智能计算系统ppt知识汇总

第1章-绪论

一、人工智能发展现状

1. **技术突破与应用**
   - **大模型进展**：ChatGPT 推出 5 天用户超百万，DeepSeek-R1 推理性能对标 OpenAI o1 且成本低 27 倍，Sora 实现文生视频。开源模型如 DeepSeek-V3（6710 亿参数，预训练成本 557.6 万美元）推动技术普及。
   - **典型应用**：图像风格迁移、语音合成、自动驾驶、代码生成（CodeForces 比赛超越 99.9% 程序员）、药物研制等。
2. **技术分层体系**
   - 从底层到应用分为芯片层（CPU/GPU/MLU）、系统层（PyTorch/DeepSpeed）、算法层（Transformer）、应用层（大模型 / 广告推荐）。
3. **发展历程**
   - **三次热潮**：符号主义（1950s 专家系统）、连接主义（1980s 反向传播）、大模型时代（2010s 至今，以 GPT 系列为代表）。
   - **关键事件**：1956 年达特茅斯会议、2012 年 AlexNet、2016 年 AlphaGo、2022 年 ChatGPT。

二、智能计算系统演进

1. **三代系统特征**
   - **第一代（1980s）**：LISP 机 / Prolog 机，面向符号处理，因 AI 寒冬衰落。
   - **第二代（2010s 至今）**：CPU + 智能芯片的异构系统（如寒武纪 MLU、Google TPU），解决通用 CPU 算力瓶颈。
   - **第三代（未来）**：面向强人工智能，需超大规模计算能力，融合大模型与认知智能。

2. **第二代系统核心技术**
    - **芯片**：寒武纪 MLU100（2018 年国际峰值速度领先）、MLU270（性能提升 4 倍），首创深度学习指令集（2016 年）。
    - **算力需求**：大模型参数从 AlexNet 的 6 千万增至 DeepSeek-R1 的 6710 亿，算力需求呈指数级增长。

三、关键技术与优化方法

1. **大模型核心技术**
    - **Scaling Law**：模型性能与参数规模、数据量呈幂律关系，算力增加可降低测试 Loss。
    - **混合精度训练**：如 DeepSeek 使用 FP8，减少 GPU 内存占用，加速训练。
    - **混合专家架构（MoE）**：训练时 6710 亿参数，推理时仅激活 370 亿，降低计算成本。
2. **系统优化策略**
    - **稀疏注意力 vs FlashAttention**：前者通过掩码减少计算量（O (n・k)），后者优化内存访问（带宽需求降至 O (n・√n)）。
    - **自动生成技术**：机器 5 小时自动生成 32 位 RISC-V CPU（启蒙 1 号），编译器后端 / 机器描述文件自动化生成。

四、实验与技术平台

1. **实验环境**
    - **硬件**：集成 4 个智能处理器簇的 DLP 平台，峰值算力 128T。
    - **软件**：PyTorch/TensorFlow 框架、BCL/Bang 智能编程语言。
    - **云平台**：paas.extrotec.com（100 小时使用限制）。
2. **实验内容**
    - 分阶段涵盖神经网络设计、编程框架应用、DLP 运算部件设计、大模型优化（如 FlashAttention 算子实现）。

五、技术趋势与挑战

1. **未来方向**
    - 第三代系统探索：目标是通过超大规模计算系统实现强人工智能，需打通感知到逻辑的鸿沟。

- 自动化设计：大模型生成处理器、操作系统（如 AutoOS 优化内核配置提升 25.6% 性能）。

2. **现存挑战**
   - 深度学习局限性：泛化能力、逻辑推理、可解释性不足。
   - 算力成本矛盾：GPT-4 训练花费超 1 亿美元，开源模型（如 DeepSeek）降低技术门槛。

第2章-深度学习基础

一、机器学习基础

1. **线性回归模型**
   - **单变量线性回归**：($H_w(x) = w_0 + w_1x$)，通过最小二乘法拟合直线。
   - **多变量线性回归**：($H_w(x) = \sum_{i=0}^{n} w_i x_i = \hat{w}^T x$)，处理多特征输入。
   - **损失函数**：均方差损失 ($L(\hat{w}) = \frac{1}{2m}\sum_{j=1}^{m}(H_w(x_j) - y_j)^2$)，通过梯度下降法迭代优化参数：($\hat{w} = \hat{w} - \eta\frac{\partial L(\hat{w})}{\partial \hat{w}}$)

2. **机器学习与深度学习的关系**
   - 人工智能 > 机器学习 > 神经网络 > 深度学习 > 大模型，深度学习通过多层神经网络实现复杂特征提取。

二、神经网络基础

1. **神经元模型**
   - **生物神经元类比**：树突（输入）、轴突（输出）、突触（权重）。
   - **人工神经元**：输入加权求和后经激活函数输出，如感知机模型：($H(x) = \text{sign}(w^T x + b)$) 损失函数为误分类点到超平面的总距离：($L(w,b) = -\sum_{x_j \in M} y_j(w^T x_j + b)$)。

2. **多层神经网络（MLP）**
   - **结构**：输入层→隐层→输出层，全连接方式连接，引入激活函数实现非线性映射。
   - 激活函数：
     - **sigmoid**：($\sigma(x) = 1/(1+e^{-x})$)，将输入映射到 (0,1)，但存在梯度消失问题。

- **ReLU**：($f(x) = \max(0, x)$)，缓解梯度消失，计算高效，广泛应用于深层网络。
- **GELU/ SiLU**：大模型中常用，如 GPT 系列使用 GELU，DeepSeek 使用 SiLU，提升模型表达能力。

三、神经网络训练方法

1. **正向传播**
   - 以两层网络为例：
     1. 输入 x 经权重 ($W^{(1)}$) 和偏置 ($b^{(1)}$) 计算隐层：($h = G(W^{(1)T}x + b^{(1)})$)。
     2. 隐层经 ($W^{(2)}$) 和 ($b^{(2)}$) 计算输出：($\hat{y} = G(W^{(2)T}h + b^{(2)})$)。
2. **反向传播**
   - 通过链式法则计算梯度并更新权重，以交叉熵损失为例：
     - 二分类交叉熵：($L = -\frac{1}{m}\sum(y \ln \hat{y} + (1-y)\ln(1-\hat{y}))$)。
     - 梯度更新：($w = w - \eta\frac{\partial L}{\partial w}$)，如对($w_{2,1}^{(2)}$)的偏导：($\frac{\partial L}{\partial w} = -(y - \hat{y}) \cdot \hat{y}(1-\hat{y}) \cdot h_2$)

