推理系统系列

推理流程全景

ZOMI

Talk Overview

1. **推理系统介绍**
   - 推理系统与推理引擎
   - 推理系统的工作流程
   - 推理系统生命周期管理
2. **模型小型化**
   - NAS神经网络搜索
   - CNN小型化结构
   - Transform小型化结构
3. **离线优化压缩**
   - 低比特量化
   - 二值化网络
   - 模型模型剪枝
   - 模型模型蒸馏
4. **部署和运行优化**
   - 图转换优化（算子融合/重排/替换）
   - 并发执行与内存分配
   - 动态batch与bin Packing

Talk Overview

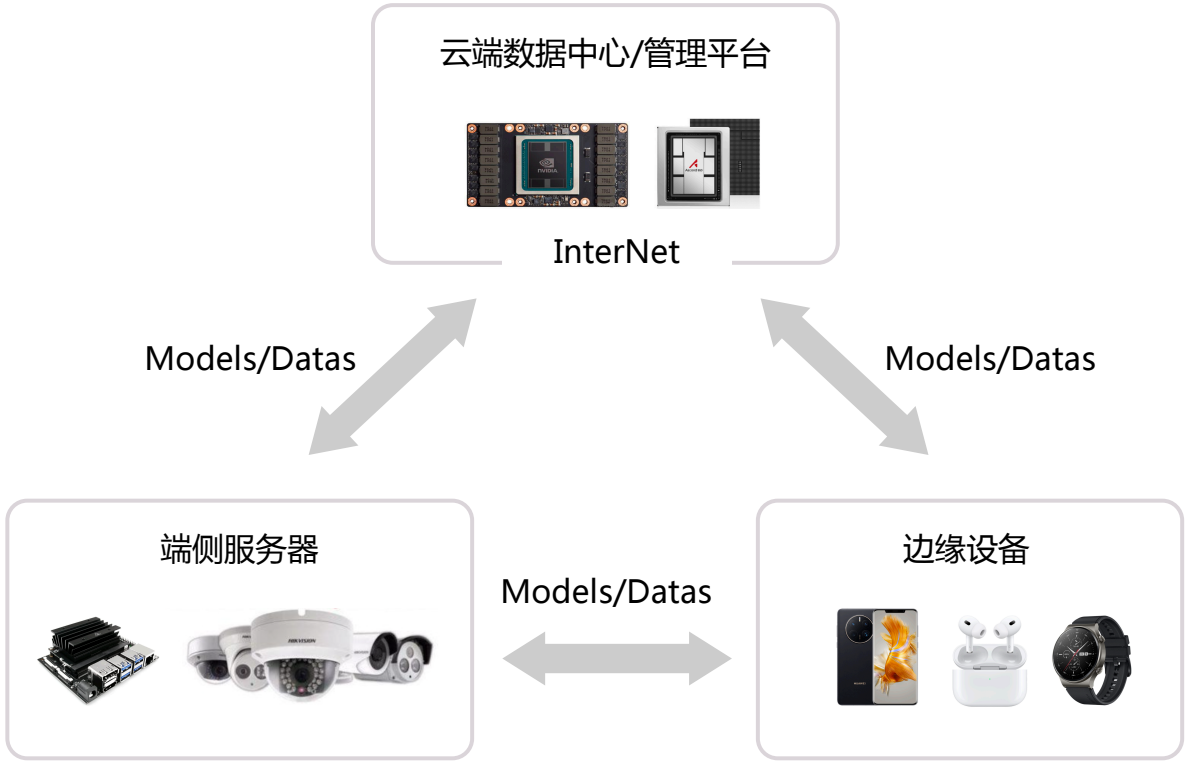
1. 推理流程全景

- Difference on Deployment status – 部署态区别
- Cloud deployment workflow – 云测推理流程
- Edge Deployment mode – 边缘部署方式

