



南京大學

本科畢業論文

院 系 软件学院

专 业 软件工程

题 目 结合边缘计算的云 AI 平台的

设计与实现

年 级 2016 级 学 号 161250162

学生姓名 谢寅鹏

指导老师 刘海涛 职 称 讲师

提交日期 2020 年 5 月 31 日

南京大学本科生毕业论文(设计、作品)中文摘要

题目：结合边缘计算的云 AI 平台的设计与实现

院系：软件学院

专业：软件工程

本科生姓名：谢寅鹏

指导老师（姓名、职称）：刘海涛讲师

摘要：

本论文将阐述一种结合边缘计算的云 AI 平台的设计和实现。本系统提供联邦学习的支持。首先，系统将通用 AI 模型分发给边缘计算平台。然后，边缘计算平台利用端设备提供的事件数据训练本地 AI 模型。在完成本地 AI 模型的训练后，边缘计算平台将本地 AI 模型上传至云端。集中的本地 AI 模型可以使用联邦学习方法获得更好的通用 AI 模型。这样就完成了一次迭代。

本系统主要提供通用 AI 模型的发布与本地 AI 模型的部署、训练和上传两类服务。通过本系统，用户降低了用于通讯和计算的花销，减少了泄露隐私的风险，兼顾了 AI 模型的通用性和个性化。

本人参与了该系统的开发全过程，具体内容如下：

（1）完成用户需求的分析整理，选出最关键的需求

（2）完成系统的总体设计，并对关键组件提前调研。

（3）在框架和 SDK 下，完成 ModelCloud, EdgeX Foundry 等组件的开发。

关键词：边缘智能；边缘计算；联邦学习；数据隐私；人工智能；机器学习

南京大学本科生毕业论文 (设计、作品) 英文摘要

THESIS: Design and Implementation of AI Cloud Platform Combined with Edge Computing

DEPARTMENT: School of Software

SPECIALIZATION: Software Engineering

UNDERGRADUATE: Xie Yinpeng

MENTOR: Lecturer Liu

ABSTRACT:

This paper conducts the design and implementation of a cloud AI platform combined with edge computing. The system supports federated learning method. Firstly, it distributes the general AI model to the edge computing platforms. Secondly, the edge computing platform uses the event data provided by the end devices to train the local AI model. After the training of the local AI model, the edge computing platforms upload the local AI models to the cloud. Centralized local AI models can use federated learning method to obtain better general AI models. The above procedures constitute an iteration.

This system mainly provides two kinds of services: the release of general AI model and the deployment, training and upload of local AI model. Through this system, users can reduce the cost of communication and computing, decrease the risk of privacy disclosure, and take into account the universality and personalization of AI model.

This author participated in the whole development process of the system. The specific contents are as follows:

- (1) Complete the analysis and sorting of users' requirements and select the most critical requirements
- (2) Complete the overall design of the system and investigate the key components in advance.
- (3) Complete the development of model cloud, edgex foundry and other components under the framework and SDK.

KEY WORDS: Edge Intelligence; Edge Computing; Federated Learning; Data Privacy; Artificial Intelligence; Machine Learning

目 录

目 录	III
1 绪论	1
1.1 背景和意义	1
1.1.1 背景	1
1.1.2 意义	2
1.2 国内外发展现状	3
1.2.1 国外发展现状	3
1.2.2 国内发展现状	4
1.2.3 发展趋势	4
1.3 论文主要工作内容	5
1.3.1 工作目标	5
1.3.2 工作内容	6
2 相关技术介绍	7
2.1 Tensorflow ^[1]	7
2.2 Keras	7
2.3 边缘计算	8
2.4 MongoDB	9
2.5 EdgeX Foundry ^[2]	10
3 需求分析	12
3.1 用例分析	12
3.1.1 用户注册与登录	12
3.1.2 本地模型的选择与部署	13
3.1.3 本地模型的推理、训练与上传	15
3.1.4 全局模型的发布与迭代	17
3.2 系统功能性需求分析	18
3.2.1 账户的注册、修改	18

3.2.2	会话的建立	18
3.2.3	全局模型和本地模型的提交、修改、删除	19
3.2.4	全局模型和本地模型的查询和获取	19
3.2.5	本地模型的部署、推理、训练和上传	20
4	系统总体设计	21
4.1	组件划分与设计	21
4.1.1	端设备	22
4.1.2	边缘计算平台	22
4.1.3	云仓库	23
4.2	数据库设计	23
4.2.1	全局模型集合	23
4.2.2	本地模型集合	24
4.2.3	用户集合	25
4.2.4	会话集合	25
4.2.5	文件集合	26
4.2.6	事件集合	26
4.3	系统流程	27
4.4	数据流程	28
5	系统详细设计与实现	30
5.1	Model Cloud 组件	30
5.1.1	DAO 模块	31
5.1.2	Service 模块	32
5.1.3	APIServer 模块	33
5.2	EdgeX Foundry 组件	35
5.2.1	EMQXServer 模块	36
5.2.2	EdgeXClient 模块	37
5.2.3	EventCreator 模块	38
6	系统测试及功能验证	40
6.1	测试目的	40
6.2	测试内容	40
6.3	测试结果	41

7 结论与展望	42
7.1 结论.....	42
7.2 展望.....	43
参考文献	44
致 谢	47

第一章 绪论

1.1 背景和意义

每个研究的背后都有其时代背景和时代意义。抓住时代背景的研究能够充分利用好时代红利，使得研究能够得到更广泛的支持和认可；抓住时代意义的研究能够为时代做出真正有用的贡献，更有甚者，甚至能开创一个新的领域。

本小节介绍研究所处的时代背景和时代意义。

1.1.1 背景

当今正式通讯技术变革的时代。5G 技术的突破带来了增强移动宽带、超高可靠超低延时通信和海量机器类通信。

随着物联网^[3]的普及，同时也是万物互联^[4]时代的到来，越来越多的信息是由分布广泛而分散的移动设备和物联网设备产生的，这个数量甚至超过了大型云数据中心。Cisco 公司预测未来会有大量的无线设备接入网络，并带来大量的数据^[5]。比如说，大量生产的无人驾驶汽车^[6]和无人驾驶汽车生产的大量数据^[7]，波音飞机产生的大量数据^[8]，电动车产生的大量数据^[9]，监控摄像头产生的数据^[10]。根据爱立信公司的预测，到 2024 年，全球总共 40ZB 的互联网数据中将有百分之 45 的数据产生自物联网数据。如何将这么庞大的数据从边缘端卸载到云端会是一个很棘手的问题，因为这会引起极端的网络堵塞^[11,12]。

这些数据往往都是不需要长期存储的临时数据^[13]。因此，一种更实用的办法就是直接在边缘端处理用户的需求。这就是新计算范式——边缘计算^[14,15]的诞生过程。

边缘计算的话题横跨很多领域的很多概念和技术，比如说面向服务计算，软件定义网络，计算机架构。边缘计算范式的目的是促使提供服务 and 进行计算的计算资源和通讯资源从云端转移到边缘端，从而避免不必要的通信延迟并确保更快的用户反馈。边缘计算是当前一个很火的领域。

AI 技术在越来越广泛的领域发挥着越来越不容忽视的作用^[16]。大数据支持了许多对数据量和计算力需求极大的技术，深度学习就是其中之一。在过去

的十年里，AlexNet 和 DNNs 获得了巨大的成功。可以学习数据的深层表示的它们成为了最流行的机器学习架构。深度学习已经在近些年成为最受欢迎的 AI 策略，其代表是 DNNs 和它的分支们，比如说 CNNs、RNNs 和 GANs。深度学习在很广泛的领域里产生了突破，包括计算机视觉、语音识别、自然语言处理和棋盘游戏。

除此之外，硬件架构和平台以一个很快的速度进步，有望满足计算集中型深度学习模型的需求。特殊的加速器被设计出来，以完善吞吐率和功耗。总而言之，以深度学习的突破和硬件架构的升级为驱动力，AI 技术将获得持续的成功和发展。

随着边缘端数据的生产能力和消费能力的逐渐凸显，边缘计算应运而生。只有把边缘计算与 AI 技术相结合，才能充分利用边缘端的种种有利条件。将边缘计算和 AI 结合，这也就引出了边缘智能^[17]的概念。边缘智能成了一个新鲜的课题，横跨众多概念和技术，成为了最活跃的研究领域之一。

选择与边缘计算并不意味着必须完全抛开云计算。如今，云计算已经成为了很成熟的技术^[18]。业界有很多成熟的对云计算的支持，比如 google 文件系统^[19]，MapReduce 编程模型^[20]，分布式系统 Hadoop^[21]，分布式计算框架 Spark^[22]，等等。然而，传统云计算无法解决带宽和延迟的问题^[23]。边缘计算则很擅长处理这一点。

1.1.2 意义

结合边缘计算能极大缓解网络带宽与数据中心压力。普通的云 AI 平台要求用户将训练、推理所需的数据上传到数据中心。这些数据大多是临时数据，一经训练、推理就立刻被抛弃。这是一种巨大的浪费，然而碍于边缘端的计算力不足，不得不将数据发送到数据中心进行处理，但这有涉及到另一项阻碍，即网络带宽的匮乏。如果网络带宽不足，将数据发送到数据中心仍然是一种不恰当的选择。

如果选择结合边缘计算，那么只需要在边缘端部署一些服务器，就可以直接在离边缘端很近的地方处理用户的数据。这样一来既分担了数据中心的压力，也减轻了带宽的压力。

移动设备在计算、存储和电量等资源上的匮乏是其固有的缺陷，云计算可以为移动设备提供服务来弥补这些缺陷。但是，网络传输速度受限于通信技术的发展，复杂网络环境中更存在链接和路由不稳定等问题，这些因素造成的延

迟过高、抖动过强、数据传输速度过慢等问题严重影响云服务的响应能力。

而边缘计算在用户附近提供服务，近距离服务保障较低的网络延迟，简单的路由也减少网络的抖动。

同时，5G 时代正在到来，多样化的应用场景和差异化的服务需求，对 5G 网络在吞吐量、延迟、连接数目和可靠性等方面提出挑战。

边缘计算技术和 5G 技术相辅而成，边缘计算技术正以其本地化、近距离、低时延等特点，助力 5G 架构变革，而 5G 技术将是边缘计算系统降低数据传输延时、增强服务等响应性能的必要解决方案。

结合边缘计算能保护隐私数据，提升数据安全性：物联网应用中，数据的安全性一直是关键问题，调查显示约有百分之 78 的用户担心他们的物联网数据在未授权的情况下被第三方使用。云计算模式下所有的数据与应用都在数据中心，用户很难对数据的访问与使用进行细粒度的控制。

而边缘计算则为关键性隐私数据的存储与使用提供基础设施，将隐私数据的操作限制在防火墙内，提升数据的安全性。

1.2 国内外发展现状

研究和产品是统一的。对于研究来说，了解国内外发展状况，在其它研究的基础上进行研究能专注于自身的创新点和避免重复的操作；对产品来说，了解国内外发展现状，能够避开前人的错误，同时也能了解自身产品的发展前景。

云 AI 平台与边缘计算的结合研究已经被产品化过了，本节主要介绍国内外的云 AI 平台与边缘计算结合所诞生的产品。

1.2.1 国外发展现状

AWS IoT Greengrass^[24] 支持使用在云中创建、训练和优化的模型，并在设备本地执行机器学习推理。借助 IoT Greengrass，用户可以灵活地使用在 Amazon SageMaker 中训练的机器学习模型，或使用用户在 Amazon S3 中存储的已经预先训练好的模型。

机器学习使用根据现有数据所学习（该过程称为训练）的统计算法，以便对新数据做出决策（该过程称为推理）。在训练期间，将识别数据中的模式和关系，以建立模型。该模型让系统能够对之前从未遇到过的数据做出明智的决

策。优化模型过程中会压缩模型大小，以便快速运行。训练和优化机器学习模型需要大量计算资源，因此与云是天然良配。但是，推理需要的计算能力要少得多，并且往往在有新数据可用时实时完成。要想确保用户的 IoT 应用程序能够快速响应本地事件，则必须能够以非常低的延迟获得推理结果。

IoT Greengrass 为用户提供了两全其美的解决方案。用户可使用在云中构建、训练和优化的机器学习模型，并在设备上本地运行推理。例如，用户可在 SageMaker 中构建预测模型以用于场景检测分析，对其进行优化以便在任何摄像机上运行，然后部署该模型以便预测可疑活动并发送警报。在 IoT Greengrass 上运行推理过程所收集到的数据可发送回 SageMaker，然后就地标记，并用于不断提高机器学习模型的质量。

1.2.2 国内发展现状

Link IoT Edge 支持以边缘应用的方式，将云端的机器学习模型部署到边缘端，并在边缘端执行机器学习推理（ML Inference）。此功能非常适用于在边缘端处理实时性强、数据量大的服务（例如计算机视觉识别）。

用户可以在阿里云机器学习平台 PAI 或者任何其它地方训练用户的推理模型，把训练好的模型和相关代码托管在阿里云的函数计算、对象存储（Object Storage Service, OSS）或者容器镜像（Container Registry）等服务中，然后在 Link IoT Edge 边缘实例中以边缘应用的方式，将模型部署到网关。在网关上使用本地模型执行推理后把结果上传到阿里云物联网平台。

1.2.3 发展趋势

移动手机和平板电脑对于现在的许多人来说是基本的计算设备。在大多数情况下，这些设备很少离开它们的拥有者。这种丰富的人机交互和强大的传感器相结合，提供了数量庞大的私人数据。从这些数据上训练出来的模型一定更加智能，但是把这些数据存储在一个集中的位置也意味着风险和责任。

联邦学习^[25]提出了一种新的可能——它可将训练数据分散在移动设备上，并借助一个协调服务器聚合通过本地训练获得的更新。这是对 2012 年白宫报告中有关消费者数据隐私提出的重点收集或者说数据最小化的直接应用。未来，它们不会再包含除训练数据外的任何信息，并且这个量会越来越少。这种方式一个典型的优点就是使模型训练不再需要之间获取原始的训练数据了。

当然，对协调训练的服务器来说仍需要一定的信任，并且根据模型和算法

的具体情况，模型更新仍可能包含私人信息。然而，对于那些基于客户端数据来完成训练目标的应用，联邦学习通过将攻击对象限制在设备而不是设备和云来降低隐私和安全上的风险。

