通信比小于硬件的计算能力与通信能力的比值时，应用是通信受限的。

软件资源的评价一般发生在第一次响应应用需求时，该评价一般是描述性的。如：支持哪些算法和模型，支持语言是什么，底层平台是什么，支持哪些硬件设备，可编程性，可扩展性，上层支持的接口等。如果大型系统的计算模型固定 (例如，迭代计算)，可以比较不同软件资源的执行效果。

当硬件资源充足，需要综合比较全系统的各个硬件单元时，它的评价指标与领域内应用密切相关。例如专门的图计算系统，会用单位时间边的吞吐量表示图计算的性能。还可以评价领域基本计算单元的执行效率，常见的深度学习性能指标有卷积层用时、深度神经网络基本算子 benchmark 等。

(c) **技术指导原则**

解决人工智能运行环境中的动态问题，可以追溯到人工智能的类脑本质——近似和随动。如表 4.5，近似的方法可以满足系统层面的多个目标，降低系统执行的多方面开销，但可能对应用层面的目标有损失。随动的方法可以实时调整状态，对资源利用率等方面的目标满足地更好，但实时调度可能导致响应开销较大。

表 4.5　技术原则举例

	优点	缺点
近似	降低执行开销	损失准确率等应用层面的目标
随动	实时响应，提高资源利用率	响应有开销，调整有代价

(d) **技术集合**

动态人工智能运行环境架构中，动态响应的主要技术包括：

动态调度 最基本的响应策略是将计算调度至最适合其执行的结构上运行。将模型合理调度至硬件环境中执行有多种方式，无论是框架、库，还是语言－编译器－运行时，都需要明确任务需求和资源环境能力，并做合理的分配和调度。

动态并行化 数据和模型规模的增加以及单机系统环境的资源受限导致了应用的并行/分布式需求。典型的应用并行策略包括：数据并行、模型并行、数据拆分、图并行。

近似计算 当应用层面的目标降低、系统层面的目标提高 (例如降低分析的准确程度但用时更少) 时，通过近似的方法可以达到要求。近似计算可以用在对数据的分析上和对模型的调整上。

4.4 协同执行

人工智能运行环境大多以层次化的软件栈实现应用的执行，如图 4.2中左侧蓝色列表。为了满足动态运行环境的需求，协同执行需要实现“动态响应”的内容，完成调度。

包含数据 (多维数据特征模型) 与计算 (多层次模型表示) 的“人工智能模型表示”是协同执行的要素之一。协同执行需要将人工智能模型表示通过运行环境调度至当前实时条件下的硬件资源完成实际执行。

本节将讨论模型表示和模型到硬件的映射方式。

4.4.1 模型表示

人工智能运行环境中执行的起点一般为模型表示 (Model Representation)。由于人工智能领域框架众多，描述计算的方法也多种多样，模型描述的方法依赖于模型实现的载体。

面向神经网络的模型表示已由 ONNX 作为中间表示所统一，可以相对便捷地互相转换。面向其他机器学习方法的模型表示在各环境中都有 API 支持，ONNX 也试图将其融入自己的体系定义了一套表示环境 ONNX.ai。

表 4.6　模型表示方法列举

表示方法	支持环境	支持模型
keras API	keras, tensorflow	神经网络
tensorflow API	tensorflow	神经网络
prototxt	caffe, caffe2	神经网络
mimodel	coreML	神经网络、树、SVM 等
Spark ML	spark	机器学习
scikit-learn API	scikit-learn	机器学习
LightGBM	LightGBM	树
XGBoost API	XGBoost	基于梯度的机器学习
ONNX	pytorch, caffe2, CNTK, MXNet , scikit-learn 等	神经网络，机器学习

模型表示的主要 API 设计依赖于基本算子和基本执行方式，模型的内容包括计算表示和参数数据。

已有的模型表示是串行描述的、单一 (或少量) 文件的，计算和参数或合并描述或分开描述。由于常见的模型描述规模有限，如此简单描述暂可以应对，但若未来模型体量爆炸式增长，需要复杂的、分布式的描述，还需要对现有的表示方法作出适当调整。

基于 API 或者标准格式的模型表示针对计算执行之前的描述或者训练好模型的传递，对象是静态模型。模型的动态特征往往通过框架或者运行时系统的内部实现，缺少可交互或迁移的表示方法。动态特征对于交互的需求还不明显，目前基于运行时系统内部的表示能满足其一般需求。但若未来的模型需要交互式的动态表示方法，需要对现有的模型表示添加动态描述能力，例如时间维度的表示，或扩展的 API。

4.4.2　动态调度映射

明确模型表示方法之后，模型的构建需要充分分析和分解需求，并融合输入数据。动态模块构建在 3.2.5中已有讨论。模型调整需要综合上层的应用需求和下层的资源环境变化，可以由用户直接动态干预，也可以由用户设定基本规则，在执行阶段自动调整。

模型映射至具体的体系结构有 AOT(Ahead Of Time) 和 JIT(Just In Time) 两种方式，不仅与应用的需求和特征有关，也与包括软硬件在内的系统环境有关。

AOT 模式下，所有的映射都在执行前完成，而 JIT 模式可以边执行边映射。在大多数条件下，这两种方式区别不大，都能完成已有简单模型的执行，其编译所在的硬件环境是 (或可以精细控制) 执行所在的环境。例如，人工智能编译器 TVM[97] 的主要模式是 JIT，可以在编译的同时调度执行，控制 CPU，通过远程过程调用 (Remote Procedure Call, RPC) 机制调度 GPU(服务器端、嵌入式)、FPGA 等一系列硬件。一些大型应用，不需要 JIT 方式交互，只需要 AOT 方式部署至环境执行；而验证性的、规模有限的应用需要实时更新执行内容并了解执行结果，JIT 方式更恰当。

系统环境复杂多样，许多大型系统只能通过 AOT 模式交互，而很多小型或嵌入式

系统需要 AOT 模式提前优化。例如典型的超级计算机环境中，应用需要提前部署、一次执行，没有复杂的交互过程。嵌入式终端设备上受限于性能和功耗，运行的人工智能应用需要精简优化，常见于推理过程。

至于支持动态性的能力，客观上 JIT 方式可以实时看到硬件状态以决定映射策略，但 AOT 模式同样可以将动态策略预先设计使整个运行时系统在 AOT 生成并融入执行环境，运行过程中并不因模型映射的时机有所差异。

无论是模型构建、表示，还是模型的调整指令，都是描述过程 (即使有动态调整)，不涉及到执行过程，因此可以通过 API 定义，也可以通过领域专用语言来表示。

为了形成执行闭环，在以上执行过程中，还需要实时监测系统状态和动态需求，并作适当的应对。

4.5 小结

本章设计了基于“需求感知、动态响应、协同执行”闭环的动态人工智能运行环境架构，该架构描述了由数据、模型和资源组成的动态人工智能运行环境的运行机制。

在“需求感知”部分，我们分析了应用层面和系统层面的目标，并形成了目标矩阵。应用层面的目标常被分解为多个子问题的目标，例如回归问题、分类问题、聚类问题、图模型等，目标多以计算参数达到阈值的形式体现。系统层面的目标不仅包括当前应用的执行效率，还包括系统本身的资源利用、功耗以及系统中多应用的协同效果等。

针对“动态响应”，我们首先讨论了响应的时机，之后提出了响应流程。在响应流程中，需要计算实际需求、确定实时资源，并选择响应的技术。我们讨论了近似和随动的技术指导原则，用以应对人工智能中的动态需求。最后我们列举了典型的响应技术如直接调度、并行、近似等。我们将在下一章基于具体应用讨论响应技术。

在“协同执行”中，我们讨论了执行的对象——模型表示，和执行的方式——动态调度映射。由于执行需要针对具体的应用设计，具体的执行策略将在下一章讨论。

第5章　动态人工智能运行环境关键技术

前一章讨论了动态人工智能运行环境架构，并描述了其运行机制。本章将讨论该架构中的关键技术。

本研究力求选取的关键技术涉及动态人工智能运行环境的核心，全面涵盖数据、模型、资源三个组成部分。针对数据，本研究选择数据执行过程中的近似估计这一具有执行动态性的任务；针对模型，本研究选择模型执行过程中稀疏调整这一动态执行方式；针对资源，本研究分别选则人工智能应用执行的两个阶段——预处理与实际执行，分别选择资源的两个层次——系统级和芯片级，研究资源约束条件下任务的拆分优化与调度。具体的关键技术如表 5.1。

表 5.1　关键技术选择

	技术	技术原则选择
针对数据	基于邻域的局部近似方法	近似
针对模型	基于动态指导的模型稀疏化方法	近似 + 随动
针对资源	自适应的协同预处理调度策略	随动
针对资源	基于人工智能计算流图的动态自适应调度模型	随动

第 5.1到 5.4节将分别从以下几个角度讨论面向动态特征的人工智能计算环境中的关键技术。

- 原始数据的分析。针对人工智能应用中规模比较庞大的一类关系型数据，提出基于邻域的局部近似方法。
- 模型动态稀疏化。针对模型可调整、可稀疏化的动态需求，结合体系结构环境变化和用户需求，提出基于动态指导的模型稀疏化方法。
- 人工智能预处理。将人工智能数据预处理的需求融入人工智能运行环境中协同设计，并对超级计算机系统这一具体的体系结构环境提出协同预处理方案。
- 全流程动态调度。利用对数据、模型和体系结构的动态综合建模，设计具有动态自适应功能的人工智能调度模型。

5.1　基于邻域的局部近似方法

关系型数据 (图数据) 是人工智能领域的重要输入，也是近来人工智能扩展的重要方向。如何完成图嵌入问题，将图数据转化成为人工智能中神经网络的典型输入是一

个难点。已有的方法主要在于全局关系向量化，但当图规模极大时，遍历全图的成本极高。由于数据的关联性强且关联特征不确定，大规模图数据的分析是一个动态问题，近似方法可以在一定程度上完成图的部分分析，尤其是估计图中的点在全局关联数据中的中心度位次。因此利用局部信息粗略地估计某节点在全局中的中心度是一个值得研究的问题。

本节提出一种简单而轻量的算法来获得节点的近似中心度。给定采样的邻居或节点的邻域，该算法可以估计它在整个图中的近似排名，确定它在图结构中是否重要，以及重要程度。给定节点的子集及其对应领域，该算法可以快速对这些节点进行近似重要性排序。它可以通过采样简化并扩展到分布式环境。该算法基于图数据中节点的邻域和语义信息，由于只得出更少的信息，其计算量相比如 PageRank 等全局算法少了数量级，并且可以通过优化进一步减少计算。此方法在超大规模关联数据估计和边缘计算中将发挥重要作用。

5.1.1 中心度邻域分析

关系型数据 (图数据) 由点和边组成，在本节中会根据情景同时用点、节点、顶点表示图中的点，用边、连接表示连接两个点之间的边。为了分析图数据的中心度，采用表 5.2中的图数据 (来自于 SNAP[16]，Clueweb[108]，Sogou[6])，这些数据由维度关联关系 (2 维) 构成，数据本身没有其他维度的属性信息，但连接关系中蕴含了一定的邻域信息。由于图数据只包含结构信息且目前人工智能的图嵌入数据规模还不是特别巨大，这些图计算中经常使用的标准数据类别涵盖人物/社会关系、信息索引关系、地理交通关系等多种关联关系，也可以为大规模人工智能图嵌入等关系分析提供一定的参考。

表 5.2 图数据集

Graph	# nodes	# edges	Description
p2p-Gnutella04	10,876	39,994	Peer to peer network
web-NotreDame	325,729	1,497,134	Web graph
roadNet-CA	1,965,206	2,766,607	Road network
web-Google	875,713	5,105,039	Web graph
ClueWeb12	978.4M	42.57B	Web graph
Sogou	271.9B	12.25T	Web graph

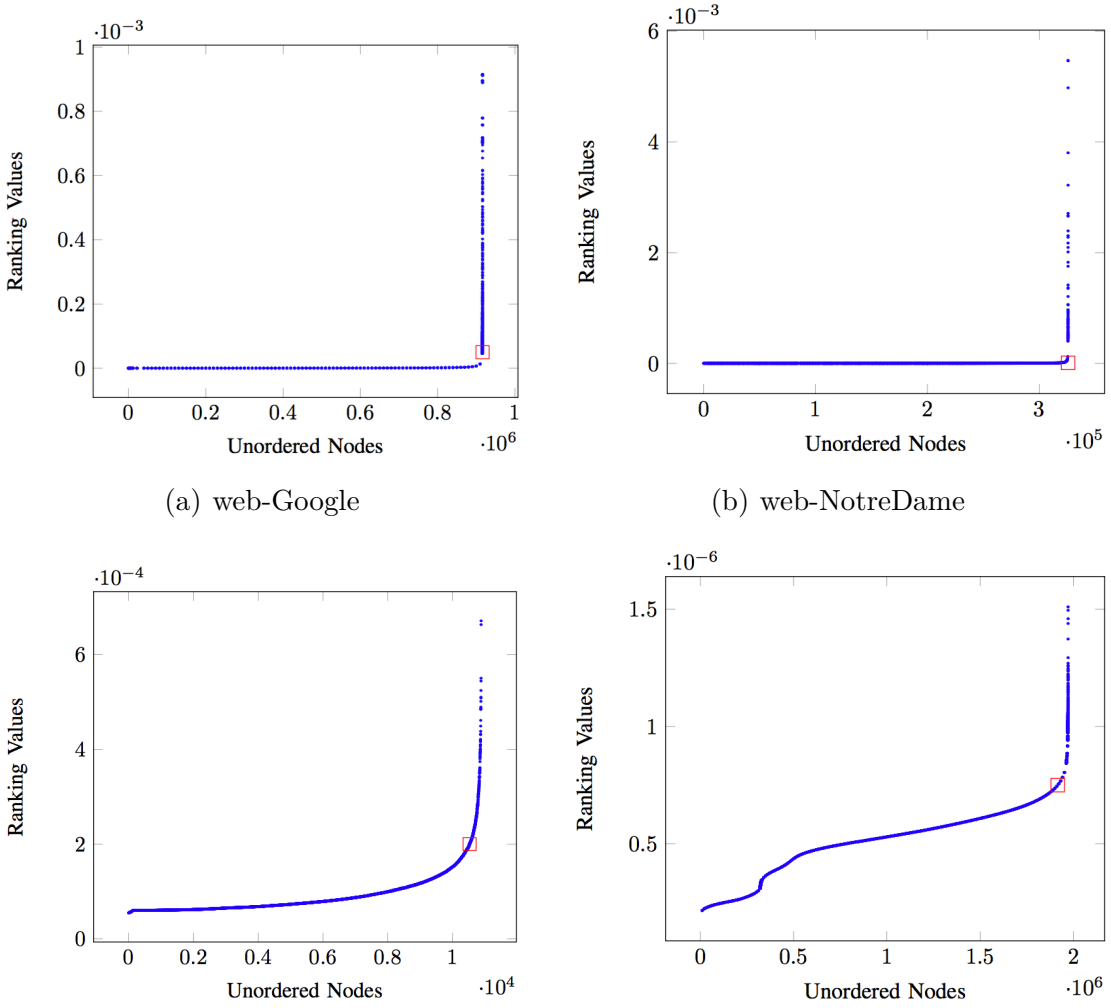
M: million, B: billion, T: trillion

(a) **中心度分布**

中心度的分布可以启发中心度估计方法的设计。

经过基本 PageRank 算法后，不同图中各节点的中心度分布如图 5.1，所有节点的 rank 值的和是 1。大部分节点的 rank 值 (即中心度的度量值) 较小，只有极少部分节点

的 rank 值较高。其分布比“幂律”分布[160] 更倾斜。



(a) web-Google (b) web-NotreDame

(c) p2p-Gnutella04 (d) roadNet-CA

图 5.1 图中节点的 rank 值分布 (PageRank 算法度量)

web-Google 图中，rank 值大于 10^{-5} 的节点数量小于 1%，只有 33 个节点 (0.003%) 的 rank 值大于 5×10^{-5}。web-NotreDame 图与前者分布相同。p2p-Gnutella04 图中，只有 2 个节点的 rank 值极大，而 rank 值较大的节点少于 10% ；前 10% 节点的 rank 值的和是 0.0384 。在 roadNet-CA 图中 (图 5.1(d))，居于“中间 rank 值”的节点不可忽略，高 rank 值的节点数仍小于 3%。设 rank 分布图的二次拟合曲线中斜率变化最快处为“分割点”，高 rank 值和低 rank 值的分割点如图 5.1中红色方块所示。

表 5.3 中列举了图 5.1中 rank 值高于阈值的节点比例。可以看出，该比例极小。在图结构分析中，高 rank 值的节点更有意义，在关联数据中属于重要节点，在图网络中也是需要重点考虑划分关系的节点。因此，在图节点结构属性近似中，应该着重区分高 rank 值的节点与其他节点。

表 5.3　高 rank 值节点比例

G	web-Google	web-NotreDame	p2p-Gnutella04	roadNet-CA
T	$5 \times 10{-}5$	10^{-4}	2×10^{-4}	$7.5 \times 10{-}7$
P	0.003%	0.0052%	3.25%	2.8%

G: Graph; T: Threshold (rank value); P: Proportion

(b) **邻域节点分布**

邻域 对于图中的绝大多数节点而言，其邻域极小。我们分析了 SNAP[16] 数据集中的 68 张图，其中的 98.5% 平均度数 (每个节点的平均邻居个数) 小于 100，94.9% 的平均度数小于 50，即使表中最大图 Sogou 的平均度数也只有 45。

节点间的距离 为了表示图中节点的特征，我们首先定义“节点间距离”。此处的节点指的是图中任意两个节点，它们可以有边连接也可以没有。表示图中节点之间距离的方法很多，例如：

文字距离 Literal distance: 产生的来源是遍历图。如果图中节点的序号是通过遍历获得的，那么该图是遍历图，两个节点间的文字距离即二者序号的差值。典型的遍历图包括网页链接图 (web graph) 和部分引用图 (citation graph)。

语义距离 Semantic distance: 图中的节点可以具有语义属性，例如道路图 (road graph) 中的坐标，网络图 (web graph) 中节点的属性——url 和引用图中的作者单位 (organization of authors)。这些语义属性之间的距离可以根据实际情况提前定义。

边距离 Edge distance: 图中的边上若有属性，此属性可能与距离相关。例如在交通图或道路图 (road graph) 中边代表地理距离。然而，边距离相比于文字距离和语义距离，能提供的“全局”信息有限，甚至不包含任何全局信息。

图 p2p-Gnutella04 是遍历图，其索引本身有距离的含义。图 5.2列举几个不同 rank 值的节点上邻域节点的分布。

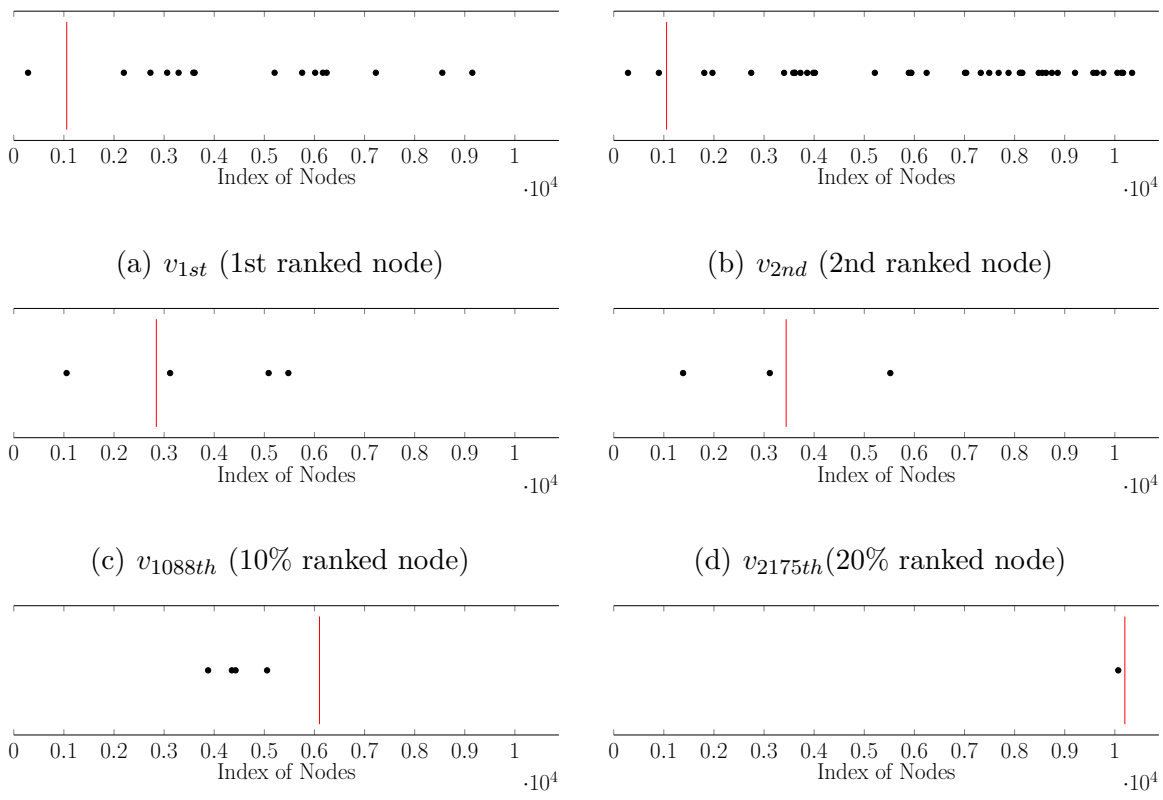
如图 5.2，红竖线表示当前节点的索引所在位置。高 rank 节点的邻居 (入边邻居 in degree neighbors) 分布在整个索引范围内近乎均匀 (广泛)，邻域内节点个数远多于其他节点 (rank 最高的节点包含 15 个邻居，rank 值第二的节点有 40 个邻居，所有节点的平均邻居个数为 4)。rank 值居中节点的邻居数要远小于 rank 值较高的节点，而其邻居分布更具有局部性。低 rank 值节点的邻居数更少，分布更局部。

(c) **邻域内的归类度量**

邻域节点的分布可以为该节点在图中的中心度提供重要参考，然而即使是邻域节点的分布也需要大量详细的信息，将邻居节点归类度量可以简化分布信息。

节点的类别: 可以通过语义间接获得，例如将语义属性或文字属性 (文字距离) 分槽，根据每个节点到索引最小节点间的距离将节点按照所属槽归为不同的类别。

将 Sogou 图中的点按照索引分为 40960 个类别，索引为 i 的点将被划分进第 c_i 类，其中 $c_i = \lfloor i/40960 \rfloor$。随机选取了 92 节点，分析它们邻域所含节点分布。



(a) v_{1st} (1st ranked node)　(b) v_{2nd} (2nd ranked node)

(c) v_{1088th} (10% ranked node)　(d) v_{2175th}(20% ranked node)

(e) v_{4350th} (40% ranked node)　(f) v_{8701st} (80% ranked node)

图 5.2　p2p-Gnutella04 图数据中典型节点的邻域节点 (in degree) 分布

图 5.3中横轴表示选择的 92 个点，纵轴表示该点邻域内节点的分布情况——非空类比例。一个点邻域内的每个节点都属于前述定义的 40960 类中的一类，对每个类别所包含的该邻域节点计数，若计数值大于 1，则该类别非空，所有非空类别个数占全部 40960 类的比例即为**非空类别数**。

最右侧点的邻居个数 (邻域内节点数) 为 99,150,514, 这些邻居分布在所有 40960 类中的 33696 个类别中，这也意味着该点邻域内节点分布近乎均匀。如图 5.3所示，只有 3.2% 的节点其邻域节点分布在超过 3/4 比例的类别中, 78.9% 节点其邻域节点分布在少于 0.9% 比例的类别中。

5.1.2　关系型数据中近似中心度度量方法

本节提出基于邻域信息的近似中心度度量方法。度量值可以用数值、排序位次或其他参数表示。

(a) 方法描述

方法的主要思想是为节点的邻居个数和邻域节点分布打分。节点 v_i 的得分可以用函数 f 表示:

$$S_{v_i} = f(|Neib(v_i)|, Distribution(Neib(v_i))) \tag{5.1}$$

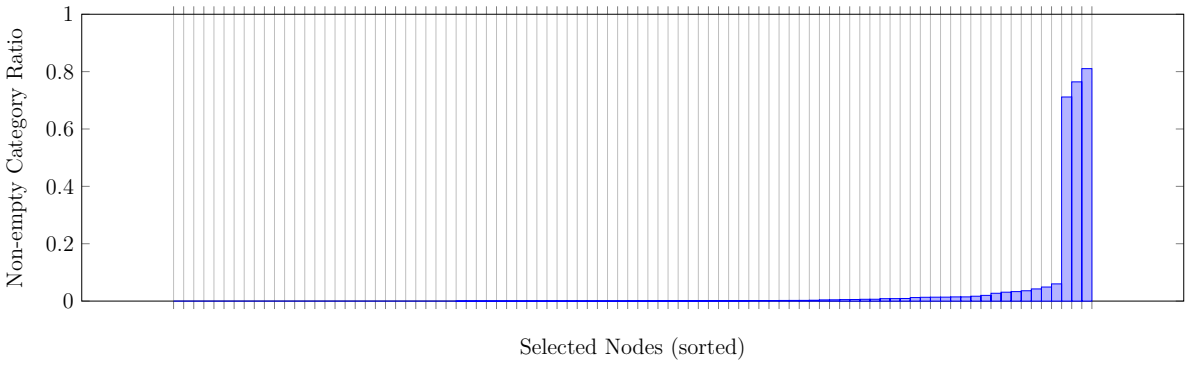


图 5.3　Sogou 图中所选点中邻域内非空类别的比例

(b) **归类技术**

为了描述邻域内的分布，本文提出一种基于距离的归类技术。

假设要将所有节点归为 $|C|$ 类，点 v_i 包含邻居 v_j，二者距离是 $dis(v_j, v_i)|$，则 v_j 所属类别 c_j 可以表示为:

$$c_j = \frac{|C| \cdot dis(v_j, v_i)}{\max_{v_k \in Neib(v_i)}\{dis(v_k, v_i)\}} \tag{5.2}$$

