南京大學

研 究 生 毕 业 论 文
（申 请 硕 士 学 位）

论 文 题 目 神经网络测试数据优先级度量系统的设计与实现

作 者 姓 名 张子杰

学科、专业名称 工程硕士（软件工程领域）

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授，冯洋 助理研究员

2021 年 **5** 月 **19** 日

学　　号：**MF1932252**

论文答辩日期：**2021** 年 **5** 月 **20** 日

指 导 教 师：　　（签字）

The Design and Implementation of Test Case Prioritization System for Neural Networks

by

Zijie Zhang

Supervised by

Professor Zhenyu Chen, Assistant Researcher Yang Feng

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER OF ENGINEERING
in
Software Engineering

Software Institute
Nanjing University

May 19, 2021

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：神经网络测试数据优先级度量系统的设计与实现

工程硕士（软件工程领域）专业 2019 级硕士生姓名：张子杰

指导教师（姓名、职称）：陈振宇 教授，冯洋 助理研究员

摘　要

随着深度学习领域中若干新技术的涌现，基于神经网络的多种模型越来越多地被应用到实际生活中，包括人脸识别、自动驾驶等。但是，神经网络由于其黑盒性，无法保证其决策可靠性，这在自动驾驶等生命攸关的场景中显得尤为重要。目前通常使用大量的测试用例以保证神经网络在不同场景下的安全性，但这会导致十分高昂的数据标注成本。因此如何对深度学习软件的测试用例进行优先级排序以挑选出关键用例成为一个亟需研究的问题。

本文针对上述问题，设计了基于基尼不纯度的测试用例度量系统。用户使用本系统创建度量任务，上传模型与测试数据集、配置参数并提交。系统会将一系列具有依赖关系的操作建模成有向无环图。在获取到神经网络模型推理结果后，系统按照基尼不纯度的公式计算出度量值，并以此作为用例排序时的比较依据。任务执行完成后，用户可获得与之关联的分析报告。此外，用户可以选择一部分高优先级的样本对模型进行重训练。

本系统划分为模型编码转换模块、数据预处理模块、度量算法模块、度量报告生成模块以及计算引擎模块。系统使用了前端框架 Vue.js 和后端框架 Flask 进行开发，使用 ONNX 开放模型编码格式解决多框架间的兼容性问题，利用 Kubernetes 实现系统的高可用与弹性可扩展性，利用 Airflow 调度中间件实现系统的模块化开发并保证任务执行的原子性。为保障任务执行的可靠性与数据存储的安全性，系统使用云端对象存储系统负责关键数据的读取与写入。

该系统已在慕测人工智能基础平台中上线。通过用户问卷调查实验，用户体验的满意度达到了 92.8%。基于基尼不纯度的测试用例度量系统能够有效地筛选出神经网络无法明确判决的测试用例，帮助用户检查数据特征工程是否完备。此外，利用高优先级数据进行重训练，能够进一步地提升模型泛化能力。

关键词：神经网络；测试用例度量；信息熵；基尼不纯度

南京大学研究生毕业论文英文摘要首页用纸

THESIS: The Design and Implementation of Test Case Prioritization System for Neural Networks

SPECIALIZATION: Software Engineering

POSTGRADUATE: Zijie Zhang

MENTOR: Professor Zhenyu Chen, Assistant Researcher Yang Feng

Abstract

With the emergence of several new technologies in the field of deep learning, a variety of neural network-based models are increasingly being applied to real life, including face recognition, autonomous driving, etc. However, neural networks cannot fully guarantee reliability due to their black box nature, which is particularly important in life-critical scenarios such as autonomous driving. Currently, a large number of test cases are usually used to ensure the safety of neural networks in different scenarios, but this will lead to very high data labeling cost. Therefore, how to prioritize the test cases for deep learning software in order to select the key cases has become a problem that needs to be studied urgently.

Aiming at the above problems, this thesis designs a prioritized system for test cases based on gini impurity. Users use this system to create measurement tasks, upload models and test data sets, configure parameters and submit them. The system will model a series of dependent operations into a directed acyclic graph. After obtaining the inference result of the neural network model, the system computes the measurement value according to the formula of gini impurity, and uses this as the basis for comparison when sorting the test cases. When the execution of the task is finished, the user can get a related analysis report. In addition, users can select a part of high-priority samples to retrain the model.

The system is divided into a model encoding conversion module, a data preprocessing module, a measurement algorithm module, a measurement report module, and a computation engine module. The system uses Vue.js as the front-end framework and Flask as the back-end framework for development. It also uses the open neural network

exchange(ONNX) format to solve the compatibility problem between multiple frameworks. Equipped with Kubernetes, the system can achieve high availability and elastic scalability. Airflow is used as a scheduling middleware to achieve modular development and atomicity of task execution. To ensure the reliability of measurement tasks and the security of data storage, the system uses an object storage system on the cloud to read and write critical data.