Google 早就做过了相关的尝试。它在用户手机上部署 AI 模型，将用户在手机上的输入内容用作模型的训练数据。这样一来就完成了符合个体用户输入习惯的 AI 模型，并通过预测用户的输入来提高效率。然而，个人的数据往往不够大，无法训练出更好的模型；然而出于用户隐私考虑，也不能将用户数据上传到属于 Google 的云计算中心。这就出现了一种折中，将 AI 模型上传到云计算中心，通过联邦学习得出准确率更高的模型，使所有人收益。

边缘端产生的数据由边缘端处理，而不是上传到云端处理，这暗合边缘计算的思想。

1.3 论文主要工作内容

显然，国内外最常见将边缘计算与云 AI 平台的结合方法是将 AI 模型存储于云端，在边缘端需要推理时将 AI 模型从云端部署到边缘端，从而避免了推理所需要数据的带宽占用和云端计算力负担。这让边缘计算的用户之地变得有限。

对于 AI 的代表技术深度学习，计算力和数据量的需求主要来自于训练和推理。这种结合方式减轻了源自推理的负担，却没有减轻源自训练的负担。如果训练数据产生于边缘端，那么在上述结合中，这些数据仍然要上传到云计算中心，占用大量的带宽和云计算力。

本论文将参考 Google 联邦学习的思想和实践，设计和开发一个系统，使得 AI 模型的训练和推理都能够直接在边缘端完成，不需要再上传到云端，同时充分利用分散各处的数据以保证 AI 模型的准确率。

1.3.1 工作目标

本论文将论述如何设计和实现一个系统。这个系统将利用与边缘计算的结合赋予云 AI 平台更多的优点。

在这个系统中，AI 模型的训练应当在边缘端完成训练的。

在这个系统中，众多边缘端的数据能够合力完成 AI 模型的训练。

1.3.2 工作内容

这里首先要引出两个概念，即全局模型和本地模型。

全局模型是同一功能领域的 AI 模型的一类通用模型。用户在开发个人 AI 模型的时候，可以选用这一通用模型作为初始模型，再用自己的个性化数据进行训练，从而得到更加个性化的 AI 模型。与此同时，用户训练好的个性化 AI 模型对于通用模型的优化有很大的作用，可以选择将自己的模型上传给通用模型的开发者，供其用于下一次通用模型的迭代。我们可以发现，个性化模型可以有很多，通用模型却只有一个并影响着所有的个性化模型，因此将通用模型称为全局模型，将个性化模型成为本地模型。这更有利于理解他们之间的关系。

本论文主要将设计和实现两个组件，即云仓库和边缘计算平台微服务。

云仓库将全局模型存储于云端，支持用户对全局模型的上传、下载等功能。

边缘计算平台微服务可以挂接在边缘计算平台上，代行本地模型的部署、预测、训练和上传，相当于边缘计算平台和云 AI 模型仓库之间的代理。

联邦学习的部分本应该是云端的职责，但为了降低系统的复杂度，将其作为一个独立的组件，可以部署在任意位置。

第二章 相关技术介绍

2.1 Tensorflow^[1]

TensorFlow 是一个端到端平台，无论是专家还是初学者，都可以轻松地构建和部署机器学习模型。TensorFlow 也是一个完整的生态系统，可以帮助使用机器学习解决棘手的现实问题。

TensorFlow 提供多个抽象级别，因此用户可以根据自己的需求选择合适的级别。用户可以使用高阶 Keras API 构建和训练模型，该 API 让用户能够轻松地开始使用 TensorFlow 和机器学习。如果用户需要更高的灵活性，则可以借助即刻执行环境进行快速迭代和直观的调试。对于大型机器学习训练任务，用户可以使用 Distribution Strategy API 在不同的硬件配置上进行分布式训练，而无需更改模型定义。

TensorFlow 始终提供直接的生产途径。不管是在服务器、边缘设备还是网络上，TensorFlow 都可以助用户轻松地训练和部署模型，无论用户使用何种语言或平台。如果用户需要完整的生产型机器学习流水线，请使用 TensorFlow Extended (TFX)。要在移动设备和边缘设备上推断，请使用 TensorFlow Lite。请使用 TensorFlow.js 在 JavaScript 环境中训练和部署模型。

TensorFlow 可以构建和训练先进的模型，并且不会降低速度或性能。借助 Keras Functional API 和 Model Subclassing API 等功能，TensorFlow 可以助用户灵活地创建复杂拓扑并实现相关控制。为了轻松地设计原型并快速进行调试，请使用即刻执行环境。TensorFlow 还支持强大的附加库和模型生态系统以供用户开展实验，包括 Ragged Tensors、TensorFlow Probability、Tensor2Tensor 和 BERT。

2.2 Keras

Keras 是为人类而不是机器设计的 API。Keras 遵循了减少认知负载的最佳实践：它提供了一致和简单的 api，它最小化了常见用例所需的用户操作数量，并提供了清晰和可操作的错误消息。它还有大量的文档和开发人员指南。

Keras 建立在 TensorFlow 2.0 之上，是一个行业实力框架，可以扩展到大型 GPU 集群或整个 TPU pod。这不仅是可能的，而且很容易。Keras 充分利用 TensorFlow 平台的部署能力。用户可以将 Keras 模型导出到 JavaScript 以直接在浏览器中运行，导出到 TF Lite 以在 iOS、Android 和嵌入式设备上运行。通过 web API 服务 Keras 模型也很容易。

Keras 是 Kaggle 排名前 5 的获奖团队中使用最多的深度学习框架。因为 Keras 让用户更容易进行新的实验，它让用户比竞争对手更快地尝试更多的想法。用户就是这样赢的。Keras 是 tightly connected TensorFlow 2.0 生态系统的核心部分，涵盖了机器学习工作流的每个步骤，从数据管理到超参数训练到部署解决方案。Keras 被 CERN, NASA, NIH 和世界上更多的科学组织使用。Keras 具有实现任意研究想法的低级灵活性，同时提供可选的高级便利功能以加快实验周期。

由于其易用性和对用户体验的关注，Keras 是许多大学课程选择的深度学习解决方案。它被广泛推荐为学习深度学习的最佳方法之一。

2.3 边缘计算

边缘计算是将计算放在边缘端进行的一种新型计算模式。边缘计算中的“边缘”是相对的概念，是指从数据源到云计算中心路径之间的任意计算、存储和网络资源。将这条路径上的资源看作是一个“连续统”，这条路径上的任意资源节点都可以是边缘。

边缘计算的核心理念是“计算应该更靠近数据的源头，可以更贴近用户”。边缘计算中的“边缘”与数据中心相对，这里“贴近”一词包含许多种含义。首先可以表示网络距离近，由于网络规模缩小，带宽、延迟、抖动等不稳定因素都易于控制与改进。还可以表示为空间距离近，这意味着边缘计算资源与用户处在同一个场景之中，根据这些情景信息可以为用户提供个性化的服务。空间距离与网络距离有时可能并没有关联，但应用可以根据自己的需要来选择合适的节点。

网络边缘的资源主要包括手机、个人电脑等用户终端，WiFi 接入点、蜂窝网络基站与路由器等基础设施，摄像头、机顶盒等嵌入式设备，Cloudlet 等小型计算中心等。这些资源数量众多，相互独立，分散在用户周围，将其称之为边缘节点。边缘计算就是要将空间距离或网络距离上与用户临近的这些独立分散的资源统一起来，为应用提供计算、存储和网络服务。

如果从仿生的角度来理解边缘计算，可以做这样的类比：云计算相当于人的大脑，边缘计算相当于人的神经末梢。当针刺到手时总是下意识地收手，然后大脑才会意识到针刺到手，因为将手收回地过程是由神经末梢直接处理地非条件反射。这种非条件反射加快人的反应速度，避免受到更大的伤害，同时让大脑专注于处理高级智慧。未来是万物互联的时代，不可能让云计算成为每个设备的“大脑”，而边缘计算则是让设备拥有自己的“大脑”。

2.4 MongoDB

本系统主要是用 MongoDB 作为数据库管理系统。MongoDB 是一种 NoSQL 数据库管理系统^[26]。这里的 NoSQL 指的是 Not Only SQL，不仅仅是关系型数据库。

本系统的数据均存储于 MongoDB 中，这为系统的未来集成提供了便利。如果未来要进行系统集成，可以直接将数据存储层剥离出来，作为新系统的数据基础。除此之外，MongoDB 还对数据聚合提供了原生的支持，也支持 MapReduce 等第三方组件。系统管理员可以通过 mongo shell 方便地了解系统状态和分析数据。

mongod 是 MongoDB 的主要守护进程。它处理数据请求、管理数据访问和执行后台管理操作。MongoDB 设计之初就是要支持分布式的。它对多个 mongod 进程共同运作系统给予原生的支持。通过分布式的设计，系统可以很方便的横向扩展其数据处理能力。mongod 进程是最基本的数据库管理进程，直接进行本地数据的访问。

MongoDB 支持副本集。MongoDB 中的副本集是一组包含相同数据的 mongod 进程。副本集提供冗余和高可用性，并且是所有生产环境的部署的基础。副本集中只有一个主节点支持读写操作，其他的节点仅仅可选的支持读操作。当副本集中有一个节点的数据意外丢失，就可以通过其他节点中的副本恢复数据。除此之外，当其它节点允许读操作时，也能一定程度上提高读写效率。

MongoDB 支持分片。分片是一种将数据分发到多个机器上的策略。MongoDB 使用分片来支持极大数据集和高流量操作。分片可以将一个数据请求分解成多个数据请求发送到多个机器上执行，并将多个机器的执行结果收集到一起返回给请求者。分片能够充分利用多个机器的存储和计算能力。相对于纵向扩展，横向扩展能以更低的成本提高数据访问效率。

2.5 EdgeX Foundry^[2]

EdgeX Foundry 是一个位于网络边缘的供应商中立的开源软件平台，它与设备、传感器、执行器和其他物联网对象的物理、日常工作世界进行交互。

其目的是为工业物联网边缘计算构建一个通用框架。EdgeX 平台能够并鼓励快速增长的物联网解决方案提供商在一个可互操作组件的生态系统中协同工作，以减少不确定性，加快上市时间，并促进规模化。

EdgeX 带来了这种迫切需要的互操作性，使监视物理世界项目、向它们发送指令、从中收集数据、将数据穿过雾移动到云中，在云中存储、聚合、分析数据，并将其转换为信息、驱动和操作变得更加容易。因此，EdgeX 使数据能够向北移动到云端，也可以横向移动到其他网关，或者返回到设备、传感器和执行器。

EdgeX Foundry 的有四个主要的为服务层，核心服务层、支持服务层、导出服务层和设备服务层。

核心服务层将边缘的北侧和南侧层分隔开，包含四个主要的组件。Core Data 负责从南侧对象收集的数据的持久性存储库和相关的管理服务。Command 促进和控制从北面到南面的驱动请求。Metaadata 负责有关连接到 EdgeX Foundry 的对象的元数据的存储库和相关管理服务。提供提供新设备并将其与所属设备服务配对的功能。Config and Registry 负责向其他 EdgeX Foundry 微服务提供有关 EdgeX Foundry 和微服务配置属性（即初始化值存储库）中关联服务的信息。

支持服务层包含广泛的微服务，提供边缘分析和智能，并为 EdgeX Foundry 本身提供服务。正常的软件应用程序任务，如日志记录、调度和数据清理由支持服务层的微服务执行。规则引擎、警报和通知微服务都在 SS 层中，因为它们在核心服务层之上运行。本地分析功能（今天作为一个简单的规则引擎实现）也位于这一层。

设备服务层的微服务通过每个物联网对象本机的协议与设备、传感器、执行器和其他物联网对象通信。DS 层将 IoT 对象生成和通信的数据转换为通用的 EdgeX Foundry 数据结构，并将转换后的数据发送到核心服务层，以及 EdgeX Foundry 其他层中的其他微服务。

导出服务层的微服务使非网关客户端能够订阅他们感兴趣来自端设备的数据，并通知数据的交付地点和时间 and 传递数据的格式和形状。

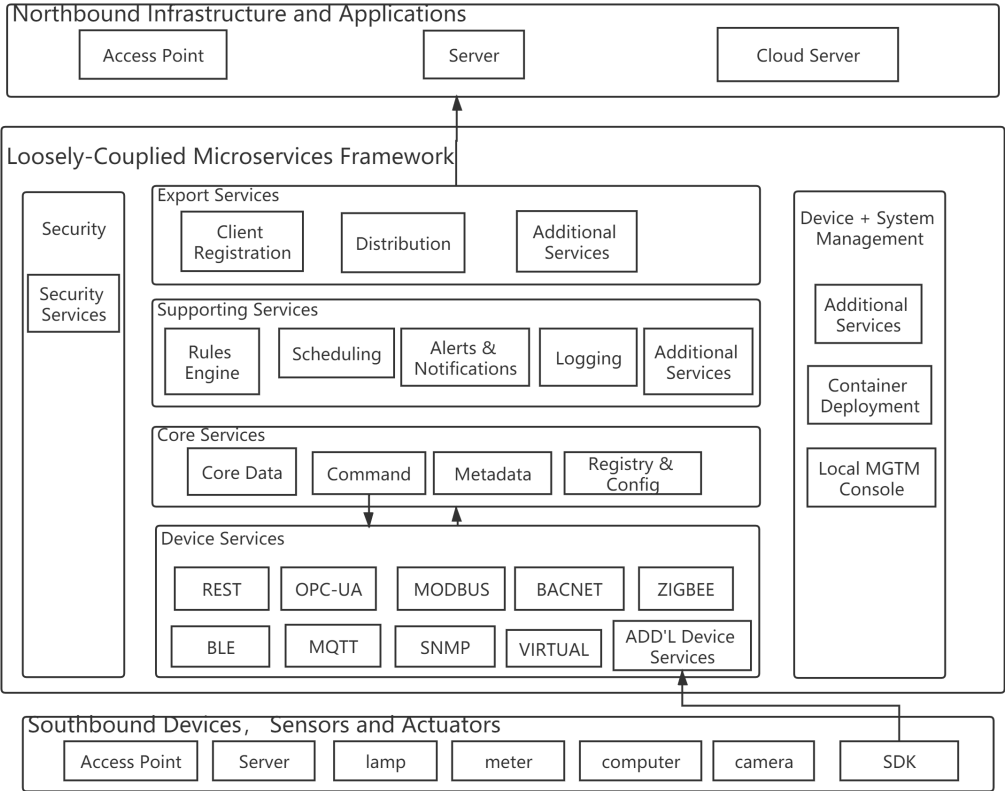


图 2-1: 架构图

第三章 需求分析

3.1 用例分析

在实际场景中，人在使用云 AI 系统会遇到以下几种场景，用户注册与登录，本地模型的选择与部署，本地模型的推理、训练与上传和全局模型的发布与迭代。正确的分析这些用例有助于获得系统需求。