四、神经网络设计基础

1. **隐层设计**
   - 隐层数量和神经元个数影响模型复杂度：
     - 过少：欠拟合，无法提取复杂特征。
     - 过多：过拟合，计算成本高。
     - 经验法则：隐层神经元数取输入维度的 1.5-2 倍。
2. **损失函数选择**
   - **均方差（MSE）**：适用于回归任务，但与 sigmoid 联用时梯度易消失。
   - **交叉熵**：适用于分类任务，与 sigmoid 联用可缓解梯度消失，如：($L = -\sum y_i \ln \hat{y}_i$)

五、过拟合与正则化

1. **过拟合现象**

- 模型在训练集上误差低，但测试集上误差高，原因是模型拟合了噪声。

2. **正则化技术**
   - **L2 正则化**：在损失函数中添加权重平方和：($\tilde{L} = L + \frac{\theta}{2}|w|^2$)，迫使权重趋近于 0，降低模型复杂度。
   - **L1 正则化**：添加权重绝对值和：($\tilde{L} = L + \theta|w|_1$)，可使权重稀疏（部分为 0）。
   - **Dropout**：训练时随机丢弃部分神经元，避免神经元间过度依赖，如以 50% 概率丢弃隐层单元。
   - **提前终止**：监控验证集误差，当误差不再下降时停止训练，避免过度拟合。

六、交叉验证

1. **数据集划分**
   - **训练集**：拟合模型参数；**验证集**：调优超参数；**测试集**：评估模型泛化能力。
2. **验证方法**
   - **K 折交叉验证**：将数据集分为 K 份，每次用 K-1 份训练，1 份测试，取平均误差，适用于小数据集。
   - **留一法（Leave-One-Out）**：每次用 n-1 个样本训练，1 个测试，计算量极大，仅适用于极小规模数据。
   - **时间序列交叉验证**：按时间顺序划分训练集和测试集，避免未来数据泄露，适用于金融、气象等场景。

七、深度学习发展与关键技术

1. **深度学习定义**
   - 2006 年 Hinton 提出，通过多层神经网络实现特征的层次化提取，依赖算法（BP、Dropout）、大数据和算力（GPU/TPU）。
2. **典型网络结构**
   - **Transformer 中的 FFN**：两层全连接层，如：($\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2$) 大模型中常替换 ReLU 为 GELU 或 SiLU，提升非线性表达能力。

第3章-深度学习应用-part1

一、卷积神经网络（CNN）基础

1. CNN 核心组件

卷积层：

- **局部连接与权重共享**：减少参数数量，避免过拟合（如 1024×1024 图像的全连接层参数约 10^9，卷积层可降至 10^4）。
- **卷积运算**：多输入通道卷积（如 6×6×3 输入 + 3×3×3 卷积核→4×4×1 输出），可转换为矩阵乘法加速计算。
- **关键参数**：核大小（K）、步长（s）、填充（pad），输出尺寸公式：$W_o = \left\lfloor \frac{W_i + 2\text{pad} - K}{s} \right\rfloor + 1$

池化层：

- **Max Pooling**：保留特征最大值，提升鲁棒性（如 2×2 窗口，步长 2）。
- **作用**：降维、减少计算量、控制过拟合。

全连接层：

- 特征映射为一维向量，用于分类（如 VGG16 的 4096 维全连接层）。
- **Softmax**：输出分类概率，公式：$f(z_j) = e^{z_j} / \sum e^{z_i}$。

2. 网络优化技术

参数初始化：

- **Xavier 初始化**：适用于 tanh 激活，方差$Var[W] = 2/(n_i + n_{i+1})$。
- **Kaiming 初始化**：适用于 ReLU 激活，方差$Var[W] = 2/n_i$。

梯度下降变种：

- **Adam**：结合一阶矩和二阶矩估计，自适应学习率（公式：$w_{t+1} = w_t - \eta \hat{m}_t / \sqrt{\hat{r}_t + \epsilon}$）。
- **AdamW**：解耦权重衰减与梯度更新，减少过拟合（大模型训练常用）。

正则化：

- **Dropout**：训练时随机丢弃神经元（如 50% 概率），抑制过拟合。
- **BatchNorm**：归一化激活值为标准正态分布，加速收敛（公式：$\hat{x}_i = (x_i - \mu)/\sqrt{\sigma^2 + \epsilon}$）。

二、图像分类经典网络架构

1. AlexNet（2012）

创新点：ReLU 激活、LRN 局部归一化、Max Pooling、Dropout。

性能：ImageNet top5 误差 15.3%，首次证明深度 CNN 的有效性。

2. VGG（2015）

结构：全 3×3 卷积层堆叠（如 VGG16 有 13 个卷积层），通过多层小卷积替代大卷积（如 2 个 3×3 卷积等价 5×5 卷积，参数更少）。

优势：结构规整，特征提取能力强，top5 误差 7.5%。

3. ResNet（2016）

残差连接：H(x) = F(x) + x，解决深层网络退化问题，可训练 152 层网络。

性能：ImageNet top5 误差 3.57%，残差块成为后续网络基础组件。

4. Inception（2015-2017）

模块化设计：多尺度卷积并行（1×1/3×3/5×5 卷积 + 池化），通过 1×1 卷积降维（瓶颈层）。

创新：Factorization 思想（如 3×3 卷积拆分为 1×3+3×1），减少参数。

三、目标检测算法

1. 评测指标

IoU（交并比）：IoU = |A∩B|/|A∪B|，衡量定位精度（IoU≥0.5 为有效检测）。

mAP（平均精度均值）：综合召回率（Recall）和精度（Precision），公式：$AP = \int_0^1 P(r)dr$ （其中 r为召回率，P为对应精度）

2. 两阶段算法（Two-Stage）

R-CNN 系列：

R-CNN：候选区域提取（Selective Search）+CNN 特征提取 + SVM 分类，单帧检测 50s。

Faster R-CNN：引入 RPN 网络生成候选区域，共享特征提取，单帧 0.2s，mAP 67.0%。

关键步骤：候选区域生成→特征提取→分类 + 回归→NMS 去重。

3. 一阶段算法（One-Stage）

YOLO 系列：

YOLOv1：将图像分 7×7 网格，每个网格预测 2 个边界框，实时性 45FPS，mAP 63.4%。

YOLOv5：Backbone（CSPDarknet）+Neck（FPN+PAN）+Head（耦合检测头），兼顾速度与精度。

SSD：多尺度特征图检测（如 8×8 和 4×4 特征图分别检测小目标和大目标），结合 Anchor 机制，mAP 74.3%。

四、网络优化与设计技巧

深度可分离卷积：分为 Depthwise（逐通道卷积）和 Pointwise（1×1 卷积），参数量大幅减少（如标准卷积参数量$K^2C_iC_o$，深度可分离为$K^2C_i + C_iC_o$）。