部署态区别

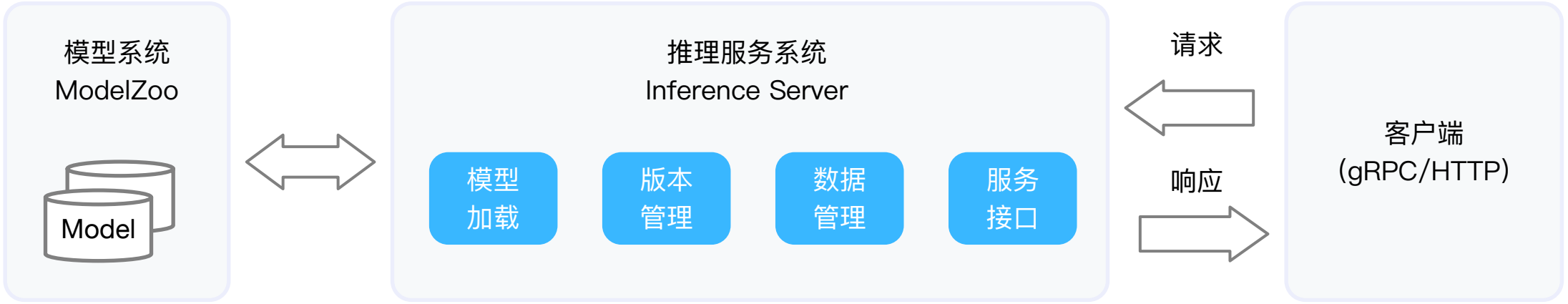
部署态

- 推理系统一般可以部署在云或者边缘。云端部署的推理系统更像传统 Web 服务，在边缘侧部署的模型更像手机应用和IOT应用系统。



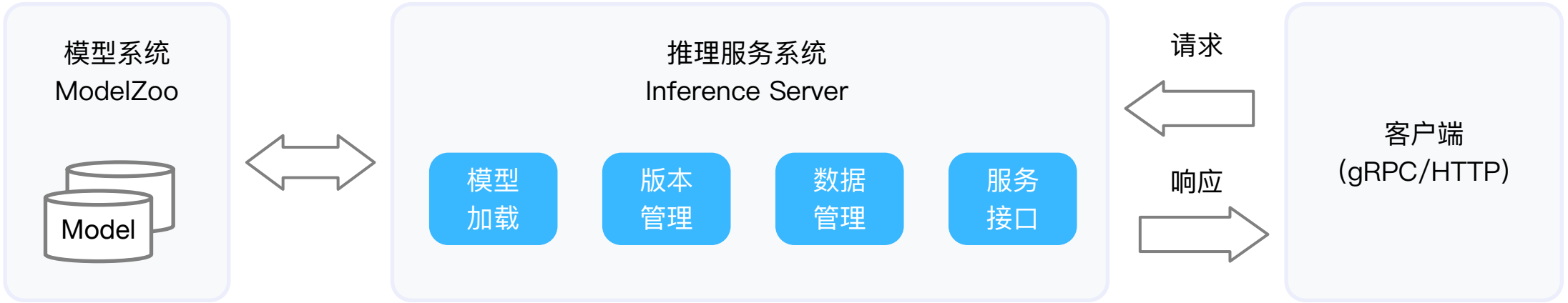
部署态： Cloud 云端

- 云端有更大的算力，内存，且电更加充足满足模型的功耗需求，同时与训练平台连接更加紧密，更容易使用最新版本模型，同时安全和隐私更容易保证。相比边缘侧可以达到更高的推理吞吐量。但是用户的请求需要经过网络传输到数据中心并进行返回，同时使用的是服务提供商的软硬件资源。



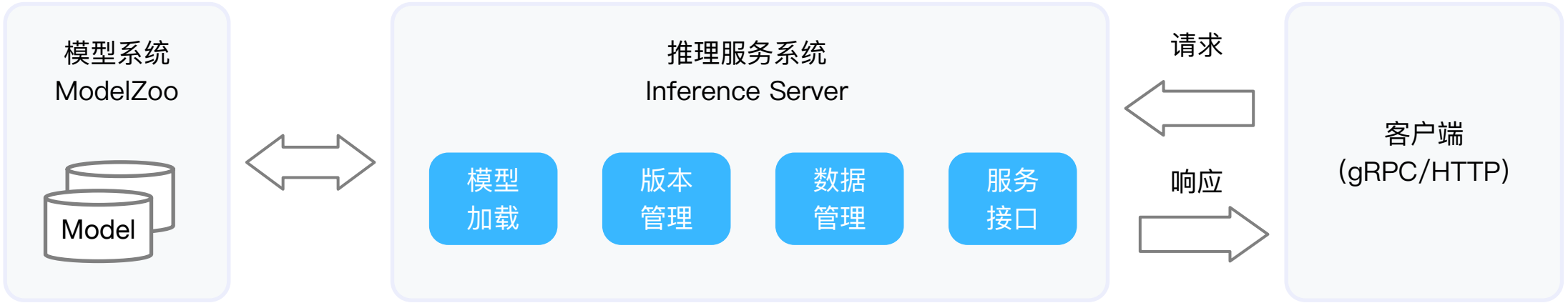
部署态： Cloud 云端特点

- 对功耗、温度、 Model Size 没有严格限制
- 有用于训练和推理的强大硬件支持
- 集中的数据管理有利于模型训练
- 模型更容易在云端得到保护
- 深度学习模型的执行平台和AI 框架统一



部署态： Cloud 云端问题

- 云上提供所有人工智能服务成本高昂
- 推理服务对网络依赖度高
- 数据隐私问题
- 数据传输成本
- 很难定制化模型



部署态： Edge 端侧

- 边缘侧设备资源更紧张（例如，手机和 IOT 设备），且功耗受电池约束，需要更加在意资源的使用和执行的效率。用户的响应只需要在自身设备完成，且不需消耗服务提供商的资源。

部署态： Edge 端侧挑战

- 严格约束功耗、热量、模型尺寸小于设备内存
- 硬件算力对推理服务来说不足
- 数据分散且难以训练
- 模型在边缘更容易受到攻击
- DNN平台多样，无通用解决方案