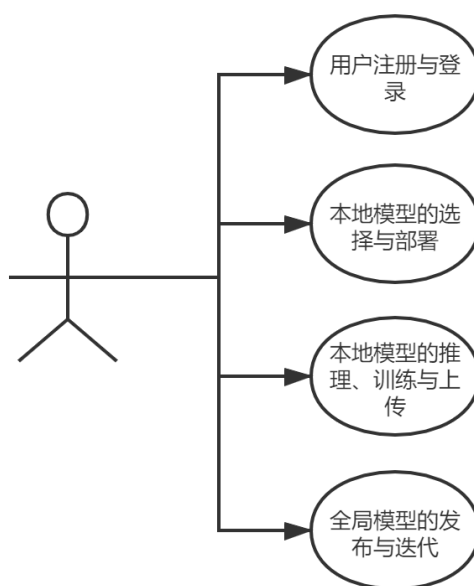


图 3-1: 全局用例图

3.1.1 用户注册与登录

用户身份在系统中起到重要的作用。用户身份是某个全局模型或本地模型的重要标签，其它用户可以根据这个标签对模型进行了解和筛选；用户身份是大多数操作的权限控制的依据，系统依照默认的或者用户定义的规则和做出操作者的身份来做出相应的反应；用户身份的引入也比较符合人的思维习惯，便于人性化的管理。

用户身份与众多的其它概念有着重要关系。用户身份与全局模型和本地模型之间有着拥有者和被拥有者的关系；用户身份与网络会话之间有着建立者与被建立者的关系。

用户注册时要提交许多属性。其中最关键的属性是密码，只有账号和密码一一对应才能证明用户的身份。其次还有很多附加属性，比如邮箱，如果未来有找回密码的功能时，可以根据邮箱发送找回密码的连接。在用户属性这一方面还有很多可以扩展的功能。

用户登录的本质是开启一个短期会话，会话有很多作用。用户在与系统交互的时候需要表征自己的身份，这就要求用户发送自己的账号密码来证明自己地身份。事实上，网络通信不是绝对安全的，而且每次都发送自己的账号密码既浪费通信资源，也不方便。这就体现除了会话的优势。只要建立好一个会话，之后只需要发送会话号码服务端就知道用户的身份了。与此同时服务器还可以在用户的会话上存储一些数据，以维持一个连续的交互过程。

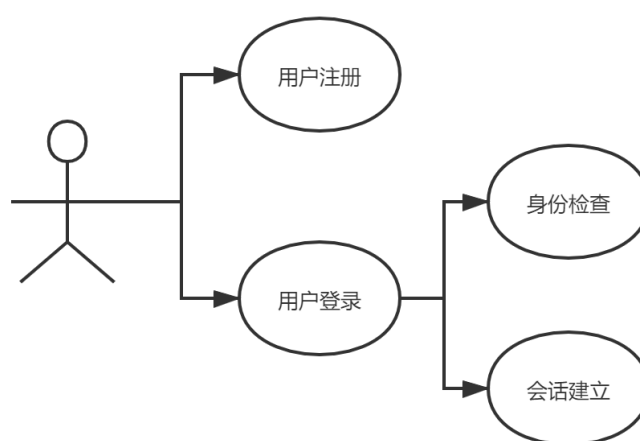


图 3-2: 用户用例图

3.1.2 本地模型的选择与部署

本地模型是边缘计算平台微服务的核心。无论是利用本地数据对本地模型进行训练，还是提交相关信息给本地模型进行预测，这前提都是要有一个符合用户需求的模型。然而本系统作为一个平台，并不会明确死那些模型可以上传到平台上供人下载。这在提高泛用性的同时，也为用户找到自己所需要的模型提高了难度。

用户在选择用作本地模型的全局模型时，需要参考本地模型的标签和解释。本地模型和全局模型从不同的角度去看有着不同的属性。

本地模型和全局模型本质是 AI 模型，因此拥有与 AI 模型结构相关的属性，比如 AI 模型所应用的技术、对输入数据和输出数据的规定；本地模型和全局模型是用户提交的数据，因此应该拥有于提交数据相关的属性，比如提交

的时间、是谁提交的；本地模型和全局模型在部署到本地模型和全局模型代理上时就是一个提供服务的功能实体，因此拥有与 AI 功能相关的属性，比如说该 AI 模型适用于什么场景、应当在什么条件下使用以及如何使用该 AI 模型。

给出这些标签和解释的目的不仅仅是为了方便用户找到适合自己的全局模型作为本地模型和全局模型的基础，更是为了更准确的对模型进行定义，以标准化的方式对它们进行管理。

用户在选择用作本地模型的全局模型时，需要使用多样化的搜索功能。无论本地模型和全局模型的属性定义的有多么的完善，用户也不是计算机，不可能自己从海量的信息中筛选出自身想要的模型。很多时候，用户甚至对自己想要的 AI 模型是什么样的都没有明确的概念。因此，系统应当代替用户思考用户可能会出于什么目的去搜索模型，这种搜索关注的属性有哪些，并据此设计出相应的搜索方法——不常用的搜索方式仍然是必要的，所以粗粒的搜索方法仍然要有，但不应该占据主要地位。

之前在设计全局模型和本地模型的属性时提到过依据观察的角度来分类讨论，设计搜索方法也能够利用这一方法。系统可以设计并列的几种面向领域的搜索方法，这些搜索方法之间可以取交集或并集，甚至异或和取反——然而用户不太可能使用这么复杂的搜索方法，多数情况下最多也只是用到交集。这些领域包含 AI 技术领域，提交方式领域和功能实体领域。选择性的组合这三个领域的搜索能够帮助用户高效地找出自己需要地模型。

另外，现实场景中有很多异质的数据，它们需要得到不同的配合。系统中的全局模型和本地模型并没有规定 AI 技术只能用哪一种。而且，大部分用户并不是技术专家，像是部署 AI 模型这种工作他们做不了同时也不想做，这就需要系统代用户对选择好的模型进行部署。

用户部署所获得的本地模型时，需要自动启用相关的运行环境。虽然工业界已经有了很多成熟 AI 框架，但是为保障少数人利益，系统仍然要对一些不那么常见但是仍然有不少人使用的 AI 框架进行适配。使用何种框架可以从两个角度判断，一个是由用户在 AI 模型的属性上定义自己所使用的框架并由系统查看，另一个是检查用户所选择模型下载下来后的文件格式或者连带的表征格式的注解。这两种都是可以的，但如果两个都采用、检查两遍那就更完美了。

第一次进行模型部署时会消耗比较多的通信资源。但是在第二次以后应该通过缓存来缓解这一负担。

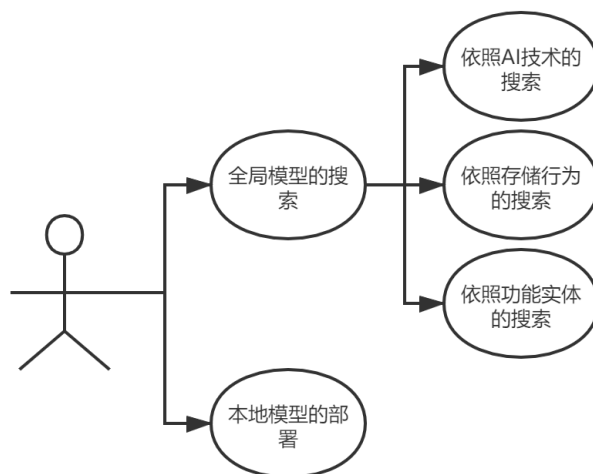


图 3-3: 模型部署用例图

3.1.3 本地模型的推理、训练与上传

对于端设备的使用者来说，能够免费使用 AI 技术，创建符合自己使用场景的 AI 模型并借此进行推理就是最大的诉求了。他们理想中的场景可以用三个词形容，“免费”、“AI 推理”和“无门槛”。因此可以说本地模型的推理和训练对于他们来说是最重要的功能。

对于全局模型的发布者来说，能够免费使用用户数据训练自己的模型，同时避免触犯隐私法，这就是他们最大的诉求了。然而这一诉求的前提就是训练好的本地模型需要上传到云端上，发布者才能获取并利用它们训练更优秀的模型。因此可以说本地模型的上传对于他们来说是最重要的功能。

推理和训练在资源占用上有着对抗的关系。从众多的实际场景中我们可以知道，边缘计算的优点之一就是延迟低，这在 AI 模型的使用这个场景中指的是推理的低延迟而不是训练的低延迟。实践上传统的结合边缘计算的云 AI 平台是要将边缘端产生的训练数据上传到云端进行训练的，这势必造成比本设计多得多的延迟和通信量。可见推理和训练的处理优先级的差距之大。

更多的场景里，推理请求是突发请求，伴随小批量的推理数据；训练请求是常规请求，伴随着大批量的训练数据。从 AI 技术的角度上看，推理所消耗的资源一般比训练消耗的资源小的多，尤其是 AI 技术的代表深度学习。因此，在推理和训练竞争资源的时候，应该让推理居于优先地位。

推理和训练在功能上也是相辅相成的。全局模型的发布者可能会担心，本地模型的使用者如果仅仅使用全局模型进行推理，而不将个人的数据投入训练

该怎么办呢？这里误会了一点，推理和训练并不是两个独立的过程。从整个时空上看，有推理数据就必然有对应的训练数据。AI 技术的原理是从历史数据中发现某些数据之间的某些关系，进而利用这些关系，在只有部分数据的条件下推理出全部的数据。用户使用 AI 技术的推理功能的原因在于用户之掌握了残缺的信息，却希望推出补充的信息，并利用补充的信息达成一些目的。当用户使用补充的信息试图去达成那些目的的时候，势必大体会带来两个结果，成功或失败。如果成功了，证明预测和现实相符；如果失败了，证明预测和现实相悖。这本身就是一种可以用于训练的数据，只不过产生的时间比起推理要延后。同时，利用实际结果对本地模型进行个性化的训练也符合用户的利益。因此，可以说推理和训练会伴生存在。

本地模型的上传是全球模型发布者的诉求，但在数量超过诉求时是要经过筛选才能上传或者被发布者选作新一代全局模型的训练数据的。本地模型的上传最终是为联邦学习服务的。

Google 公司曾在自己的手机用户上做过联邦学习的实践。这场实践中有众多实验者参与其中。他们在不上传本地键盘输入数据到云端的条件下，完成了键盘输入预测的 AI 模型开发。在这场实践中也遇到了很多的问题。比如说本地模型的训练会增大手机的负载，于是规定只有空闲且处于充电状态的手机参与联邦学习；比如不同人群的输入习惯和输入数据数量展示出严重的不平衡，这在统计学中称为不平衡、非独立同分布的模型，这一点仍有待更好的解决；比如网络的延迟会让一些本地模型的上传失败，于是将之抛弃。

全局模型的开发有着更高的门槛，所以全局模型和本地模型相比必然处于少数的地位。如果在未来系统能得到充分的发展，本地模型的供应量必然高于需求量，如何选择出高质量的本地模型就成了问题。

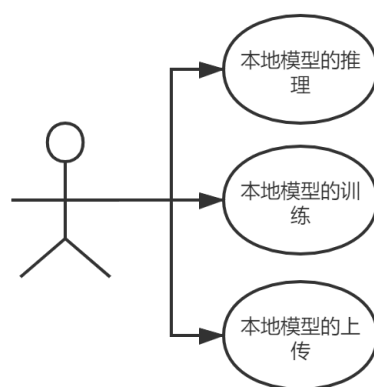


图 3-4: 模型服务用例图

3.1.4 全局模型的发布与迭代

AI 模型的开发者只有先称为全局模型的发布者，即将自己的 AI 模型以全局模型的身份发布到平台上，才可能收集到经过用户数据训练后的本地模型。全局模型的发布是整个联邦学习过程的起点。只有发布做好了，建立起云端与边缘端的桥梁，才能够完成所有参与者的互利互惠。

全局模型和本地模型有两个重要的身份。其一是它们在本系统中的身份，是作为一个数据单元的概念存在的。当用户登入本系统，他们对全局模型和本地模型的理解是系统定义的两类资源。系统的定义忠实地体现资源地抽象含义。其二是它们在现实应用中的身份，是作为一个功能单元的概念存在。用户在真正从平台获得全局模型和本地模型时，获得的不是抽象的描述，而是随时可以启用 AI 模型，是可以完成推理和训练任务的工具。其存在的意义、使用的方法完全由用户来决定。

因此系统将全局模型的发布分为两个部分，一个是全系统通用的对全局模型和本地模型的抽象描述，一个是功能完备、形式自定的 AI 模型实体文件。用户可以先提交模型的抽象描述，在系统中建立了相应的实例，再将相应的实体文件挂接在实例上。用户也可以根据模型的抽象描述获得初步的了解，再将模型下载下来投入实际的使用。