特征金字塔（FPN）：自上而下融合高层语义与低层定位特征，提升小目标检测能力。

损失函数设计：

交叉熵：解决 Sigmoid+MSE 的梯度消失问题。

CIoU 损失：$L_{CIoU} = 1 - IoU + \rho^2(b, b^{gt})/c^2 + \alpha v$，优化边界框回归。

五、核心技术对比

任务 / 算法	代表模型	核心创新	性能 / 特点

图像分类	ResNet	残差连接，解决深层网络退化	top5 误差 3.57%，152 层深度
目标检测（两阶段）	Faster R-CNN	RPN 网络替代 Selective Search	mAP 67.0%，单帧 0.2s
目标检测（一阶段）	YOLOv5	多尺度检测 + Anchor-free 设计	实时性 + 高精度（COCO mAP 50.7%）
网络优化	AdamW	解耦权重衰减与梯度更新	大模型训练必备，减少过拟合

第3章-深度学习应用-part2

一、Transformer在图像处理中的应用

1. **ViT（Vision Transformer）**

- **核心思想**：将图像划分为16×16像素块（Patch），展平后添加位置编码，输入Transformer编码器，通过全局注意力建模图像特征。
- **结构**：
  - 输入：224×224图像→14×14×768 Patch序列（196个Patch）。
  - 关键层：Multi-Head Attention + MLP，输出通过Class Token分类。
- **挑战**：图像尺度变化大时计算复杂度高（如224×224图像序列长度为196，是BERT的100倍）。

2. **Swin Transformer**

- **创新点**：
  - **分层窗口机制**：将图像分块，仅在窗口内计算自注意力（W-MSA），通过滑动窗口（SW-MSA）连接相邻窗口，降低计算量。
  - **Shifted Window**：交替使用规则窗口和偏移窗口，增强跨窗口信息交互。
- **优势**：适用于图像分类、目标检测等任务，参数量比ViT更少。

二、生成模型技术

1. **生成对抗网络（GAN）**

- **核心架构**：
  - **生成器（G）**：从随机噪声生成假样本，目标是欺骗判别器。
  - **判别器（D）**：区分真实样本与生成样本，目标是正确分类。

- **训练过程**：
  - 极小极大博弈：$\min_G \max_D V(D,G) = \mathbb{E}x \sim pdata[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$。
- **问题与改进**：
  - **模式崩溃**：生成样本缺乏多样性，可通过Wasserstein GAN（WGAN）缓解。
  - **梯度消失**：训练早期修改生成器损失为$-\mathbb{E}[\log D(G(z))]$。

2. **扩散模型（Diffusion Models）**

- **双向过程**：
  - **正向扩散**：逐步向图像添加高斯噪声，直至变为纯噪声（$x_0 \to x_T$）。
  - **反向去噪**：从噪声中恢复图像，通过U-Net预测噪声并迭代去噪。
- **代表模型**：
  - **Stable Diffusion**：在隐空间中训练，降低计算量（如512×512图像压缩为64×64隐向量）。
  - **DALL-E 2**：结合CLIP的文本-图像对齐，通过文本生成高分辨率图像。
- **应用**：文本生成图像（如ControlNet支持条件控制）、图像修复、视频生成（SORA）。

3. **变分自编码器（VAE）**

- **核心原理**：
  - 编码器将图像映射到概率分布（$q(z|x)$），解码器从分布采样重构图像（$p(x|z)$）。
  - 损失函数：重构误差 + KL散度（$KL(q(z|x)||p(z))$），确保分布接近标准正态分布。

三、视觉-语言模型（VLM）

1. **CLIP（Contrastive Language-Image Pre-training）**

- **训练方法**：
  - 对比学习：对齐4亿图像-文本对，通过文本编码器（Transformer）和图像编码器（ResNet/ViT）学习跨模态嵌入。
- **推理能力**：

- 零样本预测：将分类标签转为文本提示（如“A photo of a dog”），计算图像-文本相似度。

2. **BLIP/BLIP-2**

- **BLIP**：
  - 三阶段训练：图像-文本对比（ITC）、文本-图像匹配（ITM）、语言建模（LM），提升多模态理解与生成。
- **BLIP-2**：
  - 冻结预训练视觉编码器和大语言模型（LLM），仅训练轻量Q-Former，降低训练成本，支持视觉问答、图像描述。

四、智能体系统（Agent Systems）

1. **核心组件**：

- **规划（Planning）**：
  - 子任务分解：思维链（CoT）、思维树（ToT）将复杂任务拆分为子步骤。
  - 反思（Self-Reflection）：通过ReAct、Reflexion纠正决策错误。
- **记忆（Memory）**：
  - 短期记忆：上下文窗口（如提示工程）；长期记忆：外部向量存储（检索增强生成RAG）。
- **工具使用**：
  - HuggingGPT：用LLM作为控制器，调用HuggingFace模型完成任务（如文本生成、图像分类）。

2. **协议标准**：

- **MCP（Model Context Protocol）**：统一AI应用与工具的接口，类似USB-C标准。
- **A2A（Agent2Agent）**：定义智能体间通信规范，支持跨系统协作。

五、神经网络量化技术

1. **量化目标**：

- 减少模型参数量（如FP32→INT8，存储降低4倍）、加速推理、降低能耗。

2. **量化方法**：

- **按对称性**：

- 对称量化：$Q = \left\lfloor \frac{R}{2^p} \right\rfloor$，适用于对称分布数据。
- 非对称量化：$Q = \left\lfloor \frac{R-B}{2^p} \right\rfloor$，引入偏移量B拟合非对称数据。

- **按粒度**：
  - 分层量化：整层共享量化参数；分通道量化：各通道独立参数，精度更高。
- **按训练阶段**：
  - 训练后量化（PTQ）：无需微调，直接校准数据分布（如GPTQ）。
  - 量化感知训练（QAT）：微调量化模型，提升低位宽精度（如INT4）。

3. **大模型量化**：

- **仅权值量化**：GPTQ、AWQ，优化权值加载时间（如GPTQ将176B参数量化至INT3）。
- **神经元+权值量化**：SmoothQuant，通过通道放缩平衡量化难度。
- **KV缓存量化**：KIVI，对KV Cache分块量化，优先保留近期高相关数据精度。