The system has been deployed into MoocTest artificial intelligence infrastructure platform. Through questionnaire survey, the satisfaction of our user experience reaches 92.8%. The test case prioritization system based on gini impurity can effectively filter out test cases that the neural network cannot make clear decisions, and help users inspect whether the data feature engineering is complete. In addition, using high-priority data for retraining can further improve the generalization ability of the model.

keywords: Neural network, Test case prioritization, Information entropy, Gini Impurity

目　　录

插图清单

附表清单

第一章　引言

1.1　项目背景与意义

人工智能理论及其应用在近十几年以来飞速发展，其中以神经网络为基础的深度学习方法在计算机视觉、知识图谱、信息检索与推荐等领域得到了广泛的应用 [1–3]，其在部分场景下性能评价指标远超传统算法。神经网络的基本特点是基于数据驱动，即通过拟合输入数据的分布特点来预测独立同分布下真实数据的标签。正由于这一特点，几乎所有的神经网络模型都需要大量的数据及标签来学习并验证这种内在的分布规律。这在自动驾驶等生命攸关的应用领域显得尤为重要。因此，需要对这类软件进行大量的验证测试以保证其可靠性。

在实际工程中，通过各种传感器采集到的数据大多是无标签的，因此项目人员需要花费大量的时间成本和人力成本来完成数据的标注工作。常见的解决办法是利用迁移学习或者小样本学习。将在某个领域内学习到的模式或经验应用到其他相关但不同领域中的技术被称为迁移学习 [4]。迁移学习利用重训练的技术，可以有效地解决带标签训练数据样本过少的问题。小样本学习是元学习理论的一种，指利用有限的标签数据来完成监督学习 [5]。这两种方式本质上都是从大量未标记数据中均匀采样一部分，对这一部分数据打上标签后，利用相关算法训练模型以拟合该数据集。这类方法的缺陷在于数据采样的好坏将直接决定模型的性能。较少的数据样本可能会导致该部分数据无法真实地反应数据集分布规律，也可能会使模型对数据集产生过拟合。

针对以上的问题，本系统利用基于统计信息的测试用例度量技术来解决以上难点。测试度量技术在传统软件测试中发挥了重要的作用，能够有效地判定哪些测试用例对系统贡献率较大。主要的度量指标包括基于需求的覆盖率、基于代码的覆盖率等 [6]。但是，传统度量指标无法直接应用于神经网络模型，原因是相比于传统软件直接将业务逻辑表达在代码控制流中，在深度学习软件中，所有的业务规则都通过特征工程嵌入到神经网络的参数中，因此上述指标无法有效地区分不同测试用例对提升网络泛化能力的贡献。考虑到神经网络数据驱动的特点，本系统从统计角度出发，利用基尼不纯度值作为不同测试用例

优先级高低的度量标准。本系统生成的度量报告从多角度对结果进行分析，包括度量标准值分布的柱状图、度量标准值与数据集类别的热力图、度量结果中前 10 个与后 10 个用例的可视化结果等，用户根据该报告能够检验数据预处理及特征工程是否完备、模型结构是否合理等。此外，通过对度量后的部分数据进行标注并重训练，相比于对测试数据随机采样，前者能更有效地提升模型的泛化能力。

1.2 国内外研究现状

本系统主要为深度学习软件中测试用例的度量技术，所以本节将详细介绍目前已有的神经网络测试技术以及传统软件中的测试用例度量技术。

1.2.1 基于对抗样本的神经网络测试技术的研究现状

对抗样本生成技术由 Ian J.Goodfellow 在 2014 年提出，是指在原有输入图片中加入不为人眼可见的细小扰动，诱导神经网络模型输出明显错误的分类结果 [7]。对抗样本生成技术已在多个领域中得到应用，例如在数据扩增领域中生成新样本、利用对抗式的训练方法提升神经网络的鲁棒性、针对神经网络系统的对抗攻击等。学术界提出了多种有关对抗样本的生成算法。Szegedy 等人首次证明了可以通过对图像添加少量的、人眼无法感知的轻微扰动可使神经网络作出误分类，并给出了对应的算法实现 [8]。Goodfellow 等人在尝试解释对抗样本的产生原因时，认为扰动造成的影响在神经网络中具有累积效应，即存在某个特定的方向，使得扰动所造成的影响能够在网络推理时被快速放大，从而造成模型作出误分类。根据这种假设，Goodfellow 等人提出了 FGSM 算法，实现了对抗样本的快速生成 [7]。Papernot 等人首次提出了基于 L_0 范数生成对抗样本的算法，相比于使用 L_2 或 L_∞ 范数会导致扰动散布在整张图像中，使用 L_0 范数可以做到仅改变几个像素点的值就使得模型产生误分类的行为 [9]。Su 等人提出了基于差分进化算法的单像素攻击，通过仅修改图片中单个像素点的值，实现对神经网络模型的攻击，这种攻击方式隐蔽性强，肉眼难以察觉 [10]。

在神经网络测试中，对抗样本生成技术通常用来在已有标签类下产生新的测试样本，以弥补缺少真实标签的数据样本的不足 [11]。但是，单纯地使用上述的生成算法会导致生成的数据样本特征过于生硬、不自然并且不含有领域意义。因此，Zhao 等人提出了从流形角度出发，在特征域上进行扰动叠加，并通

过生成对抗网络来完成不同域之间的映射转换 [12]。此外，在结合防止训练过拟合的条件下，Xie 等人提出利用对抗样本作为训练数据，对神经网络模型进行调优训练的方法。通过在批归一化层增加额外的训练参数，可以保证神经网络模型同时学习原始数据样本和对抗数据样本这两种不同分布，从而有效地解决训练过拟合的问题 [13]。

1.2.2 测试用例优先级度量技术的发展现状

在大型工业软件的开发中，程序员倾向于采用结构化或面对对象的思想进行程序的编写工作，通常会在此过程中编写大量的单元测试用例以验证该模块的可靠性。随着软件规模与复杂度的不断增加，在软件开发的后期阶段，花费在回归测试上的时间越来越可观。例如在某电气工业软件中，测试用例包含约 20 000 行代码，运行全部的测试用例需近 7 周的时间 [14]。在这种背景下，软件工程师希望能够将测试用例按照某种标准进行排序，优先运行高优先级的测试用例，以此提高软件的开发效率。测试用例优先级度量技术能以更高的效率来提升错误检测率、特定代码段的错误检测率、代码覆盖率等指标 [15]。