另外，全局模型的发布未必是从无到有的发布，也可能是在原有全局模型的基础上开发出更优秀的模型，以一个更新的版本发布。这就是全局模型的迭代。

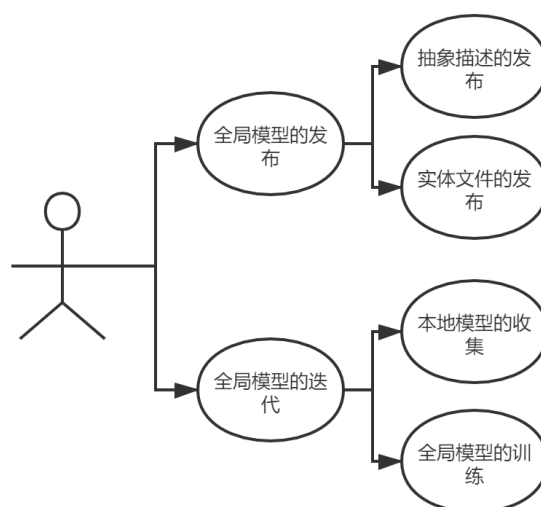


图 3-5: 模型迭代用例图

3.2 系统功能性需求分析

3.2.1 账户的注册、修改

系统接受用户提交的账户信息，并建立一个新的账号，并将账号的登录方式返回给用户。

用户应当提交一些关键性的信息作为登录的依据，而系统通过这些关键性的信息作为判断请求登录者的身份。在本系统中，用户将设置密码，作为对用户身份的验证。而用户身份的表征则可以利用 id 和邮箱两种方式。

使用邮箱作为表征用户身份的方式有三点好处。其一，邮箱一般为个人所有，因此不容易冲突。其二，邮箱是用户日常使用的数据，不容易遗忘，且避免了起名字的烦恼，用户体验极佳。其三，邮箱可以帮助密码的找回，在请求修改密码时只需要向用户邮箱发送一条确认邮件就能确认用户的身份。

然而用户的邮箱不是一成不变的，如果用户出于某些目的而修改了邮箱，系统数据容易出现前后不一致的情况。这时候就必须有系统直接分配用户 id，不允许改变。因此，id 作为表征用户身份的终极手段，邮箱作为表征用户身份的辅助手段。

用户应当提交一些个性化的信息作为身份的塑造。在本系统中，用户不仅是操作系统的人，还是与其他人交流的人。人是一种社会动物，适当的公开一些个性化的信息塑造形象能够加快人与人之间相互的了解。本系统本质上服务于促进本地模型使用者和全局模型发布者的互利互惠事业。这一切都要建立在两者的互相了解和互相信任上。

系统接受用户提交的新的账户信息，对指定的账号进行修改。

在用户信息的修改中，用于身份检验的关键信息的修改之前应当先检验用户身份，确认是本人操作。其中的邮箱可以被修改，但是 id 不能被修改。如果密码遗忘可以通过邮箱找回。

3.2.2 会话的建立

系统接受用户提交的账号和密码，并建立一个新的会话，并将会话 id 返回给用户。

系统通过接受和解析用户发送的会话 id，即可确定用户身份。这样以来用户就不需要每次提交请求都需要带上用于验证用户身份的账号和密码了。这降

低了操作的复杂性，提高了账号和密码的安全性。

系统允许用户在会话存储一些变量，用于维持连续的操作。用户的一些大的操作是由许多小操作一起组合而成的，小操作之间有着共同的联系。

3.2.3 全局模型和本地模型的提交、修改、删除

这三类操作都是是读写操作中的写操作，会在第一次或所有次数的请求中对系统中的数据造成改动。这使得三者具有一些共同点。

在操作前进行权限检查，只允许模型的所有者修改模型。模型的创建者也是所有者，拥有着对模型的最高控制权限。并且，模型的拥有者也可以管理是否允许其他人访问模型，以何种方式、何种条件去访问模型。

不允许进行对模型的修改，而是以创新的全局模型或者在原有全局模型的基础上创建新版本模型作为代替。全局模型是众多本地模型的基础，如果要修改的话，牵扯的范围太过广泛，不如创建一个新的全局模型。

这三类操作的对象有两类，一类是抽象的属性，一类是实体文件。

模型的属性包含三个方面，AI 技术相关属性，数据单位相关的属性，功能实体相关的属性。AI 技术相关属性定义了模型所使用的 AI 技术的类型、解释、运行环境等信息。数据单位相关属性定义了数据单位的发生时间、执行者、id、备注等信息。功能实体相关属性定义了功能实体的输入数据格式、输出数据格式、功能解释、训练参数等信息。这里只是列出了一些，之后要结合领域知识进行详细设计。

实体文件要满足 AI 模型的功能要求，主要包含推理、训练。功能实体在接收到合法的推理数据时，应当能依据一定的规则，给出以推理数据为基础的推断。如果能知道实际结果，应当能和模型的推断结合，得出误差，从而优化模型的规则，即对模型进行训练。

3.2.4 全局模型和本地模型的查询和获取

模型的查询和模型的提交是相对的需求。

从三个方面支持查询，分别是 AI 技术相关属性，数据单位相关属性，功能实体相关属性。

在 AI 技术相关属性的查询中，用户主要是对 AI 技术的类型进行查询。AI 技术往往会比较擅长一些领域，不擅长另一些领域。用户依据其对 AI 技术的理解，依据 AI 技术和实际需求的契合程度完成对 AI 技术的选型，将之作为一

个查询条件，从而能够找出更适合的模型。

在对数据单位相关属性的查询中，有些用户会关注模型的所有者是谁，根据对所有者的态度决定对模型的态度，从而选择自己需要的模型。有些用户会关注模型的提交时间和版本号，因为一般更新出现的模型、更新版本的模型会比早就出现的、旧版本的模型更优秀；不过相对的，有些用户会反其道而行，以追求模型的稳定性。

在对功能实体相关属性的查询中，功能为用户最关注的点。用户因为想使用 AI 模型的功能才在云仓库中查询自己所需要的模型。

模糊查询模式能够帮助用户满足不太明确的需求。模糊查询的“模糊”可以体现在很多方面。比如语义的模糊，当用户要求查询包含某个单词的模型时，系统也返回包含用户该单词的同义词的模型。比如领域的模糊，同一个模型的数据有很多种类的属性，不需要指定是哪一个属性，直接在所有属性的值中进行查询。

多个查询结果进行聚合完成高级查询。系统提供了很多中查询方法，这些方法可以以集合的方法聚合在一起。比如将依据 AI 技术相关属性的查询结果与依据数据单位相关属性的查询结果求交集。这样一来就获得了符合两套查询条件的查询结果。

3.2.5 本地模型的部署、推理、训练和上传

系统接受用户对指定全局模型的部署请求，将全局模型文件下载到本地并部署。

系统接受用户对本地模型提交的推理数据，使用本地模型进行推理，返回推理结果。

系统接受用户对本地模型提交的训练数据，使用本地模型进行训练。

系统接受用户对本地模型的上传请求，将本地模型上传到云端。

第四章 系统总体设计

4.1 组件划分与设计

该图交代了各组件的部署。系统中有三个主要的部署位置，端设备，边缘计算平台，云服务器。

端设备中部署了 User App。用户应用由于部署在端设备上，所占有的资源比较匮乏，如果有比较消耗资源的计算任务，比如 AI 模型的推理和训练，尤其是训练，就很难完成任务。这时候可以将计算任务提交到最近的或者最空闲的边缘计算平台上的边缘计算平台上的服务来处理。

边缘计算平台部署了 EdgeX Foundry 和 EdgeX Client。EdgeX Foundry 是沟通端设备的基础。由于端设备的种类繁多，支持的协议也五花八门。而边缘计算平台就使用对应的设备服务和它们进行交互，并将收集到的数据转化成统一的格式，存储于平台上，或者转发给注册在平台上的客户端。EdgeX Client 是注册在边缘计算平台上的一个微服务。这个代理向下间接接收端设备的请求和数据，向上于云仓库建立连接，对仓库中的全局模型和相应本地模型进行数据访问。它在边缘计算平台上充分利用本地的计算资源进行 AI 模型的推理和训练，充分发挥了边缘计算的优势。

云服务器部署了 Cloud Warehouse 组件，它由四个模块组成，分别是 Database, DAO, Service, APIServer。这四个模块以分层架构组成一个整体，为云仓库。

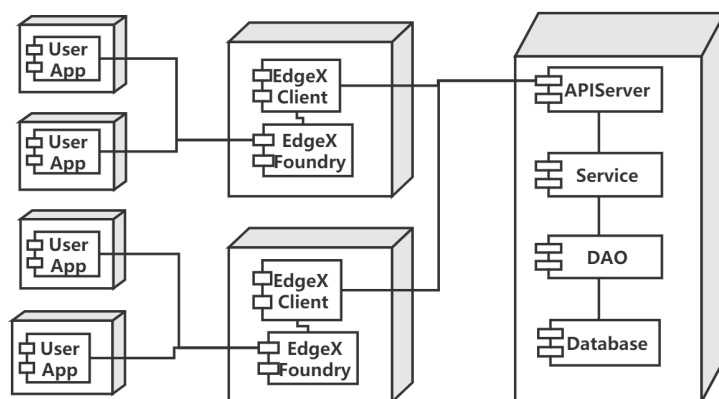


图 4-1: 部署图

4.1.1 端设备

认识一个端设备可以从三个角度，分别是如何获取信息，如何处理信息，如何发送信息。近年来，物联网技术逐渐成熟，在数据的输入、处理和输出领域中出现了三种关键技术。

端设备通过传感器技术获取外部信息。这是一种从自然信息源中获取信息，并将之转化成可以识别和处理的信号的技术。与普通的个人计算机所不同的是，这种数据输入的方式一般不是由网络或者键盘一类的输入设备输入的，它常常通过一些能感知物理特征的感知器来获取信号，并将这些信号转化成通用的数字化信号。当这些感知信号转化成数字化信号并将之存储时会出现一个矛盾。那就是所获得数据量的庞大和处理数据能力的匮乏。

端设备通过嵌入式系统技术处理数据。端设备也可以有计算机系统。但是相对于个人计算机上的功能强大的通用操作系统，端设备上的嵌入式系统更像是功能不全的专用操作系统。端设备往往计算能力弱、存储能力小，而且电量也经常是有限的。嵌入式系统在部署在端设备上时要考虑如何充分利用设备的计算能力、存储能力和电力，给出独一无二的方案。即使如此，嵌入式系统仍旧是五脏俱全的操作系统，能够胜任处理数据的任务。

端设备通过无线射频技术完成数据的传输。端设备可以通过发送射频信号自动获取彼此的身份。

结合这三项技术，端设备接入互联网，走入物联网的时代。

4.1.2 边缘计算平台

边缘计算平台使用不同协议连接端设备。端设备往往处于硬件条件匮乏的状态。受限与此，端设备会采用五花八门的通信协议。通信协议的复杂性导致不同类型端设备与普通的服务器不能通过互联网直接交流，这时候就需要边缘计算平台与端设备先行连接，并将从端设备上收集到的数据转化成其它通用设备认可的统一模式，间接搭建其两者之间的桥梁。同时，边缘计算平台也能够对从此处经过的数据流进行分析和控制。

边缘计算平台以事件的形式收集端设备的数据。事件是对端设备所执行的动作的一种抽象。一个事件包含两类属性，一类是事件的触发属性，一类是事件的内容属性。事件的触发属性解释哪个设备触发了事件、在什么时间出发了事件等问题。事件的内容属性解释了触发的事件是什么的问题。边缘计算平台

会将从端设备收集到的数据进行事件格式的统一形式。在转发时，也是转发的事件而非原始数据。

边缘计算平台将收集到的数据转发给在平台上注册的微服务。边缘计算平台本身是一个平台，并直接实现各种功能上的扩展服务。这就需要在平台上注册丰富的微服务。通过平台转发数据到边缘计算平台微服务上来完成功能的实现。

边缘计算平台微服务接受并处理来自边缘计算平台的转发数据。这种转发数据是以事件的格式发送的，同时使用最常见、最统一的网络通讯协议发送。当微服务收到数据时可以立即处理，也可先将数据保存下来选择一个恰当的时间点进行处理。

边缘计算平台微服务完成对本地模型的各项操作。微服务上部署了本地模型。它可以接受用户关于本地模型操作的种种请求，比如推理和训练。微服务将以合理的顺序尽可能地完成这些计算任务。

边缘计算平台微服务与云仓库建立数据连接。微服务建立本地模型的基础全局模型仍旧需要从云端获得，所以必须要与云仓库建立连接。然而，由于结合了边缘计算，系统的大部分计算任务都能在本地完成，因此大大减少了与云端的通信次数。

4.1.3 云仓库

云仓库接受全局模型和对应本地模型的上传和下载。云仓库的根本职责就是提供模型的存储和获取。用户可以对云仓库中地数据自由地进行符合权限的操作。

4.2 数据库设计

本小节介绍关键数据模型结构。

4.2.1 全局模型集合

数据集合的设计分三个方面，AI 技术方面，数据单位方面和功能实体方面。全局模型实体的属性在 AI 技术方面上面设置了一个 `type`，用于判断 AI 模型的技术类型；在数据单位方面设置了 `name`、`permission` 等属性，用于表征全局模型在云仓库中数据存储单位的身份；在功能实体方面设置了 `labels`、

description 等属性，可以由用户定义功能的具体细节。

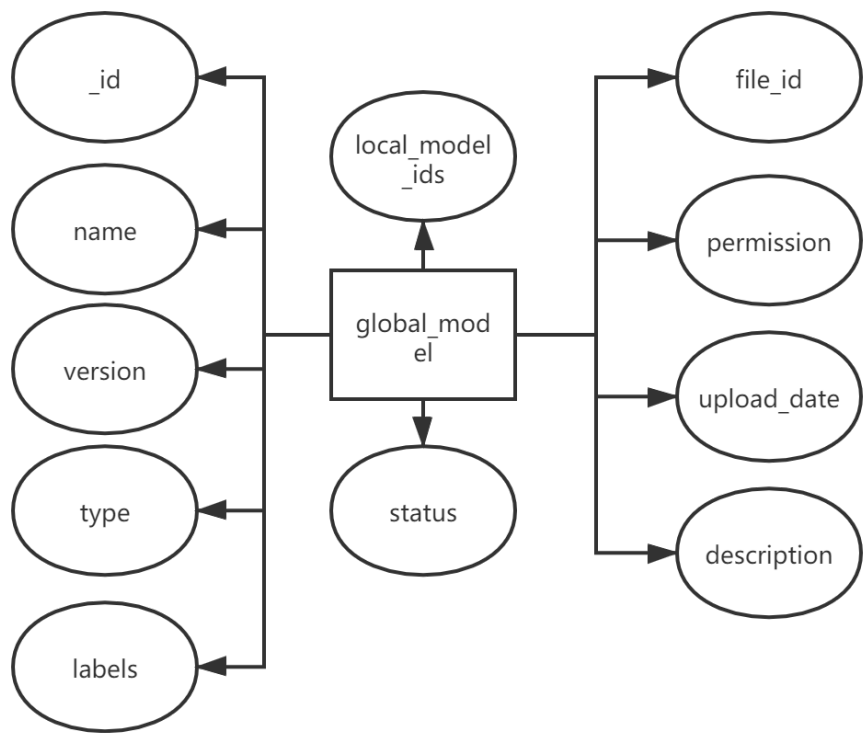


图 4-2: 全局模型实体图

4.2.2 本地模型集合

本地模型的数量相对全局模型格外庞大，同时有不少属性是继承全局模型的。因此本地模型的属性应当尽可能地少，添加 update date 作为基本地信息，更多地集中于 message 提交备注和可以自行定义地 labels 这样比较自由灵活的属性上。