六、关键技术对比

任务/模型	核心创新	典型应用
图像生成	扩散模型的反向去噪	文本生图（Stable Diffusion）
多模态理解	CLIP的跨模态对比学习	零样本图像分类
智能体决策	规划+记忆+工具调用的协同	自动驾驶、聊天机器人
大模型压缩	量化与稀疏化	边缘设备部署

第3章-深度学习应用-part3

一、适合图像处理的卷积神经网络

1. 卷积神经网络总体结构

2. 神经网络优化方法

- 激活函数（如 Leaky ReLU）
- 损失函数设计（如 MSE、CIoU 损失）

3. 图像分类卷积神经网络（未详细展开）

4. 目标检测卷积神经网络

（1）YOLO 系列

- YOLOv1：单回归问题建模、S×S 网格划分、Bounding Box 预测（x,y,w,h,confidence）、类别概率计算、网络结构（基于 GoogleNet）、优缺点（速度快 / 背景误判少 / 邻近物体检测差）
- YOLOv5：Backbone（CSPDarknet）、Neck（FPN+PAN）、Head（耦合检测头）、CIoU 损失函数
- YOLO 系列发展历程（v1 到 v12）及性能对比

（2）SSD 算法

- 核心思想：结合 Anchor Box 与多尺度特征图检测
- 多尺度特征图应用：大特征图检测小目标，小特征图检测大目标
- Anchor Box 设计：宽高比、尺度计算公式
- 与 YOLO 的对比（精度与速度平衡）

5. 图像风格迁移（Driving Example）

- Gatys 方法：内容损失（Conv4 层欧式距离）与风格损失（Gram 矩阵）、VGG19 网络应用
- 实时风格迁移：图像转换网络训练、感知损失函数、速度提升对比

二、适合语音 / 文本处理的循环神经网络

1. 循环神经网络（RNN）基础

- 结构特点：隐藏状态 h (t) 依赖前一时刻状态与当前输入
- 正向计算过程：激活函数（tanh/ReLU）、参数共享机制
- 输入 - 输出结构：one-to-many、many-to-one、many-to-many

2. RNN 的梯度问题与优化

- 梯度消失 / 爆炸原因：链式求导中的连乘项
- 解决方案：梯度截断、LSTM/GRU 模型改进

3. 长短期记忆模型（LSTM）

- 核心结构：遗忘门、输入门、输出门、单元状态更新公式
- 变体：窥视孔连接、耦合输入门与遗忘门

4. 门控循环单元（GRU）

- 结构简化：合并单元状态与隐藏状态、更新门与重置门设计

- 与 LSTM 对比：参数量少 / 训练速度快 / 表征能力差异

5. 序列到序列模型（Seq2Seq）

- 编码器 - 解码器架构：RNN/LSTM 作为基础单元
- 注意力机制：相似度计算（α_ij）、注意力汇聚（c_i）、自注意力与交叉注意力区别
- 应用场景：机器翻译、对话系统、自动文摘

三、从深度学习到大模型

1. Transformer 架构

（1）核心组件

- 自注意力机制：缩放点积注意力（QKV 计算）、掩码处理、多头注意力拼接
- 位置编码：正弦余弦函数计算 PE (pos,2i) 与 PE (pos,2i+1)
- 层归一化（Layer Norm）：缓解梯度消失、加速收敛
- 前馈网络（FFN）：ReLU 激活、升维 - 降维结构

（2）编码器与解码器

- 编码器：N 层多头自注意力 + FFN，无掩码
- 解码器：带掩码自注意力 + 交叉注意力 + FFN，推理时自回归生成

2. 大模型架构分类

- Encoder-Only（如 BERT）：双向注意力、预训练（完形填空 + NSP）、微调应用于 NLU 任务
- Decoder-Only（如 GPT 系列）：因果注意力、自回归预训练、少样本学习（Zero-shot/One-shot/Few-shot）
- Encoder-Decoder（如原始 Transformer）：Seq2Seq 任务专用

3. GPT 系列大模型

- GPT-1/2：预训练 - 微调框架、无监督语言模型、参数规模扩展
- GPT-3：175B 参数、45TB 数据、涌现能力（模型规模与性能关系）、提示学习（Prompt Engineering）
- GPT 后续发展：ChatGPT（RLHF 微调）、GPT-4（多模态）、InstructGPT（人类反馈强化学习）

4. Llama 系列大模型

- 架构特点：Decoder-Only、RMSNorm、SwiGLU 激活函数、RoPE 旋转位置编码
- 版本迭代：Llama1/2/3 的参数、数据量、上下文长度对比

5. DeepSeek 大模型

- MLA（多头潜在注意力）：KV 压缩、性能优于 MHA
- MoE（混合专家模型）：专家分组、稀疏激活机制、计算效率优化

第3章-智能计算系统-预训练后训练

一、前置知识

1. 基础概念

- 词元与词元化（Tokenization）
- 词嵌入（Word Embeddings）
- Transformer 解码器架构（Decoder-only）

2. 模型与算力度量

- 模型参数量（如 7B=70 亿参数）
- 计算量（FLOPs，1 TFLOPs=10^{12} 次浮点运算）
- 数据量（Token 数，如 DeepSeek V3 使用 14.8T Tokens）

3. 训练基础

- 训练 / 验证 / 测试数据集划分
- 参数更新、学习率、Batch Size

二、大模型训练流程

1. 预训练与后训练阶段

- 预训练：无监督学习获取基础能力（如语言理解）
- 后训练：指令微调 + 偏好优化（对齐人类偏好）

2. 阶段对比

阶段	数据形式	数据量	GPU 需求	目标
预训练	文本数据	>1T Tokens	100-10000 张	学习基础常识与结构

后训练	成对问答数据	>10K 输入输出	≥1 张	增强任务特异性与安全性

三、预训练基础知识

1. 无监督训练范式

- 自回归预测：给定前 N-1 词预测第 N 词
- 损失函数：$L_1(\mathcal{U}) = \sum_i \log P(u_i|u_{i-k}, ..., u_{i-1}; \Theta)$

2. 数据处理

- 数据获取：Common Crawl（250T）、C4（800GB）等
- 数据清洗：启发式过滤（去重、去毒）、模型过滤（FastText）
- 数据打包：使用<BOS>/<EOS>拼接文本提升效率

3. 效果观测

- 监控 Loss 曲线
- 测试集评估（ARC、MMLU、GSM8K 等）
- 去重处理（N-gram、MinHash、Rouge-L）

4. 训练技巧

- 继续预训练（Domain Adaptation）
- 学习率调度（Warmup、Linear Cooldown）
- 数据比例优化（如金融模型中领域数据与通用数据配比）