当计算出节点的所有邻居类别后，可以简单定义函数 f 为计算每一类中数量的加权和。如:

$$f(|Neib(v_i)|, Distribution(Neib(v_i))) = \sum_i w[i]|C[i]| \tag{5.3}$$

线性函数较为简单，此外还可以用基于多项式的函数定义 f。

在一些图数据中，节点邻域内类别分布不同，不同的分布导致该节点在全图中的 rank 值不同，因此可以通过机器学习或者预定义来得到函数 f。

算法 5.1 Approximate ranking algorithm

输入: graph, node v_i, number of vector category

输出: Rank Value x_i of node v_i

1: Graph formatting;
2: Initialize tmp vector C;
3: for v_j in neighbor set of v_i do
4: $\quad c_j \leftarrow$ category of v_j ;
5: $\quad$ add v_j to $C[j]$;
6: end for
7: $x_i \leftarrow f_{valuefunction}(C)$;

算法 5.1将指定节点的邻域归类。与基于中心度的算法 (如 PageRank) 类似，节点最终的 rank 值由其邻居决定。除了距离之外，其他属性信息也可以作为归类的依据。

在该算法中，邻域内所有的节点都被归类至向量中，作为函数 f 的输入。由于数据没有依赖关系，其中的 for 循环可以完全并行化。

算法的并行包含两个层次：节点邻域内部的并行以及指定节点集中所有节点的并行，如算法 5.2和5.3。

算法 5.2 Distributed approximate ranking algorithm for one node

parfor v_j in subset of $Neib(v_i)$ do
    $c_j \leftarrow$ category of v_j ;
end parfor
$x_i \leftarrow f_{valuefunction}(c_1, c_2, \cdots, c_j, \cdots, c_{|\Gamma(v_i)|})$;

算法 5.3 Distributed approximate ranking algorithm for a subset of nodes

输入: graph, subset G_{sub}, number of vector category
输出: Rank Value x_i of node v_i

Graph formatting;
parfor v_i in G_{sub} do
    Initialize tmp vector C;
    for v_j in neighbor set of v_i do
        $c_j \leftarrow$ category of v_j ;
        add v_j to $C[j]$;
    end for
    $x_i \leftarrow f_{valuefunction}(C)$;
end parfor

然而，如果图的每个节点都需要计算 rank 值，那么基于传播的算法（如 PageRank）仍然是优先选择。

(c) 采样技术

当图的规模增加时，高度数节点的邻域极大、访问代价高。通过采样技术可以减少实际访问邻域节点数量，减少计算量。

采样的方法多样，简单的随机抽样或系统抽样足以达到效果。设定采样阈值 t，当邻域内节点数超过该阈值时，开始采样，且样本数设置为 t。如果邻域内有多个类别，则按类别采样，且采样可并行。

以下将列举一个简单的采样方法：在实数 $[0,1)$ 范围内，随机选取 t 个数 $x_1, x_2, \cdots, x_t$。设邻域内的所有节点可被重新编号为 $v_0, v_1, \cdots, v_{n-1}$，我们选择其中的 $v_{\lfloor x_1 \times n \rfloor}, v_{\lfloor x2 \times n \rfloor}, \cdots, v_{\lfloor x_t \times n \rfloor}$ 作为采样点。为了保证对数据分布的合理推断，需要保证采样的随机性和科学性。

对于邻域较小的节点，采样的需求不大，且不当使用可能会导致错误。

(d) **复杂度分析**

基本算法的计算复杂度是 $O(|Neib(v_i)|)$，其中 $Neib$ 表示邻域。该计算复杂度远小于一般中心度度量方法的计算复杂度 $O(|E|)$，其中 $|E|$ 表示图的边数。例如，常见的网页链接图 (web graph) 边的规模会达到万或者亿的级别，而平均邻域规模只有 12[16]。且在真实数据中，度数分布具有幂律特性[161]，只有极少部分节点有较大邻域，从而需要采样技术。因此，$O(|Neib(v_i)|)$ 的实际值很小。此外，如果大规模图在分布式环境中分布恰当，那么对于高度数节点邻域的采样没有数据依赖，容易并行，且并行负载较为均衡。

5.1.3 实验分析

实验需要证明方法的正确性和有效性。

评价函数 f 设为:

$$f(C) = w_a \sum_{j=1}^{j<=t} z(c_j) + w_b g(C) + w_c Neib(v_i) + w_d |Neib|_{avg} \tag{5.4}$$

其中, w_a, w_b, w_c, w_d 是权值, $C = \{c_1, c_2, \cdots, c_t\}$,

$$z(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

该函数的参数包括目标节点邻域中的非空类总数、每个类中的元素个数，其邻域规模，以及全图的平均邻域规模。

(a) **实验环境和实验数据**

计算节点包含 64 个 16 核 Intel Xeon E5-2680 的服务器，64GB DDR4，300GB 存储空间，读写带宽 1GBps，操作系统为 RHEL6.6。

实验数据采用表 5.2中的数据。

(b) **正确性验证**

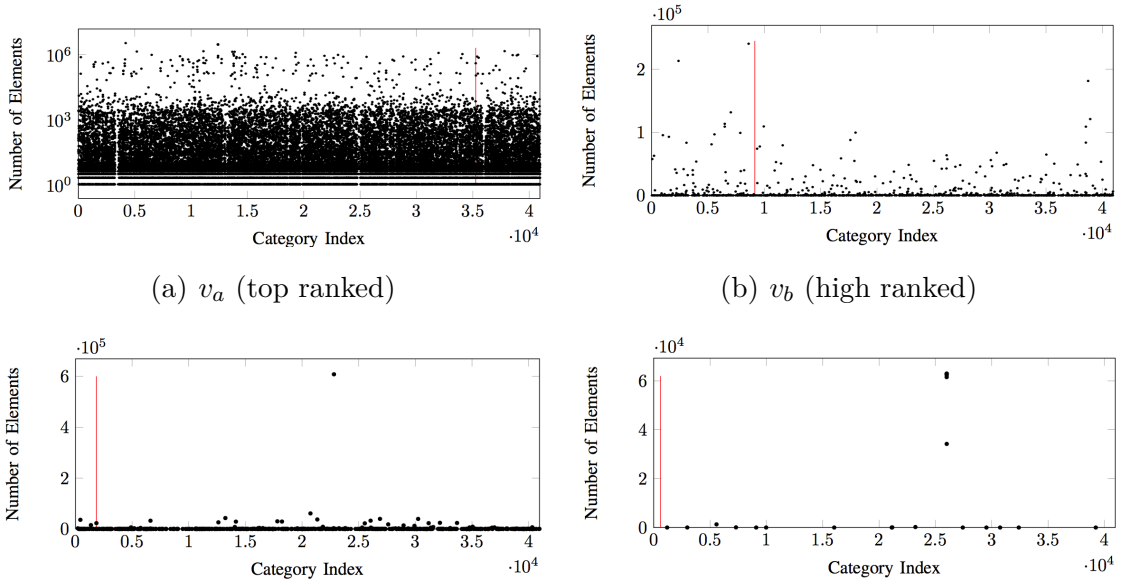
实验可以有效地区分出高 rank 值节点和低 rank 值节点。节点原始的 rank 值越高，估计越准确。从图 5.1可以看出，只有不到 1% 的节点具有高 rank 值。

通过本方法成功识别出给定 4 个实验图 (web-Google, web-NotreDame, p2p-Gnutella04, roadnet-CA) 中 rank 值前两位的节点。而 rank 值排在前 0.1% 的节点中，本方法可以成功识别出 91%，并归类为高 rank 节点，剩下的被识别为中等 rank 值的节点，与实际 rank 值差距不大。

(c) **归类技术评价**

在更大规模的图 Sogou 中，我们进行了基于分类的评价。

Sogou 图是基于网页链接的图，且节点编号按照 url 排序 (host address + url semantics)，因此其索引具有语义信息。如图 5.4，每个节点的邻域都按照同样的归类方法被划分为 40960 个类别，我们选取了代表性的几个节点 (v_a, v_b, v_c, v_d) 来说明其邻域分布。



(a) v_a (top ranked)　(b) v_b (high ranked)

(c) v_c (middle ranked)　(d) v_d (low ranked)

图 5.4　图节点邻域 (Sogou 图)

实验中的评价函数 f 与前述定义类似。最高 rank 值的节点邻域非空类多 (76%) 且每一个类中元素多。较高 rank 值节点邻域非空类数量远小于前者，但仍不可忽视 (2.7%)，每一类中的元素虽然远小于前者元素，但仍远大于均值。中等 rank 值节点邻域非空类数量更少 (1.4%)，每类中元素更少。而低 rank 值节点邻域规模极小且局限在极小范围内 (非空类 0.04%)。

(d) 采样评价

在 Sogou 图中，最高度数的节点邻域包含 3 万亿条边 (个邻局节点)，此规模是全图规模的 25%，正如图 5.4(a) 所示。大图存储在分布式环境中，且计算环境也是分布式结构，针对该点和类似节点采用分布式采样技术。

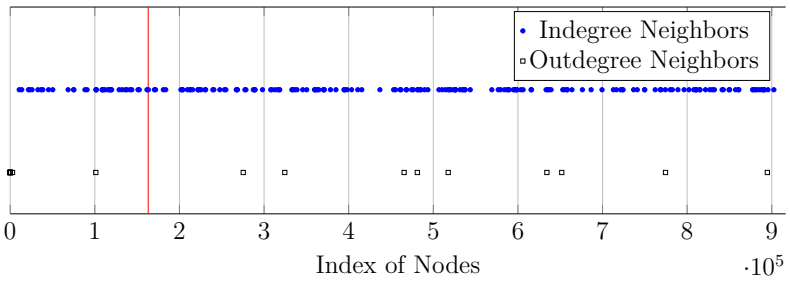
由于图按照节点序分块存储，因此本实验中用基于固定步长 (针对节点序号) 的采样技术。在绝大多数情况下，采样可以正确地评估节点邻域。

图5.5(a) 刻画了 web-Google 图中的一个高 rank 节点的邻域。经过采样得到图5.5(b)，203 个入邻居中有 11 个被选择出来，这 11 个样本均匀地分布在如图中 x 坐标的 9 个分割槽中，分布与原始分布几乎相同。经过评价函数 f，该节点仍被评价为高 rank 值节点。实验证明了采样技术在保证近似结果的基础上可以显著降低计算量。

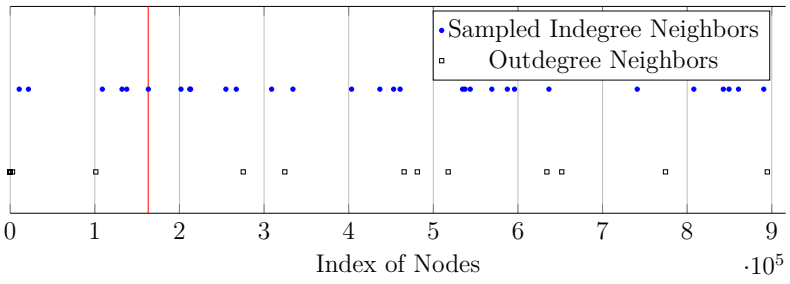
(e) 性能评价

Sogou 图规模巨大，有序存储为 1024 个规模相同的块。采用 CSC (Compressed Sparse Column) 格式存储，而转化为 CSR (Compressed Sparse Row) 格式代价巨大。通过 64 个分布式计算节点处理邻域，由于没有数据依赖关系，所有的计算完全并行。

读入全图需要约 1 小时。如表 5.4：执行 PageRank 需要超级计算机 1 小时的数据读入和 20 分钟的计算；采用近似方法，对于全图度数最高的点，只需要在 64 个计算节



(a) Entire Neighborhood



(b) Sampled Neighborhood

图 5.5　高度数节点采用采样技术前后邻域信息对比 (web-Google 图)

表 5.4　近似算法性能比较

方法 (数据)	**读入时间**	**执行时间**	**执行环境**
全局 PageRank(Sogou)	1 hour	5 轮 × 5min/轮	超级计算机 (40960 个节点)
Bressan 算法[61] (roadnet-CA)	–	1s	Intel Xeon E5620
		0.08s	NVidia GTX 1070
本文近似算法 (Sogou，图 5.4中 v_a 点)	1 hour	5 min	64 个计算节点
本文近似采样算法 (Sogou，图 5.4中 v_a 点)		1 min 以内	1 个计算节点

点的集群上执行 5 分钟，其他点开销更小，平均只需要不到 10 秒；采样技术也会显著降低开销。

5.1.4　小结

本节面向关系型数据提出了基于邻域的局部近似方法：对邻域进行基于语义的分类，通过价值函数准确的近似了关系型数据中局部节点分析的动态问题。该方法计算开销相比于全局方法减少了数量级，对于千亿条量级的数据，在 64 个节点的计算环境中，平均每个点的近似分析只需不到 10 秒，图中最大点的分析只需 5 分钟。

但是，对邻域进行分类是一项艰巨的任务，需要多次试验。评估函数的设计需要依赖于专业的知识。

5.2 基于动态指导的模型稀疏化方法

前文已讨论过人工智能模型执行具有多层次的动态特征，其中稀疏性也体现在模型的多个方面。在实际应用中，常常会遇到资源受限、时间受限、精度受限的人工智能训练需求，且这些需求可能是实时产生、动态变化的。利用稀疏化的方法，可以为实际应用执行时复杂多变的情况提供高效的应对策略。它可以压缩模型、减少数据需求、减轻计算负担，甚至可以更方便、更快捷地得到模型的执行反馈。此外，一些应用本身就需要获得稀疏的模型，但直接稀疏设计可能无法达到期待的结果，在执行过程中的稀疏化是一种较为经济便捷的方式。

本节将利用人工智能模型的可稀疏特征，提出基于动态指导的模型稀疏化方法，以应对动态人工智能运行环境中的模型动态调整问题。本节将首先分析模型的可稀疏性，接着分析常用的稀疏化策略，随后根据人工智能动态运行环境中的需求提出基于动态指导的模型稀疏化方法并进行实验。

5.2.1 人工智能模型的稀疏特征分析

直观地理解，“稀疏特征”与模型中“0”的比例关系密切。而人工智能模型包括数据、模型结构等多个方面。因此首先初步定义：

稀疏度：单位计算结构中，0 作为参数所占总参数的比例。在人工智能模型中，数据稀疏度包含输入数据、模型参数等数据类别的稀疏度，结构稀疏化通过参数设 0 使之不发挥作用。

为了对人工智能模型中的稀疏度有初步的理解，首先分析各种典型模型的结构和参数。这里仍用 ONNX 模型集合[36] 中的模型作为基本的分析对象，该模型集合中包含预训练的模型参数，也可为典型的参数分布分析提供参考。

模型的参数稀疏度来自两个方面：人工智能领域的模型设计者所设计的稀疏模型和运行时系统对于模型进行的稀疏化改动。前者需要专业的知识，而后者是我们研究的重点。本节将在没有应用知识的条件下，让运行时系统尽可能地对模型进行合理的稀疏化改动和执行。

基本算子的稀疏性分析在第 3.2.7节已有陈述。

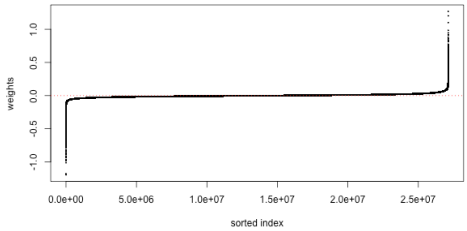
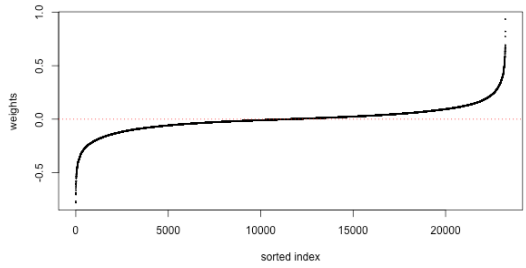


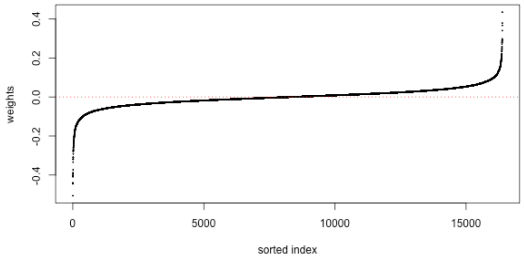
图 5.6　GoogleNet Inception V3 中参数分布情况

图 5.6是 GoogleNet Inception V3 预训练网络所有参数的分布情况。从图中可以看

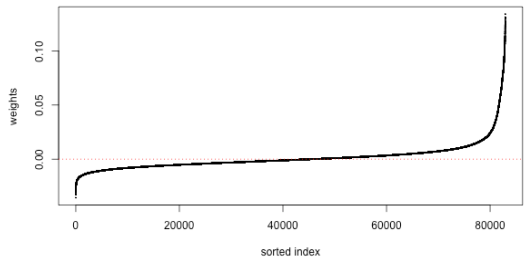
出，绝大多数参数在 0 附近，只有极少数参数的绝对值较大。实际上只有 0.4% 的参数绝对值大于 0.1(即最大参数的 10%)，44.3% 的参数绝对值小于 0.01(即最大参数的 1%)。



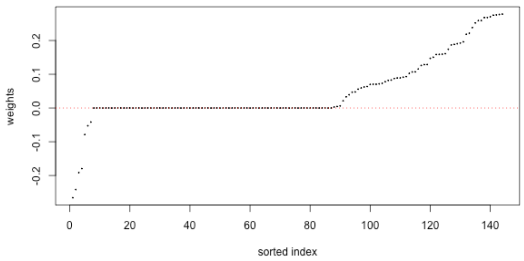
(a) AlexNet conv1



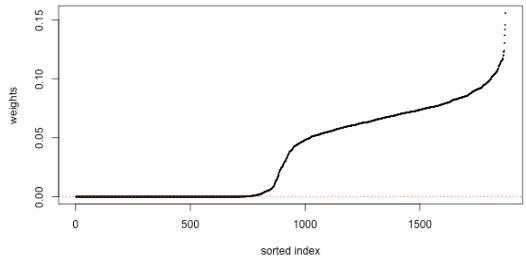
(b) GoogleNet Inception V3 Branch3 conv2d



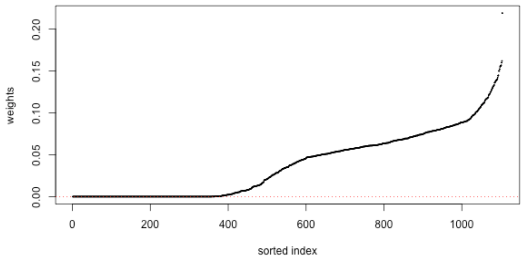
(c) DenseNet161 denseblock4 denselayer18 conv2



(d) DenseNet161 denseblock1 denselayer2 norm1



(e) DenseNet161 denseblock3 denselayer32 norm1



(f) DenseNet161 denseblock4 denselayer2 norm1

图 5.7　典型模型中算子的参数分布

图 5.7(a-f) 分别刻画了典型预训练模型中卷积和归一化算子的参数分布。

针对 AlexNet 网络 5 个卷积单元，GoogleNet Inception V3 网络 98 个卷积单元和 DenseNet161 网络 160 个卷积单元的参数分析：大部分参数都在 0 附近，只有少部分绝对值较大的参数。绝对值较大的参数在算子参数中的比例在 5% 到 25%，因此接近”0”的参数比例在 75% 以上。

归一化单元只在部分模型中存在，其中参数有 30% 或以上是 0，其余数据呈近似线

性分布。

从以上例子可以看出，人工智能模型中的确有一定比例的数据等于 0 或接近 0。等于 0 的数据自然可以不做计算，而接近 0 的数据能否直接近似成 0，下文将进行进一步实验。

5.2.2 模型稀疏化策略

稀疏性表示计算单元本身的稀疏度，而通过一些稀疏化方法，可以增加计算单元的稀疏度从而减少数据量、计算量或模型连接关系。

在传统的机器学习领域，稀疏化方法包括 L1 正则化、稀疏贝叶斯方法等。这些方法也是以深度神经网络为代表的人工智能模型稀疏化的重要指导。

(a) **数据稀疏化**

L1 正则化 L1 正则化与 L2 正则化一样，是权重衰减的常见形式。一般的 L1 正则化的主要操作是求权值向量中的各元素绝对值之和。用在损失函数的计算中，原始参数需要与带系数的 L1 正则化项求和，例如 $X' = X + \lambda \sum |w_i|$。因此，后者用在损失函数项，以对参数进行一些限制。在传统的机器学习中，L1 正则化可以产生稀疏模型 (即稀疏的权值系数)，被用来进行特征选择。

稀疏贝叶斯方法 稀疏贝叶斯方法作为实现参数的简约表示用在回归和分类中，意图只保留很少非零权重作为参数模型。稀疏贝叶斯学习中，假设所要求解的对象符合参数化的高斯分布，通常会设置某个阈值，将趋近于 0 的参数置为 0（稀疏化）。这样，在各已知条件和对解的先验分布进行假设的基础上，利用贝叶斯规则可求得后验高斯分布的参数，所求的解由后验高斯分布的平均值给出。

掩码法 通过设置掩码 (mask)，将数据中符合掩码约束的部分数据过滤出来进行计算，从而忽略其他数据。掩码法还有包括特征提取等不同的计算方式和变形。

参数量化 通过用更少的比特数表示参数，可以实现更小的内存开销。

(b) **结构稀疏化**

人工智能模型中，结构也由参数来表示。结构稀疏化指：将靠近 0 的参数修改为 0，以删除在神经元意义下的该神经元。

结构稀疏化是压缩神经网络的重要方法，但经过结构变化的神经网络往往会面临精确度下降的问题，一个解决方案是将训练数据重新输入压缩网络训练，优化参数。

另外一种结构稀疏化的方法是直接训练具有稀疏连接特征的神经网络 (稀疏进化训练)：在训练的初始阶段，确定好神经元数量之后随机设定一个稀疏连接；每一轮训练之后删除权值最小的连接，再随机加入一条新连接；直到某一结束条件。此方法易扩展，可适应任意大的模型[162]。

此外，还可以将大数据上广泛使用的关联数组代数 (Associative Array Algebra) 应用于深度神经网络，以构建更大规模的稀疏神经网络[163]。

(c) **执行稀疏化**

人工智能模型的执行有多种迭代方法：SGD, Adagrad, Adadelta, Adam, Adamax, Nadam 等。在执行阶段，稀疏化有多种方法。其一是惰性参数更新，例如并行执行的模型参数可以经过多轮之后再更新。此外，在训练期间执行动态稀疏图[164]，每次迭代中仅选择少量具有高选择性的神经元，可以显著节约内存并减少操作数量。

对于机器学习领域的算法，如决策树或者 EM 等，可以从离散数据中提取单向或双向计数，以稀疏执行。或用朴素贝叶斯对推断聚类，以稀疏执行[165]。

5.2.3 基于动态指导的模型稀疏化方法

(a) **设计意义**

前一章所述的动态人工智能运行环境框架中，需求感知、动态响应与协同执行一起构成了执行闭环。基于动态指导的模型稀疏化方法所涉及的研究内容是针对某些动态需求产生的一种响应方式。

(b) **基于动态需求的指导**

方法名称中“动态指导”来自于动态需求，实时的需求可以由根据前文所建立的“目标矩阵”描述，而实时的资源状态可以通过前文所述的动态响应中对资源的监测方式获得，并由多层次资源特征模型所实时描述。

模型稀疏化需要关注的实体资源需求包括：计算力 c，存储空间 m 等；目标需求包括时间 t，精度 p 等。稀疏化目标下的上述因素与稀疏化之前的模型对上述因素的需求相比，可以得到改进参数 r_c, r_m, r_t, r_p:

$$r_c = \frac{c_{sparsemodel}}{c_{currentmodel}}, r_m = \frac{m_{sparsemodel}}{m_{currentmodel}}, r_t = \frac{t_{sparsemodel}}{t_{currentmodel}}, r_p = \frac{p_{sparsemodel}}{p_{currentmodel}} \tag{5.6}$$

这些需求，以及未来可能出现的更多需求，为模型的改动提供了不同维度的参考信息。

(c) **模型稀疏化方法**

模型执行的结构可以表示为表 5.5。

表 5.5 模型执行结构与稀疏化建模

人工智能多层次模型表示	举例		稀疏化方法 f	稀疏化影响
外层执行方式	SGD		f_{iter}	$(r_c, r_m, r_p, r_t)_{iter}$
基本功能模块组合图				
基本功能模块				
内层计算流图	Alexnet		f_{graph}	$(r_c, r_m, r_p, r_t)_{graph}$
基本算子	conv	输出数据	f_{unit_out}	$(r_c, r_m, r_p, r_t)_{unit\_out}$
		结构参数	f_{unit_weight}	$(r_c, r_m, r_p, r_t)_{unit\_weight}$

稀疏化方法 f 是由多层执行结构组成的一整套策略，每层结构都可以包含多个基本单元，每个单元可以使用 0 或多个稀疏化方法。因此，一系列方法产生的影响需要综合评价。

在实际应用中，稀疏化需求的产生来源于多个方面，例如：运行环境本身的状态 (资源使用率等)，用户对于执行的新需求 (希望实时调整模型结构、执行时间、执行精度、执行开销等)。这些需求往往可以通过系统资源环境接口或者用户反馈直接获得。

设计人工智能模型并运行是一个多次反馈、实时调整的过程。动态指导体现在对于动态需求的实时响应。

综合动态需求和模型稀疏化策略两方面的模型，可以得到基于动态指导的模型稀疏化方法的表示：

$$\arg\min\{\int_t |g_{demand}(t) - g_f(t)|\} \tag{5.7}$$

方法的目标是尽量满足实时需求。其中 g_f 可以表示资源、期望时间和精度随时间变化的函数，需求部分 g_{demand} 可通过参数 r 求得。

实际执行时，“实时需求”除了实时产生之外，还包括额外的输入需求，上式的积分过程难以实际计算。为此，将模型稀疏方法表示分解到每一个需求的时刻 t_i 得到新的表示 $\arg\min|g_{demand}(t_i) - g_f(t_i)|$ 。