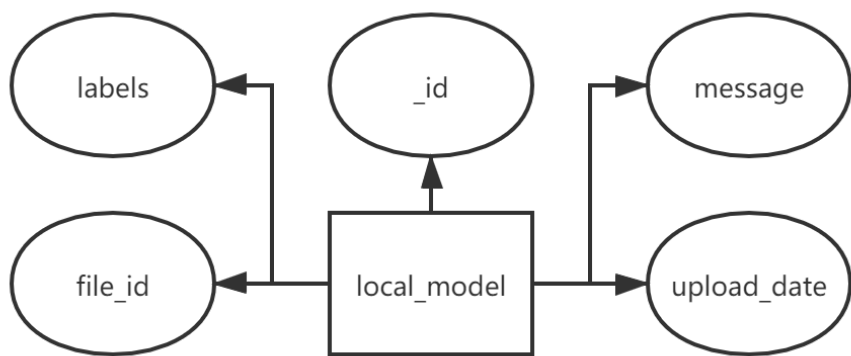


图 4-3: 本地模型实体图

4.2.3 用户集合

用户集合关注于身份验证、形象塑造和与其它数据单元的关系上。在身份验证上，设置了 `password`，用于在登录和修改用户关键信息时确保是本人在操作。在形象塑造上设置了 `name`，用户可以塑造自己的形象，从而流畅地与其他用户交流。在与其它数据单元地关系上设置了 `global model ids` 和 `local model ids` 用于表示用户与全局模型和本地模型间的所属关系。

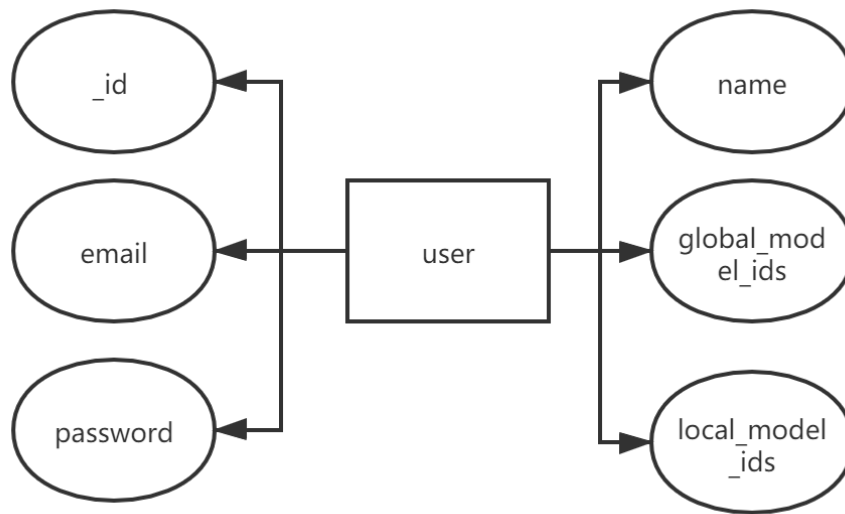


图 4-4: 用户实体图

4.2.4 会话集合

会话是对用户和系统之间交互的抽象。其中，`user id` 用来表征用户身份。使用 `attributes` 灵活存储一些会话中可能短期持久使用的数据。

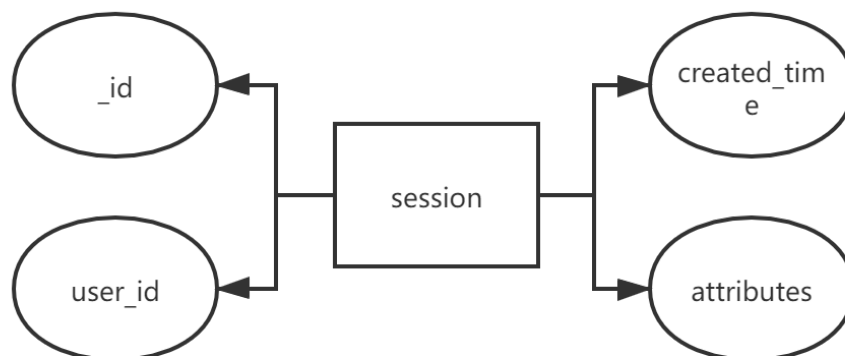


图 4-5: 会话实体图

4.2.5 文件集合

主要表现形式为能够完备地实现 AI 模型所有功能的数据集合，在本系统中主要指人工神经网络高阶应用程序接口 `keras` 所导出的层次数据格式的 `h5` 文件。

在 MongoDB 中，文件存储采用 Grid FS，它将文件存储在两个集合中，`chunks` 和 `files`。其中 `chunks` 存储文件分裂出的二进制块，`files` 存储文件的元数据。这两个集合都有数据库管理系统原始的定义，因此不需要也不建议用户自定义数据字段。但是，用户可以通过 `gridfs` 接口来指明一些数据字段。

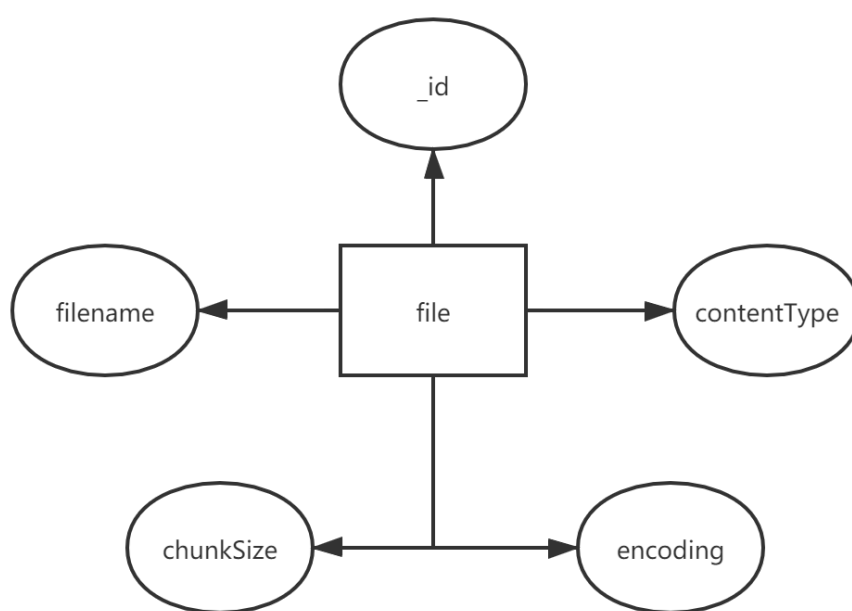


图 4-6: 文件实体图

4.2.6 事件集合

事件是端设备用于向边缘计算平台提供数据的一种行式。这里的事件是端设备所触发的事件。

因为事件是由设备触发的，因此事件的属性中一定包含有端设备名。除此之外还可以有触发事件的事件、环境和位置等信息，这里做了简化，同时也是为了配合边缘计算平台所要求的规范，这里只包含端设备名。

除此之外，事件多包含的最重要的信息就是事件的属性。由于端设备的功能比较简单，并且信息主要来源于对自然世界的感知，因此可以用多个属性来度量事件的属性。

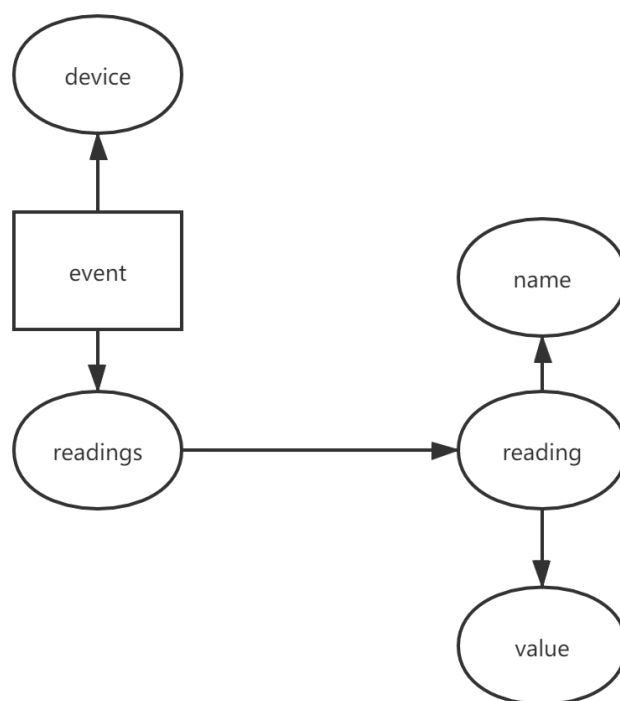


图 4-7: 事件实体图

4.3 系统流程

该时序图交代了完成一次联邦学习过程中组件之间的交互。

图中有三类组件，分别是联邦学习组件、边缘计算平台微服务组件和云仓库组件。

第一步，联邦学习组件向全局模型仓库提交了原始的全局模型。该 AI 模型定义好了结构并且进行了初步的训练，其功能更倾向于通用性。

第二步，边缘计算平台微服务从全局模型仓库处下载和部署了本地模型。AI 模型从云端转移到了边缘端，体现了边缘计算的思想。

第三步，边缘计算平台微服务训练和向全局模型仓库上传了本地模型。AI 模型在本地利用用户数据进行训练。用户数据在用于训练后便完成了使命。而训练好的本地 AI 模型如果符合条件且拥有着有意愿的话可以提交到云仓库，用于新全局模型的训练。

第四步，联邦学习组件从全局模型仓库下载了所需的本地模型，通过联邦学习训练出了新的模型，并再次提交到全局模型仓库。借由云仓库的优势，所需的本地模型可以下载到网络上的任意位置，完成训练任务后可以方便地上传回平台，完成一次迭代。

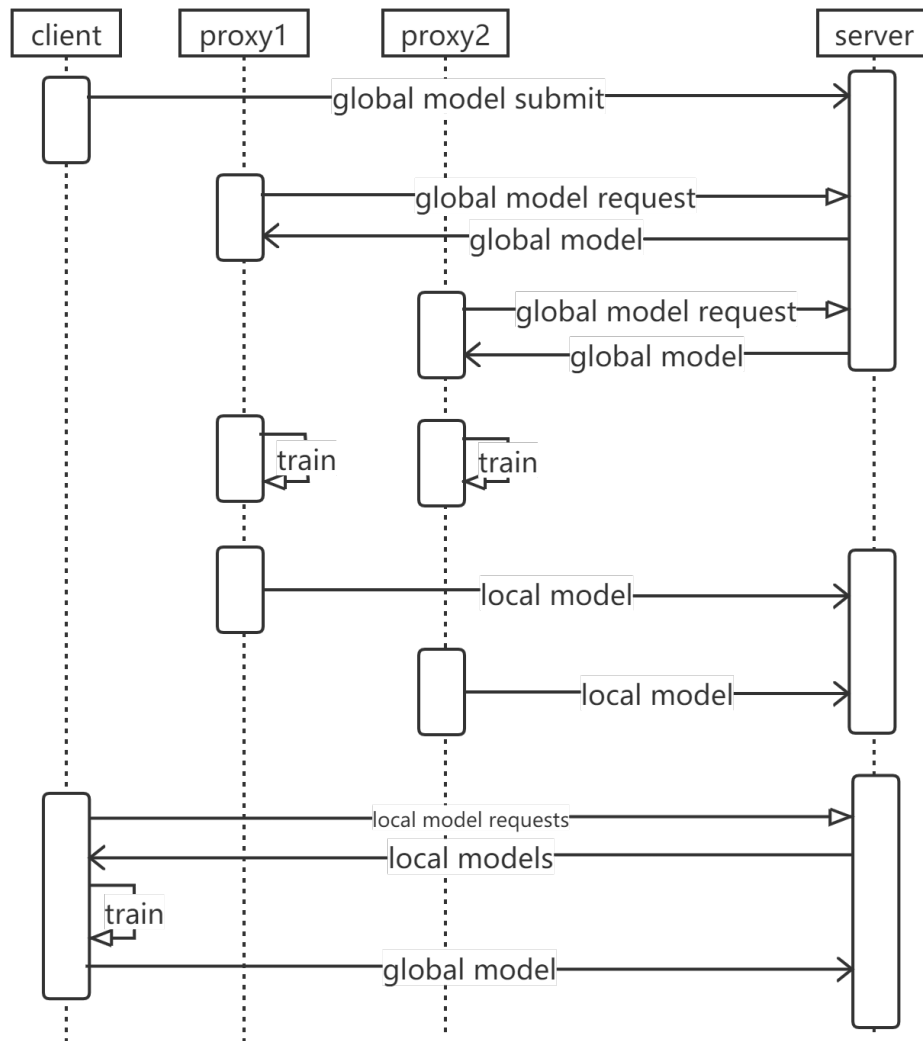


图 4-8: 时序图

4.4 数据流程

该数据流图交代了在一次联邦学习过程中数据的转化。

图中有五类数据，分别是未经训练的本地模型、经过训练的本地模型、未经训练的全局模型、经过训练的全局模型和用户数据。

第一步，从云数据库中取出了未经训练的全局模型。此时，全局模型的查找主要依据于全局模型的属性进行查找，而全局模型的获取则是以实体文件的形式获取的。

第二步，未经训练的全局模型被下载和部署到了边缘处，转化成了未经训练的本地模型。通过建立本地模型的实例，定义属性和挂接实体文件，来完成从全局模型到本地模型的转化。

第三步，未经训练的本地模型利用来自于边缘数据库中的用户数据进行训练，转化为了经过训练的本地模型。此时的本地模型根据训练数据对 AI 模型中各种变量和参数进行了调整。模型更加成熟和具有个性化。