目前，学术界已经提出了若干种测试用例优先级的度量算法。为衡量这些度量算法的实际效率，通常使用回归测试中的平均 bug 检测时间，即 average rate of fault detection(APFD)。常见的度量算法包括以下几种：全量分支粒度覆盖率度量、增量分支粒度覆盖率度量、全量语句粒度覆盖率度量、增量语句粒度覆盖率度量、全量潜在错误度量、增量潜在错误度量 [16]。全量分支粒度覆盖率度量是指在分支粒度下，对单个测试用例所执行过的分支进行计数，根据每个测试样本所执行的分支总数进行排序。对于具有 n 个分支的程序，该度量算法的时间复杂度经过分析为 $O(n \log n)$。增量分支粒度覆盖率度量是指在全量分支度量的基础上，重点考虑能够覆盖当前尚未执行的分支，即算法迭代地考虑每一个测试用例，记录下该用例所能新覆盖的分支，直到所有分支被覆盖或者当前用例为最后一个时算法停止。增量分支度量能够快速发现一些执行不频繁的分支，并且将其度量优先级拔高，因此能够有效地在较少的测试用例下提高平均错误检测率。考虑到需要迭代所有的测试用例，因此在具有 n 个分支的程序中，算法的时间开销为 $O(n^2)$。全量与增量语句粒度覆盖率度量是把覆盖粒度从分支缩小为语句，其他过程不改变，优点在于执行路径覆盖更为全面。全量潜在错误度量从故障发生概率的角度出发，即揭露出程序语句上的 bug 不仅要求该测试用例能够执行到该语句，还要求该测试用例能够实际触发该 bug，

因为存在着测试用例执行到该语句但是并不能触发 bug 的情况，这在数值计算软件中很为常见。全量潜在错误度量通常采用变异分析技术来实现，相比于上述的度量算法，其计算过程也更加繁琐复杂 [17]。增量潜在错误度量将逐步减少不同测试用例在触发相同 bug 时的奖励值，其他与全量潜在错误度量算法类似。经过实验证明，这些度量算法能够有效提高平均故障查找时间，提升大型软件回归测试时的效率 [6]。

1.3 本文主要工作内容

本文通过研究针对深度学习模型的测试用例度量技术，并结合云原生相关技术，实现了针对神经网络模型的测试用例度量系统，系统生成的度量报告能够辅助用户检查并验证当前模型中数据预处理操作与特征工程的合理性，度量后的测试用例经过人工标注并结合重训练技术，可进一步提升模型的泛化能力。当前，由深度学习理论衍生出的各种 AI 模型已经广泛应用于各大公司的产品业务中，极大地提高了相关业务的生产效率，但是在模型迭代改进环节，由于线上产生的数据缺少真实标签，往往需要耗费大量的人力来完成数据的标注工作。此外，并非所有的数据都有益于模型泛化能力的提升，因此该过程已经成为深度学习模型运维的瓶颈之一。为了解决模型迭代优化成本高、数据标注利用率低、流程自动化能力弱的问题，本系统尝试将传统软件测试领域的测试用例度量技术引入到深度学习中并针对深度学习数据驱动的特点作出一定改进，有效地降低了数据标注的成本，同时基于重训练的手段能够有效提高地模型泛化能力，进而解决模型迭代性能提升慢这一瓶颈问题。本论文重点讨论基于信息熵的深度学习模型测试用例度量系统的设计以及具体实现，并给出算法可靠性和执行效率的验证结果。在工程实现上，本系统前后端分离，前端使用 Vue.js[①]框架，后端使用轻量级的 Flask[②]框架，测试度量算法封装为独立的计算引擎，这使得系统具备更好地扩展性与移植性。此外，为了能够同时处理多个测试度量任务，本系统采用 Apache Airflow[③]作为任务调度平台，将用户提交的测试用例度量任务建模为有向无环图，以并发地方式进行执行。通过利用 Kubernetes[④]容器编排引擎，系统具备高度弹性可扩展的架构，可根据任务量进行资源的弹性管理，保证了硬件资源的高效利用。

[①] https://vuejs.org
[②] https://flask.palletsprojects.com
[③] https://airflow.apache.org
[④] https://kubernetes.io

基于基尼不纯度的深度学习模型测试用例度量核心技术主要分为以下几个步骤：首先，加载并校验用户提交的模型文件与未标注的数据集。其次，针对该数据集完成一次前向传播的计算，记录下神经网络输出层向量的结果。然后，对神经网络的输出层计算 Softmax 结果，再针对每一个样本计算其基尼不纯度。最后，对整个数据集进行排序，以每个样本的基尼不纯度大小作为排序的依据。其值越大，说明该模型对样本的分类决策越不确定。因此对这类样本进行人工标注并进行重训练能够有效地提高模型的泛化能力，并大幅度地降低标注数据的成本。

1.4 本文的组织结构

本文的组织结构如下：

第一章是引言。本章阐明了项目的背景和意义，调研了测试用例优先级排序技术在传统软件工程测试场景中的应用现状，以及本文为解决已存在问题提出的方案和主要工作内容。

第二章是技术概述。本章通过与相关技术的对比，介绍了本系统主要选择和使用的核心概念和技术方案。

第三章是系统需求分析与概要设计。本章将先从项目背景、系统开发目的和目标以及系统提供的功能等方面介绍系统整体情况。然后，分析了系统的功能和非功能需求，并基于此给出了系统用例图以及详细的用例描述，再结合系统架构图和 4+1 视图给出系统的总体架构设计以及模块划分，并对各模块进行简单设计描述，包括架构设计、流程设计和概念类图设计。

第四章是系统详细设计与实现。本章以第三章为前提，对模型编码转换模块、数据集预处理模块、度量算法模块、度量报告生成模块、计算引擎模块进行了详细的设计，绘制了顺序图和类图，并展示了关键代码片段。