$f = \bigcup_j f_j$ 是采用的所有稀疏化方法的全集，有的方法可能嵌套在其他方法之中。更进一步，该稀疏化方法集合 $\bigcup_j f_j$ 可用函数表示，但难以准确求解，因此提出近似求解方式。

算法 5.4 稀疏化方法近似方式

输入: r_c; r_m; r_p; r_t;

输出: U

1: 计算 c, m, t, p 的实际需求;
2: 初始化 U, U_c, U_m, U_t, U_p 为 $\varnothing$;
3: for require in $\{<r_c, r_m, r_t, r_p>\}$ do;
4: for f in $\{\{f_{unit\_weight}\}, \{f_{unit\_out}\}, \{f_{graph}\}, \{f_{iter}\}\}$ do
5: 寻找满足 require 条件的多种 f 并评价其影响，加入集合 $U_{require}$;
6: end for
7: end for
8: 选择满足最多需求条件的 f 集合: $U \leftarrow U_c \bigcap U_m \bigcap U_t \bigcap U_p$，当后者为空时选取满足其中主要需求的方法集合作为 U;
9: 评价 U 中各方法的综合影响，并酌情调整;

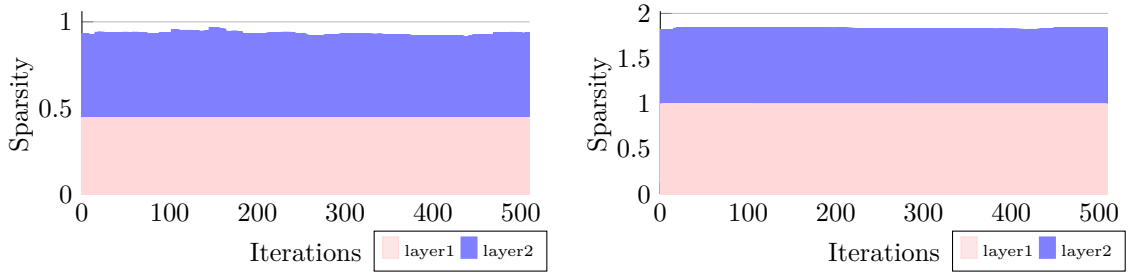
在近似求解过程中，需要综合比较不同稀疏方式的开销和效果，对照目标矩阵进行选择。

5.2.4 实验分析

(a) 实验环境

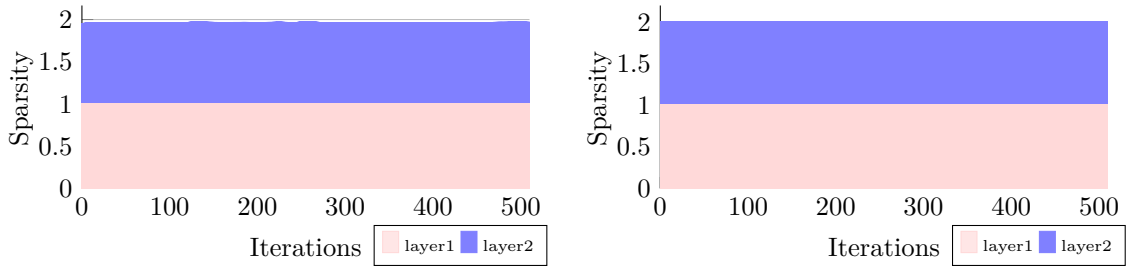
实验在 Intel Core i7，16 GB 1867 MHz DDR3 上完成。C++ 编译器为 clang(基于 Apple LLVM version 7.0.2 (clang-700.1.81))，原始框架 caffe 的版本为 128797eb。

目前的人工智能训练任务绝大多数都是数据并行，单机能够容纳一个完整的模型。系统级的资源状态粒度较粗，而芯片级的处理器利用率、内存占用率等状态更能精细反应当前处理器上执行任务的实时需求。此外，通过合理组合多个芯片级的任务调整，可以应对更大规模的系统级需求。因此，在单机上完成本实验能说明本方法的效用。



(a) Sparsify weight (simple alexnet, mnist5, s=0.1)

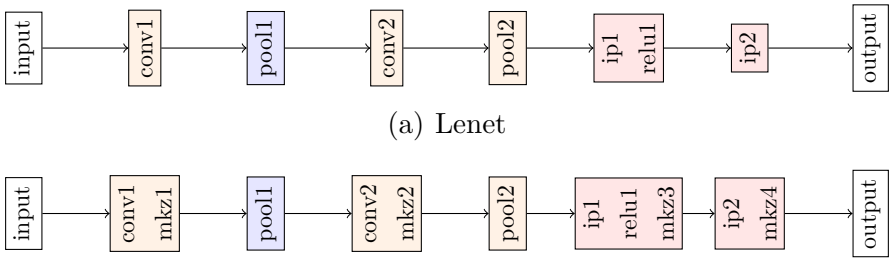
(b) Sparsify weight (simple alexnet, mnist5, s=0.25)



(c) Sparsify weight (simple alexnet, mnist5, s=0.5)

(d) Sparsify weight (simple alexnet, mnist5, s=0.75)

图 5.8　稀疏化实验迭代过程 (模型参数权值)



(a) Lenet

(b) Sparsify unit of lenet

图 5.9　稀疏化实验迭代过程 (模型结构变换)

(b) 结构参数稀疏化

实验用 c++ 实现，采用简化的 AlexNet 网络，进行手写数字识别。训练数据是简化版的 Mnist(包含 0-4 共 5 种手写数字)。网络共包含 34500 个权重参数，其中 layer1 包含所有的卷积层共 32000 个参数，layer2 包含所有的全连接层共 2500 个参数。

定义稀疏度参数为 s，其取值范围是 $[0, 1]$，用来稀疏化 $w = s \times |w_{max}|$ 以下的参数。

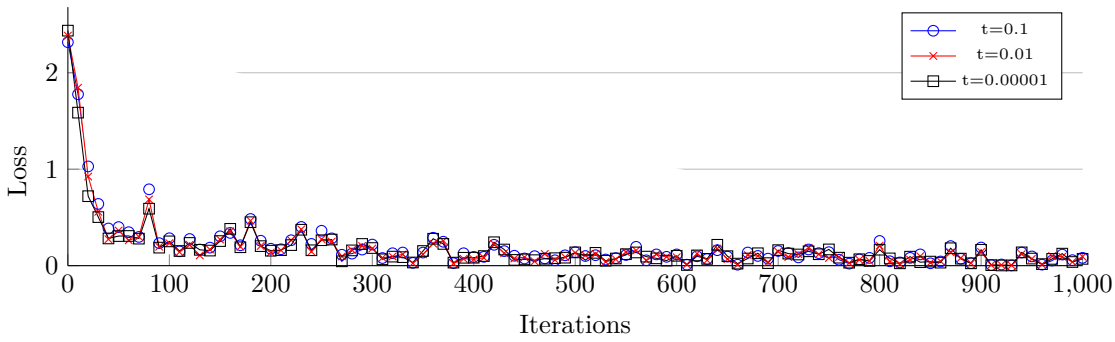


图 5.10　稀疏化实验迭代过程 (模型算子输出 Sparsify unit out data (lenet, mnist))

稀疏度 $p_s = c_{w.below\_threshold}/c_{w.all}$，设定权值绝对值在最大权值绝对值的 s 比例以下的参数为 0。

图 5.8表示在设定稀疏度参数为 0.1，0.25，0.5，0.75 条件下，迭代过程中的稀疏度变化。由于有 2 个 layer，每个 layer 的最大稀疏度为 1，因此总的比例在图中以和的形式呈现。可以看出，保留绝对值较大的权值可以显著地改变模型参数地稀疏度，即使只删除 $s = 0.25$ 的参数，总稀疏度也接近 0.999。

(c) 输出参数稀疏化

实验采用 caffe 框架作为基础环境，对于 Lenet 网络进行 Mnist 手写数字识别。将基本算子的输出部分稀疏化作为一个特殊的 layer 插入到原始网络中进行实验。如图 5.9，在 Lenet 网络的卷积层、池化层、全连接层中进行输出数据稀疏化 (添加 mkz 层) 的实验，原始网络结构和实际网络结构分别为 (a)(b) 两部分。

实验的运行结果如图 5.10，设定基本算子输出部分绝对值值分别在 0.1, 0.01, 0.00001 以下的数据为 0，描述迭代过程 loss(判断收敛的条件) 曲线。可以看出，对于基本算子输出数据的微小调整基本不影响实验结果。

(d) 基于动态指导的模型稀疏化

动态稀疏化需要动态运行环境的支持。目前已有的人工智能框架大多是静态执行，其调度粒度为一次训练，虽然 DyNet 等动态框架支持动态计算图，但需要在每轮迭代之前重新构建计算图，引发的开销较大。

图 5.11和 5.12 基准实验是需求被满足且无变化的模型执行过程，如 base 一列。模拟一个简化的真实场景：在计算执行开始时，不作稀疏要求，在计算执行到特定轮数时，设置稀疏化调整需求分别为 slight(在不影响模型的条件下稀疏化)、0.001(稀疏阈值为 0.001，后同)、0.01、0.1、halfnum(稀疏度为 0.5)、0.5(稀疏阈值为 0.5)。子图 (a)-(e) 分别为不同的算子模块的情况，子图 (f) 为全模型的情况。可以看出稀疏化基本可以按照指导要求进行调整。

只调整卷积算子对于总模型参数的影响很小 (图 5.11(f))，因为绝大多数参数都在 FC 算子中；但对于总计算需求的影响符合稀疏需求 (图 5.12(f))。在训练过程中，稀疏化的目标主要是减少计算量，因此针对计算占比较高的算子稀疏化是有意义的。

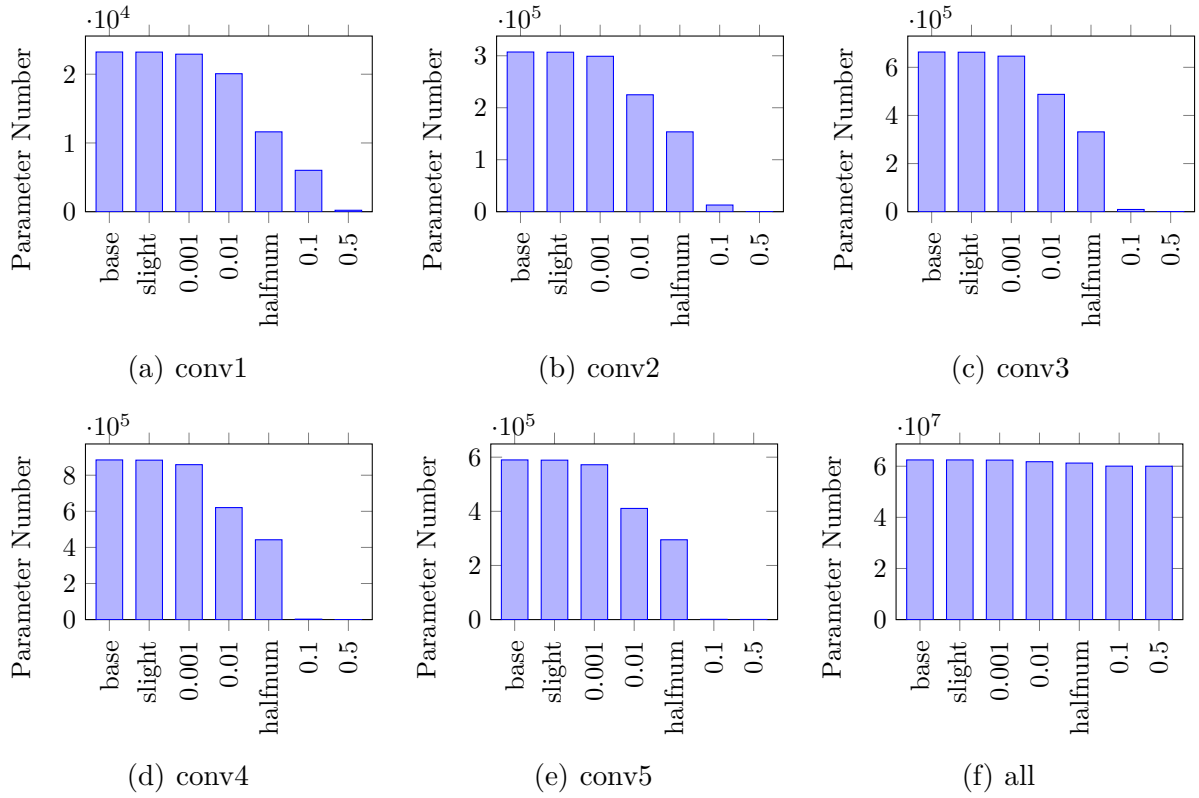


图 5.11　动态稀疏化实验 AlexNet 一次调整-参数量

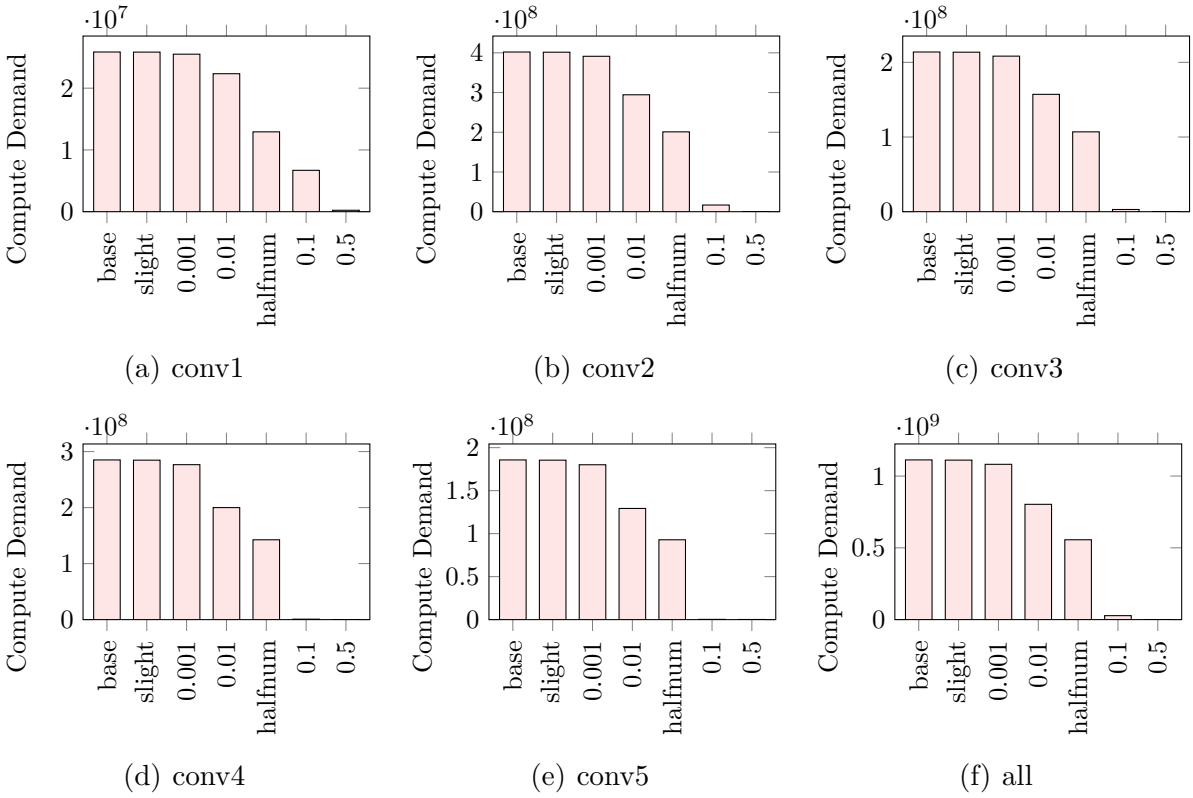
因此实验采用 caffe 作为基础框架并修改，为了响应动态需求，在执行的每一轮 (epoch) 开始前加入指导检测。

图 5.13是 Lenet 和 Cifar10 quick 两个网络在 Mnist 和 Cifar10 两个数据集上训练中稀疏度变化过程。经过一定迭代轮数之后，在图中拐点出现稀疏化需求，后进行调整。该图分别统计了经过卷积 (conv)、正则化 (relu)、归一化 (ip) 这几类模块数据的稀疏度变化曲线。

5.2.5　小结

本节针对实际应用需求动态可变和模型可调整、可稀疏化的特征，提出了基于动态指导的模型稀疏化方法：针对实时需求，调整模型的结构和输出参数。实验证明，采用设置权重阈值的方法进行结构参数的稀疏化时，只需设置相对较小的阈值就能获得极大的稀疏度；对基本算子输出数据稀疏化时，输出数据的微小调整基本不影响模型的效果；动态稀疏化可以根据动态需求实时调整模型的稀疏度，从而减少模型的计算、数据等需求，完全不影响模型的稀疏化可以减少约 0.07% 到 0.1% 的参数和计算，设置稀疏阈值为 0.01 已经可以减少接近 20% 的参数和计算。因此动态稀疏化技术可以在保持模型基本功能的同时为复杂多变的应用执行情况提供高效的应对策略，以适应用户和体系结构的需求或变化。

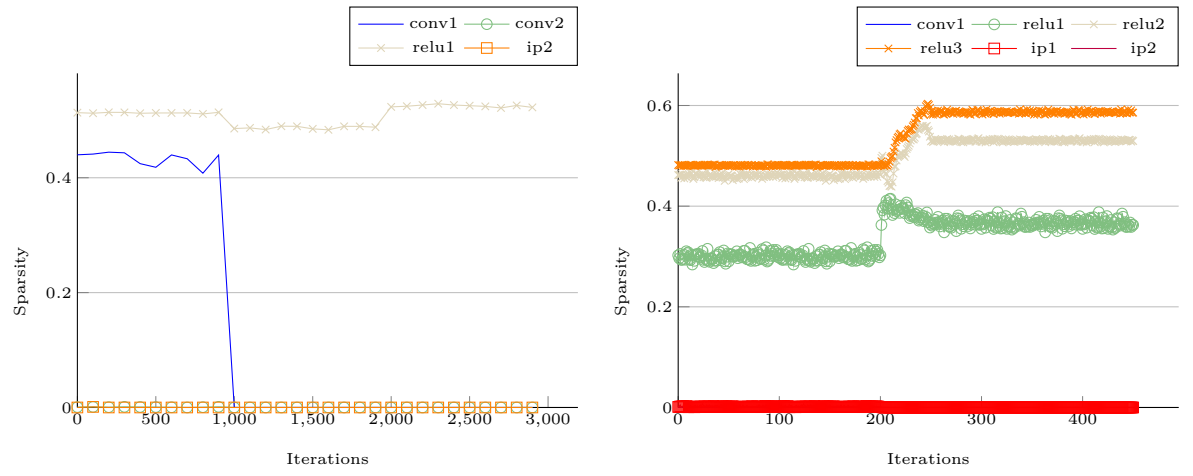
然而，应用的目标不仅仅是稀疏化一个方面，当因为稀疏化而满足了存储、计算、时间等目标时，用户还需要平衡诸如训练效果这样的目标。因此，虽然实验可以证明稀



(a) conv1　(b) conv2　(c) conv3

(d) conv4　(e) conv5　(f) all

图 5.12　动态稀疏化实验 AlexNet 一次调整－计算量



(a) Lenet, Mnist　(b) Cifar10 quick, Cifar10

图 5.13　模型训练过程中的数据稀疏度变化

疏化的确带来了预计的好处，也的确能够较少地影响应用效果，但在具体的场景下还需要用户判断或者实时监测系统需求，人工智能模型需要不断交互地进行调整和训练。

5.3 自适应的协同预处理调度策略

体系结构复杂而多样，高性能计算机系统 (HPC) 最初为科学计算设计，应用在气象地震、石油石化、国防军工等方面。作为一类具有强大计算、通信、访存能力的系统，HPC 具有高内存、紧耦合、高效互连的架构，其计算能力受到了数据相关性强的人工智能应用的高度青睐，成为了后者的重要计算环境。人工智能各领域的应用广泛出现在 HPC 系统中，且比重逐渐增加[166]。HPC 成为人工智能的重要计算环境，其上的应用类型也从计算密集型为主发展到计算密集型和数据密集型并重。

在人工智能应用的运行环境中除了执行模型，如何处理数据也是关键。而数据预处理过程具有数据密集型应用的特征，对于人工智能应用的顺利执行至关重要。在计算能力之外，HPC 系统的存储能力、数据吞吐能力越发重要。数据预处理作为人工智能等应用全流程执行中的重要一环，在数据量大的条件下不可忽视。然而，由于大部分 HPC 系统并不是专为人工智能数据处理设计，在将应用移植运行时往往会遇到一系列问题。数据量大而引发的高吞吐需求就是其中之一，很多 HPC 系统与数据来源分离，数据的高效接入也是挑战。为此，如何将高性能计算系统与高吞吐系统协同利用，在 HPC 系统现有条件下高效完成人工智能数据预处理，并且合理利用系统资源，是在此类系统中进行协同设计的重要挑战。

本节将从人工智能应用中数据处理的步骤出发，就 HPC 这一类典型的体系结构，设计存储协同计算的预处理方法，并基于神威太湖之光系统，完成实验验证，使 HPC 系统在合理有效吞吐数据的同时，能够完成适当计算，减轻后续人工智能模型执行的负担。

5.3.1 HPC 上人工智能应用数据预处理流程分析

随着数据规模的增长，我们对数据的分析越来越全面和精细。许多人工智能应用已经不仅仅局限于“计算”本身，其数据的采集、预处理、存储、计算和展示都是重要部分。

数据密集型的人工智能应用如数据挖掘，本身就需要全流程的协同。而计算密集型的人工智能应用，如科学计算中的天文数据分析，其数据量越来越大，对于数据预处理的需求也逐渐增加。数据预处理过程单位计算量对应的数据量更大，对于带宽等资源的需求更高，如何将硬件环境与其服务的应用源数据有机整合，是计算环境设计中值得研究的问题。

(a) HPC 系统级资源建模实例

为了充分挖掘 HPC 的能力，我们需要采用系统级资源模型对 HPC 建模，从而分析 HPC 的基本结构和各结构能满足的计算需求。

一般而言，HPC 分为计算系统和支持计算的外围系统。HPC 系统除了基本的计算单元之外，还包括存储、网络等资源。文件系统是大规模数据进入 HPC 后的第一挑战，HPC 的文件系统主要是 Lustre[152]、GPFS[167]、PVFS[168] 等分布式全局共享文件系统。在文件系统中，数据分割与分布会影响到预处理时以及后续 IO 的效率。一般少量数据

的处理可以用数据库完成基本操作，然而超大规模的数据库性能有限，也缺少对于 HPC 系统的支持。此外，很多 HPC 系统同时服务于多个应用，原始数据不能长时间留存在存储系统中，所以重量级的数据库不适合数据预处理，而与预处理协同的轻量级数据分布策略更适合。

在计算的全流程中，各部分并不是始终高负荷作业。在一部分高负载运转时，另一些部分可能负载较轻。如果能利用低负荷作业的资源完成部分计算，将会分担数据密集型应用的部分计算量，有效降低计算过程所需的总时间。

支持计算的外围系统往往包含管理节点，或者轻量级的计算单元。这些结构有的对外可见，会有空闲或轻负载的情况。例如，在保证服务 IO 作业满带宽运行的同时，IO 管理节点的计算资源仍有空余；在保证存储一致性响应正常的同时，存储管理节点也能空闲一部分计算资源。这给我们的后续设计提供了可能。此外，存储协同计算本身还可以通过弱连接的协同计算部件优化处理过程。随着数据产生的速度和体量高速增长，硬件环境的设计与利用将不仅仅停留在计算单元更高效的层面，而是向各部件协同高效发展，与应用的关系也更加紧密。

(b) **数据处理流程**

计算密集型应用一般只在计算分析部分需要较强的计算资源，而数据密集型应用在整个流程中，对计算资源都有需求。计算密集型应用计算过程占据了整个流程的大部分时间。其数据输入较少，中间计算需求大，数据因计算而产生，但结果数据量有限，对于吞吐能力要求不高。

数据密集型应用的一般流程包括数据的预处理、存储、分析和展示不同模块。每一步都需要不同的计算资源，虽然分析部分是重点，但每一步的计算资源需求都不可小觑。

对于 HPC 而言，其全流程设计往往针对计算密集型应用，缺少对数据预处理的支持。数据密集型应用的预处理过程有时耗时巨大，有时需要所有数据协同处理，但直接利用 HPC 往往过于奢侈，如果在数据接入之前预处理完毕，则需要大量额外的计算资源，甚至到达 HPC 本身的规模。而随着传感器数量的增加，精度的提高，一些以模拟为主的计算密集型应用的预处理过程也日趋复杂，需要更强的预处理支持。

如果用 HPC 作为以上应用的计算资源，需要整合其计算流程，并充分挖掘 HPC 各个部件的计算能力，适应全系统的耦合特性，使计算资源和计算需求匹配。

5.3.2 预处理过程中典型计算模块分析

数据密集型应用中，一般的预处理包括数据接入、数据清洗、格式转换、压缩／解压缩等，这些计算模块的顺序往往与应用本身有关。计算量 $z(n)$ 为 $O(n)$ 到 $O(n\log n)$ 不等[169]。

数据接入和采样: 数据接入 (导入) 依赖于接入设备、接入带宽。接入的数据可以来自于网络、直连的传感器或者外接数据库等。对于大规模的数据，在接入的同时进行预处理的采样分析十分必要，可以验证或改进后续处理的流程。