四、尺度定律（Scaling Law）

1. 核心结论

模型效果与算力、数据、参数量成幂律关系
公式：
$L = (C_{min}/2.3 \cdot 10^8)^{-0.050}$（算力与损失关系）

2. 关键发现

模型形状（层数、隐层维度）影响弱，规模影响强
过拟合普遍性：需平衡模型大小与数据量
样本效率：大模型用更少数据达相同效果

3. 应用场景

算力预算分配：每 10 倍算力需 5 倍模型大小 + 2 倍数据
预测模型表现：如 Chinchilla 通过 Scaling Law 优化参数量

五、涌现能力（Emergent Abilities）

1. 定义与特征
小模型不具备、大模型突现的能力（如逻辑推理）
表现：准确率非线性突变（如 MMLU 任务）

2. 原因分析

指标非线性（如 Exact Match 对长序列敏感）
人类任务本身的非线性（如选择题、代码生成）

六、指令微调（Instruction Tuning）
1. 范式对比

上下文学习（In-context Learning）：GPT-3 少样本推理
传统微调（Pretrain-Finetune）：BERT 针对单一任务
指令微调：多任务泛化（如 FLAN 模型）

2. 数据构建

人工标注：高质量成对问答（如 LIMA 使用 1000 条数据）
模型合成：Alpaca 通过 GPT-4 生成 52K 指令数据
进化算法：Evol-instruct 通过深度 / 广度进化生成指令

3. 关键技巧

数据质量优先：多样性、正确性、难度适配
多模态对齐：如 LLaVA 结合视觉与语言数据

七、基于人类反馈的强化学习（RLHF）

1. 强化学习基础
策略梯度（Policy Gradient）与优势函数（
$\hat{A}_t = R_t - V(s_t)$）
PPO 算法：通过裁剪约束策略更新（
$L^{CLIP}(\theta) = \min(r_t \hat{A}_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)\hat{A}_t)$）

2. RLHF 流程

奖励模型训练：基于 Bradley-Terry 模型（$p(y1 > y2|x) = \frac{\exp(r(x,y1))}{\exp(r(x,y1))+\exp(r(x,y2))}$）
策略优化：结合奖励模型与 KL 散度约束

3. 优化变体

DPO（直接偏好优化）：省去奖励模型，直接优化偏好损失
RLAIF（AI 反馈强化学习）：用 LLM 替代人类标注
宪法 AI（Constitutional AI）：基于手工准则对齐模型

八、测试时扩展（Test-time Scaling）

1. 搜索派（Parallel）
Pass@k：k 次采样中至少 1 次正确的概率
奖励模型筛选：ORM（结果奖励）与 PRM（过程奖励）
搜索策略：Beam Search、Lookahead Search

2. 唠叨派（Sequential）

思维链（CoT）：强制模型生成推理过程
R1 模型：结合准确率奖励与格式奖励（Question+Think+Answer）

3. 关键发现

小模型可通过多采样弥补参数量不足
推理算力分配：简单任务优先用小模型搜索，复杂任务需大模型预训练

九、参考工具与框架

- 训练框架：LLaMA-Factory、OpenRLHF、trl
- 数据集：PRM800K、GSM8K、HumanEval

第4章-编程框架使用

一、编程框架概述

1. 为什么需要编程框架

- 深度学习算法复杂性：梯度计算、多层结构共性操作封装
- 提升开发效率：封装基础操作（卷积、池化等）
- 硬件优化支持：针对硬件特性优化计算效率

2. 主流编程框架

- PyTorch、TensorFlow、Keras、Caffe、PaddlePaddle、Mindspore
- 发布时间与核心特点对比

二、PyTorch概述

1. 起源与发展

- 前身Torch（LuaJIT接口，2002年）
- 2017年Facebook开源，基于Python生态

2. 设计原则与优势

- 基于Python、易用性、高性能
- 顶会论文占比超80%（EMNLP、ACL等）

3. 版本迭代

- 关键版本更新：1.0（2018.11）、2.0（2023.3）、2.6（2025.1）
- 新增功能：编译模式、移动端支持、模型量化

三、PyTorch编程模型及基本用法

1. NumPy基础

- 数组创建：np.array、np.arange、np.eye
- 形状操作：reshape、resize
- 数学函数：sin、cos、exp、clip

2. 张量（Tensor）操作

- 创建方式：torch.tensor、torch.ones/zeros、torch.rand
- 类型转换：to()、cpu()、numpy()
- 设备管理：device='cuda'/'dlp'，to(device)
- 数据格式：NCHW（PyTorch）与NHWC（TensorFlow）

3. 计算操作

- 原位操作：add_()、mul_()
- 广播机制：维度扩展规则
- 常用函数：add、mul、matmul、gram_matrix

4. 计算图

- 动态图 vs. 静态图：PyTorch动态图特性
- 自动求导：requires_grad、backward()
- 叶节点与非叶节点：is_leaf、retain_grad

四、基于PyTorch的模型推理实现

1. 数据读取与预处理

- 图像读取：PIL、OpenCV、torchvision.io
- 格式转换：transforms.Compose、ToTensor、Normalize

2. 模型构建

- 自定义模型：继承nn.Module，定义__init__和forward
- 预训练模型：torchvision.models.vgg19等
- 模型组件：Conv2d、ReLU、Linear、Sequential

3. 模型调试与优化

- 打印模型结构：print(model)、summary
- 可视化工具：TensorBoard、add_graph
- 模型剪枝：prune.L1Unstructured、global_unstructured
- 模型量化：quantize_dynamic、动态量化

五、基于PyTorch的模型训练实现

1. 数据加载

- Dataset与DataLoader：自定义数据集类
- 内置数据集：CIFAR10、ImageNet、MNIST

2. 损失函数与优化器

- 内建损失函数：MSELoss、CrossEntropyLoss
- 自定义损失函数：继承nn.Module

- 优化器：SGD、Adam、LBFGS，step()与zero_grad()

3. 训练流程

- 计算图构建与反向传播：backward()
- 梯度控制：torch.no_grad()、detach()
- 模型保存与恢复：state_dict、检查点文件（.tar）

六、驱动范例：非实时风格迁移

1. 算法原理

- 内容损失与风格损失：VGG19特征提取
- 损失函数公式：内容损失（conv4）与风格损失（conv1-5）

2. 代码实现

- 图像加载与预处理：image_loader、imshow
- 模型构建：Normalization、ContentLoss、StyleLoss
- 优化过程：run_style_transfer、LBFGS优化器