第五章是系统测试与分析。本章说明了系统测试环境，描述功能和非功能测试目标、过程以及最终结果，并就结果进行分析总结。

第六章是总结与展望。主要总结了本文的工作，并对系统的优点和缺点加以说明，展望了测试用例度量技术在神经网络领域的未来发展。

第二章　相关技术概述

本章节对系统中所使用到的关键技术和相关问题的解决方案以及选择原因作简要概述。本系统通过使用 Airflow 中间件将用户度量需求建模为一张有向无环图，图中结点任务包括模型文件编码格式转换、数据集编码格式转换、模型的推理计算，以及最终计算结果的合并与排序。其中，模型编码采用 ONNX[①]格式，数据采用经过 Pickle 序列化 npy 编码格式进行存储，模型推理选用针对多硬件平台进行优化的 ONNXRuntime 库，计算结果合并与排序通过 PySpark 的 MapReduce 过程来实现。

2.1　数据预处理技术

2.1.1　常见的数据预处理方法

在机器学习中数据集预处理归属于特征工程的一部分。模型性能由数据质量直接决定。在采集获取的真实数据中，通常会存在各式各样的缺陷，包括数据值缺失、自然噪音、由于人工操作失误导致的异常点等。这些缺陷有害于算法模型的训练，可能导致损失函数曲线无法收敛。

常见的数据预处理方法通常包含以下几种：归一化、标准化、中心化、数据扩增等 [18, 19]。

归一化是指将样本值按照一定的比例进行缩放，使其落入某个指定的区间中。此外，归一化会消去数据的量纲限制，将其转换为纯数值，这就可以将不同种类的特征进行比较或加权。在模型的训练阶段，归一化会带来如下的两个优点：提升模型训练参数的收敛速度和提升模型的预测精度。前者是因为归一化后在梯度更新时，优化器能快速找到正确的梯度下降方向，而不会产生“之”字形路线，这会显著地提高损失函数的收敛速度，如图 2-1所示。后者是因为当各特征值的取值范围相差较大时，那么较小取值范围的特征维度对结果所产生的权重也就越小，这就会导致一定的精度损失，因此数据集经归一化处理

[①]https://github.com/onnx/onnx

后，均衡了各特征维度对决策结果的贡献，有效地提高了模型的泛化能力。

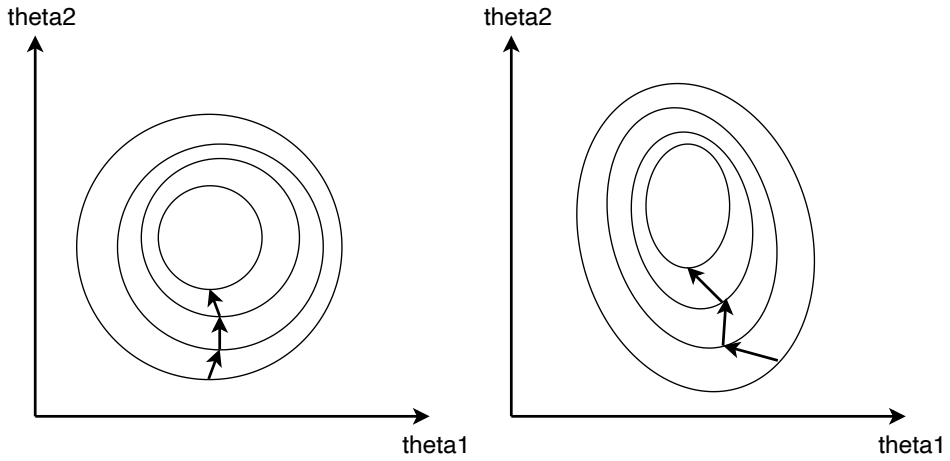

图 2-1: 标准化与未标准化的梯度更新曲线

归一化的方法包括将一列数据变换到某个固定区间中，例如为了将原始数据的分布变换到 [0, 1] 区间中，可进行如下的线性变换：

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{2-1}$$

其中，x, 是变换后的数据，需要处理的数据用 x 表示，样本数据中的最小值用 x_{min} 表示，样本数据中的最大值用 x_{max} 表示。

标准化是指将原始数据依据均值和标准差进行变换处理，常使用 Z-score 标准化，其计算如下式所示：

$$x, = \frac{x - \mu}{\sigma} \tag{2-2}$$

其中，μ 是样本集的均值，σ 是样本集的标准差。标准化的优势在于能够将不同量级量纲的数据统一到同一个维度，以保证数据之间的可比较性，缺点是仅通过有限样本数据得到的均值和标准差会存在一定的偏差。

数据扩增是近年来一种新颖的数据预处理技术，它是指在不实际增加原始数据的前提下，通过对已有数据样本施加变换算子，生成多份数据 [20]。数据扩增通过增加训练阶段的样本数、平衡不同种类样本分布不均、丰富数据多样性，以此提升模型性能。数据扩增的原则是不能引入与分类类别无关的样本且扩增后的样本标签不发生变化。目前，已有数据扩增的方法可以分为单样本扩增和多样本扩增这两类。单样本扩增包括：图像旋转、仿射变换、灰度化、图像裁切、高斯模糊、风格转换以及生成对抗网络等；多样本扩增包括

SamplePairing，图像融合等，通过在高维空间按照一定的规则对若干张图片进行合成，从而产生新的图像 [21]。实验证明，通过使用数据扩增技术，能有效地提升模型在训练集上的表现。这种方法适用于数据样本不足时的情形。