数据清洗、补全与噪声识别: 数据清洗指检测、修正或去除不正确或不完整的数据，

有时也作过滤。与特定领域无关的数据清洗，可能涉及到去重。而很多数据清洗方法都是领域相关的，例如通过已知该领域的数据分布或者数据范围确定异常值。

数据补全主要针对不完整数据，尤其是高维数据。有的缺失值可以通过本数据源或者其他数据源推导而得，有的缺失值可以直接手工重新标记。

噪声识别可以采用回归或者分段光滑化的方法[170]，由于算法较为复杂，可能涉及到领域数据和迭代算法，预处理阶段较难处理，该步骤可以在数据分析的时候完成。

数据转化的种类繁多，有数据归一化、数据索引以及格式转化。

数据归一化，需要对所有的数据求极值，再计算对应比例。设数据量是 n，则计算量是 $2n$，吞吐量是 $2n$。求极值和计算分别都可以完全并行，没有数据依赖关系，但二者之间有数据相关性，并行计算之后的每部分极值需要通过选举求极值并将极值广播。

数据索引需要对所有数据依照某规则建立索引，并给数据赋索引值。建立索引的方法有繁有简，有的索引需要将数据转化为特殊的格式，有的索引需要将数据更简单的表示。索引可以根据数据到来的顺序建立，也可以根据数据本身的可排序性建立。如果按照到来顺序，需要维持一个查找表，额外的空间复杂度最多可以达到 $O(n)$，单次查找时间会逐步增加到 $O(\log n)$。如果按照某方法排序，可以不需要额外的空间，但一般排序算法的复杂度至少为 $O(n \log n)$，后续查表虽然单次时间仍为 $O(\log n)$，但可以批量查找并优化查找策略使访存连续。如果数据服从某种分布，还可以利用转换函数直接将数据映射成索引，只需 $O(n)$ 计算量和 $O(1)$ 的额外开销。数据索引有时可以通过数据库实现，数据库包括 MySQL 等关系型数据库，MongoDB[171] 等非关系型数据库。其中，非关系型数据库还包括图数据库如 Neo4j[172] 等。在数据规模较大时，即使数据库规模能够容纳这些数据，数据库的插入和查询操作也会耗时巨大，难以高效完成数据转化工作。在 HPC 系统中，由于应用的多样性，单一功能的数据库难以满足需求，而数据库本身的开销也不可忽视，

降维是通过特征变换函数将高维特征转化为低维特征，常见的降维方法包括主成分分析 (PCA)、线性判别分析 (LDA) 等。其计算复杂度一般较高，需要全局数据的矩阵计算，并行化时依赖程度较高，难以用” 轻量级计算” 来完成，需要用到高性能的计算资源。

格式转化是为了更方便地进行后续计算，一般的格式转化包括两类，一类需要全局数据，一类只需要转换函数。前者对空间的要求很高 (甚至达到全局规模 $O(n)$)，计算复杂度也高 (甚至达到 $O(n \log n)$)。后者一般只对数据一次作用转换函数即可，易并行，数据几乎没有依赖关系。

数据整合包括数据组合、数据格式统一、数据去重、数据分块等。

组合和统一格式只需要对于输入的数据流水执行整合操作即可。数据去重需要更高的计算开销或者空间开销，如果用排序的方式去重，需要计算量为 $O(n \log n)$；如果用哈希索引去重，空间开销为 $O(n)$，还可能有冲突造成的额外开销。

对于大规模分布式计算，数据分块/数据分布关系到后续访问效率以及计算过程中的通信。需要根据数据和计算特征以及体系结构条件做出适当的设计。

对于一些数据，可以处理为压缩的方式，以减少后续传输的数据量。压缩的设计需要权衡压缩/解压的开销与传输减小开销。

5.3.3 预处理协同计算设计

本设计采用多层次资源模型中的系统级模型；并将人工智能数据预处理过程抽象化为具有数据流关系的动态模块模型；运用自适应技术对预处理过程进行协同调度，实例化为超级计算机系统 (神威太湖之光 HPC 平台[42]) 和超大规模预处理数据。

本研究利用“存储协同计算”，通过 HPC 的 IO 代理节点，在存储端对 HPC 的 IO 负载较轻时，提供数据密集型应用的预处理支持。

将预处理过程设在存储管理节点的优势是：HPC 的计算节点往往承担许多应用，机时有限，而预处理的流程计算种类较多，过程复杂，可能需要较长时间。存储节点和存储管理节点的容错性能更好，也能够有效支持高吞吐的数据量。存储管理单元的软件支持更丰富，更适合复杂多样的预处理流程。

人工智能应用在 HPC 系统中的整个计算流程一般是：数据导入、预处理、读入计算节点、计算、写回存储节点。对应的资源分别是：网络入口、存储节点、预处理计算资源、HPC 计算节点网络、HPC 计算节点、HPC 计算节点网络、存储节点。我们可以看到，除了计算部分，其他部分的处理都不在 HPC 计算节点上。

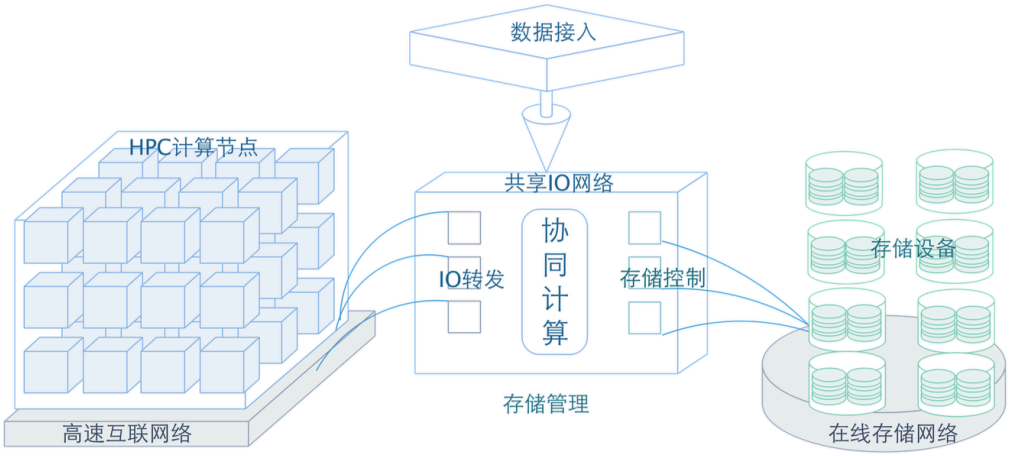


图 5.14　HPC 系统资源与数据处理流程图

HPC 设计主要以其计算节点的计算能力更高为目标，其接入带宽的设计致力于满足计算密集型应用如科学计算的需求。面对数据密集型应用尤其是数据预处理过程中数据量更大，计算量没有显著增长的情况，数据接入存储设备，以及数据由存储设备传输至 HPC 计算节点都会面临巨大的挑战。尤其是数据读入部分，将海量数据读入 HPC 的存储单元开销很大，预处理需要紧密耦合在存储单元中，否则 HPC 接入带宽将会限制后续处理。

存储管理节点的业务包括存储控制和数据转发，负责数据吞吐，对数据密集型应用尤为重要。不同层次的存储管理业务 (如存储控制、IO 转发等) 有可能由一层物理设备

完成，也有可能分散在多层设备上。存储管理节点的负载与它服务的数据量有关，一般情况下，其负载远小于其计算能力。因此可利用存储服务节点的空闲资源完成人工智能应用大规模数据的预处理计算。基本流程如图 5.14，数据进入存储资源时，“协同计算”就地进行预处理，之后按照应用需求选择暂存或者直接转发至 HPC 计算单元。

存储协同计算的设计是利用存储单元，对数据就地进行预处理等操作。协同计算从数据接入开始，至数据进入 HPC 计算单元为止。如图 5.15，一般包括接入、采样、清洗、归一化、索引、组合、去重、分布设计等。这些流程可根据不同应用做适当调整。

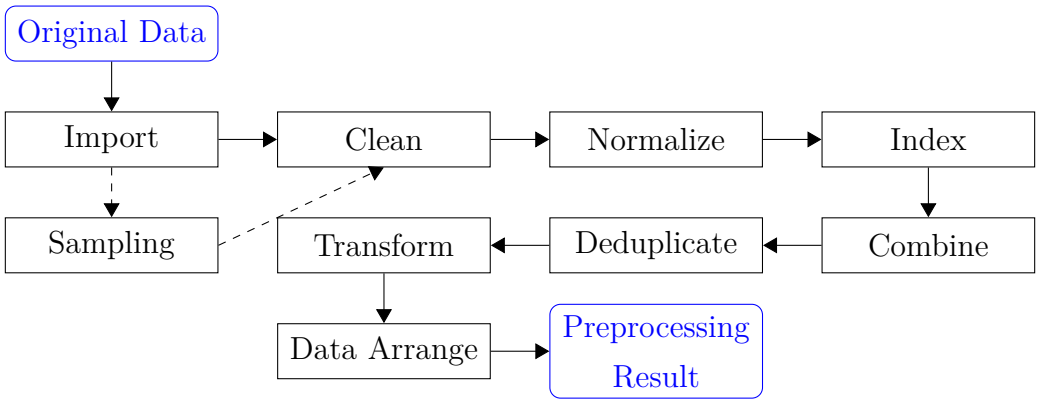


图 5.15　预处理的一般流程

为了评价数据密集型应用每一步的计算设计，我们采用“基于模块的数据流图”对整个处理流程建模。之后再根据预处理的目标确定预处理的设计方案。

对于流程中的每个模块，依据其入数据量、出数据量以及中间过程计算量构建模型。设各模块此三者分别为 m_i, n_i, $f(m_i)$。则该模块 i 对应的计算时间为

$$t_i = t_{in} + t_c + t_{out} = \frac{m_i}{b_{in}} + t_c(f(m_i)) + \frac{n_i}{b_{out}} \tag{5.8}$$

即读入时间、计算时间、写出时间之和。其中读入带宽 b_{in}, 写出带宽 b_{out}，当多节点并行处理时，该参数对应为聚合带宽。

数据处理的全流程开销可以表示为 $t = \sum_{i=1}^{N} t_i$，流程共 N 步。我们希望每一步能根据其数据量和计算量选择合适的计算资源。

如果相邻的两个流程之间可以流水作业，那么前一个流程的 t_{out} 和后一个流程的 t_{in} 都可以省去，可能需要少量额外的管理开销。

在实际应用中，预处理过程具有较强的“问题特性”，每个模块的算法、操作与应用和数据关系密切，不能一概而论。但在同一个应用中，数据预处理的各个模块之间由于以“数据流”作为连接方式，具有一定的相关性，且这种相关性在数据移动的占主要开销的情况下更强，因此“协同预处理”具有普遍的含义。

用目标矩阵可以描述多样的应用约束条件。其中最小化总处理时间 t 是常见的要求。鉴于不同的资源稀缺性和对各流程的适应性不同，不同子模块的目标也会多样化。有时，为了让 HPC 计算节点运行高效，我们需要最小化计算节点所含流程的 t_c。有时，

数据密集型应用某些流程的数据量较大，我们需要最小化 t_{in}，将流程安排在离存储更近的地方。

在预处理阶段，我们更关注系统层面的目标，而应用层面的目标一般只有“完成”这一个维度。该阶段的常见目标为：在约束的时间内，完成预处理计算，对资源的开销不影响所利用的协同计算资源本身的任务，同时，希望总的预处理时间最短。为此，需要充分了解各阶段硬件资源的参数与流程的数据量、计算量，并充分挖掘计算的并行。减少预处理时间的方法包括：合理减少需处理的数据、减少写盘的次数、降低预处理计算的复杂度、增加预处理计算资源、计算和访存相互隐藏等。

5.3.4 实验分析

我们用一种典型的人工智能数据预处理应用——二维数据排序索引并转换——做样例分析。二维数据的两个维度格式相同，其预处理过程需要提取每个维度的所有数据统一过滤排序去重，得到连续数据并建立索引表，再将原始数据转换成索引表示的整数对。实验选取了实际应用中的图数据 (边列表)，每条边表示一条连接关系，其中每个维度都是连接关系中的一方。该应用计算量适当，除了对 IO 带宽有需求之外还需要适宜的预处理计算资源，适合分析协同计算的能力。

(a) **实验环境**

应用需求：初始的压缩数据大小接近 400TB(解压后预计 1PB 以上)。在数据未展开之前不能确定数据的重复率，对最终数据只有粗略估计。数据量大、数据相关性强以及其不确定性是该应用预处理阶段面临的主要问题。

软硬件环境：神威太湖之光有 144 个可用的 IO 服务器，上面部署虚拟机来管理文件访问，考虑到预处理读写数据带宽受限，以及存储管理节点的其他应用和服务，本次各个计算流程用到了 64 到 80 个计算节点 (虚拟)。这些节点共享存储，但是彼此之间不能互连，当单节点不能完成计算时，只能通过存储文件进行数据交互。具体参数如表 5.6。每个盘阵两个控制器，60 块 1.2TB 磁盘，以 RAID6（8+2）模式构建为 6 组卷。每个卷提供约 330MB/s 的读写带宽。服务器通过 rdac(多路径软件）访问盘阵。正常情况下每个服务器访问 3 个卷，总共约 1GB/s 的总带宽；某个服务器故障情况下，由其相邻的伙伴节点接管其卷，提供服务。网络为 FDR 的 Infiniband。

(b) **实验设计**

数据密集型应用的预处理协同计算需要从数据接入开始设计，直到其进入 HPC 计算单元。整个过程中，利用存储管理节点协同计算。

400TB 压缩数据通过网络接入，边读入边解压/格式转换/粗去重后写盘存储，全排序/去重/建索引后再存储，最后查表/重新构建原数据，此时可以根据需要直接导入计算节点。

原始数据量大、排列无序且有一定的重复性，在一次流水式读入处理的条件下，无法全局去重，故排序去重阶段至少要有一次中间数据存储。而由于各节点之间无法交互，初始阶段分段处理 (按照数据的前 n 比特分成 2^n 段相互之间有序的数据)，这样使后续

表 5.6　软硬件环境

IO 服务器	太湖之光 IO 服务器
可用 IO 服务器个数	64 至 80
各服务器空闲内存	16GB
本地存储	300GB
每个服务器读写带宽	各约 1GBps
存储	Lustre 文件系统，PB 级容量
接入带宽	4GBps
HPC 计算节点双向通信带宽	112Gbps

计算完全可并行。

流水设计：流水的思想在于充分利用数据处理的每个步骤的空闲资源 (带宽或者计算资源)，对于不需要全局视角的计算 (如仅依赖本条数据的过滤、分段、格式转换和数据按照流式处理的解压过程)，在前后步骤的吞吐速率匹配时，将数据一次性读入内存，尽可能多地完成计算步骤。解压缩时，对于每一个文件的解压缩需要维持一个查找表，且关键部分 (查表) 不能并行，总计算量固定，单位时间数据输出量恒定。格式转化时，针对解压缩后的每一行数据，都执行相同的计算。为了验证设计的可行性，我们的单元实验都以数据集中的一份压缩文本文件为单位 (12GB)。下同。

表 5.7　一个文件不同处理的用时

步骤	**用时**	**读**	**写**
拷贝数据	2min	12GB	12GB
单独解压缩 + 存储	4min 35s	12GB	52GB
单独格式转换 (本地读 + 写)	6min 49s	52GB	25GB
解压 + 格式转换 + 分段	4min 32s	12GB	25GB
解压 + 格式转换 + 分段 + 粗排序去重	7min 12s	12GB	23GB

从表 5.7可以看出，流水虽然会引发少量的开销，但相比于每一步单独计算会更高效。由于在分段过程粗排序去重的效果并不显著 (对于真实数据只能去重 5% 以内)，在实际运行时，我们仅“分段”。将粗排序去重转移到下一步，待收集到更多数据之后再去重，其效果显著 (去重 50% 以上)。

多核并行：多核并行能够大大压缩计算时间，但由于并行能力与硬件相关，本实验数据只针对本环境有效。由于流水的设计，并行度需要考虑本流程及其他相关流程的协同。此外，还需要考虑其他硬件资源的限制。解压缩部分，受限于读入带宽，一个 CPU 在并行度为 4 的时候处理速度达到饱和 (表 5.8)。第二级流水是格式转换，虽然理论上可以高并发度执行，但依赖于入口读速率，所以只设计了 2 路实现 IO 与计算隐藏。剩

余的协同计算资源用作下一级流水——分段和粗排序去重。

表 5.8　不同并行度测试

用时 / 并行度 / 步骤	1	2	4	8
解压缩	4min 30s	4min 35s	4min 35s	10min+

表 5.9　不同分段数测试

分段数 / 步骤	1024	2048	4096
流水一总用时	7m19s	7m37s	7m52s
内存开销 (带缓冲)	8.2G	18G	30G

由于硬件资源同时支持其他服务，每个节点实际可用的内存只有 16G 左右。如果用不带缓冲的文件读写，用时开销大幅增加。故选择写带缓冲的 1024 个分段数进行数据分段 (表 5.9)。排序部分，考虑到双缓冲开销和计算节点的其他服务开销，每次排序的数据量设置在 8GB，用 16 的并行度充分利用 16 核资源。

幂律特征处理：很多真实世界的数据呈现幂律特征，本次选择的数据也属于这一类。直接分段时，有的段数据量巨大，有的段数据量很小。实验中分 1024 段时，最大段的数据量是平均值的百倍以上。此外，我们发现这些“大点”(数据量大的段) 的处理，即使单独对其更细致分段仍会遇到“幂律问题”。因此，我们在去重阶段首先按照末端 (即数据的末若干位) 分段，末端的数据近乎均匀分布，没有幂律特性，对于分段去重有算法上和运行时间上的保证。

在排序建表阶段，我们仍需要按照升序排序，需要解决“幂律问题”。前一阶段的去重已经大大减少了数据量，总的数据从 400TB 的压缩数据减少为 4TB 的二进制数据。在此规模上，继续分段排序，对“大点”特殊处理，建立索引表。

下一阶段“反查”，需根据索引表查找原始数据的对应表示，并形成结果数据。为了反查的效率 (分段查找每一段规模相当)，反查表仍需要按照“末端”索引。所以在建立索引表之后需要将其对应成反查表。

综上，去重阶段“去幂律化”，排序建表阶段根据幂律特征区别对待“大点和小点”，最终成功完成了此类幂律数据的处理。

(c) 全流程实验结果

最终的流程如图 5.16。整体步骤仍然包括接入解压缩、格式转换、排序去重建索引，以及查表。但在细节和具体数据处理方面添加了前文所述的优化步骤。最后一步查表的次数由数据本身的维度决定：如果仅是一维列数据，直接查表得到结果；如果是二维关系数据，需要在初始分段中先按照某一维分段，第一次查表后流水按照另一维分段，再进行第二次查表得到最终数据。本应用数据为二维，所以需要两次查表。

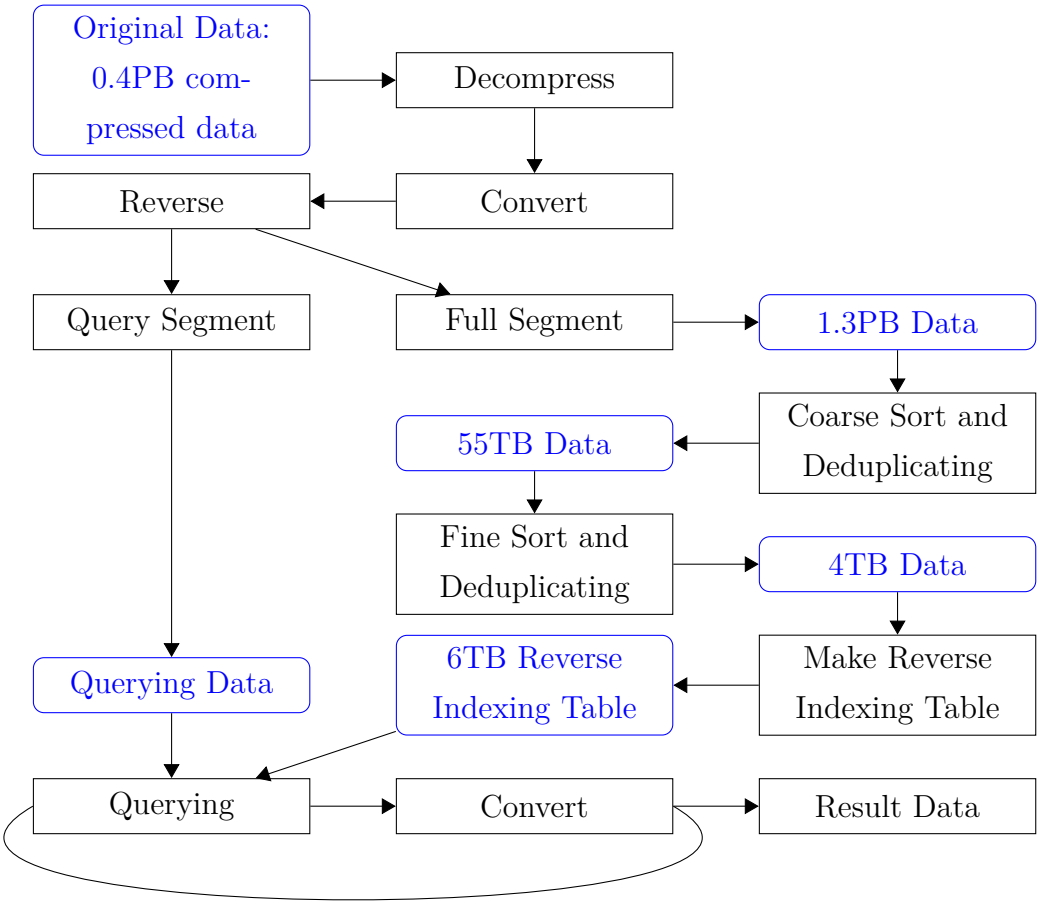


图 5.16　实验最终流程图

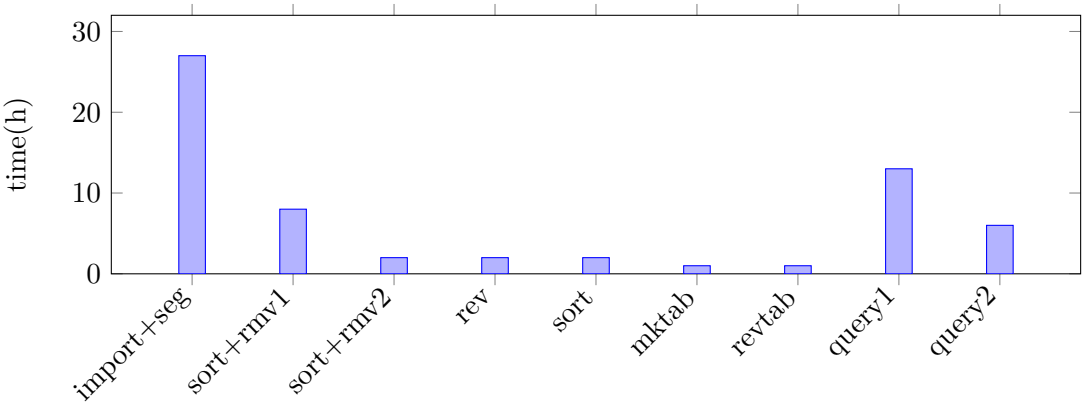


图 5.17　全流程各阶段用时 (依次是接入反转分段、粗排序去重、全排序去重、反转、排序、索引编号、建反索引、查表－第一次、查表－第二次)

表 5.10　全流程各阶段使用资源

import+seg	sort+rmv1	sort+rmv2	rev	sort	mktab	revtab	query1	query2
接入节点，16 个节点	64 个节点	64 个节点	64 个节点	64 个节点	80 个节点	80 个节点	64 个节点	16 个节点

在不影响协同计算部件基本服务的基础上，本应用所有数据完成预处理整个流程的运行用时在 56 小时以内。如图 5.17，其中，受限于接入带宽 (4GBps)，接入数据到分段完成至少用时 27 小时。除此之外，查表的用时也较长，这是因为查表需要吞吐总数据量 (400TB 压缩数据过滤后再压缩表示，约 400TB)，而之前的排序会去重，仅需要处理不重复的数据 (将二维数据的每个维度当作一个数据，去重)。预处理的全过程采用存储单元协同计算，HPC 计算节点和存储资源的其他常规服务没有受到影响。最终结果数据量约 135TB，全部读入 HPC 计算节点需要 1 小时左右。只使用 HPC 计算节点进行数据预处理，针对 1.3PB 的原始数据仅排序过程就需要 $37500CG \cdot h$(核组小时)。

5.3.5 小结

本实验协同利用超级计算机系统的各种资源，进行人工智能应用数据预处理设计，与只用 HPC 计算节点相比，节约了数万 HPC 核组机时，且协同计算没有影响其他基本服务，利用轻量级的处理方式，在 56 小时之内完成了 PB 级数据预处理流程。

人工智能应用中的预处理多为数据密集型操作，其层次多、种类多，在协同计算方案设计时需要考虑各个层次的衔接关系，充分利用硬件资源，提高处理效率。在实际应用中，模块的操作效率与数据强烈相关，需要根据数据的特征做实时调整。许多预处理的数据本身具有一定的领域特征，这些领域特征可能为后续的分析带来特殊的方法。

5.4 基于人工智能计算流图的动态自适应调度模型

在运行环境中，如何将具有动态特征的应用与系统紧密结合，如何满足全流程中的动态需求是关键的动态问题。

无论是应用本身的实现、各种人工智能框架，还是运行时环境、操作系统，都试图为应用如何在系统中高效运行提供解决方案。然而，随着实际应用的不断变化，系统环境的不断变化，需求的不断变化，在人工智能运行环境中亟需一套合理的机制来满足这些变化以更好地满足不断涌现的新需求。

5.4.1 基于基本算子的模型拆分

在已有的人工智能环境中，执行、调度严重依赖于框架本身，在超大规模的条件下，模型能否拆分、怎样拆分完全依赖于框架的处理。然而，很多框架对于分布式环境的支持能力有限，即使支持也多针对数据并行的条件。

模型和数据规模都在增加，未来会不会超出单一硬件单元的限制？当数据并行难以满足爆炸的规模时，模型如何合理拆分？当发展变化的模型本身出现不同的特点时，如何有效拆分成为一个值得研究的问题。并且，模型拆分这项技术本身不仅能指导人工智能框架中的模型映射与调度，还可以对其他运行时环境提供支撑。