第四步，众多经过训练的本地模型上传到云端，经过联邦学习，转化为一个经过训练的全局模型。这里会获取本地模型 AI 模型中的关键变量，根据这些数据对全局模型 AI 模型进行训练，在不需要直接接触训练数据的条件下，完成全局模型的优化和迭代。

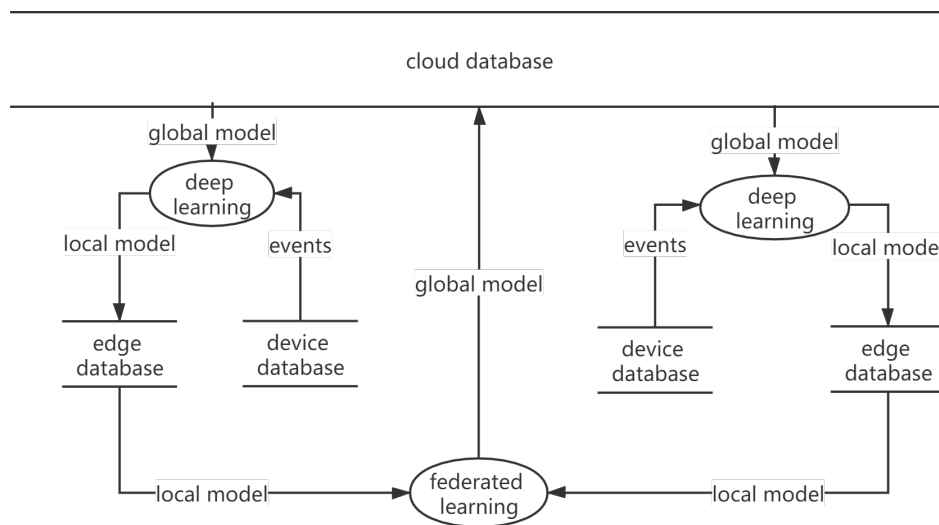


图 4-9: 数据流图

第五章 系统详细设计与实现

5.1 Model Cloud 组件

Model Cloud 组件提供全局模型和本地模型的存取，并提供权限控制。

Model Cloud 组件采用分层架构，自下而上包含数据访问层、业务逻辑层和用户接口层，分别对应 DAO 模块、Service 模块和 API Server 模块。层与层之间允许隔层交互。

Model Cloud 组件和 EdgeXClient 组件直接共同完成了云和边的协同。

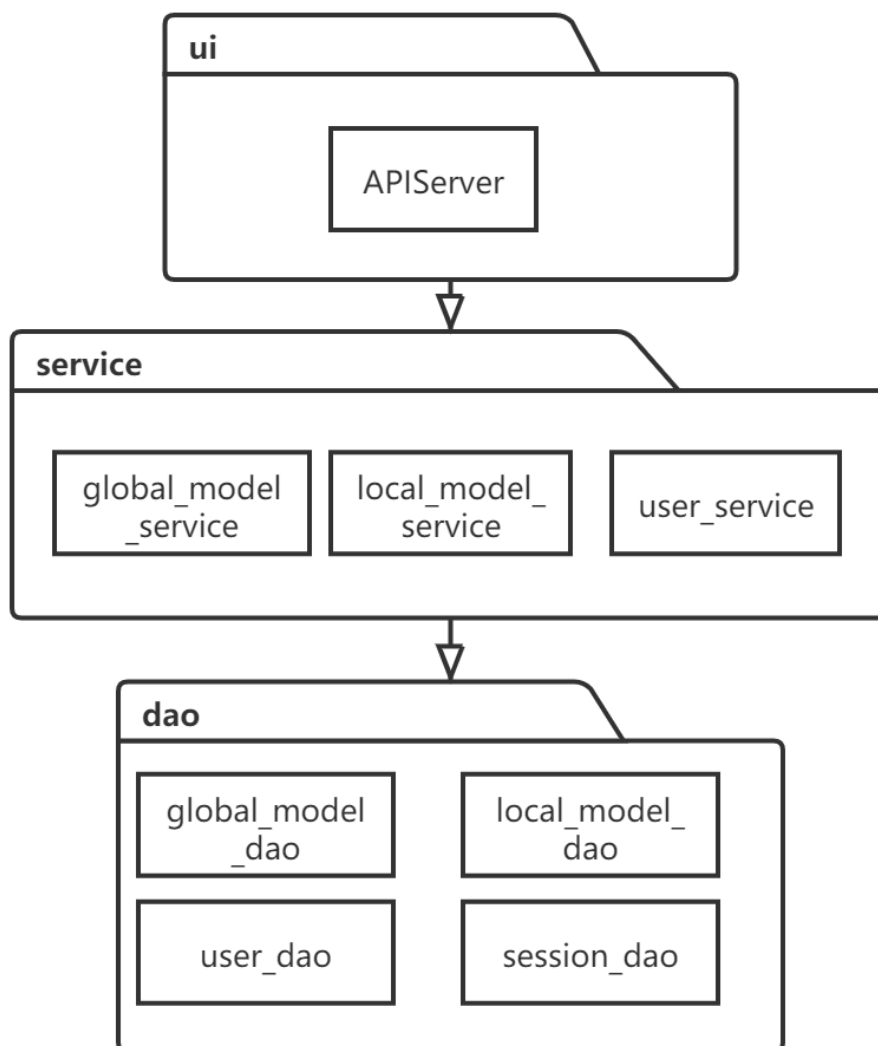


图 5-1: 包图

5.1.1 DAO 模块

数据访问层的作用是将数据库操作封装起来，对下层通过调用数据库驱动程序来从数据存储层获取数据，对上层通过设计良好的接口支持业务逻辑层，本质是一种对数据存储和业务逻辑的屏蔽。

这种屏蔽不仅仅是简单的隐藏细节，提高编码效率。这种屏蔽更源自于这两层对灵活性和安全性的重视程度的矛盾。

数据存储重视安全性，轻视灵活性；而业务逻辑则重视灵活性，轻视安全性。因此数据访问层要为数据存储层的安全性负责，限制对数据存储的自由操作；又要为业务逻辑的灵活性负责，定制多样化的、功能完备的数据接口。

数据库访问层的接口大体包含增、删、改、查，接口将在这四个大方向上予以设计。下面简要解释这些操作在本系统中的定义。

插入是提交文档，添加到指定集合中。若文档中含有原集合中存在的主键，则插入失败，若文档中不包含主键，则由数据库管理系统自动生成一个。

删除是提交查询条件，删除符合查询条件的文档。若无符合条件的文档，则视为成功，但为无效操作。

更新是提交一个查询条件和一个修改描述，依据修改描述修改符合查询条件的文档。

查询是提交一个查找条件，返回符合查询条件的文档。若无符合条件的文档，则视为成功，但返回空集。

数据访问模块。数据访问层的功能来自于数据库驱动程序。因此驱动程序所提供的接口能提供完备的数据访问功能。

但是，对于业务逻辑层来说，简单的使用驱动程序所提供的接口既不安全也不便捷。因此驱动程序所提供的接口不应该直接作为数据访问层的接口暴露给业务逻辑层，但又迫于提供完整数据访问功能的需要不得不提供一些功能强大的接口。

因此，应当将驱动程序所提供的接口做进一步的封装，同时业务逻辑层也要减少这类接口的使用。与此同时，数据访问层还要定制一些经常被业务逻辑层调用的接口。另外，全局模型和本地模型的接口很相似。

业务逻辑模块。业务逻辑主要处理两种业务逻辑，输入和输出。

对于输入来说，原则上，一部分属性和实体文件是必须提交的；一部分属性可以根据网络会话的属性推测出来；一部分属性是可选的，即使不填系统也会允许或者填入默认值。实际情况中，为了提高用户操作的灵活性，一些必填

的属性和实体文件也可以暂时不提交，但是这会后续的功能。

对于输出来说，一些数据出于安全性的考虑，是不能够返回给用户的；一些数据出于减少流量和提高用户体验的考虑，应该由用户决定是否要返回；一些数据体现了系统的核心业务逻辑，必须要返回。业务逻辑模块应该根据这些原则，将不规范的用户输入转化成数据访问层能接受的规范输入，将朴素通用的数据访问层数据转化成有益于业务逻辑运作的系统输出。

MongoDB 与其说是非关系型数据库，不如说是不仅仅是关系型数据库。以关系作为基本单位来设计数据接口不失为一种结构清晰的方法。

下面的代码依据 id 来获取模型文件的输出流，是该模块的主要功能之一。

```
@check_id
def find_file_one_by_id(self, file_id):
    out_stream=self._fs.get(file_id)
    return out_stream
```

图 5-2: 按 id 查找文件

5.1.2 Service 模块

业务逻辑层（Business Logic Layer）无疑是系统架构中体现核心价值的部分。它的关注点主要集中在业务规则的制定、业务流程的实现等与业务需求有关的系统设计，也即是说它是与系统所应对的领域（Domain）逻辑有关，很多时候，也将业务逻辑层称为领域层。

系统将提供对全局模型和本地模型的托管服务。系统在业务逻辑层所提供的服务仍然是以资源为中心设计的，基本上和数据接口保持一致，但是直接体现着业务逻辑。

相对于数据访问层，业务逻辑层直接服务于用户的需求。用户需求的灵活性也决定了业务逻辑层的灵活性。业务逻辑的本质仍旧是修改数据库中的数据或者查看数据库中的数据，但和数据访问层相比，有以下几点不同。

数据查询更加细节，需要定义所有用户可能使用的接口，即使使用的次数很少；数据聚合更加复杂，需要从原始的数据中提取出有价值的数据；数据格式更加多样，以适应用户体验、减少数据传输量和隐藏数据结构等需要。不仅如此，一次接口的调用可能会对数据存储造成多方面的复杂的影响，需要妥善对待。

虽然付出如此多的代价，业务逻辑层仍然收获了灵活性作为回报。

下面的代码依据 id 来获取模型文件，是该模块的主要功能之一。

```
def download_global_model_file(self, session_id,
    global_model_id):
    user_id = self.session_dao.find_one_by_id(session_id)
    user_id = user['user_id']
    if not user_id:
        return None
    file = self.global_model_dao.find_one_by_id(global_model_id)
    file_id = file['file_id']
    return self.global_model_dao.find_file_one_by_id(file_id)
```

图 5-3: 下载全局模型

5.1.3 APIServer 模块

APIServer 模块是一个 Web 应用。它通过接收 Http 请求，返回 Http 回复来完成与用户间的交互。

为了提高开发效率，人们开发了很多种 web 框架。它们都有一些相似点。

web 框架由路由开端。这里的路由和计算机网络上的路由有相似之处，但大部分是不同的两个概念。Http 请求中有一个参数叫路径。当 Http 请求提交到 Web 服务器时，首先要检查这个路径，再根据这个路径和开发者定义的一些规则将 Http 请求转发到相应的处理函数上。这就被称为一次路由。

路由本身是很简单的功能，但为方便开发，人们在路由规则上提供了很多选择。比如说，路径中可以包含可变的参数，当路径发到路由上是可以解析处这个参数，并在请求的处理过程中利用这个参数。比如说，可以用正则表达式的语法表示许多种路径，路由检查路径是否符合正则表达式，如果符合就转发到对应的处理函数上。

紧接着路由的是对 Http 请求的处理。Http 请求中包含很多的数据，也有着很多的类型。这得从 Http 协议说起。Http 协议规定 Http 请求包含许多内容，其中最关键是路径、请求头、主体。在路径中可以添加查询参数，交由处理函数解析。请求头中包含了对请求的自描述。主体部分则可以包含各种各样的数据，甚至二进制数据也是可以的。处理函数通过不同的办法对主体进行解析。

对 Http 请求的处理并不一定是一次完成的。有可能需要经过多道处理来为完成一个 Http 请求的处理。这往往是出于构建一个合理的框架，提高编码效率的目的所做的。

由此可见，一个 web 应用要完成一系列的基础工作，而这些基础工作是所

有 web 应用都会有的。如果重复开发的话会降低开发效率，有些人就开发出了所有开发者都能使用的 web 框架。

本系统选用的 web 框架是 Bottle。它是一个部署迅速的、功能完善的、方便使用的 web 框架。在 Bottle 中，一个 web 应用即使一个对象。web 应用对象在构造好之后可以多次部署，也可多个 web 应用组合成一个大应用，很适合实验性的软件项目。

作为一个用户接口模块，APIServer 原则上是不处理业务逻辑的。用户接口模块的职责应该是促进人机交互，将用户的语言忠实地转化成计算机所能理解的语言。

这在本系统中的体现就是从 Http 请求中提取业务逻辑层所需要的数据，组织成 Service 模块可以理解的形式，转发给 Service 模块，将业务逻辑的执行交由 Service 模块进行处理。

在 Service 模块处理完成后，APIServer 会受到其处理的结果，有时候描述了操作的状态，有时候是要返回给用户的数据。这时候 APIServer 就要把这些返回值转化成用户端能接收的形式，包装成 Http 回应返回给用户。

在 APIServer 的接口设计中有个常用的风格，即 REST 风格。Http 协议对 Http 请求的规定中有个数据叫做方法名。常见的 Http 请求方法名有 Get, Post, Put, Delete，它们与数据库中的查、增、改、删很相似，但又不完全一样。由于 Http 是无状态的协议，在方法的分类上它更关注方法的安全性和幂等性，即如果发送一次请求，系统是否会有什么变化，如果在发送一次请求后接着发送相同的请求，系统是否会发生什么变化。

REST 风格的核心是面向资源思想。在本系统中，所谓的资源主要是全局模型数据、本地模型数据和用户数据。围绕这三项资源并结合一些产业界的范例就能设计出良好接口。

下面的代码依据 id 来下载模型文件，是该模块的主要功能之一。

```
@app.get('/global_models/files/<global_model_id>')
def global_model_file_get(global_model_id):
    global_model_service = GlobalModelService()
    content = request.params
    file = global_model_service.download_global_model_file(
        session_id=content['session_id'],
        global_model_id=global_model_id
    )
    return file
```