2.1.2 数据预处理在本系统中的应用

在本系统中，默认提供了若干种常见的数据预处理操作算子，包括归一化、灰度化、均衡化、异常样本检测等。但是，数据预处理的思路丰富多样，尤其是针对用户私有数据集。为了能够让用户的预处理需求得到充分地表达，本系统结合领域驱动设计，提供了用于自定义预处理算子的领域特定语言（Domain Specific Language，DSL），用户可以根据实际的预处理需求，基于 DSL 编写相应的预处理操作定义，系统会在预处理阶段进行解析，并将其插入到预处理模板代码对应的抽象语法树中，最终利用 astor 包将修改后的抽象语法树转换为 Python 代码 [22]。在执行度量任务时，如需使用到该预处理逻辑，系统会调用相关的 Python 代码，以完成指定的数据预处理操作。

2.2 基于统计信息的测试用例度量算法

2.2.1 信息熵与基尼不纯度

熵的定义来自香农的信息论，是指接收到每条信息中包含信息的平均量 [23]。信息的定量化研究一直被科研工作者所关注。Hartley 作为信息论的先驱，他首先研究并提出采用消息出现概率的对数测度作为离散消息的信息度量单位。例如，给定一个具有 N^m 种取值的等概率信源，其信息度量公式为

$$I = \log N^m = m \log N \tag{2-3}$$

由于包含信息的符号最主要的特征是不确定性，而客观信源概率统计上的不确定性是其主要来源，因此上述公式可作为等概率信源条件下信息度量的一个特例。即对于事件集合 X，其中某个随机发生的事件 $X = x_0$ 的概率为 $p(x_0)$，则这个事件所包含的信息量为

$$I(x_0) = \log \frac{1}{p} = -\log p(x_0) \tag{2-4}$$

其物理意义是事件发生的概率越高，则事件所包含信息量越小；反之，事件发生的概率越低，则说明该事件蕴含的信息量越大。

根据如上信息量的定义以及熵的定义，衍生出信息熵的概念。信息熵用来度量随机变量不确定性，是所有信息量的期望。信息熵 H 可表示为

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \tag{2-5}$$

信息熵可以表示一个事件集合的不确定程度。熵越大，表示不确定性越大。

基尼不纯度用来衡量一个数据集 D 的纯度，被定义为一个随机选中的样本在子集中被分错的可能性 [24]。由以上定义，可得到基尼不纯度的表达式：

$$\begin{aligned} I_G(p) &= \sum_{i=1}^{n} p_i \sum_{k \neq i} p_k \\ &= \sum_{i=1}^{n} p_i(1 - p_i) \\ &= \sum_{i=1}^{n} (p_i - p_i^2) \\ &= \sum_{i=1}^{n} p_i - \sum_{i=1}^{n} p_i^2 \\ &= 1 - \sum_{i=1}^{n} p_i^2 \end{aligned} \tag{2-6}$$

从信息熵的角度出发，它也表示一个事件集合的不确定程度，因此也可作为度量样本集合纯度的指标之一。若 n 表示总类别数，$p(x_i)$ 表示第 i 类别样本的出现概率，那么当集合中每类样本的出现概率接近时，集合的混乱程度越高，其纯度也越低。事实上，如果在 $p(x_i) = 1$ 处对 $-\log p(x_i)$ 作一阶泰勒展开，可以得到

$$-\log p(x_i) \approx -\ln p(x_i) \approx 1 - p(x_i) \tag{2-7}$$

此时，信息熵的表达式可以近似表达为

$$H(X) \approx -\sum_{i=1}^{n} p(x_i)(1 - p(x_i)) = 1 - \sum_{i=1}^{n} p(x_i)^2 \tag{2-8}$$

可以看出，基尼不纯度可近似为信息熵公式在 $p = 1$ 处进行一阶泰勒展开的结果，因此在实际计算中，信息熵与基尼不纯度均可用来衡量数据集的纯度值。

2.2.2 测试用例度量算法的基本思想

测试用例度量的核心目的是挑选高价值的测试用例，这些测试用例能够帮助开发人员快速地揭示系统中的 bug 或者定位 bug，而无需像回归测试，只有运行全部的测试用例才知道问题所在。在传统的软件工程中，通常采用代码覆盖率作为测试用例度量的标准，例如分支覆盖率、语句覆盖率等 [25]。覆盖率之所以能够在传统软件中的测试度量表现中取得良好的效果，是因为在传统软件中，与产品息息相关的业务逻辑都被显示地用代码进行描述，因此，当测试用例能够覆盖该条语句或者分支时，就表示这部分业务逻辑已经被测试用例的设计者考虑到，所以覆盖率高的测试用例其度量结果的优先级也更高。但是，在许多使用神经网络进行决策的深度学习软件中，逻辑不再被代码显示地表达出，而是蕴含在若干神经元的权重参数中，这就直接导致覆盖率度量指标无法在深度学习软件中发挥作用 [26]。事实上，实验也能够证明，当神经网络对单个测试用例进行推理时，99% 的神经元都被激活，因此需要从新的角度出发来解决这个问题。