基于基本算子并结合体系结构能对人工智能模型进行不同粒度的、量化的分析。因此基于由基本算子构成的中间表示可以对模型进行从分布式到并行设计不同粒度的拆分，也可以依据体系结构信息对模型进行合理地调度。

(a) 已有的并行方式

数据并行 数据并行主要应对数据规模大，模型规模不大的应用执行。每个节点使用不同的训练数据训练一个复制的模型。每个节点模型参数相同，每轮迭代后，合并所有节点的参数梯度，更新模型，并将更新后的模型分发到各节点, 进行下一轮迭代。典型的例子包括超大规模样本量下的神经网络训练。

模型并行 模型并行主要应对模型规模较大，难以和对应数据同时放在一个计算单元的情况。将模型进行拆分，拆分块在不同节点上并行执行，实现大规模网络的加速。典型的例子包括超大规模模型 (例如语音识别网络) 的执行。

数据拆分 数据拆分主要应对计算过程中高维完整数据的数据规模较大，数据之间有位置关系的情况，例如超大规模矩阵的计算。将矩阵拆分成块分布在各处理器上，各子块分别计算，过程中拆分处会发生通信，例如行列通信。

图并行 图并行主要应对图计算模型的抽象计算，用以描述数据成关联关系，计算是迭代过程的应用。最细粒度的并行可以到每一个"点"，粗粒度的并行可以扩展到"子图"。图计算模型的并行依赖关系取决于图本身的结构 (图中点与点连接的边, 或者子图与子图之间的边)。典型的例子包括超大规模关联数据 (例如社交网络) 的分析。

(b) 体系结构特征粒度与模型拆分粒度的映射

前文已就体系结构特征建立了多层次的模型，人工智能模型本身也是一个多层次的结构，如何拆分与映射需要首先考虑两者的粒度。

基于语义特征的模型拆分：模块化 基于模块的数据流图和基于算子的算子依赖图在模型表示中都可作为模型拆分的依据。模块相较于算子耦合度更高，有语义或者功能上的逻辑关系，且模块之间的组合关系多样，有的关系联系较松，相关依赖较少。因此，就拆分而言，基于模块的数据流图可以直观地拆分开来。

然而，拆分的目的是为了更好地执行。模块本身的粒度较大，当其恰好可以与硬件环境相匹配时，按照模块拆分是有意义的。但若模块的粒度超过了硬件的容纳能力，则需要更细粒度的表示方式——算子依赖图。针对算子内部的需求和算子之间的依赖关系，可以用图划分的方式拆分算子依赖图。

基于图划分的模型拆分 基于图划分的模型拆分，其拆分的标准有两个层面：计算量和数据量。

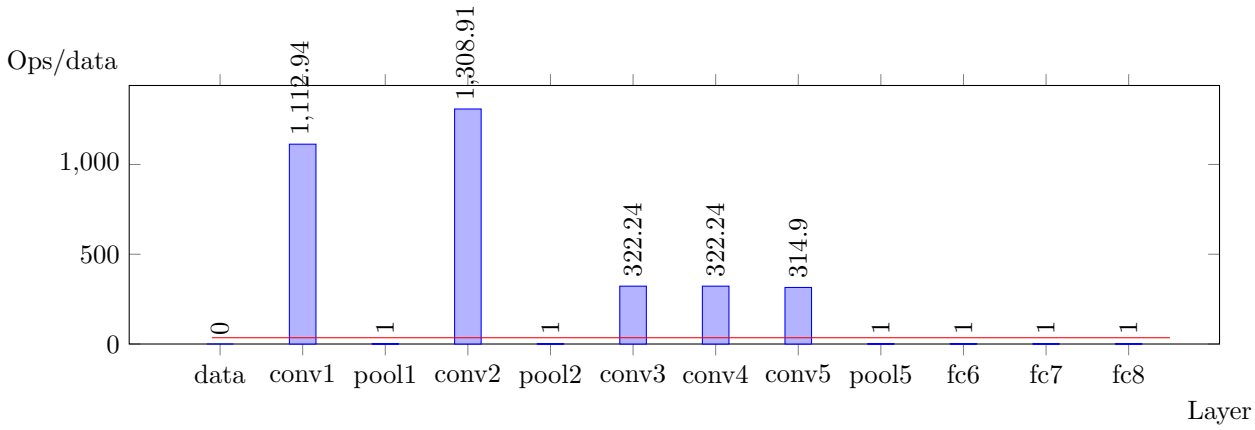
当模型能够容纳于分布式硬件环境 (存储能满足数据量需求) 时，拆分计算需根据各节点的计算能力均衡 (按计算能力成比例) 划分同时兼顾各节点的存储能力和拆分引发的通信开销。

当模型因为数据量大，单节点存储能力不能满足其执行时，拆分计算需要首先考虑每个划分单元的数据量需求与对应每个节点的数据容纳能力，并兼顾计算均衡与通信开销。

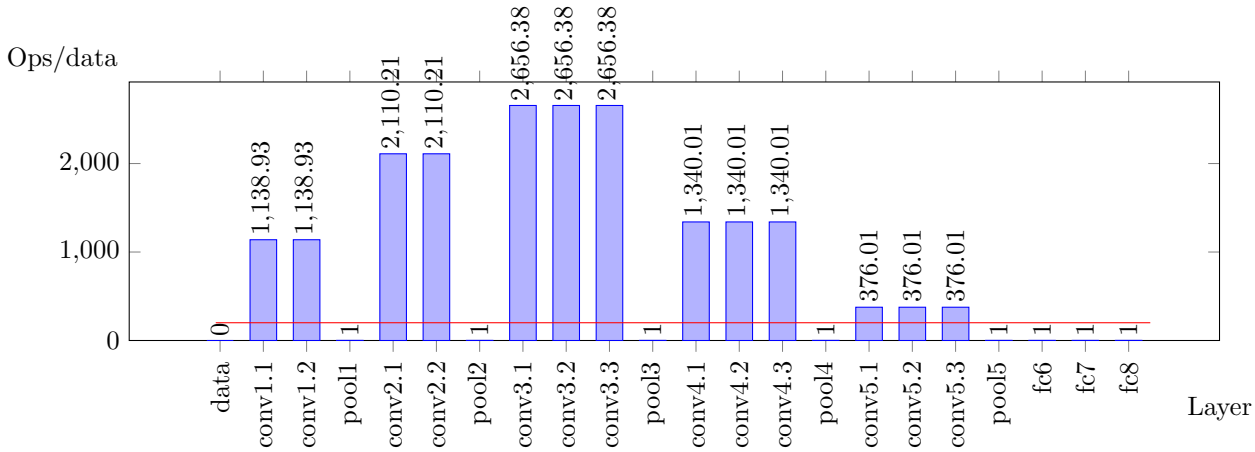
为此，需量化分析基于基本算子的计算 (计算量、数据量等) 需求。

(c) **典型人工智能模型的计算需求分析方法**

人工智能模型的计算需求分析可以**依赖于算子分析**，将模型拆分至最细粒度的基本算子，**针对基本算子构建其计算量、参数量、数据量、访存模式等需求进行建模**。



(a) AlexNet (avg=34.95,red line)



(b) VGG16 (avg=200.69, red line)

图 5.18　AlexNet 和 VGG16 各算子计算密度

图 5.18和图 5.19分别描述了 AlexNet, VGG16, GoogleNet 各个基本算子的计算密度需求。平均计算密度在图中用红色直线标出。就平均计算密度而言几种模型都能超过常见硬件 Roofline 拐点 (图 3.13)，但高密度的计算主要是卷积操作，其他操作的计算密度极低。因此鉴于各基本算子存在依赖关系，其最终计算效率除了受限于卷积的实现之外，还将受限于低密度计算部分。尤其是若低密度计算部分还受限于访存带宽，数据量的影响不可小觑。

图 5.20和图 5.22分别描述了 AlexNet 和 VGG16 各基本算子 (层) 的数据需求 (输入、输出数据量，参数量)，数据需求可以反映出基本算子对存储的需求，结合计算密度可以反映出对于带宽的需求。

以上数据需求分析的输入数据都以 ImageNet 测试集中的样本规格 (224*224*3) 为例，以 1 张图片作为一次计算输入，参数量是模型本身的特征。在此条件下，参数数据

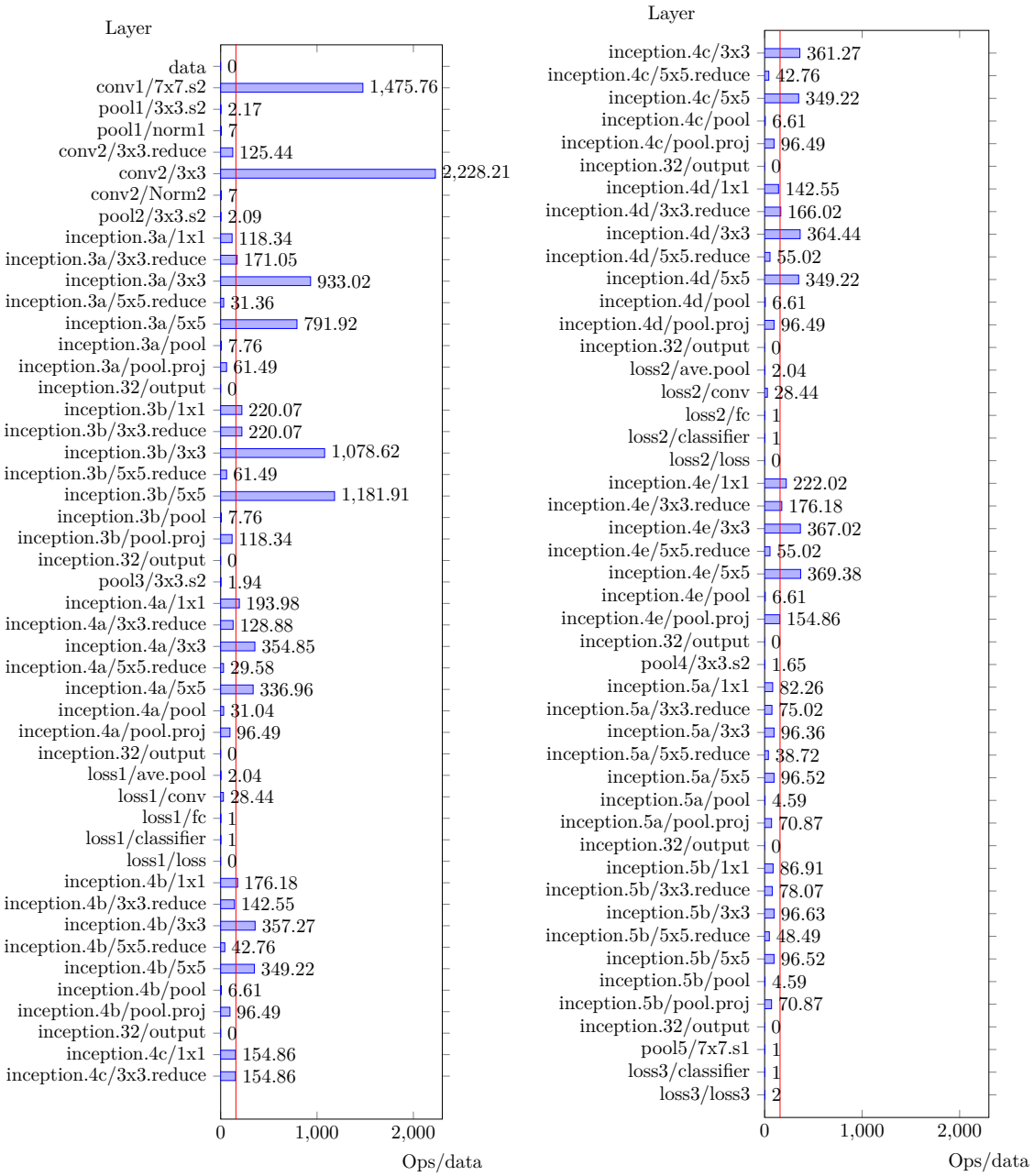


图 5.19　GoogleNet 各算子计算密度 (avg=157.90, red line)

的规模更大，在参数中，FC(全连接层) 占据了绝大多数数据量 (如图 5.21)，卷积算子参数略少，pool 等算子没有参数。当通过批处理 (batch) 技术合并多个输入一同计算，输入、输出数据量会成比例增加，参数数据量仍维持不变。在实际应用中 batch size 的常见大小为 16 或 32，一个算子内部输入、输出数据量与参数量数量没有必然的关系。

虽然模型和模型组合仍在不断发展和演化，但典型人工智能模型的基本算子发展已较为成熟，其计算需求可以根据应用的输入规模作出相对精确地估计。

人工智能模型的计算需求分析还可**依赖于框架中的计算统计工具，对模型表示或中间表示的基本单元进行计算需求统计**。模型表示或中间表示中对计算操作和数据规模有明确的定义，各框架中不乏一些统计计算量和数据量的工具，例如 THOP[173]、MXOP[174]

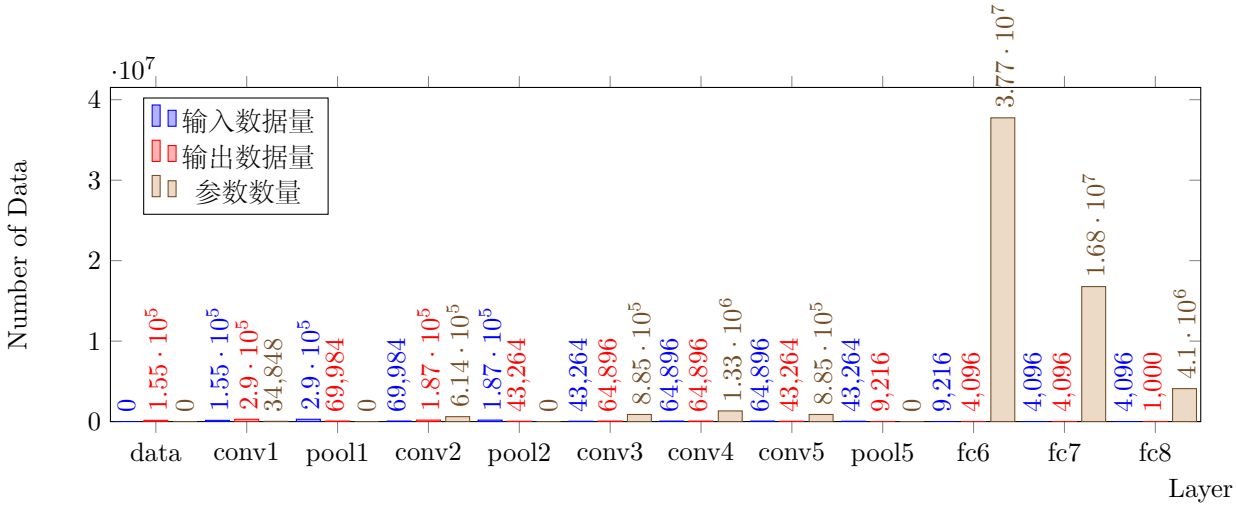


图 5.20　AlexNet 各算子数据量需求

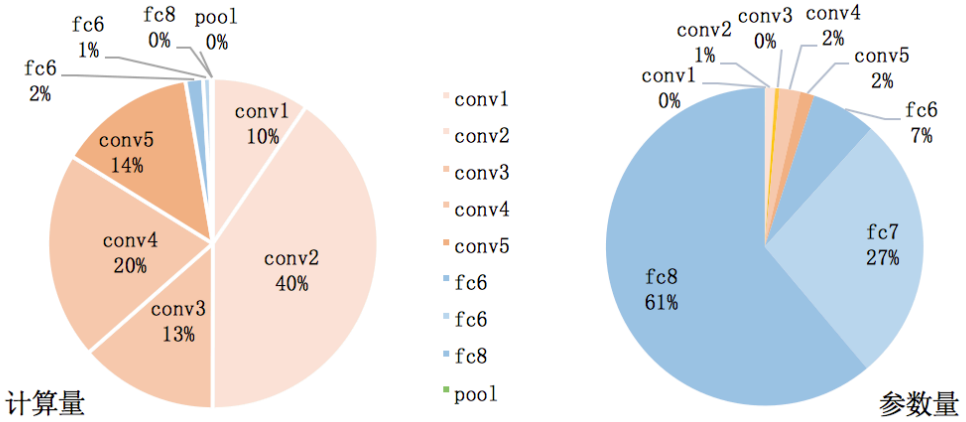


图 5.21　AlexNet 各算子计算量参数量分布

等。算子的统计除了模型中实际运行计数外，还可根据输入规模和计算量直接量化求得(如表 3.8表 3.9等)。

5.4.2　芯片级资源特征模型参数分析

体系结构模型组成多样、结构复杂，虽然上述数学模型可以完整表示其特性，但各参数的具体数值仍需要提前确定或者实时获取。即使完全量化地构建出一个体系结构模型，如何将其与应用结合，指导应用调度仍是巨大的挑战。

算子的计算密度反映的是对于硬件环境计算能力的需求，算子的数据需求量反映的是对于硬件环境存储能力的需求，访存模式已在第 3.2.6节有所讨论。结合精度需求，可以得到各基本算子对体系结构各参数的需求。

服务于调度映射的体系结构特征可以在体系结构建模的基础上，对不同的维度给予不同的优先度或权重。例如首先需要数据能够容纳于计算单元，则内存容量最为关键。算子计算密度较高时，需优先关注计算能力，较低时需优先关注访存能力。

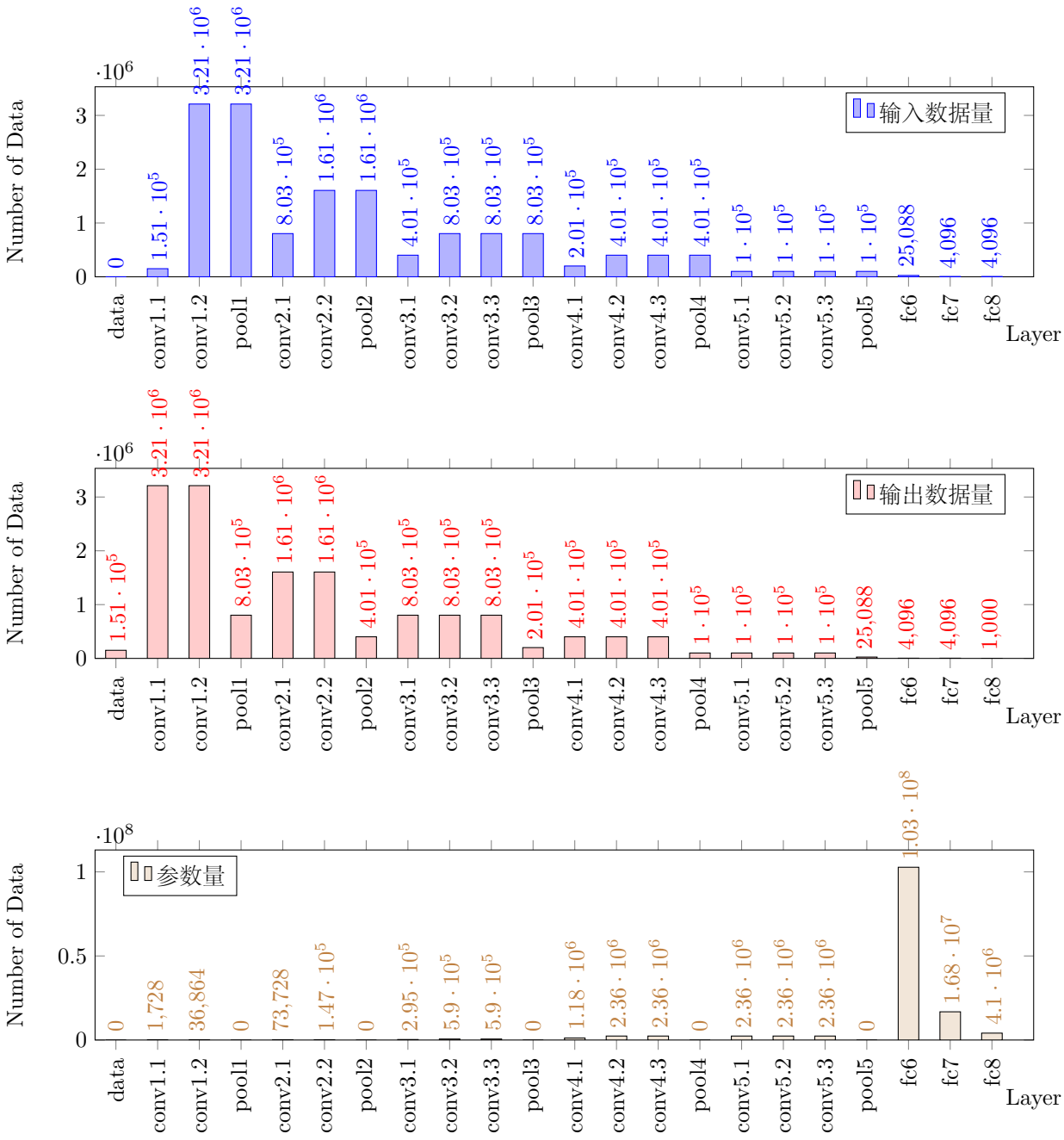


图 5.22　VGG16 各算子的数据量需求

为此，本节提出基于“预执行”的体系结构模型参数获取方案。模型可以与基于量化分析的体系结构模型相结合，为人工智能应用的拆分与调度提供指导。

(a) “预执行”的基本算子

与组成人工智能模型的基本算子类似，基于“预执行”的体系结构实用模型的执行对象也是基本算子。但由于已有的硬件结构易完全容纳人工智能基本算子，且经过融合优化的算子与基本算子有所不同，所以需要增加基本算子的测试样例规模或调整执行对象为基本模块 (如第 3.2.4节)。

(b) **基于“预执行”的芯片级资源模型参数获取**

根据给定体系结构环境，可以针对具体应用中的基本算子在不同的芯片级节点上进行轻量级的“预执行”，将计算性能、访存性能和确定的计算密度分别作为该节点对该算子单元的模型参数。

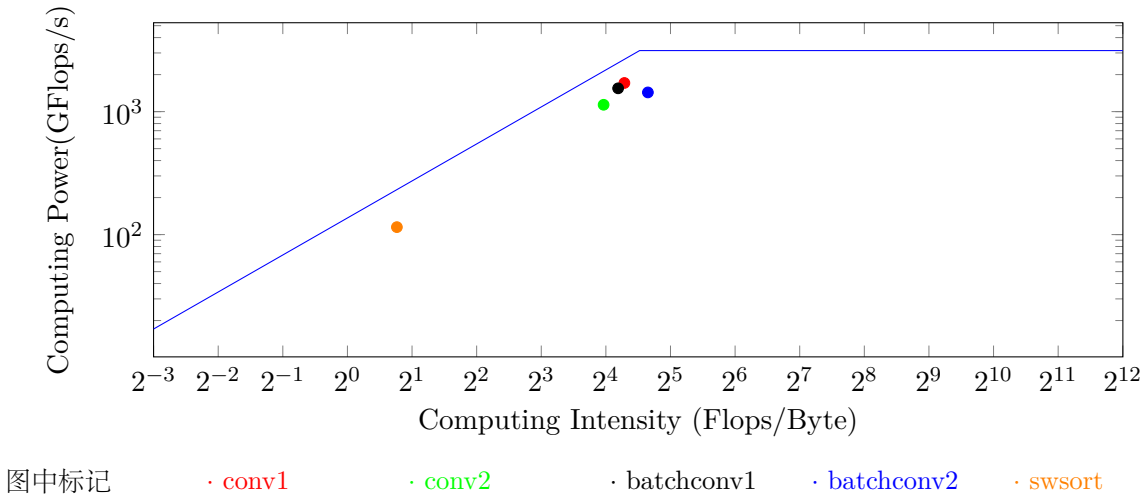


图 5.23　人工智能基本算子在 sw26010 上的运行情况

以 sw26010 为例，图 5.23中描述了几种基本算子在 sw26010 上的运行情况，sw26010 的结构包含 4 个主从核组 (CG)，每个 CG 包含 1 个主核和 64 个从核[42]。算子 conv 中 kernal size 为 3(conv1)、5(conv2)；batch size 为 32，数据完全切分，在 4 个 CG 上同时运行 4 个测试例。swsort 采用 swsort 库在 1 个 CG 上排序 1GB 双精度随机序列，扩大 4 倍作为 4CG 的近似性能结果。

最终的模型参数是包含环境中所有计算单元和应用中所有基本算子或模块的二维多通道矩阵。且在非对称多处理环境的条件下，需有额外的通信性能描述。针对动态特性，该描述需添加时序维度。

5.4.3　动态需求与动态调度策略

具有动态特征的应用与系统是未来人工智能发展的重要趋势。动态需求在第 4.2节已有分析，本节讨论动态问题的表现方式，并从调度的角度尝试解决动态的问题。

(a) **动态需求的表现形式**

数据和模型的动态特征在模型表示和中间表示层面都可以获得。数据方面的动态性可以在执行或者预执行阶段直接获得，模型层面的动态性一般以基于控制流特征的动态性描述。例如，模型在某条件下包含某模块，在其他条件下不包含该模块。

在执行时，由于目标变化而导致的动态性一般在框架内部设定，需要预先设置动态条件。例如，执行的某一阶段需要 batch size 为多少，达到设定的精度之后 batch size 变化为另一个数值。

系统和体系结构的动态变化需要运行时环境紧密配合。一方面要通过硬件资源管理

接口获得系统的状态从而适时调整其上的应用，另一方面要通过用户接口及时获得用户对系统运行的新需求从而及时调整执行状态。

以上每种需求的变化还会导致其他需求的变化，在调度中需要有联动调整的机制。

(b) **动态调度映射算法**

如何将模型中的模块需求与体系结构模型匹配是调度映射需要解决的问题。此问题可以转化为带条件的图划分问题，是 NP 问题。

本文首先采用**基于贪心的映射方法**。算法的主要思想是按照内存的需求依次映射各个算子到连续排列的计算节点上。 算法 5.5中，第 5行是算子拆分，例如将卷积算子拆

算法 5.5 基于贪心的调度映射方法

输入: 体系结构实用模型 (各节点用 $Mem[]$ 表示) ; 应用基本算子求 (整体模型用 model 表示，各算子成 DFS 序排列); 额外内存阈值 $minMem$;