图 5-4: 获得全局模型

5.2 EdgeX Foundry 组件

EdgeX Foundry 是一个边缘计算平台，采用了微服务架构。该边缘计算平台主要分为四层，设备服务层、支持服务层、核心服务层、导出服务层。

设备服务层允许添加支持不同协议、不同设备的设备服务。设备服务对上支持 REST 风格的方法调用，对下支持对设备的读和写，中间能够完成数据格式的转化。通过设备服务层，EdgeX Foundry 可以屏蔽与设备的交互细节。

支持服务层介于设备服务层和核心服务层的中间，对两者的数据交流进行监听和利用。支持服务层支持很广泛的操作，比如依靠规则引擎执行对各层的操作，如对各层的操作进行记录，以及制定各层计划的操作，等等。其设计的目标就是提供广泛的边缘分析和边缘智能功能。

核心服务层保存边缘计算平台的所有核心数据和配置，是端设备和云的中转站，在整个平台中处于核心地位。核心服务层管理着边缘计算平台中的各个微服务和伴随着的种种数据，比如微服务的注册信息，端设备的统一格式后的数据，等等。

导出服务层提供数据的导出服务。本地或者云上的微服务可以通过在导出服务层上注册，可以在核心服务层收到数据时获得数据的转发。转发的方式主要有 MQTT 和 HTTP 两种协议，其中 MQTT 更轻量化，比较适合物联网应用。

本系统利用 EdgeX Foundry 来作为边缘计算平台，并在其上模拟了端设备的事件提交和导出服务层的注册。一方面，将 mnist 数据集拆分成许多批次，以不同的设备名再次分成许多类，并包装成事件的形式，直接发送给核心服务层。这样一来，边缘计算平台就误以为收到了来此设备服务层的事件，并将这些事件储存在平台的数据库上，并将这些事件转发给导出服务层。另一方面，在导出服务层上注册部署在边缘计算平台上的 MQTT 服务器，将转发来的事件发布到话题上。之后由 EdgeXClient 来进行订阅。

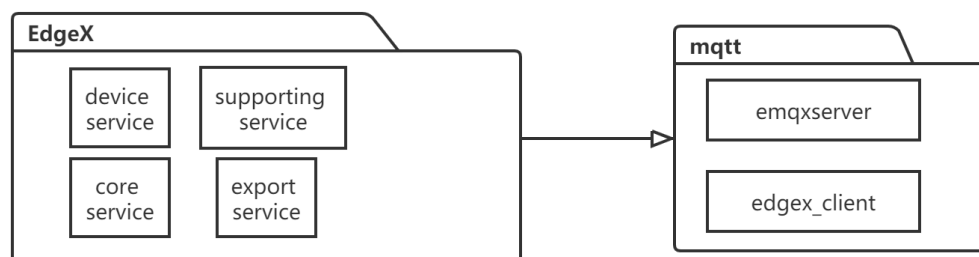


图 5-5: 包图

5.2.1 EMQXServer 模块

EMQX 是一款开源的 MQTT 消息服务器，在本系统中用于接收 EdgeX Foundry 中 export service 层发布的主题，并将其转发给订阅这个主题的客户。由客户端来完成本系统在边缘计算平台的功能。

在获知 emqxserver 的地址和端口后，EdgeX Foundry 的 export service 要注册与其对应的客户端配置，主要包括注册名称、消息服务器名称、地址、端口、订阅主题和发送格式等内容。在完成客户端注册后，EdgeX Foundry 就会把 core service 层收到的数据转发给 emqxserver，进而交付给 edgex client。

使用 MQTT 协议而非 HTTP 协议来在 export service 层注册微服务的意义在于，mqtt 协议是轻量级的数据传输协议，能够适应恶劣的硬件条件和网络状况，很适合应用于物联网系统。

```

:: Start the Windows service
:start
:: window service?
:: @\%erlsrv\% start \%service_name\%
@call :create_mnesia_dir
@call :generate_app_config
@set args=-detached \%sys_config\% \%generated_config_args\%
    -mnesia dir '\%mnesia_dir\%'
@echo off
cd /d \%rel_root_dir\%
@echo on
@start "\%rel_name\%" \%werl\% -boot "\%boot_script\%" \%args\%
@goto :eof

```

图 5-6: 启动 emqxserver

```

{
  "name": "device_event_topic",
  "addressable": {
    "name": "MQTTBroker",
    "protocol": "TCP",
    "address": "192.168.56.1",
    "port": 1883,
    "publisher": "EdgeXExportPublisher",
    "topic": "device_event_topic"
  },
  "format": "JSON",
  "enable": true,
  "destination": "MQTT_TOPIC"
}

```

图 5-7: Client Registration 发布事件配置

5.2.2 EdgeXClient 模块

EdgeXClient 同样也是一个 Web 应用，这里也选择 Bottle 作为 web 框架。

由于仅需要服务于端设备，自身的功能也很单一，因此 EdgeXClient 的接口设计起来相对简单，甚至在设计的时候采用 rpc 风格要比 REST 风格更加灵活。EdgeXClient 所提供的关键资源是本地模型，提供本地模型的部署、推理、训练和上传的功能。直接分别对这些功能设置路径处理用户的 Http 请求可以避免过度设计。

EdgeXClient 通过与 EdgeXFoundry 的连接，间接从端设备处获得以事件为统一形式的数据。这些数据可以用于推理和训练。同时，EdgeXClient 也随时准备接受部署本地模型的命令和上传本地模型的命令。

EdgeXClient 通过与 APIServer 的连接，从云仓库中获得所需的全局模型，构造本地模型和上传本地模型。这些通讯的次数少，占用的资源少，但是意义重大。只有拥有了一个云仓库，广泛分散边缘计算平台微服务才能够进行沟通，才能够合理完成同一个全局模型的开发，才能够享受集体优化的成果。

EdgeXClient 作为边与 APIServer 的云相对，完成本地模型的部署、训练和上传等事务。

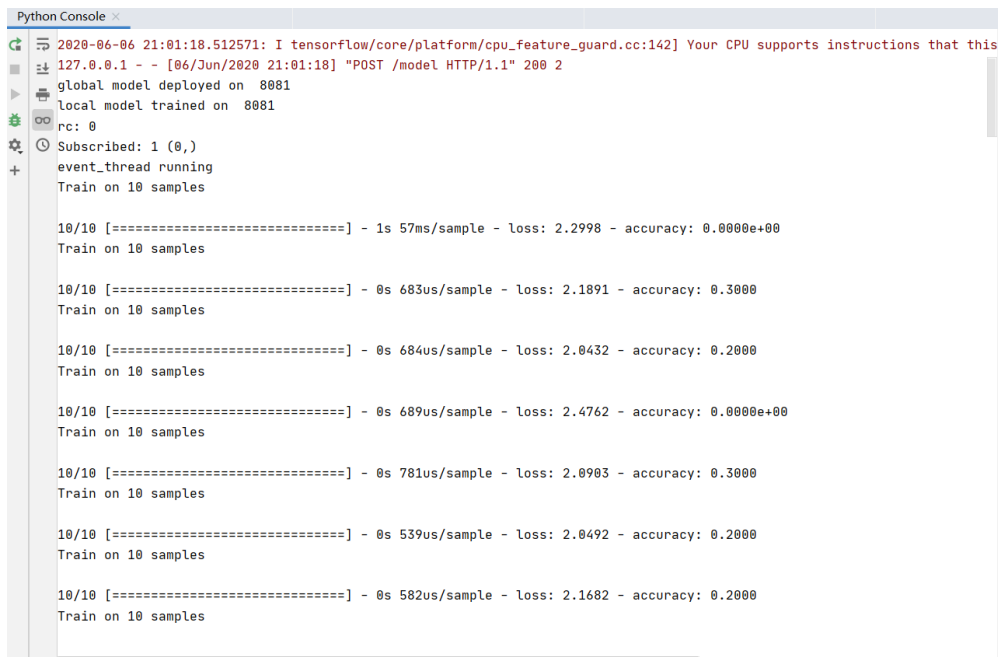
```
def train_local_model_mqtt(self):
    mqttc = mqtt.Client()
    mqttc.on_message = on_message
    mqttc.on_connect = on_connect
    mqttc.on_publish = on_publish
    mqttc.on_subscribe = on_subscribe
    mqttc.connect("localhost", 1883, 60)
    mqttc.subscribe("device_event_topic", 0)

    mqttc.loop_start()
```

图 5-8: 订阅事件

```
def on_message(mqttc, obj, msg):
    port = self.port
    event = json.loads(msg.payload)
    device_name = "device" + port[3:4]
    value = event['readings'][0]['value'].replace("'", '"')
    value = json.loads(value)
    data[port]['global_model'].fit(np.array(value['x_train']),
                                   np.array(value['y_train']),
                                   epochs=1)
```

图 5-9: 使用订阅得到的事件训练本地模型



```

Python Console
2020-06-06 21:01:18.512571: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this
127.0.0.1 - - [06/Jun/2020 21:01:18] "POST /model HTTP/1.1" 200 2
global model deployed on 8081
local model trained on 8081
rc: 0
Subscribed: 1 (0,)
event_thread running
Train on 10 samples

10/10 [=====] - 1s 57ms/sample - loss: 2.2998 - accuracy: 0.0000e+00
Train on 10 samples

10/10 [=====] - 0s 683us/sample - loss: 2.1891 - accuracy: 0.3000
Train on 10 samples

10/10 [=====] - 0s 684us/sample - loss: 2.0432 - accuracy: 0.2000
Train on 10 samples

10/10 [=====] - 0s 689us/sample - loss: 2.4762 - accuracy: 0.0000e+00
Train on 10 samples

10/10 [=====] - 0s 781us/sample - loss: 2.0903 - accuracy: 0.3000
Train on 10 samples

10/10 [=====] - 0s 539us/sample - loss: 2.0492 - accuracy: 0.2000
Train on 10 samples

10/10 [=====] - 0s 582us/sample - loss: 2.1682 - accuracy: 0.2000
Train on 10 samples

```

图 5-10: 训练本地模型的输出

5.2.3 EventCreator 模块

在系统测试的时候，需要模拟端设备收集数据并向边缘计算平台发送事件的过程，这时候后就用到了 `event creator`。使用该模块可以产生模拟的数据，并将模拟的数据包装成事件发送到 EdgeX Foundry 的 `core service` 层，模拟出端设备产生事件的情景。

在本系统中，主要研究的数据产生用于 AI 技术，因此模拟的数据为 AI 模型训练数据。数据集采用了手写数字数据集。

该模块首先将数据集分成多个批次。批次的容量可以自定义。每个批次都转化成事件的格式，包含设备和读出值两个属性。由于要模拟多个端设备，因此设备名可以按照虚拟端设备标注不同的名称。之后将各批次的事件发送到 `core service` 层即可。

```

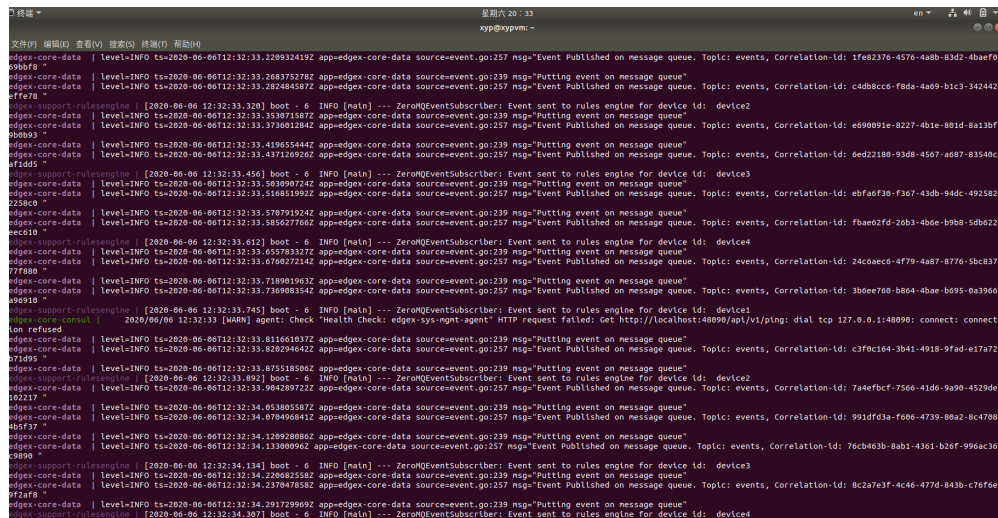
def run(self):
    print("event_thread running")
    batch_size = 10
    events = event_creator.create_photo_train_events(batch_size)
    events_size = len(events)
    for event in events:
        requests.post(self.protocol + self.host + ":" + self.port
                      + '/api/v1/event', data=event)

```

图 5-11: 发送事件到 Core Data

```
def create_photo_train_events(self, batch_size):
    batch_size = int(batch_size)
    events = []
    batches_size = len(self.x_train) / batch_size
    for batch_index in range(0, int(batches_size)):
        event = self.event.copy()
        event['device'] = 'device' + str(batch_index % 4 + 1)
        value = {
            "x_train": self.x_train[batch_index *
                                   batch_size:(batch_index + 1) * batch_size].tolist(),
            "y_train": self.y_train[batch_index *
                                   batch_size:(batch_index + 1) * batch_size].tolist()
        }
        event['readings'] = [
            {
                "name": "train",
                "value": json.dumps(value)
            }
        ]
        events.append(json.dumps(event))
    return events
```