部署态： Edge 端侧

应用层算法优化：

- 考虑到移动端部署的苛刻资源约束条件下，提供针对移动端部署的 AI 模型

高效率模型设计：

- 通过模型压缩的量化、剪枝、蒸馏、神经网络结构搜索（NAS）等技术，减少模型尺寸

移动端框架——推理引擎：

- TensorFlow Lite，MNN、TensorRT，ONNX Runtime、MindSpore Lite等推理引擎推出

移动端芯片：

- 提供高效低功耗芯片支持，如 Google Edge TPU，NVIDIA Jetson 、Huawei Ascend 310系列

端侧部署技术难点分析

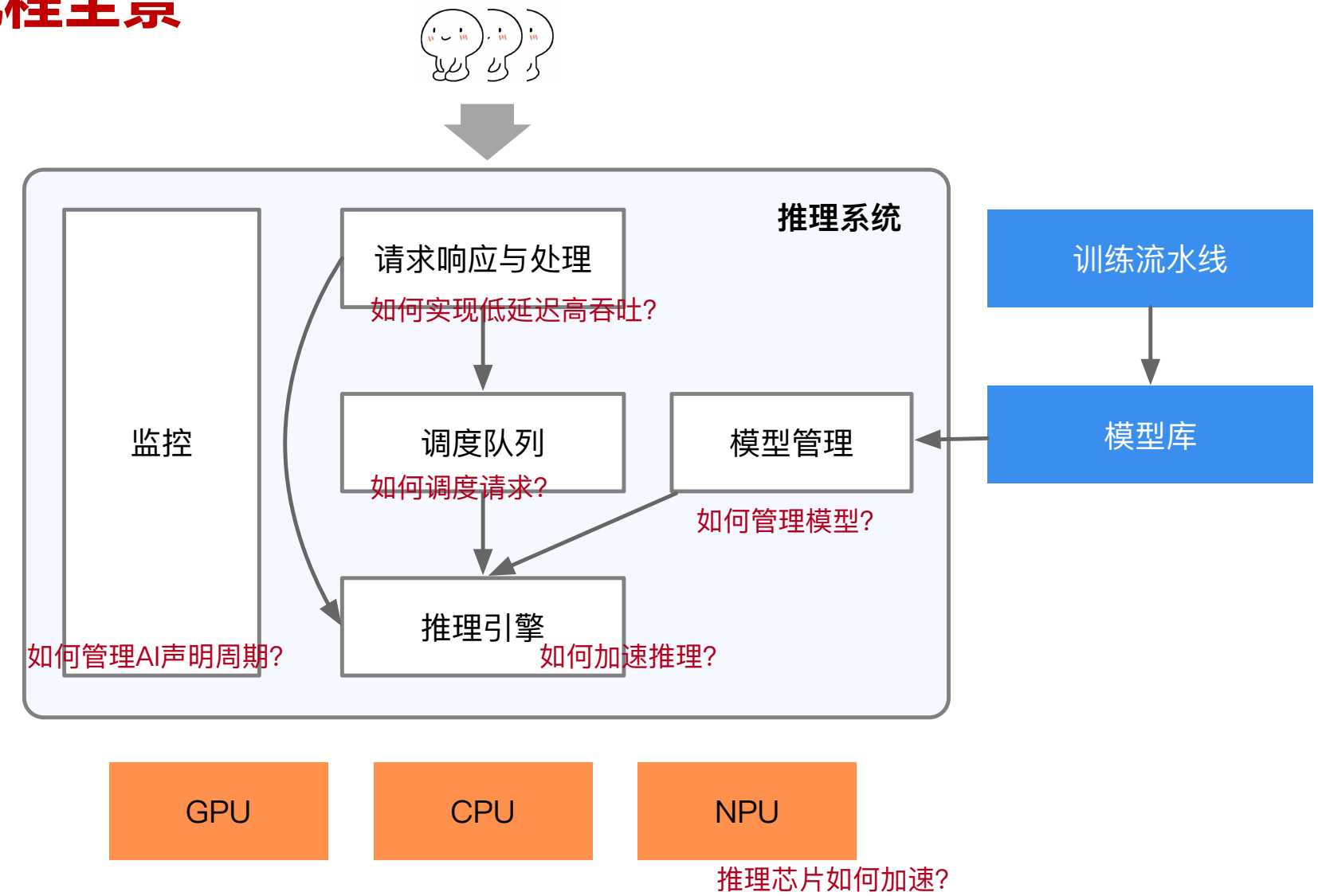
	云端部署	端侧部署
算力	算力强大（TFLOPS，并行可扩展），适合训练和推理阶段计算	算力有限，水平扩展性差，更适合推理阶段前向计算
时延	网络时延+计算开销	本地计算无网络开销或开销低，实时响应要求高
网络依赖	强依赖	弱依赖，隐私包括，联邦学习
能耗	百瓦+	几十瓦，能耗比高
系统架构	开放，高度集中	封闭，架构分散
多样性	标准化程度高，CPU/GPU/NPU	多样性芯片架构，SOC多
研发成本	配套完善，可移植性高	配套不完善，可移植性受限