七、关键概念索引

1. **张量操作**：形状转换、设备迁移、广播机制
2. **模型构建**：自定义模块、预训练模型加载、Sequential容器
3. **训练流程**：损失函数定义、优化器应用、模型保存与恢复
4. **推理应用**：数据预处理、模型推理模式（eval()）、量化与剪枝

第5章-编程框架原理

一、编程框架设计

1. 设计原则

- 简洁性：抽象机制屏蔽底层细节，聚焦算法逻辑
- 易用性：基于Python生态，命令式编程范式
- 高效性：静态图优化、硬件加速支持

2. 整体架构

- 四大核心模块：

    - 计算图构建模块：用户程序→原始计算图
    - 分布式训练模块：单设备→多设备扩展
    - 深度学习编译模块：图层级+算子级优化
    - 计算图执行模块：计算图→设备执行

二、计算图构建

1. 正向图与反向图

- 基本元素：张量(Tensor)、操作(Operation)、有向边
- 正向传播：输入→输出的前向计算过程
- 反向传播：损失函数→梯度计算的反向推导

2. 动态图与静态图

- 动态图（PyTorch）：运行时逐语句构建，即时执行，易调试
- 静态图（TensorFlow 1.x）：先构建全图再执行，性能优化空间大

3. 自动求导机制

- 求导方法对比：
    - 手动求导：链式法则，代码量大
    - 数值求导：近似计算，效率低
    - 符号求导：表达式膨胀，复杂度高
    - 自动求导：前向记录依赖，反向自动推导
- PyTorch的AutoGrad：backward()自动构建反向图，支持动态梯度计算

三、计算图执行

1. 设备管理

- 设备类型：CPU、GPU、DLP等，通过device标识符管理
- 设备操作：初始化、句柄获取、执行流同步、事件管理

2. 张量实现

- 逻辑视图：形状、步长、数据类型、布局(NCHW/NHWC)
- 物理视图：内存地址、存储分配（即时分配/内存池分配）
- 内存共享：切片操作不复制物理内存，仅修改逻辑视图

3. 算子执行

- 执行序列：拓扑排序确定算子依赖关系
- 算子实现：前端定义(Python API)、后端实现(C++)、前后端绑定
- 分派执行：根据设备类型和张量属性查找对应内核函数

四、深度学习编译

1. 编译必要性

- 解决问题：新硬件适配成本高、手动优化效率低
- 核心优势：图层级全局优化、算子级自动调优、跨平台代码生成

2. 图层级优化

- 图优化方法：
  - 子图替换：等价运算替换为更高效形式
  - 常量折叠：预计算定值表达式
  - 算子融合：合并小算子减少Kernel调用开销
  - 布局优化：NHWC/NCHW转换适配硬件加速器

3. 算子层级优化

- 调度原语：循环分块(tiling)、向量化(vectorize)、并行化(parallel)
- 自动调优：搜索空间生成、性能评估、代价模型引导优化
- 中间表示：计算与调度分离，支持跨平台代码生成

4. 常见编译器

- TVM：基于调度原语的自动调优，支持多硬件后端
- MLIR：方言机制支持多层级优化，解决碎片化问题
- XLA：细粒度算子抽象，支持GPU/TPU等加速器

五、分布式训练

1. 分布式基础

- 驱动因素：模型参数量级(如GPT-3 1750亿参数)、数据规模(TB级训练数据)
- 架构类型：
  - 参数服务器：中心化管理模型参数，适用于大规模参数场景

    - 集合通信：去中心化，通过通信原语同步参数

2. 通信原语

- 一对多：Broadcast(广播)、Scatter(散射)
- 多对一：Gather(收集)、Reduce(归约)
- 多对多：All-Gather(全收集)、All-Reduce(全归约)

3. 训练方法

- 数据并行：复制模型副本，划分输入数据，DDP/FSDP实现
- 模型并行：拆分模型参数，算子内/间并行，张量并行/流水并行
- 混合并行：结合数据与模型并行，DeepSpeed实现流水线+张量并行

4. 框架实现

- 划分模块：数据分片(DistributedSampler)、模型切分
- 通信模块：梯度平均(桶机制)、参数同步(All-Reduce)

六、本章小结

1. **核心模块**：计算图构建与执行是基础，编译与分布式训练提升性能
2. **技术趋势**：动态图易用性与静态图性能优化结合，自动编译适配多硬件
3. **关键应用**：大模型训练依赖混合并行策略，通信效率决定分布式性能

第6章-面向深度学习的处理器原理

一、通用处理器与深度学习挑战

1. 通用处理器架构

- 冯·诺依曼结构：存储程序、控制器与运算器分离
- 哈佛结构：指令与数据缓存分离，支持并行访存
- RISC架构：精简指令集，通过load/store访存

2. 深度学习执行瓶颈

- **控制开销**：循环展开、分支预测等带来的指令冗余（如卷积运算中控制指令占比超50%）
- **访存效率**：
    - 缓存失效：容量失效、临界步幅、伪共享问题

    - 访问放大：逐字访存导致内存带宽浪费（如8×8矩阵乘内存访问量达324字）
- **运算密度**：标量运算为主，向量操作占比低（运算单元仅占芯片面积1%）

二、向量处理器与SIMD优化

1. SIMD架构原理

- Flynn分类：单指令流多数据流（SIMD）
- 指令类型：
    - SWAR：固定向量长度，使用向量寄存器（如Cray-1）
    - SIMT：单指令多线程，支持分歧执行（如NVIDIA GPU）

2. 向量指令优化

- 摊薄控制开销：一条向量指令替代n条标量指令
- 典型案例：
    - 向量乘：n次运算仅需1次控制指令
    - BLAS库：三级接口提升运算密度（三级算子性能提升50倍）

三、图形处理器（GPU）架构

1. 硬件结构

- TPC（纹理处理器集群）：包含SM（流式多处理器）
- SM核心：SP（流式处理器）、SFU（特殊函数单元）
- 存储层次：共享内存、纹理缓存、显存

2. 并行计算模型

- SIMT执行：线程束（Warp）调度，支持分支分歧
- 内存模型：全局内存、共享内存、寄存器文件

四、深度学习处理器（DLP）设计原理

1. 架构创新

- **运算单元**：矩阵运算单元替代向量单元（如256×256乘累加器）
- **存储优化**：
    - 便笺存储器：替代缓存，面积减少50%，速度提升23%
    - DMA直接访存：减少CPU参与数据搬运