图 5-12: 创建事件



```

xyp@xypvm:~
Level=INFO ts=2020-06-06T12:32:33.220932419Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 1fe82376-4576-4a8b-83d2-4baef8-9bbbf8"
Level=INFO ts=2020-06-06T12:32:33.268375278Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: c4db8cce-f8da-4a69-b1c3-342442-effe78"
Level=INFO ts=2020-06-06T12:32:33.353071587Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: e690091e-8227-4b1e-801d-8a130f-90b093"
Level=INFO ts=2020-06-06T12:32:33.419655444Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 6ed22180-93d8-4567-a687-83540c-2258d0"
Level=INFO ts=2020-06-06T12:32:33.570791924Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: fbae62fd-20b3-4b0e-b908-5dbd22-1cc010"
Level=INFO ts=2020-06-06T12:32:33.655783327Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 24dc6cc0-4f79-4a87-8776-5bc837-77f880"
Level=INFO ts=2020-06-06T12:32:33.716901963Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 3bdee760-b864-4bae-b095-0a3966-969310"
Level=INFO ts=2020-06-06T12:32:33.811661037Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: c3f0c164-3b41-4918-9fad-e17a72-371095"
Level=INFO ts=2020-06-06T12:32:33.875518566Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 991df3a-f066-4739-88a2-8c4708-465f57"
Level=INFO ts=2020-06-06T12:32:34.053805587Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 76cb463b-8ab1-4361-b26f-99eac36-992a78"
Level=INFO ts=2020-06-06T12:32:34.120928862Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 8c2a7e3f-4c4e-477d-843b-c76fee-992a78"
Level=INFO ts=2020-06-06T12:32:34.133009062Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 76cb463b-8ab1-4361-b26f-99eac36-992a78"
Level=INFO ts=2020-06-06T12:32:34.291729969Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 8c2a7e3f-4c4e-477d-843b-c76fee-992a78"
Level=INFO ts=2020-06-06T12:32:34.307110000Z app=edgex-core-data source=event.go:257 msg="Event Published on message queue. Topic: events, Correlation-Id: 76cb463b-8ab1-4361-b26f-99eac36-992a78"

```

图 5-13: EdgeX Foundry 收到事件并转发的输出

第六章 系统测试及功能验证

6.1 测试目的

测试是对软件质量的维护。测试并不能保证系统绝对没有缺陷，但是测试能够尽可能少的减少缺陷。因为这个特征，测试应该从多个角度多次进行。测试可以分为单元测试和集成测试，这其实是依据粒度的测试分类。因此本系统中的测试也可以根据粒度将测试分为功能模块的测试、组件的测试和系统的测试。

测试功能模块的运行是否符合预期。创建虚拟环境，提供虚拟数据对功能模块的功能进行检查。

测试组件的运行是否符合预期。将功能模块集成为组件从外部提交请求，检查其能否完成功能。

测试系统的运行是否符合预期。将组件集成为系统，依据用户需求逐项检验。

6.2 测试内容

第一步，初始化组件。边缘计算平台由事件产生器代替，模拟端设备提交数据。其它组件都正常部署。所有的组件都单独建立线程来模拟独立的部署环境。

第二步，构造和提交全局模型。使用 Keras 高层 API 构造出结构完整、已经经过初步训练的 AI 模型。将 AI 模型加载，在云仓库中创建和提交全局模型。

第三步，本地模型的获取和部署。边缘计算平台微服务接收请求，从云仓库下载指定的全局模型，构造新的本地模型，部署在边缘端。

第四步，本地模型的训练和上传。事件产生器提供事件来模拟端设备，将事件啊发送的边缘计算平台微服务上。微服务根据这些数据进行对本地模型的训练，在训练完成后，将之上传。

第五步，本地模型的下载和联邦学习。将多个边缘计算平台微服务提交的本地模型下载到本地，并据此进行联邦学习，产生更优秀的全局模型，上传到

云端。就此完成一次迭代。

6.3 测试结果

各组件运行良好。通过观察控制台的日志可以看出，不同线程间的组件能够良好的、互不干扰的运行。

降低了通讯次数。与传统云 AI 平台相比，本系统将模型的训练也放在了边缘端，同时在边缘端进行主要的训练，所需要的与云端的通信仅仅是在提交本地模型和下载全局模型的时候。这一特性在通讯成本较高的场景中十分有利。

经过一次迭代，全局模型的准确率明显提高。在每一次迭代前估计全局模型的预测准确率。相互比较会发现，每一此联邦学习都能够提高全局模型的准确率。与一般机器学习产生的 AI 模型相类似，联邦学习所产生的 AI 模型也有准确率的瓶颈。从实验结果来看两者的最终准确率相差不大。本地模型在全局模型的基础上利用用户数据训练出个性化的模型，在用户所处的场景中会有更出色表现。这也是联邦学习的一大优点。

第七章 结论与展望

7.1 结论

本系统能减少资源消耗。用户数据不需要上传到云端，因此就减少了通信资源的消耗。AI 模型直接部署在本地，并在本地完成推理和训练，能够充分利用边缘端的计算资源，较少云端的负担。本地模型的推理和训练没有硬性的要求，因此可以选择在灵活地选择适当的场合进行。

本系统能建立广泛的互利关系。随着 AI 和大数据的蓬勃发展，数据的价值在被发掘的同时，也遭遇到了所有权上的问题。

有些科技公司认为数据本身是没有价值的，是科技公司对这些数据进行了加工这才产生了价值，因此公司采集的数据和最终的 AI 成果也应当属于科技公司。

许多民众对此表示不满，认为科技公司侵犯了自身的权益，并使用法律去维护自己的权力。

将边缘计算与 AI 结合能初步解决这一问题。用户的数据不再需要上传到云端，也就不存在隐私被侵犯的问题。同时用户利用全局模型为基础训练出的本地模型也能能够个性化地解决自身地问题，即使不放弃隐私，也能从科技的发展中获益。与此同时科技公司也能通过自身掌握的全局模型获得技术上和商业上的价值。总体来看，结合边缘计算的实践带来了双赢的局面。

本系统兼顾通用性和个性化。一个好的 AI 模型的开发成本是很高的，而且开发者必须在通用性和准确率之间做出权衡。因为一般功能强大、涉及信息复杂的 AI 模型准确率会偏低；而功能简单、领域受限的 AI 模型准确率会偏高。然而，只有兼顾涉及领域广泛和准确率高两个条件的 AI 模型才是最有价值的。

虽然不是初衷，结合边缘计算的实践给出了一套折中的方案。即将全局模型作为通用模型看待，要求其对广泛的场景具有基本的推理能力；同时本地模型利用用户数据进行训练，得出在用户所处领域中最合适的个性化模型。这样一来就兼顾了通用性和个性化两个优点。

边缘计算回答了如何将边缘端产生的数据更有效率的利用这一问题。通过

将边缘计算与云 AI 平台结合，可以降低 AI 模型训练在通信、算力和存储等方面的开销，可以保护用户数据的隐私性，可以更快速的迭代，可以获得更广泛的数据来源。

本论文所设计和实现的结合边缘计算的云 AI 平台只是非常初级的表现形式，在设计、实现和部署方面有着诸多不成熟的地方。该系统开发目的是展现边缘计算与云 AI 平台的一种可能的方式，无法投入真正的生产环境。

7.2 展望

任何新的软件产业必然有其背后的推动者。我们可以期待政府、学校和企业出于不同的目的去推动服务于边缘计算平台的云 AI 平台的开发。在本系统中仍有许多可以完善的地方。

联邦学习只是所有将数据留在边缘端而不是传输到云端进行利用的诸多方法中的一种，更不用说联邦学习本身也有很多分支 [31,32]。机器学习的方法十分多样，平台可以提供一些默认的算法供用户使用，以降低用户门槛。平台亦可以提供机器学习所需要的环境和容器。

AI 模型的管理也有很多不成熟的地方。由于机器学习在相当广泛的领域中得到了应用，因此本系统的涉众也极其广泛。应当设计成熟的权限机制来管理模型的访问。

系统的部署可以转化为分布式。系统设计的时候注意了模块化，尤其是注意数据和逻辑的一致性，因此可以利用这一点轻易地将项目改造成微服务架构。另外非关系型数据库支持分布式数据库，可以方便地扩展存储。

参考文献

- [1] 加日拉·买买提热衣木, 常富蓉, 刘晨, et al. 主流深度学习框架对比 [J]. 电子技术与软件工程, 2018, 07(4): 50–58.
- [2] HOSE J D H. EdgeX Foundry[J]. InTech, 2018, 65(2): 50–58.
- [3] K A. That “internet of things” [J]. RFID Journal, 2009, 22(7): 97–114.
- [4] David E and Culler. The Once and Future Internet of Things[EB/OL]. 2016 (2016/12/03) [2016/012/03].
http://sites.nationalacademies.org/cs/groups/cstbsite/documents/webpage/cstb_160416.pdf.
- [5] Vala Afshar. Cisco: Enterprises are leading the Internet of Things innovation[EB/OL]. Cisco, 2017 (2017/08/15) [2017/08/15].
<https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf>.
- [6] LeClaire J. HIS predicts 54 millions self-driving cars by 2035[EB/OL]. HIS, 2016 (2016/12/03) [2016/12/03].
http://www.newsfactor.com/story.ahtml?story_id=01300CTNJ8D.
- [7] Dataflop. Self-Driving cars will create 2 petabytes of data, what are the big data opportunities for the car industry?[EB/OL]. Dataflop, 2016 (2016/12/03) [2016/12/03].
<https://datafloq.com/read/self-driving-carscreate-2-petabytes-data,-annually/172>.
- [8] Finnegan M. Boeing 787s to create half a terabytes of data per flight, says virgin atlantic[EB/OL]. Finnegan M, 2016 (2016/12/03) [2016/12/03].
<http://www.computerworlduk.com/data/boeing-787s-create-half-terabytes-of-data-per-flight-says-virgin-atlantic-3433595/>.

- [9] 北京电动车辆协同创新中心. Boeing 787s to create half a terabytes of data per flight, says virgin atlantic[EB/OL]. 北京电动车辆协同创新中心, 2017 (2017/11/13) [2017/11/13].
<http://me.bit.edu.cn/sytj/syszxjs/bjddjlxtcxzx/86238.htm>.
- [10] 中国天网. 装 2000 万监控摄像头中国构建全球最大“天网”[EB/OL]. 中国天网, 2017 (2017/11/13) [2017/11/13].
http://tech.ifeng.com/a/20171004/44706732_0.shtml.
- [11] H S, P G, P F. Vision and challenges for realizing the Internet of Things[J]. Cluster of European Research Projects on the Internet of Things, 2010, 73(1): 55 – 70.
- [12] J G, R B, S M. Internet of Things (IoT): A vision, architectural elements, and future direction[J]. Future Generation Computer System, 2013, 29(7): 1645 – 1660.
- [13] Cisco Visual Networking. Cisco global cloud index:Forecast and methodology 2015-2020[EB/OL]. Cisco Visual Networking, 2018.
<https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf>.
- [14] 施巍松, 刘芳, 孙辉, et al. 边缘计算 [M]. 北京: 科学出版社, 2018.
- [15] 李钟海. 边缘计算的现状及发展 [J]. 中国公共安全, 2020, 03: 148 – 149.
- [16] 王雄. AI 技术的未来发展方向 [J]. 计算机与网络, 2020, 46(08): 38 – 40.
- [17] Shuiguang Deng and Hailiang Zhao and Weijia Fang and Jianwei Yin and Schahram Dustdar and Albert Y. Zomaya. Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence[EB/OL]. arXiv, 2019 (2019/08/15) [2019/08/15].
arXiv.
- [18] 娄焕. 云计算在计算机中的应用及发展 [J]. 计算机产品与流通, 2020, 2020(05): 116 – 160.
- [19] S G, H G, T L S. The Google file system[C] // ACM SIGOPS operating systems review. 2012: 29 – 43.

- [20] J D, S G. Mapreduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107 – 113.
- [21] K S, H K, S R. The Hadoop distributed file system[C] // 2010 IEEE 26th symposium on mass storage systems and technologies (MSST). 2010: 10 – 10.
- [22] M Z, M C, J F M. Spark: cluster computing with working sets[J]. HotCloud, 2010, 1(1): 10 – 10.
- [23] ARMBUST, MICHAEL, FOX. Above the clouds: a Berkeley view of cloud computing[J]. Eecs Department University of California Berkeley, 2009, 53(4): 50 – 58.
- [24] BUTLER B, BUTLER B. AWS Greengrass weds IoT devices with cloud[J]. Network World (Online), 2016, 53(4): 50 – 58.
- [25] Zhi Zhou and Xu Chen and En Li and Liekang Zeng and Ke Luo and Junshan Zhang. Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing[EB/OL]. arXiv, 2018 (2018/08/15) [2018/08/15]. arXiv:1905.10083.
- [26] 底慧萍, 王静宁. 浅谈非关系型数据库 [J]. 河北农机, 2019, 11(50): 50 – 58.

致 谢

白驹过隙，四年的大学时光即将结束。回望这条求学之路，既有刻苦钻研时留下的汗水，亦有学有所成时感到的喜悦。在这条路上，少不了良师益友的关怀和帮助。这里要对每一位帮助过我的人表示衷心的感谢。

首先，感谢南京大学软件学院的刘海涛导师。他在我的毕业设计完成过程中起着不容忽视的作用。在论文选题过程中，导师为我点出了论文的主旨，并在设计的完成过程中不断地强化我对论文主题的理解，使我能够紧紧抓住论文的主题，使得每一句论述都能够落到实处。在论文的撰写过程中，导师回答了我的很多疑惑。我从这些回答中汲取到了丰富的知识，使我的论文得到不断的完善。每当我遇到难以解决的问题时，导师都会不厌其烦地帮助我去理解问题、解决问题。学术领域的交流是晦涩的，却也是收获良多的。在与导师沟通的过程中，我感受到了学术研究的严谨和伟大。在论文的验收过程中，导师展现出了其严肃认真的一面。从论文排版到论文内容的组织，导师给出了细致且可操作的意见。导师更是花费了大量的个人时间用于我的论文的审阅和修改。我感激于导师包容的沟通方式和慷慨的指导帮助。

其次，感谢南京大学的老师们。我要感谢大学四年所有的任课老师。他们有着渊博的学识，有着对各自领域独特的见解。他们带领我走入知识的殿堂，是我的授业恩师。他们所传授我的知识更是我撰写论文的基石。我还要感谢南京大学软件学院的宋抃辅导员，感谢他在学习生活中给予我的帮助。

另外，感谢我的家人朋友们。他们的帮助使得我在完成论文的时候没有后顾之忧。他们的关怀和鼓舞使我在不断钻研的过程中定下心来。

最后，衷心感谢在百忙之中评阅本论文的专家老师。

是你们的帮助让我能顺利完成论文的撰写，再次表示衷心的感谢。