部署难点

- 部署维护成本高，难落地
- 模型适配、迁移难，重复开发
- 预测性能差，硬件成本高
- 高精度模型体积大，性能差

云测部署和推理方式

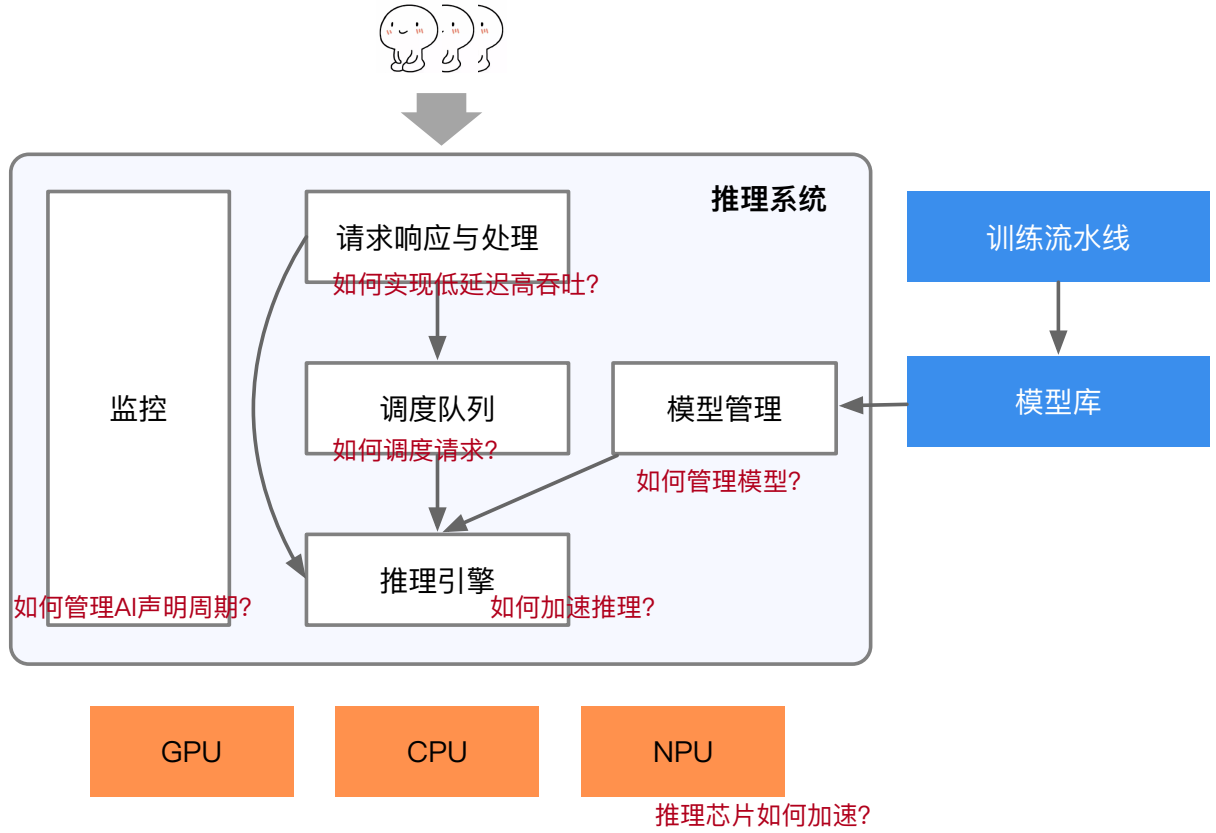
推理系统流程全景



推理系统流程全景：请求与响应处理

- 系统需要序列化与反序列化请求，并通过后端高效执行，满足一定的响应延迟。

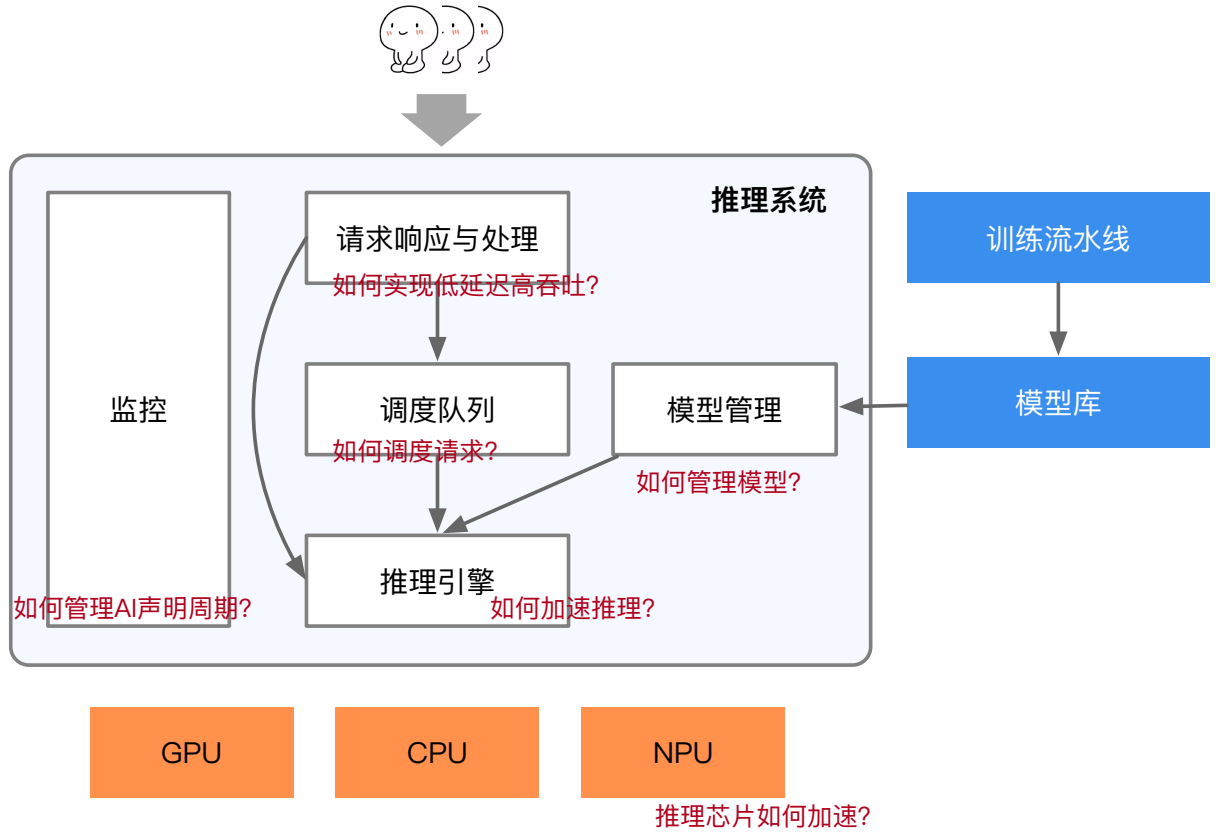
- 相比传统的 Web 服务，推理系统常常需要接受图像，文本，音频等非结构化数据，单请求或响应数据量一般更大，这就需要对这类数据有高效的传输，序列化，压缩与解压缩机制。



推理系统流程全景：请求调度

- 系统可以根据后端资源利用率，动态调整批尺寸，模型的资源分配，进而提升资源利用率，吞吐量。

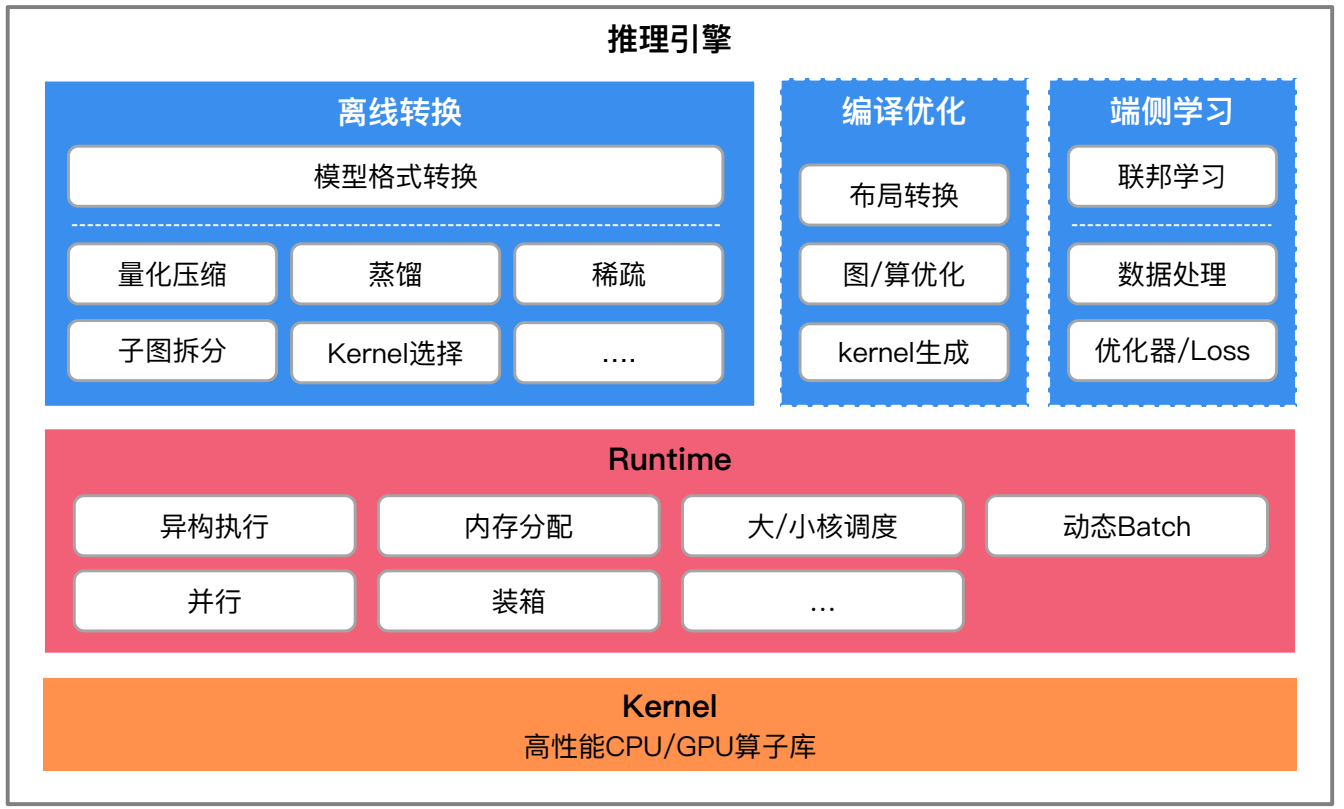
- 如果是通过加速器进行的加速推理，还要考虑主存与加速器内存之间的数据拷贝，通过调度或预取等策略在计算的间歇做好数据的准备。