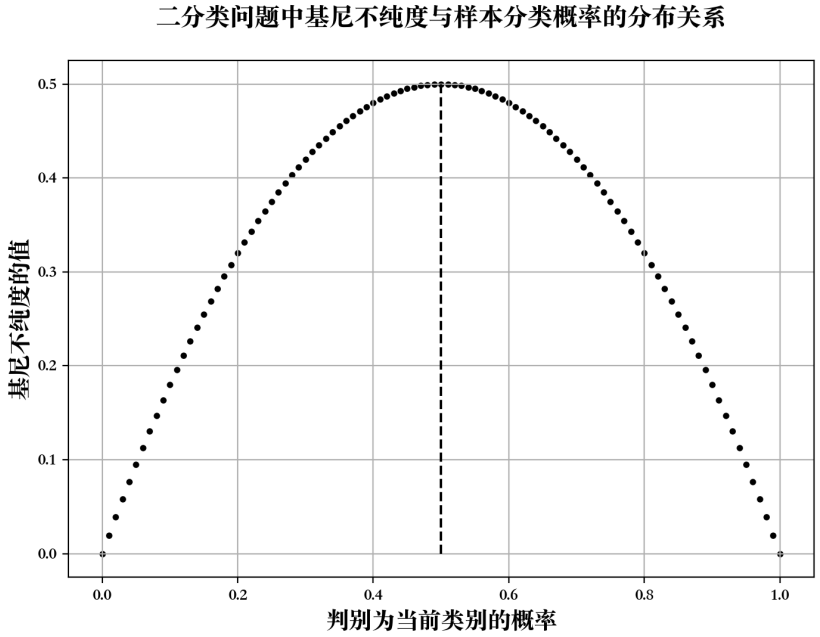

图 2-2: 二分类问题中基尼不纯度的分布

考虑神经网络的输出层向量，经由 Softmax 函数计算后，该向量中第 k 个数值的物理意义是当前样本被分类为第 k 类的概率。那么当某个样本的输出向量中存在一个明显大于其他元素的值时，可以认为神经网络对该样本的分类

具有十足的“信心”；反之，当某个样本的输出向量中所有元素的值都接近时，则说明神经网络并不确定该样本应该被分为哪一类。考虑二分类的情况，如图2-2所示，其中X轴表示该数据样本被分类为其中某一类的概率，Y轴表示当前概率下的基尼不纯度的值。从中可以看出，当该数据样本被分类为其中某一类的概率为1时，基尼不纯度的值为0，说明模型对于该样本能够很好地进行分类；当数据样本被分类的概率为0.5时，基尼不纯度的值达到极大值，说明模型并不确定该样本应该被分为哪一类，即该样本的度量优先级较高。

事实上，通过主成分分析等降维手段将决策边界可视化后，前一种情况对应于高维空间中数据样本点在分类区域的中间部分，后一种情况对应于高维空间中数据样本点在分类区域的边界线上[27]，因此可以认为类似这样处于分类边界线上的数据样本具有较高的利用价值。

2.2.3 测试用例度量算法的实现

经上述讨论，基于基尼不纯度的测试用例度量算法实现可分为如下几步：

1. 加载并预处理该数据集；
2. 利用模型在该数据集上完成推理计算；
3. 获得模型的输出向量值并经softmax函数计算，得到分类判决概率向量；
4. 根据基尼不纯度公式及分类判决概率向量，计算所有样本的度量值；
5. 对所有的数据样本进行排序，排序的键是每个样本的基尼不纯度值；

2.3 MapReduce 编程模型

2.3.1 MapReduce 的基本思想

Jeff等人在2004年时提出了一种用于大数据处理的MapReduce编程模型[28]，目的是降低应用层开发人员利用并行计算技术的门槛。MapReduce已在Google多领域业务中得到广泛使用，包括搜索引擎中有关倒排索引的计算、分布式排序算法、计数、网页链接分析PageRank等[28, 29]。MapReduce编程模型定义了一对抽象操作：map算子和reduce算子。map算子用来处理一对键值对并产生一系列的中间结果键值对，reduce算子将其这些中间结果按照键的不同进行合并。真实世界中的很多问题都可以抽象为这两种操作，如图2-3所示为利用MapReduce解决文本词语计数问题的伪代码。用户通过对已有问题进行