- **控制简化**：计数循环指令优化，减少分支预测器

2. I/O复杂度优化

- 理论模型：矩阵乘运算量≅n^2，I/O≅3n（优于向量运算的I/O≅3n）
- 分块技术：将矩阵分解为3×3区块，命中率从56%提升至78%

3. 多核扩展策略

- 分形架构：相同规则扩展核数，保持编程一致性
- 实例：寒武纪MLU370S，运算单元占芯片面积30%
- 总线仲裁：解决多核访存争用问题

五、DLP发展历程与案例

1. 历史演进

- 第一次热潮（1950s）：SNARC模拟器、Mark-I神经网络计算机
- 第二次热潮（1980s）：Intel ETANN、CNAPS数字电路
- 第三次热潮（2006至今）：
  - Google TPU：从256×256矩阵单元到多核架构
  - 寒武纪系列：从DianNao到MLU370S，运算密度提升16倍

2. 典型架构对比

处理器	运算单元	存储设计	典型应用
CPU	标量/向量	多级缓存	小模型推理
GPU	SIMT线程束	共享内存	训练加速
DLP	矩阵单元	便笺存储器	大规模训练/推理

六、关键技术总结

1. **存储优化**：便笺存储器替代缓存，解决"内存墙"问题
2. **控制简化**：计数循环指令减少控制开销，适配深度学习规律
3. **规模扩展**：分形多核架构，支持线性性能提升

第7章-深度学习处理器架构

一、总体架构

1. 核心模块划分

- 计算单元：矩阵/向量/标量运算单元协同工作
- 访存系统：便笺存储器与外部存储交互
- 通信网络：支持多核间数据传输与任务划分

二、计算单元设计

1. 矩阵运算单元

- **实现方式**：
  - 内积单元堆叠：从向量内积到矩阵乘向量/矩阵
  - 脉动阵列机：数据流驱动的矩阵计算，计算-I/O比例1:0.75
  - 结构对比：脉动阵列vs矩阵乘矩阵单元（连线复杂度与计算密度）
- **关键指标**：计算-I/O比例优化（从1:3提升至1:0.3）

2. 向量与标量单元

- **功能模块**：
  - 池化/归一化：MaxPool/AvgPool/BatchNorm硬件实现
  - 激活函数：分段线性近似计算tanh/ReLU等
  - 前缀计算：并行求和/最大值/非零计数
  - 数据重排布：排列网络（Beneš/Waksman）解决访存对齐问题

三、访存系统架构

1. 便笺存储器设计

- **核心作用**：DLP核内数据枢纽，连接计算单元与外部存储
- **优化策略**：
  - 多端口SRAM：拓宽带宽（代价：面积+50%~100%）
  - 分组SRAM：减少bank冲突（开关阵列复杂度$O(n^2)$）
  - 分离式设计：按数据类型/功能单元划分（牺牲通用性提升效率）

2. 外部存储器访问

- **DMA机制**：批量数据搬运，替代逐条load/store指令
- **软件流水线**：通过指令重排序实现计算/访存并行

- **同步控制**：显式sync指令简化硬件设计

四、通信网络设计

1. 任务划分模式

- 数据并行：全局归约（Ring All-Reduce）
- 模型并行：算子内/间并行（All-to-All通信）

2. 物理链路实现

- 总线（AXI/PCI-E）：低成本，低性能
- 片上网络（NoC）：胖树/二维环面拓扑，平衡性能与成本
- 交叉开关阵列：高性能，高成本

五、优化设计技术

1. 算法变换

- 快速矩阵乘法：Strassen/Coppersmith-Winograd算法
- 快速卷积：Winograd最小滤波算法
- 算子融合：减少访存次数（如L1-L4层合并计算）

2. 模型压缩

- 网络裁剪：非零权值筛选硬件
- 结构化稀疏：规律化非零分布（如Block Sparse）
- 串行计算：逐比特处理降低功耗

3. 近似计算

- 数值量化：INT4/FP4/BFP/PoT对数域计算
- 算法近似：低秩分解（W=A×B^T）、差分计算

4. 非传统架构

- 神经拟态计算：模拟电路实现乘累加（电压/电导/电流映射）
- 存算一体：计算发生在存储矩阵内（如RRAM阵列）

六、关键技术总结

1. **计算效率**：矩阵单元规模与计算-I/O比例的权衡
2. **访存优化**：便笺存储器分区策略与DMA流水线设计

3. **通信架构**：逻辑环状链路与物理冗余设计
4. **前沿方向**：存算一体、模拟计算等非冯·诺依曼架构

第8章-智能编程语言

一、为什么需要智能编程语言

1. 三大鸿沟

- **语义鸿沟**：传统语言（如C++/Python）与深度学习语义（如卷积、张量）的抽象差距
- **硬件鸿沟**：智能硬件（DLP）的定制运算单元（FP16/INT8）与传统语言数据类型不匹配
- **平台鸿沟**：不同硬件平台（CPU/GPU/DLP）的编程接口碎片化，缺乏统一抽象

2. 领域专用语言优势

- 提升开发效率：一条语句实现卷积运算（如`conv(input, filter, bias)`）
- 优化执行性能：直接映射硬件特性（如向量指令、片上存储）
- 跨平台可移植性：抽象共性特征，屏蔽底层差异

二、智能计算系统抽象架构

1. 硬件层次结构

- 五级架构：服务器级→板卡级→芯片级→处理器簇级→处理器核级
- 核心组件：控制单元、计算单元（矩阵/向量/标量）、存储层次（NRAM/SRAM/DDR）

2. 计算模型

- 定制运算单元：支持FP16/BF16/INT8等低位宽计算
- 并行架构：多Cluster多核协同，支持任务切分与同步

3. 存储模型

- 本地存储：NRAM（高速）、WRAM（权值存储）、SRAM（簇共享）
- 全局存储：DDR/HBM，通过DMA批量搬运

三、智能编程模型

1. 异构编程模型

- 主机端（CPU）：任务调度、数据准备
- 设备端（DLP）：核心计算（Kernel函数）
- 典型流程：分离式编程（主机/设备代码分别编译）

2. 通用智能编程模型

- **Kernel定义**：
  - 入口函数：`__dlp_entry__`（设备端）、`InvokeKernel`（主机端调用）
  - 调度类型：BLOCK（单核）/UNIONx（多核并行）
- **同步机制**：`__sync`（核内）、`__sync_cluster`（簇内）、`__sync_all`（全局）
- **内建函数**：`__matmul`（矩阵乘）、`__mlp`（全连接）