推理系统流程全景：推理引擎执行

- 推理引擎将请求映射到模型作为输入，并在运行时调度深度学习模型的内核进行多阶段的处理。

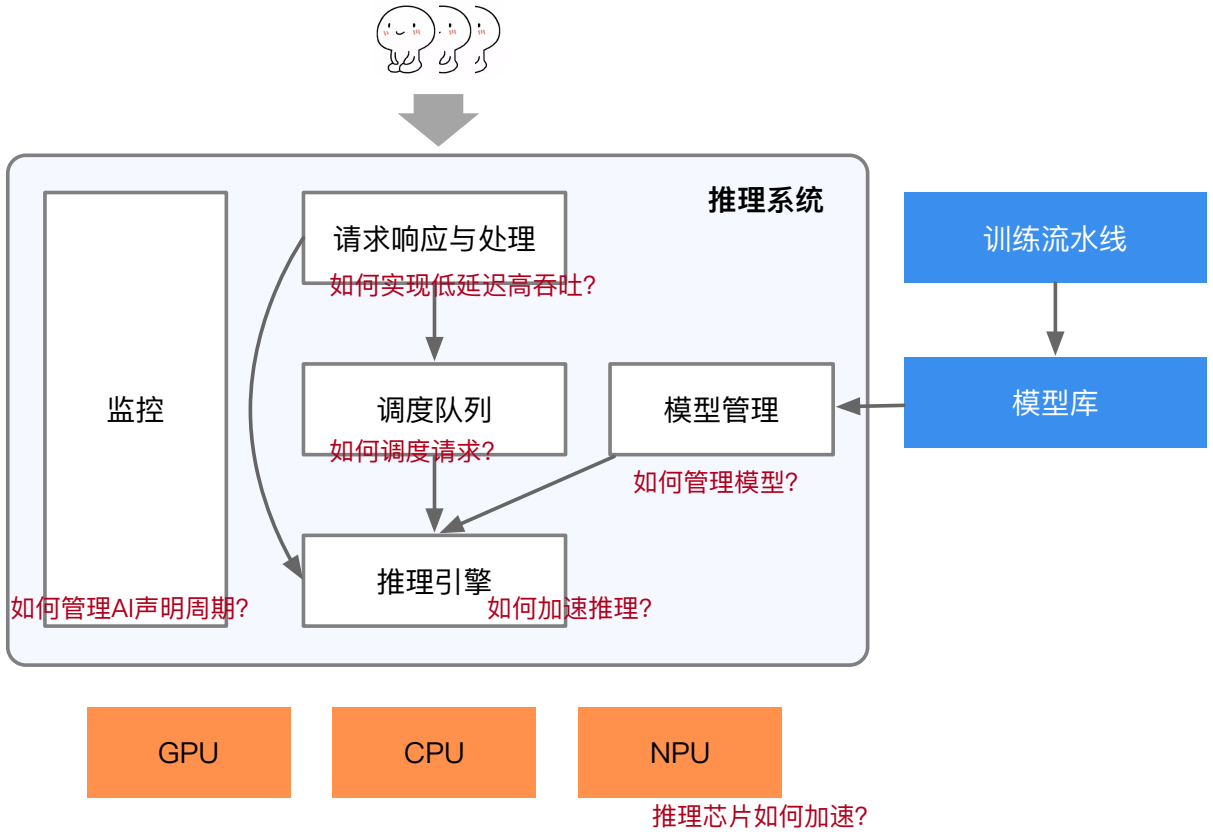
- 如果是部署在异构硬件或多样化的环境，还可以利用编译器进行代码生成与内核算子优化，让模型自动化转换为高效的特定平台的可执行的机器码。