输出: 映射关系 $s[]$;

1: $s[] = \varnothing$, Left(model) = model;
2: $currentX$ =Left(model).head, $currentC = 0$;
3: for $Left(model) != \varnothing$ do
4: while $R_{mem}(currentX) > Mem[currentC])$ do ▷ 直到新添加的内存需求可被容纳
5: $currentX \rightarrow split1$ and $split2$; ▷ 拆分成 2 模块
6: Left(model).pop_head();
7: Left(model).push_head($split2$);
8: Left(model).push_head($split1$); ▷ 替换剩余模型中的头为拆分后的单元
9: $currentX \leftarrow split1$; ▷ 重置指针为新的 Left(model) 的 head
10: end while
11: $s[currentC].add(currentX)$;
12: $Mem[currentC] -= R_{mem}(currentX)$;
13: $currentX = currentX$.next; Left(model).pop_top(); ▷ 开始映射下一个算子模块
14: if $Mem[currentC] < minMem$ then ▷ 开始分配下一个节点
15: $currentC$++;
16: end if
17: end for

分为两部分，有 2 种拆分方式，每部分包含不同的数据或者不同的卷积核。当给某单元分配算子导致所剩空间小于额外内存阈值后 (第 14行)，不再给该单元分配算子。贪心算法虽然能满足拆分的需求，但可能会引发大量额外的数据切分，从而引起通信负担。

为了充分匹配上文中体系结构实用模型的参数与模型需求，本文提出**基于机器学习中模拟退火的调度映射方法**。 如算法 5.6，初始映射 s0 为一个随机的划分和映射 (随机将模块或基本算子映射至计算节点)。算法中的函数 $E(s) = \alpha \cdot cut(s) + \beta \cdot balance(s)$ 为

算法 5.6 基于模拟退火的调度映射方法

输入: 变量 s0, kmax, emax; 函数 E, $Rneighbor$, P;

输出: s

```
1: s := s0; e := E (s)                                  ▷ 设置初始状态 s0, 初始能量 e
2: k := 0                                               ▷ 迭代次数 k
3: while k < kmax and e > emax do                       ▷ 若未达到最大迭代次数且结果不够好
4:     sn := Rneighbor (s)                              ▷ 随机修改映射状态到 sn
5:     en := E (sn)                                     ▷ 计算 sn 状态的评价分数
6:     if random() < P(e, en, temp(k/kmax)) then        ▷ 决定是否移至状态 sn
7:         s := sn; e := en;                            ▷ 迁移
8:     end if
9:     k := k + 1;
10: end while
11: return s;
```

模型分割的影响，是拆分数据量、映射中各个模块的需求与各个节点供给的匹配程度及所导致的均衡程度的函数。$Rneighbor$ 函数是一种随机的邻居映射状态，它可以包含多种随机方式：

- 随机选择一个可拆分算子并拆分。
- 随机选择一个分割中的边界算子或模块，将其移动至邻居分割。

(c) 函数组织与通信组织

为了实现调度映射的结果，需要通过函数组织和通信组织将调度策略嵌入执行过程中。AOT 模式下的函数组织和通信组织：在 AOT 模式下函数组织需通过代码生成实现。以神威太湖之光上的代码生成为例，函数组织包含 3 个层次：

1. MPI 调度层 + 模块调度层
2. 模块内算子主核函数层
3. 算子从核实现层

其中 2、3 层通过 swTVM[175] 实现，第 1 层先由本节的方法产生调度策略，再生成代码。JIT 模式下的函数组织和通信组织：在 JIT 模式下函数组织通过框架内调度实现，通信组织通过 RPC 等通信机制实现。

5.4.4 实验分析

(a) 实验环境

实验环境 1: 采用 sw26010 芯片进行实际运行实验，该处理器包含 4CG，每个 CG 包含 1 主核和 64 从核。芯片内存为 32GB，其中应用可用内存约 30GB。

实验环境 2: 模拟环境，采用单机环境进行模拟实验，其处理器为 Intel Xeon CPU E5-2650，操作系统 Red Hat Enterprise 6.6，编译器 gcc 4.4.7。

(b) **实验结果**

实验 1 测试在动态资源约束的条件下，现有人工智能模型的自适应调度情况。实验采用 swTVM 作为基础框架，在 sw26010 环境中测试。(由于 swTVM 还处于优化过程中，因此计算效率有限。)

在实际情况中，动态调度远比理想情况下生成计算流图的合理分配复杂。调度的对象不仅仅是流图中的计算部分，还包括数据：数据最初如何分布，中间执行过程又如何变化或流动。

针对 AlexNet 这一典型模型，构造数据和受限的动态资源，如表 5.11。受限的动态资源来自于 sw26010，以 1CG 为一个计算单元。包含样本数量之后，原来的模型计算

表 5.11　AlexNet 模型与构造数据和受限资源

模型	数据 (构造)	芯片级资源 (构造)
AlexNet	(样本实际规模 1.5GB) 样本数量: 16K 单样本规模: 227*227*3	计算单元数量: 4 每个计算单元计算能力: 783.36 GFlops/s 共享存储能力: 28GB

流图串联成为样本数量个模块。采用算法 5.6进行调度映射后，计算流图被拆分成为 4 块，按照数据并行的方式平均分布至各个计算单元 (因为模块间的依赖关系远小于算子间的依赖关系)。如图 5.24，不同颜色代表不同的划分。

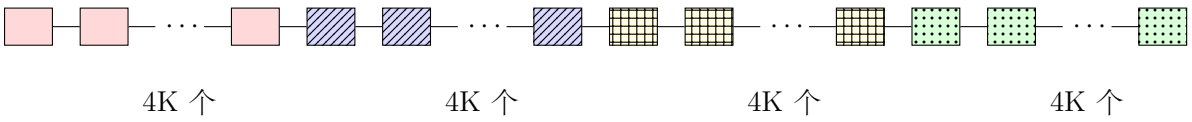


图 5.24　包含构造数据的计算流图划分

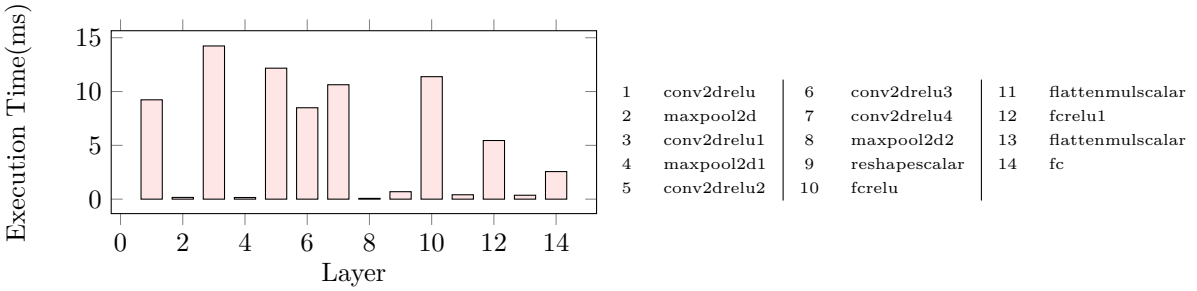
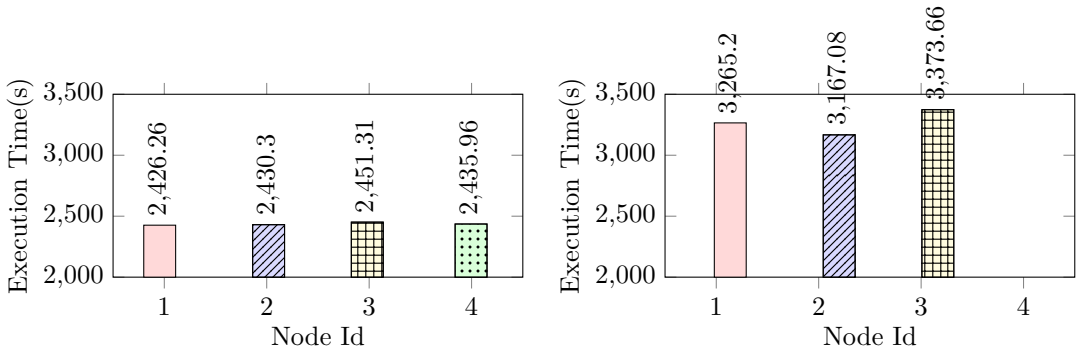


图 5.25　Alexnet 在单芯片资源 (sw26010,1CG) 上的训练单样本执行情况

图 5.25表示一个样本数据所对应的模块内部算子在 sw26010 CG 上的执行时间，这些算子都被调度进同一个执行单元。



(a) 调度前　　(b) 调度后

图 5.26　AlexNet 在构造的芯片级动态资源上运行情况

图 5.26刻画了在发现资源减少的需求后，训练一轮全样本 (epoch) 调度前与调度后所需的执行时间。可以看出，自适应调度策略可以满足该动态需求，在总体资源足够的情况下完成了应用的执行。

实验 2 测试两种调度映射方法。随机生成各算子需求较大的人工智能模型 (如表 5.12)，计算量和存储需求的单位与生成数据的表示单位成正比，只有相对意义。模型生成时采用图生成的方式构造补充的模型，以模拟结构复杂、规模庞大的未来人工智能模型。

表 5.12　人工智能计算流图调度实验用生成图与两种调度方法的需求比较

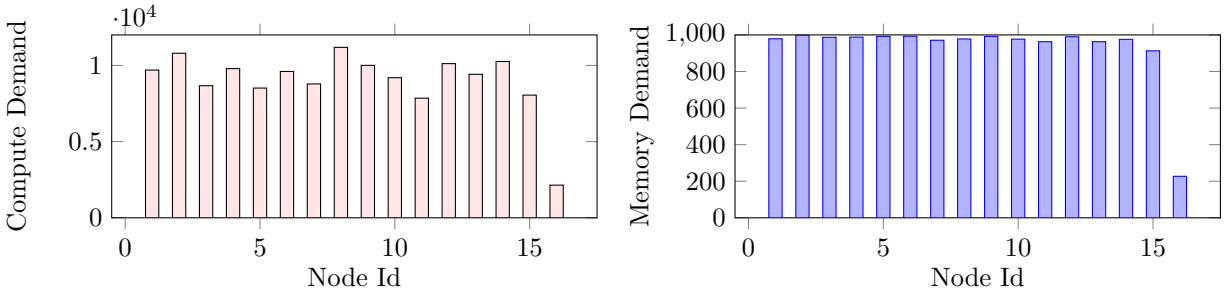
编号	算子数量	平均计算密度	模型深度	总计算量	总存储需求	通信需求－贪	通信需求－模
0	70	12 ops/data	10	42161	3446	142	100
1	269	11 ops/data	20	144064	12187	457	315
2	1039	11 ops/data	50	523974	46768	1578	1566
3	3107	11 ops/data	100	1602545	141080	2704	2730

调度结果如图 5.27，其中的模拟计算单元用不同 ID 的 Node 表示，开销为划分和调度引发的单轮通信总需求。贪心法可以较为平均地切分存储需求，偶有负载不平衡的情况；模拟退火方法可以结合均衡负载以及减少划分开销等多个目标，但其本身运行开销更大。

此外，本实验还验证了在数据并行、模型并行这两种并行方式之间，数据并行应用更广泛的原因在于并行开销较小，几乎是大部分具有非关联样本数据的人工智能应用默认并行方式。

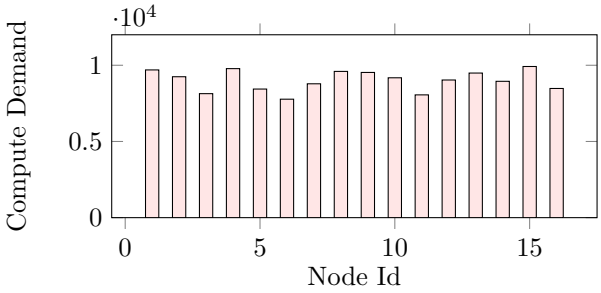
5.4.5　小结

人工智能模型的不断发展或将在不远的将来引发大量的调度需求。本节针对复杂人工智能模型在动态资源环境中运行的现实需求，提出动态自适应调度模型。该模型可以

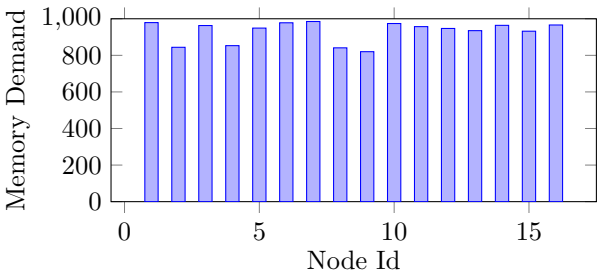


(a) 贪心法-计算需求 1

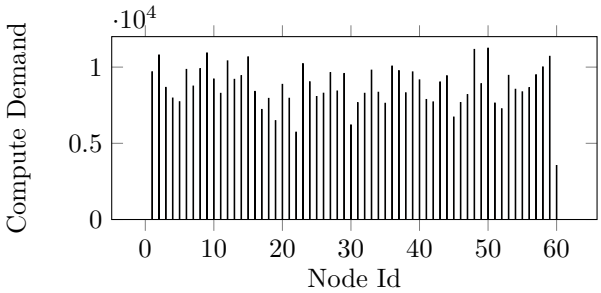
(b) 贪心法-存储需求 1



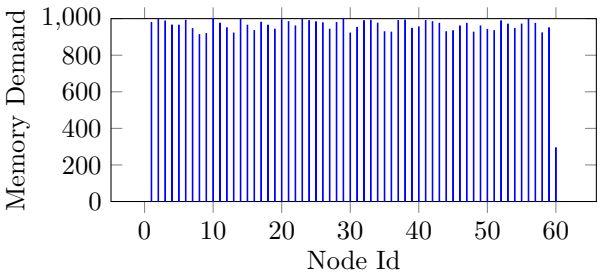
(c) 模拟退火-计算需求 1



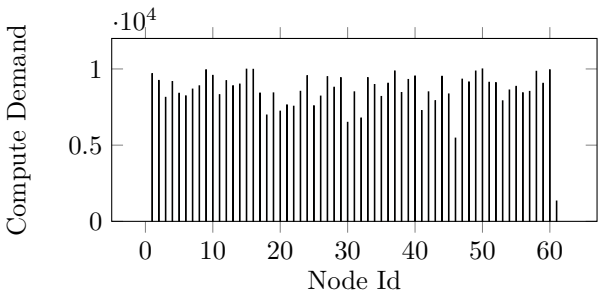
(d) 模拟退火-存储需求 1



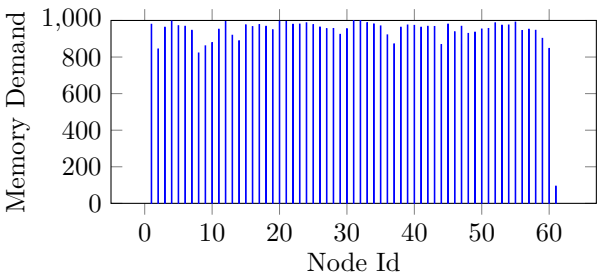
(e) 贪心法-计算需求 2



(f) 贪心法-存储需求 2



(g) 模拟退火-计算需求 2



(h) 模拟退火-存储需求 2

图 5.27　人工智能计算流图 (1,2) 的调度结果

对模型中基本单元的计算需求与芯片级的硬件资源合理匹配和映射，并在基本算子中响应执行过程中资源的动态特征。

第6章　总结和展望

人工智能已经成为各领域实现跨越式提升的重要关键技术。面对日益增长的数据量，复杂多变的应用、算法和模型，以及不断涌现的新硬件和新系统，如何有机地协同运行环境的多个方面，成为人工智能技术能够有效实现、人工智能应用能够高效执行的关键问题。

6.1　本文工作总结

人工智能环境的设计需要对其各个有机组成部分进行深入、全面的分析。仅仅从几个典型模型、几个标准数据得出的结论，不足以支撑一个真实环境的建设。本文的研究工作致力于探索并描述人工智能领域尤其是其运行环境的动态新需求，探索动态人工智能运行环境的设计方法，服务适应高性能计算系统与人工智能应用领域相结合的运行环境设计，探索新的运行环境架构从而指导下一代人工智能高性能计算系统运行环境的设计。

本文首先构建了人工智能运行环境模型，从数据、模型、资源的角度，表示了人工智能运行环境的各方面特征，全面、深入、量化地分析人工智能应用对计算环境的需求。在数据层面用多维数据特征模型，分析了实用数据集的规模并对未来可期应用的数据需求作了预估；在模型层面用多层次模型表示，综合考虑人工智能模型在执行阶段的所有层次，包括外层执行方式、基本功能模块组合图、内层计算流图，量化分析了每个层次的参数、计算需求、执行方法、规模等特征，得出人工智能模型的动态性体现在不同层次执行时的方法、参数、控制流、数据流等特征变化的结论；在资源层面用多层次资源特征模型对系统、芯片两级资源进行了特征分析和针对人工智能应用的性能建模，指出资源动态体现在执行时的状态变化以及应用对资源的需求变化中。

在人工智能运行环境模型的基础之上，设计了以“需求感知、动态响应、协同执行”为执行闭环的动态人工智能运行环境架构。该架构可以用动态响应和协同执行的方式应对人工智能应用执行过程中实时变化的目标，改变了以往运行环境中“定义——执行”的单程执行机制。

在动态人工智能运行环境架构内，选择数据、模型、资源三个方面的典型问题，围绕近似和随动等核心动态响应策略，提出和验证了 4 种关键技术。针对人工智能应用中数据关联程度较高、数据量较大的关系型数据，提出了基于邻域的局部近似方法，该方法计算开销相比于全局方法减少了 1 个数量级；针对实际应用需求动态可变和模型可调整、可稀疏化的特征，提出了基于动态指导的模型稀疏化方法，在保持模型基本功能的同时为复杂多变的应用执行情况提供高效的应对策略；针对系统级动态资源，将人工智能数据预处理过程抽象为具有数据流关系的动态模块组合，设计了自适应的协同预处理

调度策略，在超级计算机系统上为超大规模预处理数据节约了数万核组机时；针对芯片级动态资源，采用基本算子构建计算流图，设计了基于人工智能计算流图的动态自适应调度模型，以应对人工智能应用执行时来自系统、应用和用户的动态需求，在调度开销不大的情况下实现了较好的调度效果。这些关键技术在一定程度上解决了人工智能应用数据、模型、资源中的复杂动态问题。

6.2 下一步研究工作

本文所提出的动态问题都是复杂人工智能运行环境中的基本问题，值得进一步深入研究。

本文动态人工智能运行环境模型的构建和分析，需要进一步服务于下一代人工智能高性能计算环境的设计，基于可预计的应用类别、规模等信息，为计算环境的设计提供量化指标的参考。

在运行环境层面，本文设计的基于“需求感知、动态响应、协同执行”闭环的动态人工智能运行环境架构，只在原理上进行了设计，并针对每一种不同的关键技术，在现有的人工智能运行环境基础之上进行了适应性的改进。在下一步研究工作中，需要针对下一代人工智能计算环境，在基础层面进行原生的动态运行环境构建，充分整合已有的人工智能运行环境和本文设计的动态执行机制，统一上层调用接口，更灵活地应对人工智能动态需求。

在基本技术层面，面向关系型数据的局部近似方法可以扩展到更多领域或类别的数据；动态自适应调度模型还需要结合应用需求，对人工智能模型进行有意义的拆分和修改，在系统层面支持更大规模数据、模型的计算需求，面对复杂而又多样的体系架构，其抽象和运行时设计还需要针对专有特征做更细粒度的优化调度。此外，这些关键技术还需要更多应用场景的支撑，其本身也有进一步丰富和优化的空间。

随着云环境在计算中心日益重要，如何将本文提出的人工智能计算环境模型与云环境中的基本结构深度融合，甚至指导云环境中人工智能专用硬件的协同构建，也是值得深入进行的工作。

人工智能技术还在不断发展，应用领域也在不断丰富。已知的数据需求在新应用下可能会被证明需要增加或减少，可能会面临更复杂的处理方式。已有的多层次模型构建可能会遇到新的计算形式、新的抽象结构，以及新的基本计算单元。更复杂多样硬件设备的出现和端侧应用对于数据、计算的新需求，可能从根本上改变计算环境中体系结构组成方式，因此会带来新的运行环境模型。

附录 A　人工智能领域的数据

以下是人工智能领域的各类典型数据集。

表 A.1　典型自然语言处理数据集的规模

数据集	内容	规模	大小
Visual Genome [176]	图片和描述问答句对等知识	图片 108,077；位置描述 5.4 M；问答对 1.7 M；物体 3.8 M；特征 2.8 M；关系 2.3 M	15.2GB
WikiText 103 [107]	文章，短语	文章数量 28K；短语数量 103M；单词数量 268K	181MB
SQuAD2.0 [17-18]	斯坦福问答数据	100M 个问答	44MB
Question-Answer Dataset [177]	CMU 阅读理解问答数据集	558+651 个问题，每个问题多个回答 (包括回答效果评分)	8.3MB
TechTC-300 [178]	文本归类	300 个数据集；每个数据集是一个类别，包含 150-300 个文档	723MB
bAbI [179]	文本理解和推理数据集合，如	8 类数据集，每类都是问答对和相关知识	
	The (20) QA bAbI tasks	20 个 Task，每个训练数据量 100-10000，验证数据量 93-1000	
	The (6) dialog bAbI tasks	20 个 Task，训练数据量 1000，验证数据量 1000	
	The SimpleQuestions dataset	108,442 个问题和事实	
1B Word Language Model [180]	来自于新闻数据的基准语料库	1B 单词数	1.79GB
Common Crawl [181]	来自于网络爬取的数据	自 2013 年，每个月爬取约 3×10^9 个页面	PB 级

K:kilo, M: million, B: billion

表 A.2　典型图像识别数据集的规模

数据集	样本大小	元素类型	类别数量	样本数量 (训练)	(测试)	大小
MNIST [182]	28*28	B/W pixel (0-255)	10	60000	10000	1.6MB [183]
CIFAR 10 / 100 [1]	32*32	RGB pixels	10, 100	60000		176MB
ImageNet [5]	224*224	RGB pixels	图片 22K；位置 1M	15M		约 1.2TB
CTMI [184]	-	image	624	48931	-	18.3GB
Labeled Faces [185-186]	约 125x125	RGB pixels	5749	13233		233MB
MegaFace [106]	padded non-padded	RGB pixels	690K	4.7M	1M	910GB 159GB
ML-Images [187]	约 224*224	RGB pixels	11,166	17.6M	88K	估计 2TB
JFT-300M [188]		image	18291	300M		

K:kilo, M: million

表 A.3　典型关系型数据集 (图) 的规模

数据集	点个数	边个数	类别
Twitter2010	41.7m	1.47b	social network
Weibo	349.7m	44.27b	social network
UK-2014	747.8m	47.61b	web graph
Clueweb	978.4m	42.57b	web graph
Yahoo web [189]	1.4b	6.6b	web graph
Facebook12 [190]	721m	137b	social network
Twitter2012	175m	20b	social & information network [191]
Facebook	2b	400b	social network [192]
Google web	1t	-	web graph [193]
Kronecker Gen.	1.1t	4.4t	gen. graph, BFS on K, Sunway TaihuLight, BlueGene/Q [38]
Kronecker Gen.	2.2t	8.8t	gen. graph, BFS on BlueGene/Q [38]
Sogou web [6]	271.9b	12.25t	web graph
MovieLens 20M [194]	27k movies, 138k users	20m ratings, 465k tag	recommend dataset

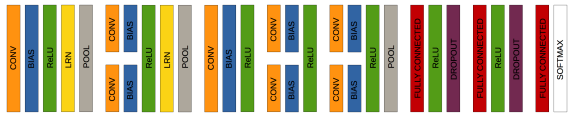
k: kilo, m: million, b: billion, t: trillion

表 A.4　其他典型数据集的规模

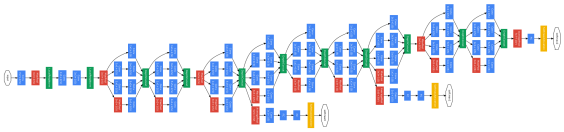
数据集	领域	内容	规模
BIT-Vehicle [195]	自动驾驶	包含车辆和其标注的交通监测图片	6 种车辆，9850 张图片
Flight simulator data [196]	飞行	飞机驾驶和模拟飞行数据	71KB
HACS [109]	视频识别	包含人行为动作的视频数据集	1.55M 个时长为 2 秒的动作视频,50K 个视频中的 140K 个已划分片段
KITTI [197]	视觉动作	运动、动作视觉数据，服务于立体声，光流，视觉测距，3D 物体检测和 3D 跟踪	GB 级别
Open Images Google [110]	位置检测	图片和其上的位置标注	1.9M 张图片上针对 600 个类别的 15.4M 个位置框 (561GB)
AlphaGo [198]	游戏	围棋对弈棋谱	160K 人类棋谱 + 30M 互博棋谱 + 人类总结的几万个模式
AlphaGo Zero [2]	游戏	围棋对弈棋谱	29M 互博棋谱
冷扑大师 [3]	游戏	德州扑克对局中的牌面信息	约 10^{160} 个决策点