四、智能编程语言基础

1. 语法与数据类型

- 基本类型：half（FP16）、int8_t/uint8_t等低位宽类型
- 张量类型：支持多维数组操作（如`__vector_add`）

2. 核心语句

- I/O操作：`__memcpy`（不同存储层次数据搬运）
- 控制流：分支/循环/同步语句（`__sync_cluster`）
- 张量计算：`__vector_relu`（向量激活）、`__bang_maxpool`（池化）

3. 编程示例

- 串行程序：向量平方计算（分块处理+NRAM优化）
- 并行程序：矩阵乘法（4核并行，任务切分）

五、智能应用编程接口

1. Kernel函数接口

- 任务切分变量：`coreId`（核序号）、`taskDim`（任务规模）
- 主机端调用：`InvokeKernel`接口与`<<<>>>`语法糖

2. 运行时接口

- 设备管理：`cnrtGetDevice`（获取设备）、`cnrtSetDevice`（设置设备）
- 队列管理：`cnrtQueueCreate`（创建异步队列）、`cnrtQueueSync`（队列同步）

- 内存管理：`cnrtMalloc`（设备内存分配）、`cnrtMemcpy`（主机-设备数据拷贝）

六、智能编程语言实例：BANG语言

1. 架构与流程

- 基于C++扩展，支持MLU硬件
- 编程流程：主机端准备数据→设备端Kernel计算→结果拷贝回主机

2. 核心功能

- 数学库：`__bang_add`（向量加）、`__bang_transpose`（矩阵转置）
- 神经网络算子：`__bang_maxpool`（最大池化）、`__bang_sigmoid`（激活函数）

3. 编译与链接

- 混合编译：主机端（clang）+设备端（bcIc）分别编译，统一链接
- 支持多架构：自动生成不同DLP型号的目标代码

七、智能应用功能调试

1. 调试接口

- 断言：`__assert`（条件不满足时终止Kernel）
- 输出：`printf`（设备端格式化输出，支持`%hf`等类型）

2. 调试工具

- BCL-GDB：支持多核切换、断点设置（函数/行号/地址）
- 功能调试流程：准备调试信息→设置断点→单步执行→变量查看

八、智能应用性能调优

1. 性能分析方法

- 硬件计数器：获取运算单元利用率、访存带宽
- Notifier接口：测量Kernel执行时间（`cnrtNotifierDuration`）

2. 调优工具

- 应用级：`bcl-perf`（生成性能报告，分析热点函数）
- 系统级：`monitor`命令（查看设备功耗、温度、带宽）

3. 优化方法

- 片上存储：利用NRAM减少访存（如向量乘优化）

- 向量化：将标量循环替换为张量计算（`__vec_mul`）
- 软件流水：三级/五级流水实现计算-访存并行
- 多核并行：按`taskId`切分任务，4核协同计算

九、智能编程语言的应用

1. 高性能库算子开发

- 流程：设计API→Kernel开发→编译封装→集成测试
- 实例：矩阵乘三级流水/五级流水实现

2. 编程框架算子集成

- PyTorch集成：
  1. 编写BCL Kernel与主机端接口
  2. 使用`DLPExtension`编译为Python扩展
  3. 在PyTorch网络中调用自定义算子（如`sigmoid`）

第9章-大模型计算系统

一、大模型概述

1. 模型分类

- **大语言模型**：基于自然语言建模（如GPT系列、BERT）
- **多模态大模型**：融合文本、图像、语音等模态（如LLaVA）
- **架构差异**：仅编码器（BERT）、编码器-解码器（T5）、仅解码器（GPT）

2. 发展历程

- 2017年Transformer提出，2018年BERT开启预训练时代
- 2020年GPT-3（175B参数）推动仅解码器架构主流化
- 2023年Llama2（70B参数）开源推动模型普及

3. 计算需求

- **参数量级**：从Transformer的213M到PanGu-Σ的1085B
- **训练资源**：GPT-3需数千块A100，训练时间数月
- **计算量公式**：训练量∝参数量×数据集规模

二、大模型算法分析

1. BLOOM-176B模型

- **架构**：70层解码器，隐藏层维度14336，112头注意力
- **训练数据**：ROOTS语料库（46种语言，166B词元）
- **并行策略**：数据并行（8）+张量并行（4）+流水线并行（12）

2. 计算特性

- **运算量**：单批量训练需34.8P FLOPs（正向+反向）
- **存储需求**：模型权重+优化器状态需3.5TB（FP32）
- **通信瓶颈**：张量并行导致ALL-Reduce通信量占比高

三、大模型系统软件

1. 训练优化技术

- **计算优化**：
  - 稀疏注意力：降低长序列计算量（DeepSpeed加速6倍）
  - 混合精度：TF32/FP16替代FP32，算力提升8-16倍
- **存储优化**：
  - ZeRO系列：分片存储优化器状态，减少75%存储
  - 重计算：用30%计算换5倍激活值存储节省
- **通信优化**：
  - 1-bit Adam：梯度量化减少5倍通信量
  - FlashAttention：融合Attention计算，降低IO开销

2. 推理优化技术

- **批处理**：连续批处理动态调度，提升吞吐率
- **KV缓存**：分页管理减少60%存储浪费
- **量化技术**：混合精度量化（权重INT8+激活FP16）

四、大模型基础硬件

1. 计算节点架构

- **单节点配置**：64核CPU+8×A100 GPU（80GB HBM2e）

- **拓扑结构**：
  - 平衡型：CPU与GPU通过PCIe Switch互联
  - 直连型：减少中间节点延迟

2. 集群架构

- **互联网络**：
  - 高速网络：英特尔Omni-Path（100Gbps）
  - 拓扑选择：胖树（高带宽）、Dragonfly（低延迟）
- **存储系统**：基于GPFS的分布式共享存储

五、大模型驱动范例：BLOOM

1. 训练流程

- **并行策略**：
  - 数据并行度8，张量并行度4，流水线并行度12
  - 48节点×8GPU=384GPU集群训练105天
- **通信组**：数据并行组、张量并行组、流水线并行组

2. 推理流程

- **自回归生成**：依赖前序输出，逐词生成
- **硬件配置**：单节点8GPU，张量并行或流水线并行

六、关键技术索引

1. **核心算法**：Transformer架构、自注意力机制、混合并行策略
2. **系统优化**：ZeRO存储优化、FlashAttention计算融合、1-bit Adam通信压缩
3. **硬件架构**：GPU集群拓扑、HBM高带宽存储、高速互联网络