推理系统流程全景：模型版本管理

- 在云端算法工程师不断验证和开发新的版本模型，需要有一定的协议保证版本更新与回滚。

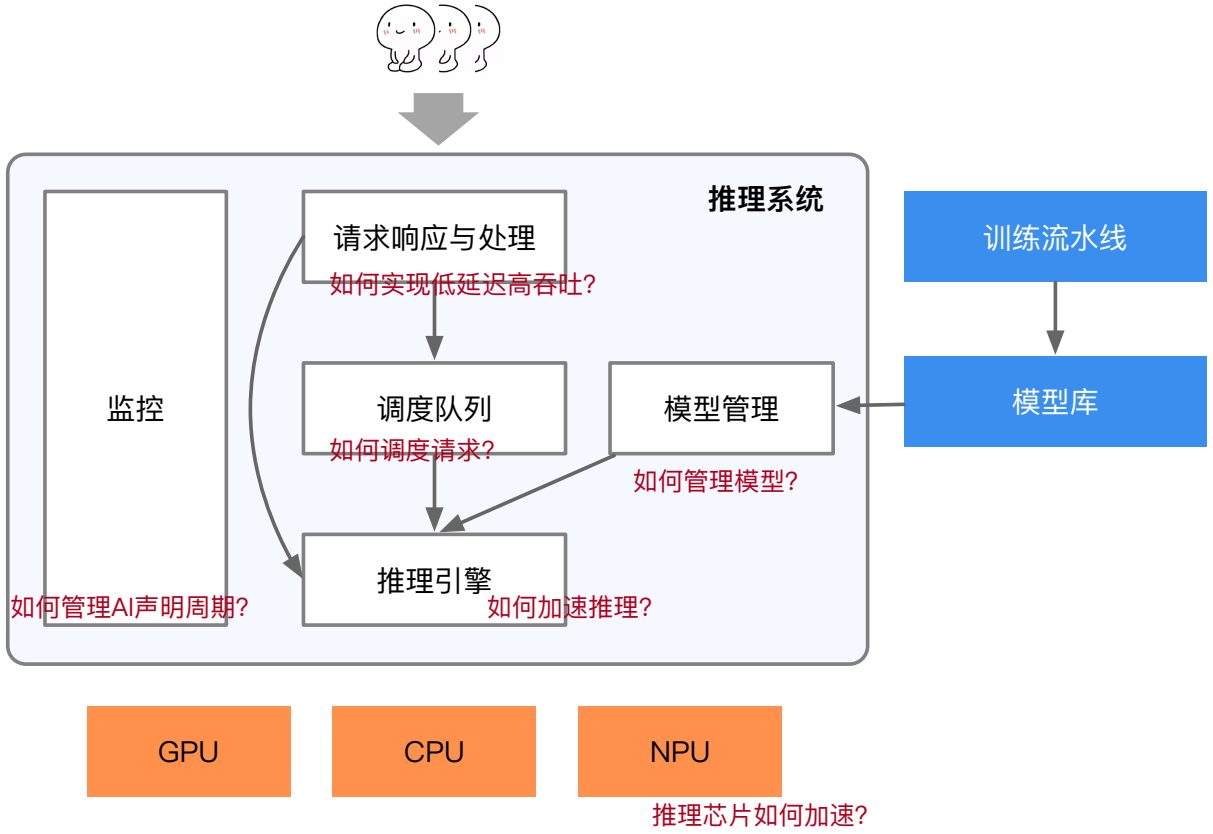
- 定期或满足一定条件的新模型不断上线替换线上模型，以提升推理服务的效果，但是由于有些指标只能线上测试，有可能线上测试效果较差还需要支持回滚机制，让模型能回滚到稳定的旧版本模型。



推理系统流程全景：健康监控

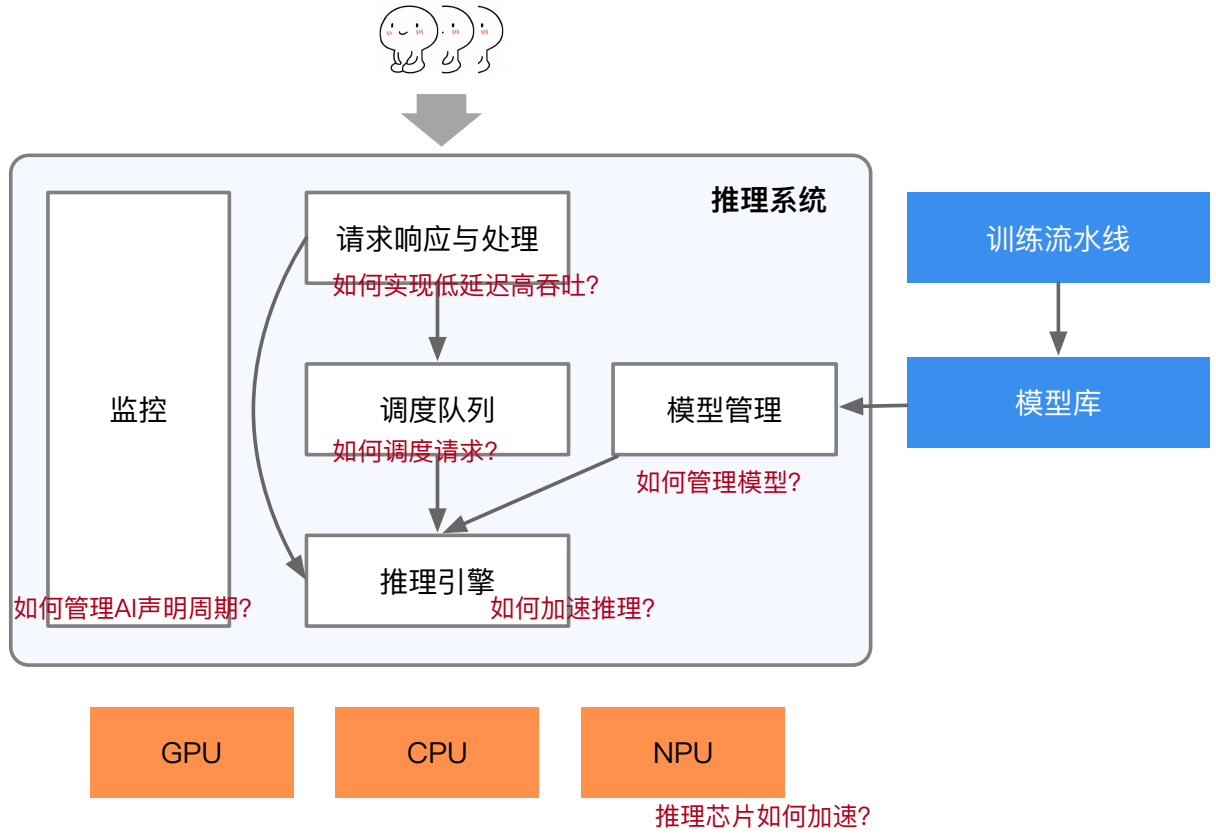
- 云端的服务系统应该是可观测的，才能让服务端工程师监控，报警和修复服务，保证服务的稳定性和 SLA。

- 例如，一段时间内响应变慢，通过可观测的日志，运维工程师能诊断是哪个环节成为瓶颈，进而可以快速定位，应用策略，防止整个服务突发性无法响应（例如，OOM 造成服务程序崩溃）。