建模，将问题的解决方案编写为 map 和 reduce 算子，框架就可以自动并行化该操作并在大规模的分布式集群中来运行，用户无需关心输入数据划分、任务调度、机器间通信等底层实现细节，不仅提高了大数据任务的执行速度，同时降低了应用层工程师使用并行技术的门槛。

```
map(string key, string value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(string key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

图 2-3: MapReduce 模型解决词计数问题的伪代码

2.3.2 MapReduce 在本系统中的应用

在本系统中有关信息熵的计算中，将采用 Apache Spark[①]框架来实现该部分的计算。Spark 是一款针对大数据处理的通用计算引擎，由 UC Berkeley AMP Lab 在 Hadoop MapReduce 的基础上开发完成。相比于 Hadoop MapReduce，Apache Spark 直接将计算结果保存在内存中，无需冗余地读写 HDFS，使得 Spark 的运行效率大幅度提高。Spark 的弹性分布式数据集使得其支持高度迭代计算和响应式编程，更加适用于机器学习与数据挖掘等迭代计算范式。

在获取到神经网络模型针对该数据集的推理结果后，系统开始执行每个样本度量标准值的计算。在本系统中，map 函数的处理逻辑如下：首先获得该数据样本的 ID 值以及输出向量具体数值，然后根据基尼不纯度的公式计算出该样本的度量标准值，最后将数据样本 ID 以及计算出的度量标准值作为新的键值对提交到系统。reduce 函数的处理逻辑如下：根据获取到的中间结果键值对，按照样本的度量值进行归并排序操作并将结果写入磁盘。最终得到的结果是根据样本度量值大小排序的样本 ID 列表。此外，为辅助最终度量报告的生成，系统在此过程中会按顺序同时记录下每个样本的输出向量、基尼不纯度以及前 10 位与末 10 位样本数据的可视化结果。

[①]https://spark.apache.org

2.4 基于有向无环图的调度中间件 Airflow

Apache Airflow 是 Airbnb 开源、用来编排、调度和监控工作流的中间件平台。Airflow 将工作流建模为以特定算子为节点的有向无环图，调度一组预先配置的 workers 按照指定的依赖关系进行执行。此外，Airflow 提供了简洁易用的监控和报警系统，以方便用户查看相应工作流的执行状态。

在 Airflow 中，定义了如下的核心概念：

1. DAGs：即有向无环图，图的点元以算子为粒度描述业务逻辑，图的线元描述点元之间的依赖关系；
2. Operators：描述了 DAG 中点元对应的算子，算子具有原子性。Airflow 内置了丰富的算子。用户也可按照业务需求自定义或组合若干已有的算子；
3. Tasks：是 Operator 类实例化的结果，对应具体 DAG 中的一个节点；
4. Task Instance：DAG 中点元的一次运行实例，每个实例保存相关的状态；

用户可以通过组合各种 Operator 来实现复杂的工作流。此外，在某些任务中需要获取外部系统的数据，Airflow 通过定义 Connection 的概念，用以建模并抽象与外部系统的连接。在很多工作流中，通过 XComs 键值对数据库，可以将不同节点之间的状态信息进行共享。Xcoms 通常用来记录工作流执行过程中的中间结果。在集群化的部署下，通常使用外部系统来负责 Airflow 中工作节点资源的分配工作，例如在使用 Celery[①]中间件时，其系统架构如图 2-4所示。 使用

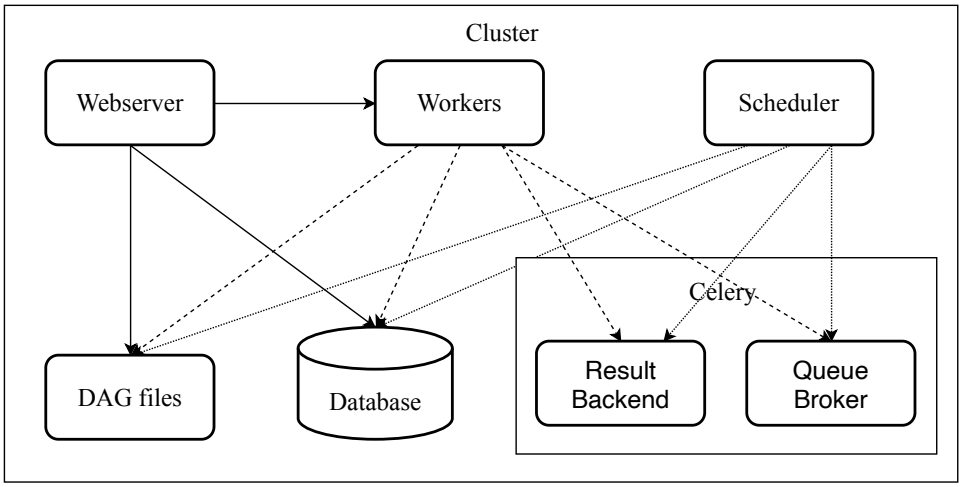

图 2-4: 基于 Celery 的 Airflow 架构图

Celery 中间件的优点是系统中的工作节点无需依赖于 Scheduler 节点，不存在单

[①]https://github.com/celery/celery

点故障，同时能够提升系统负载能力。但是，由于 Celery 的限制，系统中工作节点的个数在启动后是无法改变的，因此无法负担超过系统负载能力的工作流任务，同时在系统闲置时，资源也得不到有效利用。

因此，为了使得系统具备高度弹性可扩展的特点，通常使用基于 KubernetesExecutor 的弹性结构，由 Kubernetes 集群来负责工作节点的资源分配与回收工作。在 Kubernetes 的基础上，单个工作节点的任务调度完全可以根据系统的负载、任务所需的资源进行弹性分配，极大地提高了系统资源利用的效率。但是，由于弹性地资源分配，会导致任务执行存在一定的时延。考虑到本系统的处理对象是大规模的数据度量任务，属于批处理系统，对实时性的要求不高，因此由于弹性调度引起的响应时延可以容忍。

2.5 开放神经网络交换格式 ONNX

Open Neural Network Exchange(ONNX) 开放神经网络交换格式，旨在优化机器学习模型的推理过程，简化模型部署到生产环境所需的第三方依赖。从一般意义上讲，机器学习模型的推理计算优化困难重重，根本原因在于推理库和硬件环境的多样性。若要模型在每个平台都达到最佳性能，则需要工程师花费大量的精力针对不同的模型在不同的平台（CPU、GPU、ASIC）进行调优工作，对框架和硬件的所有不同组合逐一进行优化非常耗时，这会极大地增加工程的复杂性。ONNX 的出现给这些问题带来了可行的解决方案。

ONNX 运行时是一款高性能推理引擎，用于将 ONNX 模型部署到生产环境，并针对不同的硬件平台进行优化，适用于多个操作系统平台，它由 C++ 语言编写，并同时了提供 Python、Java、JavaScript 等多种语言的 API 接口。ONNX 运行时支持深度学习模型和传统机器学习模型，并与不同厂商硬件加速器集成，例如 Nvidia GPU 的 TensorRT。ONNX 作为开放的接口规范，是人工智能模型建立开放式生态系统的第一步，众多其他深度学习框架均提供了对 ONNX 格式的转换支持，这使得已有模型能够在不同的框架之间无缝迁移。