K:kilo, M: million

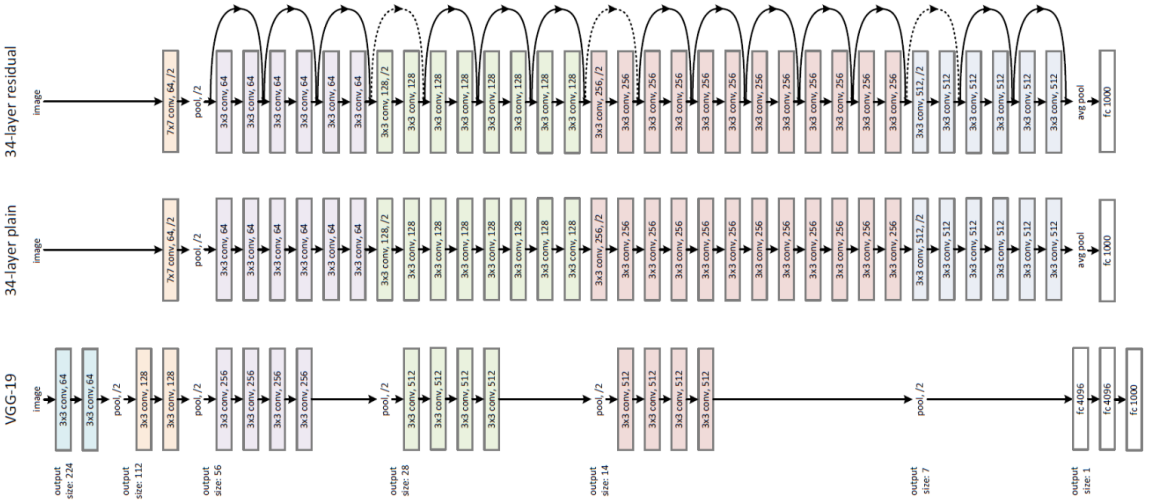
附录 B　人工智能领域的模型



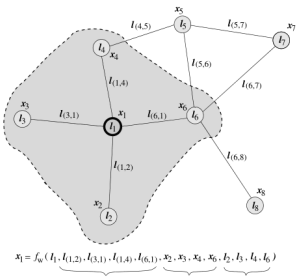
(a) Alexnet



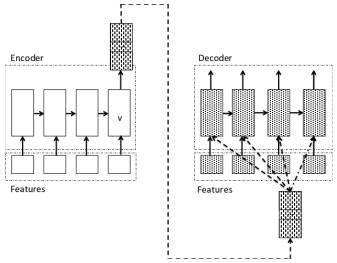
(b) GoogleNet



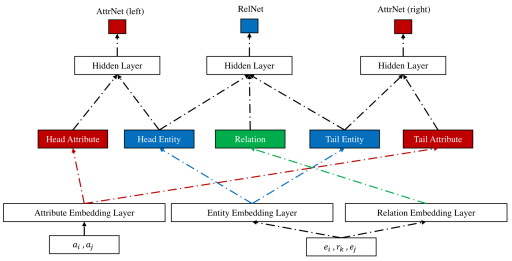
(c) VGG



(d) Graph Neural Network



(e) EncDec



(f) MultiTask Netowrk

(a) https://devblogs.nvidia.com
(b) https://i.stack.imgur.com
(c) https://cdn-images-1.medium.com
(d) http://image.mamicode.com
(e) https://www.researchgate.net
(f) https://www.groundai.com/media/arxiv_projects/7222/architecture.svg

图 B.1　一些典型的模型结构

表 B.1列举了主要的模型集合。由于人工智能模型仍在不断演进和发展，模型集合也在不断

的增加和变化中。

表 B.1　模型集合列举

模型集合	所含模型个数 (同类可合并)	主要模型类别
TensorFlow Offical Models[199]	5	决策树，CNN
TensorFlow Slim Models[200]	6	CNN
TensorFlow Research Models[201]	51	涵盖较全面
caffe model zoo[202]	44	CNN
caffe2 models[203]	16	CNN
pytorch models[204]	21	CNN, RNN, GAN, Reinforcement
keras model zoo[205]	87	涵盖较全面
The GAN Zoo[206]	501	GAN
Awesome-CoreML-Models[207]	32	CNN, RNN
MXNet model zoo[208]	20	CNN, RNN, GAN
ONNX Model Zoo[36]	45	CNN、物体识别等

以下列表列举了上述模型集合中的所有模型：

- 1d-cnn-lstm
- 1d-cnn
- A3C: Asynchronous advantage actor-critic
- adversarial-autoencoder
- adversarial crypto
- adversarial text
- Age and Gender Classification using CNN
- AgeNet
- alexnet
- AnimeScale2x
- ArcFace
- Artistic Style Transfer Gatys 2015
- Artists Recommendation (location and genre).
- async-rl
- attention ocr
- audioset
- autoencoders
- auxiliary-classifier-gan
- bidirectional-gan
- bidirectional-lstm-on-the-imdb-dataset
- boosted trees
- boundary-seeking-gan
- brain coder (reinforcement learning)
- caffenet
- capsnet
- Car Recognition
- Cascaded Fully Convolutional Networks for Biomedical Image Segmentation
- CCNN: Constrained Convolutional Neural Networks for Weakly Supervised Segmentation
- character-level-convolutional-neural-nets-for-text
- chess-reinforcement-learning
- CNN Cascade
- CNN Models for Salient Object Subitizing.
- CNN Object Proposal Models for Salient Object Detection
- cognitive mapping and planning
- (Neural) Collaborative Filtering
- colorful-image-colorization
- compression by pre-trained Residual GRU network.
- Conditional Random Fields as Recurrent Neural Networks
- conditional-gan
- context-conditional-gan
- context-encoder
- Conversational Model: Google 2015
- coupled-gans
- cvt text: semi-supervised sequence learning with cross-view training.
- cyclegan
- DCGANs
- Deep Convolutional GAN: DCGAN 2015
- Deep Hand
- Deep Jazz
- Deep Learning of Binary Hash Codes for Fast Image Retrieval
- Deep Networks for Earth Observation
- Deep Neural Network Language Models
- Deep Photo Style Transfer 2017
- Deep Q Learning Deep Mind 2015
- Deep voice: Real time neural text to speech
- deep-convolutional-gan
- deep-dreams
- deep-networks-with-stochastic-depth

- deep-reinforcement-learning-for-keras
- deeplab (labeling).
- deepmask-object-segmentation
- DeepYeast
- deep contextual bandits
- deep speech.
- delf: deep local features for image matching and retrieval.
- densely-connected-convolutional-networks-2
- densenet121
- dfi: Deep Feature Interpolation
- differential privacy.
- discogan-keras
- DocumentClassification - Classify news articles into 1 of 5 categories.
- domain adaptation: domain separation networks.
- dqn-keras
- dualgan
- DUC
- Emotion FerPlus
- Emotion Recognition
- EmotionNet.
- emotion ferplus
- Exermote.
- Facial Landmark Detection with Tweaked CNN
- fast RCNN
- faster RCNN (detectron)
- fasttext-on-the-imdb-dataset
- fivo: filtering variational objectives for training generative sequence models.
- flappybird-dqn
- FlickrStyle - Detect the artistic style of images.
- Food101 - Predict the type of foods from images.
- fractalnet-keras
- Fully Convolutional Networks for Semantic Segmentation (FCNs)
- GAN: generative adversarial networks.
- GenderNet.
- Generating sequences: LSTM 2013
- GestureAI - Recommend an artist based on given location and genre.
- Google's Neural Machine Translation System
- googlenet
- GoogLeNet cars
- HED - Detect nested edges from a color image.
- Holistically-Nested Edge Detection
- Hybrid Computing with a NN and external memory Nature 2016
- im2txt: image captioning.
- Image Captioning with attention Bengio 2016
- Image Classification: Microsoft-ResNet 2015
- Image Colorization UC Berkeley 2016
- Image Super resolution using deep convolutional networks
- image-analogies
- iMessage Spam Detection - Detect whether a message is spam.
- inception v1, v2, v3, BN
- infogan
- keras-realtime-multi-person-pose-estimation
- keras-yolov3
- keypointnet: discovery of latent 3D keypoints
- languagedialect-identification-with-multiple-character-level-cnns
- Learning Structured Sparsity in Deep Neural Networks
- learning to remember rare events: (life-long memory module)
- learning unsupervised learning: a meta-learned unsupervised learning update rule.
- Lets Keep it Simple
- lexnet nc
- lfads (autoencoder)
- lm 1b: language modeling on the one billion word benchmark.
- lm commonsense: commonsense reasoning using language models.
- locationNet
- lsgan-keras
- LSTM image captioning
- LSTM Q&A system
- LSTM sentiment analysis
- lstm-based-network-on-the-babi-dataset
- lstm-on-the-imdb-dataset-text
- lstm-text-generation
- lstm-to-predict-gender-of-a
- Making the V in VQA Matter
- marco: automating the evaluation of crystallization experiments.
- Mask-RCNN
- maskgan: text generation with GANs.
- mask rcnn 2go
- Matrix Factorization (针对电影评分数据，推荐系统)
- memory-network-on-the-babi-dataset
- Mixture DCNN
- mnist
- MobileNet - Detects the dominant objects present in an image.
- monolingual-and-multilingual-image-captioning
- Mulimodal Compact Bilinear Pooling for VQA
- NamesDT
- namignizer: recognize and generate names.
- NASNet-A mobile/large
- Network in Network 2014
- Neural Activation Constellations
- Neural Doodle 2016
- Neural Machine Translation 2014
- Neural Machine Translation: Google 2016
- neural-programmer-interpreter
- neural-style-transfer
- neural gpu: highly parallel neural computer.
- neural programmer: neural network augmented with

logic and mathematic operations.
- next frame prediction (cross CNN)
- Nin
- Nudity - Classifies an image either as NSFW (nude) or SFW (not nude)
- object detection: localizing and identifying multiple objects in a single image.
- Oxford102 - Detect the type of flowers from images.
- ParseNet: Looking wider to see better
- Pascal VOC 2012 Multilabel Classification Model
- pcl rl: reinforcement learning algorithms
- Personality Detection (sentences).
- pix2pix-2
- pixelda
- Places CNN
- Places CNDS models on Scene Recognition
- PNASNet-5 mobile/large
- popular-image-segmentation-models
- Pose-Aware CNN Models (PAMs) for Face Recognition
- PoseEstimation - Estimating human pose from a picture for mobile.
- pretraining-on-a-different-dataset
- ptn: perspective transformer nets for 3D object reconstruction.
- qa kg: module networks for question answering on knowledge graphs.
- rcnn ilsvrc13
- real-time-style-transfer
- real-valued non-volume preserving (real NVP) transformations
- rebar
- Recommender systems: DropoutNet
- ResFace101: ResNet-101 for Face Recognition
- ResNet-101 for regressing 3D morphable face models (3DMM) from single images
- ResNet50 - Detects the dominant objects present in an image.
- resnet
- reuters-topic-classification
- RN1015k500 - Predict the location where a picture was taken.
- SegNet and Bayesian SegNet
- semi-supervised-gan
- Sentiment Polarity
- SentimentVision
- seq2species
- Sequence to sequence with attention
- sequence-to-sequence-learning-for-performing
- shufflenet
- siamese-network
- simple-cnn-on-cifar10 / mnist
- single shop detection(SSD)
- single-shot-multibox-detector-keras
- Skip Thoughts sentence to vector
- smile-detection-with-a-cnn
- snapshot-ensembles
- Sound Generative models
- Spatial Transformation Network STN 2016
- Speech recognition with deep recurrent neural networks
- Speech Recognition: Deep Speech 2 2015
- squeezenet
- SSD
- stateful-lstm
- street: (name of a street by Deep RNN)
- Striving for Simplicity: The All Convolutional Net
- struct2depth: unsupervised learning of depth and ego-motion.
- structurally-constrained-recurrent-nets-text-generation
- Style Transfer - Apply artistic styles to images.
- Super resolution IEEE 2015
- super-resolution-gan
- Supervised Learning of Semantics-Preserving Hash via Deep CNN
- swivel: the Swivel algorithm for generating word embeddings.
- syntaxnet
- t-sne-of-image-cnn-fc7-activations
- tcn: Self-supervised representation learning from multi-view video.
- Text to Image Synthesis (GAN)
- TextDetection - Detecting text using Vision built-in model in real-time.
- TextRecognition - Recognizing text using ML Kit built-in model in real-time.
- textsum: sequence-to-sequence with attention model for text summarization.
- Time Series Forecasting
- Towards Principled Design of Deep Convolutional Networks: Introducing SimpNet
- transformer: spatial transformer network
- ultrasound-nerve-segmentation
- Unpaired Image to Image Translation using Cycle consistent Adversarial Network
- Using Ranking-CNN for Age Estimation
- using-pre-trained-word-embeddings
- variational-autoencoder-with-deconvolutions
- VGG 16, 19, etc.
- vid2depth (depth and ego-motion)
- video prediction (neural advection)
- Visual Dialog
- visual-semantic-embedding
- visualizing-the-filters-learned-by-a
- VQA: Visual Question Answering
- wasserstein-gan-gp
- wide-residual-networks
- wide deep
- Word2Vec
- YOLO v1,v2,v3,2016
- zfnet512

表 C.1 典型芯片参数

	型号	描述	计算能力	访存能力	内存容量	高速缓存	外部接口	功耗	(Max)ops/Byte(ops/*bit)
2014	intel E7 8894 v4	24cores, 2.4GHz	921.6 GF	max 85 GB/s	max 3.07TB	60MB cache	PCIe	165w	10.84=1.25ops/(64bit)
2011	AMD Opteron 6274	16cores, 2.2GHz		102.4 GB/s		32MB		115w	
2017	IBM Power 9	22C(AC922) 14nm, 3.07GHz			<1TB		OpenCAPI PCIe NVLink		
2014	sw26010	4CG	3.06 TF	136.51 GB/s	32GB	(LDM)64KB, 16MB	PCIe	382w?	22.97 = 3ops/(64bit)
2014	Nvidia K40	GPU, 28nm, 15SM	1.4 TF	288 GB/s	12GB	48KB/SM		235w	4.97
2017	Nvidia V100	GPU, 80SM	(DP) 7.8 TF (SP) 15.7 TF (HP) 125 TF	900 GB/s	16GB	(reg)256KB, (shared mem)96 KB	NVLink PCIe	300w	1ops/(64bit) 4.25ops/(32bit) 68ops/(16bit)
2015	TPU 1	host + 4*die	(8bit) 92 T	34 GB/s	32GB	28MB /die	PCIe	75w	2770
2017	TPU 2	4 chips	(HP) 180 TF	600 GB/s	16GB HBM				
2018	TPU 3	4 chips	(HP)? TF	? GB/s	64GB HBM		PCIe		
2016	Xeon Phi (KNL) 7290F	72 cores, 288 threads	3456 GF	400+ GB/s	384GB	36MB	PCIe	260w	
2016	intel Stratix 10	FPGA	(SP) 9.2 TF	1 TB/s	<16GB	28MB	PCIe		9.2
2016	Diannao	16 tie	(SP)2.09TF (HP)5.58TF	need 467.3 GB/s		2MB * 16	PCIe		1.1ops/(32bit) 6ops/(16bit)
2014	IBM TrueNorth		4096 cores * 256 neurns						

参考文献

[1] KRIZHEVSKY A, HINTON G. Learning multiple layers of features from tiny images[R]. [S.l.]: Citeseer, 2009.

[2] SILVER D, SCHRITTWIESER J, SIMONYAN K, et al. Mastering the game of go without human knowledge[J]. Nature, 2017, 550(7676):354.

[3] BROWN N, SANDHOLM T. Libratus: The superhuman AI for no-limit poker[C/OL]// Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. 2017: 5226-5228. DOI: 10.24963/ijcai.2017/772.

[4] DEVLIN J, CHANG M W, LEE K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[5] DENG J, DONG W, SOCHER R, et al. ImageNet: A Large-Scale Hierarchical Image Database [C]//CVPR09. [S.l.: s.n.], 2009.

[6] LIN H, ZHU X, YU B, et al. Shentu: processing multi-trillion edge graphs on millions of cores in seconds[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018. [S.l.]: IEEE / ACM, 2018: 56:1-56:11.

[7] RADFORD A, WU J, CHILD R, et al. Language models are unsupervised multitask learners[J]. OpenAI Blog, 2019, 1:8.

[8] KIPF T N, WELLING M. Semi-supervised classification with graph convolutional networks[J]. arXiv preprint arXiv:1609.02907, 2016.

[9] ANDREAS J, ROHRBACH M, DARRELL T, et al. Neural module networks[C/OL]//2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. 2016: 39-48. DOI: 10.1109/CVPR.2016.12.

[10] KRIZHEVSKY A. One weird trick for parallelizing convolutional neural networks[J]. CoRR, 2014, abs/1404.5997.

[11] Top 500[EB/OL]. 2019. www.top500.org.

[12] KURTH T, TREICHLER S, ROMERO J, et al. Exascale deep learning for climate analytics [C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018. [S.l.]: IEEE / ACM, 2018: 51:1-51:12.

[13] JOUBERT W, WEIGHILL D A, KAINER D, et al. Attacking the opioid epidemic: determining the epistatic and pleiotropic genetic architectures for chronic pain and opioid addiction[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018. [S.l.]: IEEE / ACM, 2018: 57:1-57:14.

[14] KOGGE P, BORKAR S, CAMPBELL D, et al. Exascale computing study: Technology challenges in achieving exascale systems[J]. Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Techinal Representative, 2008, 15.

[15] GEORGANAS E, EGAN R, HOFMEYR S A, et al. Extreme scale de novo metagenome assembly [C]//Proceedings of the International Conference for High Performance Computing, Networking,

Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018. [S.l.]: IEEE / ACM, 2018: 10:1-10:13.

[16] LESKOVEC J, KREVL A. SNAP Datasets: Stanford large network dataset collection[EB/OL]. 2014. http://snap.stanford.edu/data.

[17] RAJPURKAR P, ZHANG J, LOPYREV K, et al. Squad: 100, 000+ questions for machine comprehension of text[J]. CoRR, 2016, abs/1606.05250.

[18] RAJPURKAR P, JIA R, LIANG P. Know what you don't know: Unanswerable questions for squad [J]. CoRR, 2018, abs/1806.03822.

[19] Google dataset search[EB/OL]. https://toolbox.google.com/datasetsearch.

[20] 韩文弢. 时序图数据处理技术研究[D]. 北京: 清华大学, 2015.

[21] MOBAHI H, COLLOBERT R, WESTON J. Deep learning from temporal coherence in video[C]// Proceedings of the 26th Annual International Conference on Machine Learning. [S.l.]: ACM, 2009: 737-744.

[22] THERIAULT C, THOME N, CORD M. Dynamic scene classification: Learning motion descriptors with slow features analysis[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2013: 2603-2610.

[23] WANG F, LIANG J, DANG C. Attribute reduction for dynamic data sets[J]. Applied Soft Computing, 2013, 13(1):676-689.

[24] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[C]//Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States. [S.l.: s.n.], 2012: 1106-1114.

[25] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition [J]. CoRR, 2014, abs/1409.1556.

[26] SZEGEDY C, LIU W, JIA Y, et al. Going deeper with convolutions[C/OL]//IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. 2015: 1-9. DOI: 10.1109/CVPR.2015.7298594.

[27] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C/OL]//2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. 2016: 770-778. DOI: 10.1109/CVPR.2016.90.

[28] GIRSHICK R B, DONAHUE J, DARRELL T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C/OL]//2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014. 2014: 580-587. DOI: 10.1109/CVPR.2014.81.

[29] HE K, ZHANG X, REN S, et al. Spatial pyramid pooling in deep convolutional networks for visual recognition[J/OL]. IEEE Trans. Pattern Anal. Mach. Intell., 2015, 37(9):1904-1916. DOI: 10.1109/TPAMI.2015.2389824.

[30] GIRSHICK R B. Fast R-CNN[J]. CoRR, 2015, abs/1504.08083.

[31] REN S, HE K, GIRSHICK R B, et al. Faster R-CNN: towards real-time object detection with region proposal networks[C]//Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada. [S.l.: s.n.], 2015: 91-99.

[32] LIU W, ANGUELOV D, ERHAN D, et al. SSD: single shot multibox detector[C/OL]//Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I. 2016: 21-37. DOI: 10.1007/978-3-319-46448-0_2.

[33] REDMON J, DIVVALA S K, GIRSHICK R B, et al. You only look once: Unified, real-time object detection[C/OL]//2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. 2016: 779-788. DOI: 10.1109/CVPR.2016.91.

[34] GOODFELLOW I J, POUGET-ABADIE J, MIRZA M, et al. Generative adversarial nets[C]// Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada. [S.l.: s.n.], 2014: 2672-2680.

[35] JIA Y, SHELHAMER E, DONAHUE J, et al. Caffe: Convolutional architecture for fast feature embedding[C/OL]//Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014. 2014: 675-678. DOI: 10.1145/2647868.2654889.

[36] Open neural network exchange (onnx) model zoo[EB/OL]. [2019-03-01]. https://github.com/onnx/models.

[37] DONGARRA J J. The linpack benchmark: An explanation[C]//International Conference on Supercomputing. [S.l.]: Springer, 1987: 456-474.

[38] Graph 500[EB/OL]. 2017. www.graph500.org.

[39] BEN-NUN T, BESTA M, HUBER S, et al. A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning[C]//[S.l.]: IEEE, 2019.

[40] CHEN Y, CHEN T, XU Z, et al. Diannao family: energy-efficient hardware accelerators for machine learning[J]. Communications of the ACM, 2016, 59(11):105-112.

[41] JOUPPI N P, YOUNG C, PATIL N, et al. In-datacenter performance analysis of a tensor processing unit[C]//2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). [S.l.]: IEEE, 2017: 1-12.

[42] HAOHUAN F, WEI X. The sunway taihulight supercomputer: system and applications[J]. Science China Information Sciences, 2016(007):109-124.

[43] ZHANG S, DU Z, ZHANG L, et al. Cambricon-x: An accelerator for sparse neural networks [C/OL]//49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016. 2016: 20:1-20:12. DOI: 10.1109/MICRO.2016.7783723.

[44] CHAMBERLAIN B L, DEITZ S, HRIBAR M B, et al. Chapel[J]. Padua et al.[32], 2005:249-256.

[45] CHETLUR S, WOOLLEY C, VANDERMERSCH P, et al. cudnn: Efficient primitives for deep learning[J]. arXiv preprint arXiv:1410.0759, 2014.

[46] FANG J, FU H, ZHAO W, et al. swdnn: A library for accelerating deep learning applications on sunway taihulight[C/OL]//2017 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2017, Orlando, FL, USA, May 29 - June 2, 2017. 2017: 615-624. DOI: 10.1109/IPDPS.2017.20.

[47] RAGAN-KELLEY J, BARNES C, ADAMS A, et al. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines[C/OL]//ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013. 2013: 519-530. DOI: 10.1145/2491956.2462176.

[48] CHEN T, MOREAU T, JIANG Z, et al. TVM: end-to-end optimization stack for deep learning[J]. CoRR, 2018, abs/1802.04799.

[49] ROTEM N, FIX J, ABDULRASOOL S, et al. Glow: Graph lowering compiler techniques for neural networks[J]. CoRR, 2018, abs/1805.00907.

[50] ABADI M, BARHAM P, CHEN J, et al. Tensorflow: A system for large-scale machine learning [C]//12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). [S.l.: s.n.], 2016: 265-283.

[51] PASZKE A, GROSS S, CHINTALA S, et al. Automatic differentiation in pytorch[J]. 2017.

[52] TOKUI S, OONO K, HIDO S, et al. Chainer: a next-generation open source framework for deep learning[C/OL]//Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS). 2015. http://learningsys.org/papers/LearningSys_2015_paper_33.pdf.

[53] KATHAROPOULOS A, FLEURET F. Not all samples are created equal: Deep learning with importance sampling[C]//Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. [S.l.: s.n.], 2018: 2530-2539.

[54] OPATRNY J. Total ordering problem[J/OL]. SIAM J. Comput., 1979, 8(1):111-114. DOI: 10.1137/0208008.

[55] PAGE L, BRIN S, MOTWANI R, et al. The pagerank citation ranking: Bringing order to the web [C/OL]//Proceedings of the 7th International World Wide Web Conference. Brisbane, Australia, 1998: 161-172. citeseer.nj.nec.com/page98pagerank.html.

[56] GAO B, WANG T, LIU T. Ranking on large-scale graphs with rich metadata[C/OL]//Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011 (Companion Volume). 2011: 285-286. DOI: 10.1145/1963192.1963313.

[57] WELCH M J, SCHONFELD U, HE D, et al. Topical semantics of twitter links[C/OL]//Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011. 2011: 327-336. DOI: 10.1145/1935826.1935882.

[58] PFEFFER J, CARLEY K M. k-centralities: local approximations of global measures based on shortest paths[C/OL]//Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume). 2012: 1043-1050. DOI: 10.1145/2187980.2188239.

[59] RIONDATO M, UPFAL E. ABRA: approximating betweenness centrality in static and dynamic graphs with rademacher averages[C/OL]//Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016. 2016: 1145-1154. DOI: 10.1145/2939672.2939770.

[60] BRESSAN M, PESERICO E, PRETTO L. The power of local information in pagerank[C/OL]// 22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume. 2013: 179-180. DOI: 10.1145/2487788.2487878.

[61] BRESSAN M, PESERICO E, PRETTO L. Approximating pagerank locally with sublinear query complexity[J]. CoRR, 2014, abs/1404.1864.

[62] LAI S, SHAO B, XU Y, et al. Parallel computations of local pagerank problem based on graphics processing unit[J/OL]. Concurrency and Computation: Practice and Experience, 2017, 29(24). DOI: 10.1002/cpe.4245.

[63] WEI W, ERENRICH J, SELMAN B. Towards efficient sampling: Exploiting random walk strategies[C]//Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA. [S.l.: s.n.], 2004: 670-676.

[64] GJOKA M, KURANT M, BUTTS C T, et al. Practical recommendations on crawling online social networks[J/OL]. IEEE Journal on Selected Areas in Communications, 2011, 29(9):1872-1892. DOI: 10.1109/JSAC.2011.111011.

[65] STUTZBACH D, REJAIE R, DUFFIELD N G, et al. On unbiased sampling for unstructured peer-to-peer networks[C/OL]//Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference, IMC 2006, Rio de Janeriro, Brazil, October 25-27, 2006. 2006: 27-40. DOI: 10.1145/1177080.1177084.

[66] BAR-YOSSEF Z, GUREVICH M. Random sampling from a search engine's index[C/OL]//WWW '06: Proceedings of the 15th international conference on World Wide Web. New York, NY, USA: ACM, 2006: 367-376. http://dx.doi.org/10.1145/1135777.1135833.

[67] SERCU T, PUHRSCH C, KINGSBURY B, et al. Very deep multilingual convolutional neural networks for LVCSR[J]. CoRR, 2015, abs/1509.08967.