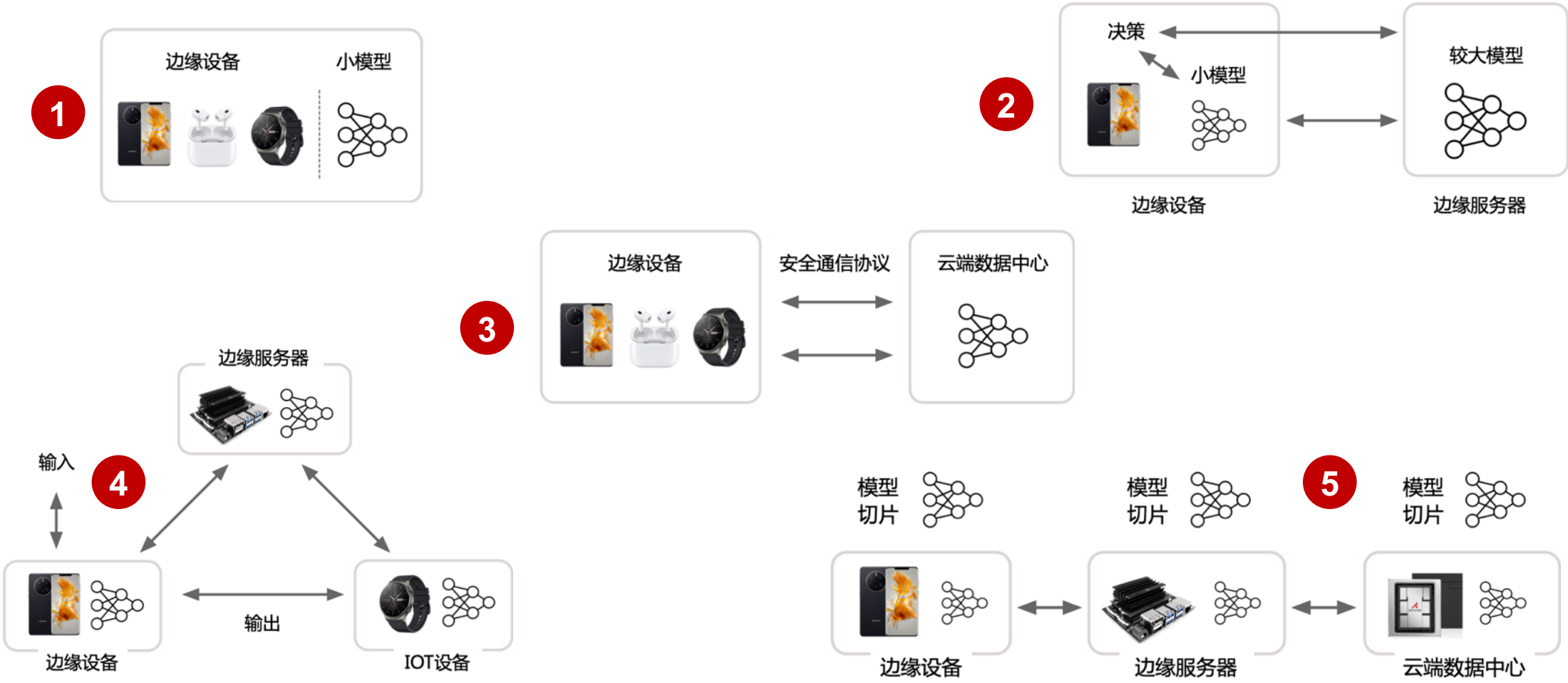
推理系统流程全景：推理硬件

- 在边缘端等场景会面对更多样的硬件，驱动和开发库，需要通过编译器进行一定代码生成让模型可以跨设备高效运行，并通过编译器实现性能优化。



边缘部署和推理方式

边缘部署与推理方式



方式1：边缘设备计算

将模型部署在设备端，聚焦如何优化模型执行降低延迟：

1. 端侧模型结构设计
2. 通过模型量化、剪枝等压缩手段
3. 针对神经网络的专用芯片 ASIC 设计

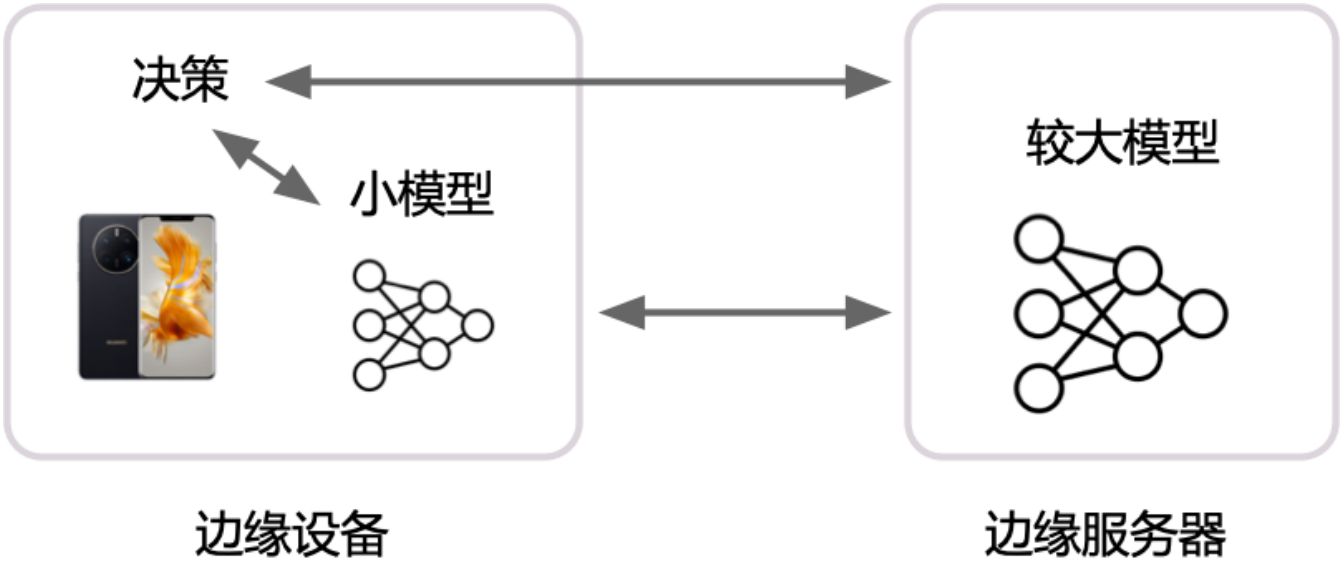
方式2：安全计算 + 卸载到云端

将模型部署于数据中心，边缘侧通过安全通信协议将请求发送到云端，云端推理返回结果，相当于将计算卸载到云端：

1. 利用云端运行提升模型安全性
2. 适合部署端侧无法部署的大模型
3. 完全卸载到云端有可能违背实时性的需求

方式3：边缘设备 + 云端服务器

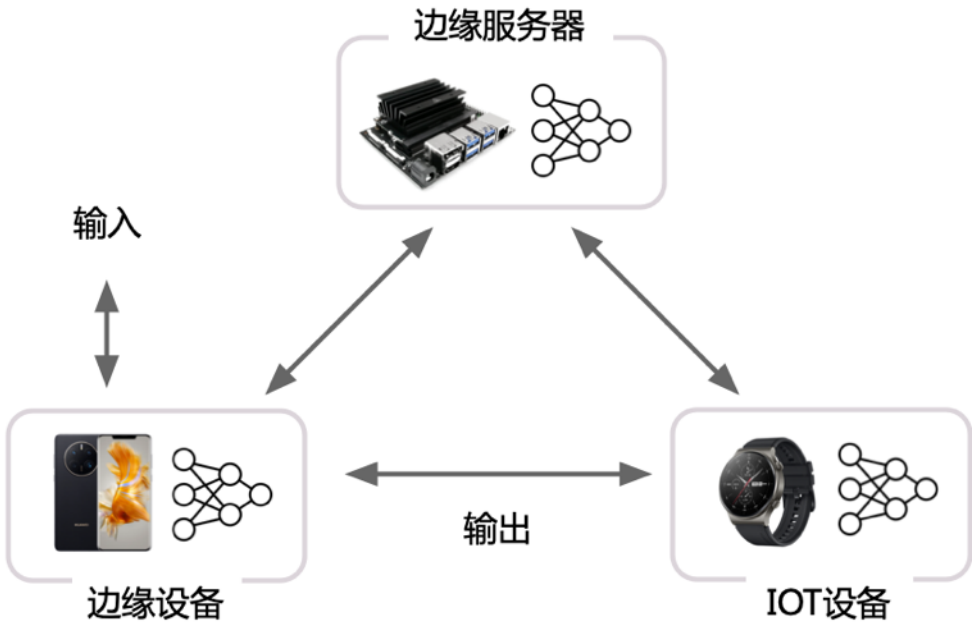
利用 AI 模型结构特点，将一部分层切（或者其 Student 模型）分放置在设备端进行计算，其他放置在云端。这种方式一定程度上能够比方式 2 降低延迟，由于其利用了边缘设备的算力，但是与云端通信和计算还是会带来额外开销。



方式4：分布式计算

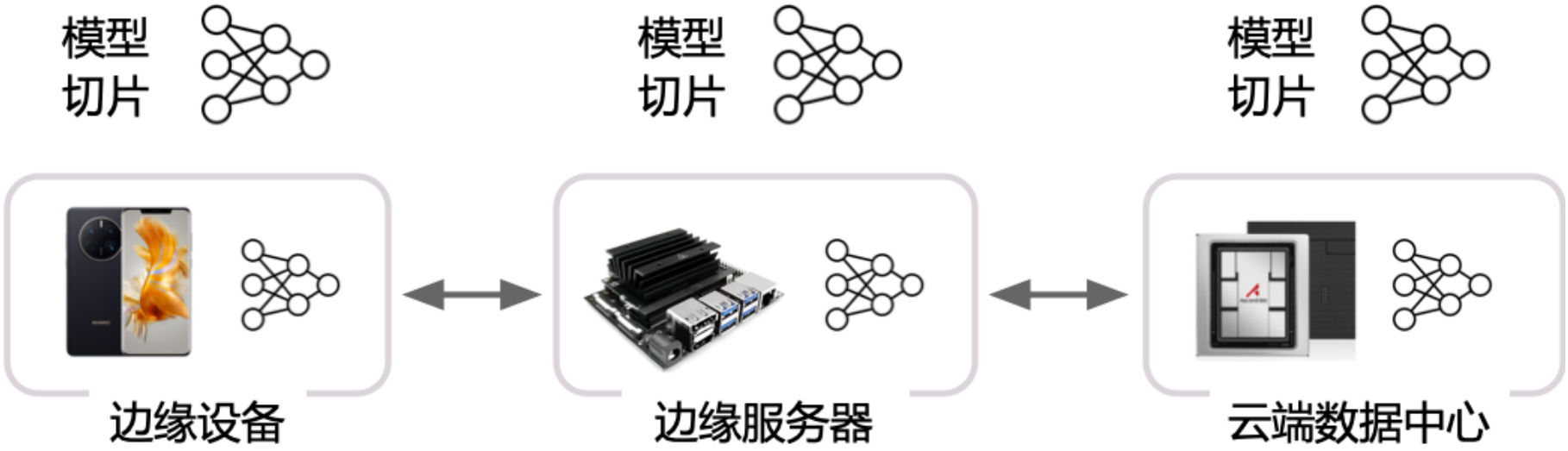
从分布式系统角度抽象问题，AI 计算在多个辅助边缘设备上切片：

- 切片策略根据设备计算能力，内存约束
- 通过细粒度的切片策略，将模型切片部署其他边缘设备
- 运行对计算模型进行调度， 并通过输入数据通过负载均衡策略进行调度



方式5：跨设备 Offloading

- 决策基于经验性的权衡功耗，准确度，延迟和输入尺寸等度量和参数，不同的模型可以从当前流行的模型中选择，或者通过知识蒸馏，或者通过混合和匹配的方式从多个模型中组合层。如较强的模型放在边缘服务器，较弱模型放置在设备。



参考文献

1. Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications
2. Clipper: A Low-Latency Online Prediction Serving System
3. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform
4. TensorFlow-Serving: Flexible, High-Performance ML Serving
5. Optimal Aggregation Policy for Reducing Tail Latency of Web Search
6. A Survey of Model Compression and Acceleration for Deep Neural Networks
7. CSE 599W: System for ML - Model Serving
8. https://developer.nvidia.com/deep-learning-performance-training-inference
9. DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING
10. Learning both Weights and Connections for Efficient Neural Networks
11. DEEP LEARNING DEPLOYMENT WITH NVIDIA TENSORRT
12. Halide: A Language and Compiler for Optimizing Parallelism,Locality, and Recomputation in Image Processing Pipelines
13. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning
14. 8-bit Inference with TensorRT
15. https://github.com/microsoft/AI-System