[68] SUN Y, LIANG D, WANG X, et al. Deepid3: Face recognition with very deep neural networks[J]. CoRR, 2015, abs/1502.00873.

[69] DUQUE A B, SANTOS L L J, MACÊDO D, et al. Squeezed very deep convolutional neural networks for text classification[J]. CoRR, 2019, abs/1901.09821.

[70] THOM M. Sparse neural networks[D]. [S.l.]: University of Ulm, 2015.

[71] LIU B, WANG M, FOROOSH H, et al. Sparse convolutional neural networks[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2015: 806-814.

[72] NARANG S, ELSEN E, DIAMOS G, et al. Exploring sparsity in recurrent neural networks[J]. arXiv preprint arXiv:1704.05119, 2017.

[73] XU L, CHOY C, LI Y. Deep sparse rectifier neural networks for speech denoising[C/OL]//IEEE International Workshop on Acoustic Signal Enhancement, IWAENC 2016, Xi'an, China, September 13-16, 2016. 2016: 1-5. DOI: 10.1109/IWAENC.2016.7602891.

[74] SALDANHA L B, BOBDA C. Sparsely connected neural networks in FPGA for handwritten digit recognition[C/OL]//17th International Symposium on Quality Electronic Design, ISQED 2016, Santa Clara, CA, USA, March 15-16, 2016. 2016: 113-117. DOI: 10.1109/ISQED.2016.7479185.

[75] NG A, et al. Sparse autoencoder[J]. CS294A Lecture notes, 2011, 72(2011):1-19.

[76] ZHANG Y, HOU X, LV Y, et al. Sparse autoencoder based deep neural network for voxelwise detection of cerebral microbleed[C/OL]//22nd IEEE International Conference on Parallel and Distributed Systems, ICPADS 2016, Wuhan, China, December 13-16, 2016. 2016: 1229-1232. DOI: 10.1109/ICPADS.2016.0166.

[77] KORDMAHALLEH M M, SEFIDMAZGI M G, HOMAIFAR A. A sparse recurrent neural network for trajectory prediction of atlantic hurricanes[C/OL]//Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016. 2016: 957-964. DOI: 10.1145/2908812.2908834.

[78] GARCÍA S, LUENGO J, HERRERA F. Intelligent systems reference library: volume 72 data preprocessing in data mining[M/OL]. Springer, 2015. DOI: 10.1007/978-3-319-10247-4.

[79] IOFFE S, SZEGEDY C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015. [S.l.: s.n.], 2015: 448-456.

[80] LIONG V E, LU J, WANG G. Face recognition using deep PCA[C/OL]//9th International Conference on Information, Communications & Signal Processing, ICICS 2013, Tainan, Taiwan, December 10-13, 2013. 2013: 1-5. DOI: 10.1109/ICICS.2013.6782777.

[81] MENG Q, CHEN W, WANG Y, et al. Convergence analysis of distributed stochastic gradient descent with shuffling[J]. CoRR, 2017, abs/1709.10432.

[82] PITTMAN R, GUAN H, SHEN X, et al. Exploring flexible communications for streamlining DNN ensemble training pipelines[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018. [S.l.]: IEEE / ACM, 2018: 64:1-64:12.

[83] 陈文光. 大数据与高性能计算[J]. 大数据, 2015, 3:1-6.

[84] TEAM AT THE UNIVERSITY OF WISCONSIN MADISON H. High throughput computing

[EB/OL]. 2019. http://research.cs.wisc.edu/htcondor/htc.html.

[85] KALMEGH P, NAVATHE S B. Graph database design challenges using HPC platforms[C/OL]// 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, November 10-16, 2012. 2012: 1306-1309. DOI: 10.1109/SC.Companion.2012.160.

[86] ISLAM N, LU X, RAHMAN M, et al. Triple-h: A hybrid approach to accelerate hdfs on hpc clusters with heterogeneous storage architecture[C/OL]//2015. DOI: 10.1109/CCGrid.2015.161.

[87] ISLAM N S, SHANKAR D, LU X, et al. Accelerating I/O performance of big data analytics on HPC clusters through rdma-based key-value store[C/OL]//44th International Conference on Parallel Processing, ICPP 2015, Beijing, China, September 1-4, 2015. 2015: 280-289. DOI: 10.1109/ICPP.2015.79.

[88] MIYOSHI T, KONDO K, TERASAKI K. Big ensemble data assimilation in numerical weather prediction[J/OL]. IEEE Computer, 2015, 48(11):15-21. DOI: 10.1109/MC.2015.332.

[89] MIYOSHI T, KUNII M, RUIZ J, et al. "big data assimilation"revolutionizing severe weather prediction[J]. Bulletin of the American Meteorological Society, 2016, 97(8):1347-1354.

[90] NGIAM J, KHOSLA A, KIM M, et al. Multimodal deep learning[C]//Proceedings of the 28th international conference on machine learning (ICML-11). [S.l.: s.n.], 2011: 689-696.

[91] ZHANG D, DING W, ZHANG B, et al. Heterogeneous deep model fusion for automatic modulation classification[J]. no., January, 2018.

[92] Xla: Domain-specific compiler for linear algebra to opti- mizes tensorflow computations[EB/OL]. https://www.tensorflow.org/performance/xla.

[93] WANG E, ZHANG Q, SHEN B, et al. Intel math kernel library[M]//High-Performance Computing on the Intel® Xeon Phi™. [S.l.]: Springer, 2014: 167-188.

[94] FANG J, FU H, ZHAO W, et al. swdnn: A library for accelerating deep learning applications on sunway taihulight[C]//2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). [S.l.]: IEEE, 2017: 615-624.

[95] DATTA K, MURPHY M, VOLKOV V, et al. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures[C]//Proceedings of the 2008 ACM/IEEE conference on Supercomputing. [S.l.]: IEEE Press, 2008: 4.

[96] BAE I, HARRIS B, MIN H, et al. Auto-tuning cnns for coarse-grained reconfigurable array-based accelerators[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 37(11):2301-2310.

[97] CHEN T, MOREAU T, JIANG Z, et al. TVM: an automated end-to-end optimizing compiler for deep learning[C]//13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018. [S.l.: s.n.], 2018: 578-594.

[98] BEN-NUN T, HOEFLER T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis[J]. CoRR, 2018, abs/1802.09941.

[99] GAUNT A, JOHNSON M, RIECHERT M, et al. Ampnet: Asynchronous model-parallel training for dynamic neural networks[J]. CoRR, 2017, abs/1705.09786.

[100] DEAN J, CORRADO G, MONGA R, et al. Large scale distributed deep networks[C]//Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States. [S.l.: s.n.], 2012: 1232-1240.

[101] LI M, ANDERSEN D G, PARK J W, et al. Scaling distributed machine learning with the parameter server[C]//11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14). [S.l.: s.n.], 2014: 583-598.

[102] NEUBIG G, DYER C, GOLDBERG Y, et al. Dynet: The dynamic neural network toolkit[J]. CoRR, 2017, abs/1701.03980.

[103] LOOKS M, HERRESHOFF M, HUTCHINS D, et al. Deep learning with dynamic computation graphs[J]. CoRR, 2017, abs/1702.02181.

[104] XIAO W, BHARDWAJ R, RAMJEE R, et al. Gandiva: Introspective cluster scheduling for deep learning[C]//13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018. [S.l.: s.n.], 2018: 595-610.

[105] MATHURIYA A, BARD D, MENDYGRAL P, et al. Cosmoflow: using deep learning to learn the universe at scale[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018. [S.l.]: IEEE / ACM, 2018: 65:1-65:11.

[106] KEMELMACHER-SHLIZERMAN I, SEITZ S M, MILLER D, et al. The megaface benchmark: 1 million faces for recognition at scale[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2016: 4873-4882.

[107] MERITY S, XIONG C, BRADBURY J, et al. Pointer sentinel mixture models[J]. CoRR, 2016, abs/1609.07843.

[108] GABRILOVICH E, RINGGAARD M, SUBRAMANYA A. Facc1: Freebase annotation of clueweb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0)[J]. Note: http://lemurproject. org/clueweb09/FACC1/Cited by, 2013, 5.

[109] ZHAO H, YAN Z, TORRESANI L, et al. Hacs: Human action clips and segments dataset for recognition and temporal localization[J]. arXiv preprint arXiv:1712.09374, 2019.

[110] PAPADOPOULOS D P, UIJLINGS J R R, KELLER F, et al. We don't need no bounding-boxes: Training object class detectors using only human verification[C/OL]//2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. 2016: 854-863. DOI: 10.1109/CVPR.2016.99.

[111] 郭建崴. 进化论系列讲座 (六) 物种与分类[J]. 化石 Fossils, 2017, 02.

[112] 生物分类法[EB/OL]. https://zh.wikipedia.org/wiki/%E7%94%9F%E7%89%A9%E5%88%86%E7%B1%BB%E6%B3%95.

[113] GROVER A, LESKOVEC J. node2vec: Scalable feature learning for networks[C/OL]//Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016. 2016: 855-864. DOI: 10.1145/2939672.2939754.

[114] KIPF T N, WELLING M. Semi-supervised classification with graph convolutional networks[J]. CoRR, 2016, abs/1609.02907.

[115] DEFFERRARD M, BRESSON X, VANDERGHEYNST P. Convolutional neural networks on graphs with fast localized spectral filtering[C]//Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. [S.l.: s.n.], 2016: 3837-3845.

[116] LESKOVEC J, CHAKRABARTI D, KLEINBERG J, et al. Kronecker graphs: An approach to modeling networks[J/OL]. J. Mach. Learn. Res., 2010, 11:985-1042. http://dl.acm.org/citation.cfm?id=1756006.1756039.

[117] DHILLON I S, MODHA D S. Concept decompositions for large sparse text data using clustering [J/OL]. Machine Learning, 2001, 42(1/2):143-175. DOI: 10.1023/A:1007612920971.

[118] JENKINSON J, GRIGORYAN A M, HAJINOROOZI M, et al. Machine learning and image processing in astronomy with sparse data sets[C/OL]//2014 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2014, San Diego, CA, USA, October 5-8, 2014. 2014: 200-

203. DOI: 10.1109/SMC.2014.6973907.

[119] YANG C H, LIU F, HUANG J, et al. Auto-classification of retinal diseases in the limit of sparse data using a two-streams machine learning model[J]. CoRR, 2018, abs/1808.05754.

[120] ANTHOLZER S, HALTMEIER M, SCHWAB J. Deep learning for photoacoustic tomography from sparse data[J]. CoRR, 2017, abs/1704.04587.

[121] JARITZ M, DE CHARETTE R, WIRBEL E, et al. Sparse and dense data with cnns: Depth completion and semantic segmentation[C]//2018 International Conference on 3D Vision (3DV). [S.l.]: IEEE, 2018: 52-60.

[122] DRUMOND R R, MARQUES B A D, VASCONCELOS C N, et al. PEEK - an LSTM recurrent network for motion classification from sparse data[C/OL]//Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2018) - Volume 1: GRAPP, Funchal, Madeira, Portugal, January 27-29, 2018. 2018: 215-222. DOI: 10.5220/0006585202150222.

[123] IANDOLA F N, HAN S, MOSKEWICZ M W, et al. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size[J]. arXiv preprint arXiv:1602.07360, 2016.

[124] CHO K, VAN MERRIENBOER B, GÜLÇEHRE Ç, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[C]//Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL. [S.l.: s.n.], 2014: 1724-1734.

[125] Create deep learning calculators based on encoder-decoder rnn using keras[EB/OL]. https://fairyonice.github.io/Create-deep-learning-calculators-based-on-Encoder-Decoder-RNN-using-Keras.html.

[126] VINYALS O, TOSHEV A, BENGIO S, et al. Show and tell: A neural image caption generator [C/OL]//IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. 2015: 3156-3164. DOI: 10.1109/CVPR.2015.7298935.

[127] KESKAR N S, MUDIGERE D, NOCEDAL J, et al. On large-batch training for deep learning: Generalization gap and sharp minima[J]. arXiv preprint arXiv:1609.04836, 2016.

[128] SZEGEDY C, VANHOUCKE V, IOFFE S, et al. Rethinking the inception architecture for computer vision[C/OL]//2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. 2016: 2818-2826. DOI: 10.1109/CVPR.2016.308.

[129] GOYAL P, DOLLÁR P, GIRSHICK R B, et al. Accurate, large minibatch SGD: training imagenet in 1 hour[J]. CoRR, 2017, abs/1706.02677.

[130] YOU Y, GITMAN I, GINSBURG B. Scaling SGD batch size to 32k for imagenet training[J]. CoRR, 2017, abs/1708.03888.

[131] KUMAR V, LUPESKO H, LIN H, et al. Dynamic training of resnet-50 with apache mxnet[J/OL]. AWS Machine Learning Blog[2018-11-26]. https://aws.amazon.com/blogs/machine-learning/introducing-dynamic-training-for-deep-learning-with-amazon-ec2/.

[132] YOU Y, ZHANG Z, HSIEH C J, et al. 100-epoch imagenet training with alexnet in 24 minutes[J]. ArXiv e-prints, 2017.

[133] ZHANG X, ZHOU X, LIN M, et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices[C/OL]//2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. 2018: 6848-6856. DOI: 10.1109/CVPR.2018.00716.

[134] LINHARES C D G, TRAVENÇOLO B A N, DE SOUZA PAIVA J G, et al. Dynetvis: a system for visualization of dynamic networks[C/OL]//Proceedings of the Symposium on Applied Computing,

SAC 2017, Marrakech, Morocco, April 3-7, 2017. 2017: 187-194. DOI: 10.1145/3019612.3019686.

[135] LOOKS M, HERRESHOFF M, HUTCHINS D, et al. Deep learning with dynamic computation graphs[C]//5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. [S.l.: s.n.], 2017.

[136] HU R, ANDREAS J, ROHRBACH M, et al. Learning to reason: End-to-end module networks for visual question answering[C/OL]//IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017. 2017: 804-813. DOI: 10.1109/ICCV.2017.93.

[137] LIU L, DENG J. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution[C]//Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. [S.l.: s.n.], 2018: 3675-3682.

[138] WU Z, NAGARAJAN T, KUMAR A, et al. Blockdrop: Dynamic inference paths in residual networks[C/OL]//2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. 2018: 8817-8826. DOI: 10.1109/CVPR.2018.00919.

[139] HE K, ZHANG X, REN S, et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification[C/OL]//2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015. 2015: 1026-1034. DOI: 10.1109/ICCV.2015.123.

[140] CLEVERT D, UNTERTHINER T, HOCHREITER S. Fast and accurate deep network learning by exponential linear units (elus)[C]//4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. [S.l.: s.n.], 2016.

[141] XU B, WANG N, CHEN T, et al. Empirical evaluation of rectified activations in convolutional network[J]. CoRR, 2015, abs/1505.00853.

[142] ASANOVIC K, BODIK R, CATANZARO B C, et al. The landscape of parallel computing research: A view from berkeley[R]. [S.l.]: Technical Report UCB/EECS-2006-183, EECS Department, University of UC Berkeley, 2006.

[143] 刘鑫, 郭恒, 孙茹君, 等. “神威· 太湖之光”计算机系统大规模应用特征分析与 E 级可扩展性研究[J]. 计算机学报, 2018.

[144] GAO W, LUO C, ZHAN J, et al. Identifying dwarfs workloads in big data analytics[J]. arXiv preprint arXiv:1505.06872, 2015.

[145] Molecular dynamics, wikipedia[EB/OL]. 2019. https://en.wikipedia.org/wiki/Molecular_dynamics.

[146] IOFFE S, SZEGEDY C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[147] 2018 中国云计算技术应用盘点：AI 正当道、IoT 崛起时[EB/OL]. [2019-01-08]. http://www.tmtpost.com/3692888.html.

[148] New study: The state of ai in the enterprise[EB/OL]. [2019-03-02]. https://cloud.google.com/blog/topics/research/new-study-the-state-of-ai-in-the-enterprise.

[149] Proceedings of the international conference for high performance computing, networking, storage, and analysis, SC 2018, dallas, tx, usa, november 11-16, 2018[C]. [S.l.]: IEEE / ACM, 2018.

[150] WILLIAMS S, WATERMAN A, PATTERSON D. Roofline: An insightful visual performance model for floating-point programs and multicore architectures[R]. [S.l.]: Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2009.

[151] HENNING J L. Spec cpu2000: Measuring cpu performance in the new millennium[J]. Computer, 2000, 33(7):28-35.

[152] SCHWAN P, et al. Lustre: Building a file system for 1000-node clusters[C]//Proceedings of the 2003 Linux symposium: volume 2003. [S.l.: s.n.], 2003: 380-386.

[153] SCHMUCK F B, HASKIN R L. GPFS: A shared-disk file system for large computing clusters[C]//Proceedings of the FAST '02 Conference on File and Storage Technologies, January 28-30, 2002, Monterey, California, USA. [S.l.: s.n.], 2002: 231-244.

[154] CARNS P H, III W B L, ROSS R B, et al. PVFS: A parallel file system for linux clusters[C]//4th Annual Linux Showcase & Conference 2000, Atlanta, Georgia, USA, October 10-14, 2000. [S.l.: s.n.], 2000.

[155] XU W, LU Y, LI Q, et al. Hybrid hierarchy storage system in milkyway-2 supercomputer[J/OL]. Frontiers Comput. Sci., 2014, 8(3):367-377. DOI: 10.1007/s11704-014-3499-6.

[156] HEICHLER J. An introduction to beegfs[M]. [S.l.]: November, 2014.

[157] BORTHAKUR D. The hadoop distributed file system: Architecture and design[J]. Hadoop Project Website, 2007, 11(2007):21.

[158] LINDHOLM E, NICKOLLS J, OBERMAN S, et al. Nvidia tesla: A unified graphics and computing architecture[J]. IEEE micro, 2008, 28(2):39-55.

[159] SHAN Y, HUANG Y, CHEN Y, et al. Legoos: A disseminated, distributed OS for hardware resource disaggregation[C]//13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018. [S.l.: s.n.], 2018: 69-87.

[160] ADAMIC L A, HUBERMAN B A. Power-law distribution of the world wide web[J]. science, 2000, 287(5461):2115-2115.

[161] FALOUTSOS M, FALOUTSOS P, FALOUTSOS C. On power-law relationships of the internet topology[C]//ACM SIGCOMM computer communication review: volume 29. [S.l.]: ACM, 1999: 251-262.

[162] MOCANU D C, MOCANU E, STONE P, et al. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science[J]. Nature communications, 2018, 9(1): 2383.

[163] KEPNER J, GADEPALLY V, JANANTHAN H, et al. Sparse deep neural network exact solutions [C/OL]//2018 IEEE High Performance Extreme Computing Conference, HPEC 2018, Waltham, MA, USA, September 25-27, 2018. 2018: 1-8. DOI: 10.1109/HPEC.2018.8547742.

[164] LIU L, DENG L, HU X, et al. Dynamic sparse graph for efficient deep learning[J]. CoRR, 2018, abs/1810.00859.

[165] CHICKERING D M, HECKERMAN D. Fast learning from sparse data[J]. CoRR, 2013, abs/1301.6685.

[166] 赵毅, 朱鹏, 迟学斌, 等. 浅析高性能计算应用的需求与发展[J]. 计算机研究与发展, 2015, 44(10): 1640-1646.

[167] SCHMUCK F B, HASKIN R L. Gpfs: A shared-disk file system for large computing clusters.[C]//FAST: volume 2. [S.l.: s.n.], 2002.

[168] ROSS R B, THAKUR R, et al. Pvfs: A parallel file system for linux clusters[C]//Proceedings of the 4th annual Linux showcase and conference. [S.l.: s.n.], 2000: 391-430.

[169] 菅志刚, 金旭. 数据挖掘中数据预处理的研究与实现[J]. 计算机应用研究, 2004, 7:117-118.

[170] 黄洪宇, 林甲祥, 陈崇成, 等. 离群数据挖掘综述[J]. 计算机应用研究, 2006, 23(8):8-13.

[171] CHODOROW K. Mongodb: The definitive guide: Powerful and scalable data storage[M]. [S.l.]: " O'Reilly Media, Inc.", 2013.

[172] MILLER J J. Graph database applications and concepts with neo4j[C]//Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA: volume 2324. [S.l.: s.n.],

2013: 36.

[173] Thop: Pytorch-opcounter[EB/OL]. [2019-03-26]. https://github.com/Lyken17/pytorch-OpCounter.

[174] Mxop: Mxnet-opsummary[EB/OL]. [2019-03-26]. https://github.com/hey-yahei/OpSummary.MXNet.

[175] LIU C, YANG H, SUN R, et al. swtvm: Exploring the automated compilation for deep learning on sunway architecture[J]. arXiv preprint arXiv:1904.07404, 2019:1-11.

[176] KRISHNA R, ZHU Y, GROTH O, et al. The visual genome dataset v1.0 + v1.2 images[J/OL]. https://visualgenome.org/.

[177] SMITH N A, HEILMAN M, HWA R. Question generation as a competitive undergraduate course project[C]//Proceedings of the NSF Workshop on the Question Generation Shared Task and Evaluation Challenge. [S.l.: s.n.], 2008: 4-6.

[178] SUGIYAMA M, IDÉ T, NAKAJIMA S, et al. Semi-supervised local fisher discriminant analysis for dimensionality reduction[J]. Machine learning, 2010, 78(1-2):35.

[179] WESTON J, BORDES A, CHOPRA S, et al. Towards ai-complete question answering: A set of prerequisite toy tasks[C]//4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. [S.l.: s.n.], 2016.

[180] CHELBA C, MIKOLOV T, SCHUSTER M, et al. One billion word benchmark for measuring progress in statistical language modeling[J]. CoRR, 2013, abs/1312.3005.

[181] Common crawl[EB/OL]. http://commoncrawl.org/the-data/.

[182] LECUN Y, BOTTOU L, BENGIO Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11):2278-2324.

[183] Mnist in csv[EB/OL]. https://pjreddie.com/projects/mnist-in-csv/.

[184] ALBERTINA B, WATSON M, HOLBACK C, et al. Radiology data from the cancer genome atlas lung adenocarcinoma [tcga-luad] collection. the cancer imaging archive.[EB/OL]. 2016. http://doi.org/10.7937/K9/TCIA.2016.JGNIHEP5.

[185] HUANG G B, RAMESH M, BERG T, et al. Labeled faces in the wild: A database for studying face recognition in unconstrained environments: 07-49[R]. [S.l.]: University of Massachusetts, Amherst, 2007.

[186] LEARNED-MILLER G B H E. Labeled faces in the wild: Updates and new reporting procedures: UM-CS-2014-003[R]. [S.l.]: University of Massachusetts, Amherst, 2014.

[187] WU B, CHEN W, FAN Y, et al. Tencent ml-images: A large-scale multi-label image database for visual representation learning[J]. arXiv preprint arXiv:1901.01703, 2019.

[188] SUN C, SHRIVASTAVA A, SINGH S, et al. Revisiting unreasonable effectiveness of data in deep learning era[C/OL]//IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017. 2017: 843-852. DOI: 10.1109/ICCV.2017.97.

[189] Yahoo! altavista web page hyperlink connectivity graph, offical site[EB/OL]. 2012. http://webscope.sandbox.yahoo.com/.

[190] BACKSTROM L, BOLDI P, ROSA M, et al. Four degrees of separation[C]//Proceedings of the 4th Annual ACM Web Science Conference. [S.l.]: ACM, 2012: 33-42.

[191] MYERS S A, SHARMA A, GUPTA P, et al. Information network or social network?: the structure of the twitter follow graph[C]//Proceedings of the 23rd International Conference on World Wide Web. [S.l.]: ACM, 2014: 493-498.

[192] CHING A, EDUNOV S, KABILJO M, et al. One trillion edges: Graph processing at facebook-scale [J]. Proceedings of the VLDB Endowment, 2015, 8(12):1804-1815.

[193] ALPERT J, HAJAJ N. We knew the web was big...[J/OL]. Offical Google Blog, 2008. http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.
[194] Movielens 20m dataset[EB/OL]. [2019-04-09]. https://grouplens.org/datasets/movielens/.
[195] DONG Z, WU Y, PEI M, et al. Vehicle type classification using a semisupervised convolutional neural network[J]. 2015:in press.
[196] HAMMANN C, HURST S, SCHMEISER S, et al. Data of evaluation different introduction forms with research flight simulator for reduction errors, mendeley data, v3[EB/OL]. 2019. http://dx.doi.org/10.17632/xcdn5nfmjh.3.
[197] GEIGER A, LENZ P, URTASUN R. Are we ready for autonomous driving? the kitti vision benchmark suite[C]//Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2012.
[198] SILVER D, HUANG A, MADDISON C J, et al. Mastering the game of go with deep neural networks and tree search[J/OL]. Nature, 2016, 529(7587):484-489. DOI: 10.1038/nature16961.
[199] Tensorflow official models[EB/OL]. [2019-04-09]. https://github.com/tensorflow/models/tree/master/official.
[200] Tensorflow-slim image classification model library[EB/OL]. [2019-04-09]. https://github.com/tensorflow/models/tree/master/research/slim.
[201] Tensorflow research models[EB/OL]. [2019-04-09]. https://github.com/tensorflow/models/tree/master/research.
[202] Caffe model zoo[EB/OL]. [2019-04-09]. https://github.com/BVLC/caffe/wiki/Model-Zoo.
[203] Caffe2 model repository[EB/OL]. [2019-04-09]. https://github.com/caffe2/models.
[204] Deep learning models in pytorch [in progress][EB/OL]. [2019-04-09]. https://github.com/sdhnshu/Pytorch-Model-Zoo.
[205] Keras model zoo[EB/OL]. [2019-04-09]. https://modelzoo.co/framework/keras.
[206] The gan zoo[EB/OL]. [2019-04-09]. https://github.com/hindupuravinash/the-gan-zoo.
[207] Awesome coreml models[EB/OL]. [2019-04-09]. https://github.com/likedan/Awesome-CoreML-Models.
[208] Mxnet model zoo[EB/OL]. [2019-04-09]. https://mxnet.apache.org/model_zoo/index.html.