ONNX 在编码格式中进行了如下定义：计算图模型、内置运算符以及标准数据类型。ONNX 格式中最顶层的结构是 Model，用于关联元数据与计算图，计算图中包含所有可执行的元素。元数据记录模型的属性，包括输入输出特征、算子版本、模型版本等。然后，ONNX 模型必须明确它所依赖的运算符集。运算符集定义了当前版本可用的操作符，所有模型会默认导入通用的运算

符集合，每个模型还会导入其域上自定义的运算符集合。接下来，ONNX 模型结构是由序列化的计算图组成，每个计算图由一组元数据字段、一组模型参数列表以及计算节点列表组成。所有的计算节点按照拓扑序排列，形成有向无环图，每个节点允许零个或多个输入以及一个或多个输出。计算图上的节点由名称、所调用的运算符的名称、输入列表和输出列表、属性列表组成 [30]。节点输出的值由节点的运算符计算，节点输入的值可以指定上一层节点输出、初始化器。根据以上的定义，ONNX 实质上提供了一种统一的中间表示，无论原模型使用的是静态图（TensorFlow[①]、Caffe2[②]）或者是动态图（PyTorch[③]、Chainer[④]，都可以转换为 ONNX 编码格式，从而方便在不同端设备上的部署和性能调优。

2.6 容器编排引擎 Kubernetes

Kubernetes 是由 Google 开源，用于集群环境中管理容器化应用的系统，旨在提供一种更加友好便捷、跨基础设施的资源管理服务 [31]。具体来说，Kubernetes 是一个用来管理容器全生命周期和调度容器运行的平台。作为用户，可以指定你的上层应用如何运行、如何与外界进行交互，可以根据当前系统的负载将服务扩容或缩容、可以在不停机地状态下实现滚动更新、可以在不同版本的应用中切换流量以进行 AB 测试等。总体来说，Kubernetes 为运维人员提供了高度灵活、可靠的容器管理平台。

Kubernetes 采用了主从架构，由集群中一台服务器担任 Master 节点。Master 节点负责担任网关的角色，通过对外暴露 API 的方式来提供服务。Master 节点的工作包括定时心跳包的检测、任务调度、协调组件间的通信等。为了避免单点故障，通常集群的设计者会将 Master 节点作高可用，例如使用另一台机器作为 Master 的热备份节点，该节点随时与主节点作状态同步更新。除主节点外，集群中其他机器均为普通节点。普通节点的主要任务是利用本地资源或外部资源来接收任务或运行任务。为了保证资源的有效利用以及应用运行安全，Kubernetes 利用了大量的虚拟化技术，以保证资源的高度隔离性，同时应用本身也无需关注运行环境，为后续的滚动更新以及流量切换提供了基础。用户通常是以配置文件的方式来描述所需要完成的任务以及所需的资源，以

[①]https://www.tensorflow.org
[②]https://caffe2.ai
[③]https://pytorch.org
[④]https://chainer.org

JSON 或 YAML 的格式提交到 Master 节点。Master 节点会根据该配置文件以及当前系统的状态对该任务进行合理的工作调度，以保证上层运用的平稳运行。

Master 节点是用户与系统进行通信的入口点，它负责接受用户的请求、调整集群网络部署、管理资源扩容与缩容等。通常使用 etcd 协议持久化相关配置信息，例如实现服务发现或者同步工作节点间的配置文件。etcd 集群同时负责领导选举并提供分布式锁服务。apiserver 担任整个集群管理的入口角色，并作为不同组件的通信渠道和信息分发的桥梁。kube-controller-manager 是用来管理集群状态的通用服务，控制器通过监测 etcd 上键值的变化来作出相应的应对行为，包括容器启停、弹性扩展等操作。kube-scheduler 根据负载均衡的策略，将任务调度到合理的节点之上。

工作节点的主要职责是通过容器来完成用户所提交的任务，因此容器的运行时环境成了必不可少的一部分，目前业界通常使用 Docker 或 containerd。容器运行时负责管理容器的启动和停止，同时实现不同容器间的资源隔离。kubelet 是工作节点用来获取集群信息、控制台信息等的入口。例如，工作节点需要借助 kubelet 来实现与 Master 节点之间的信息验证工作。

在本系统中，Kubernetes 被用来负责整个系统资源的管理工作，包括提供 Airflow Web 端运行的容器、Airflow 元数据管理的数据库容器、Airflow 执行任务时所需的工作节点容器。用户提交度量任务后，Kubernetes 会根据当前负载以及任务所需资源进行评估，分配出若干的 Pods 负责度量任务的执行。在任务完成后，Kubernetes 会回收对应资源，以保证对已有资源的高效利用。

2.7 本章小结

本章主要对在系统实现时选择和采用的技术方案进行了简单的介绍和描述。首先，介绍了常见的数据预处理算法。接着，介绍了本系统中基于基尼不纯度的度量算法，包括算法的关键思想和关键步骤。然后，介绍了在大数据处理中常见的 MapReduce 编程模型以及如何在本系统中得到应用。最后，介绍了为支持本系统实现所采用的一些第三方技术和中间件。

第三章　需求分析与概要设计

本章节将概述面向神经网络的测试用例度量系统的需求分析与设计。首先，介绍项目背景、整体情况、系统开发目的以及系统提供的功能。接着，从功能和非功能角度分析系统的需求并以此得出用例图。其次，介绍系统总体架构设计、4+1 视图以及系统不同模块边界的划分。最后，依次对系统各模块的架构设计与概念类图进行详细描述。

3.1　系统整体概述

如今，大量复杂、精巧的深度神经网络模型被算法科学家设计出并被应用到多个业务场景下，取得了不错的收益。但是模型的迭代和调优工作往往困难重重，主要集中在模型编码格式不统一、数据采集和标注成本高等。一般情况下，项目经理需要雇用外包人员或者通过众包方式来完成数据标注的工作，同时开发者需要将预训练的模型文件转换为团队内所采用的框架编码格式后才能开始调优。这种分散、琐碎、重复的处理工作阻碍了软件项目的正常开发，并且提高了整个软件项目的开发成本。

面向深度学习的测试用例度量系统提供了一个端到端的解决方案，通过内嵌主流深度学习框架的库文件，能够兼容多种框架下的模型制品。在数据预处理方面，系统支持多种常见的预处理操作并支持自定义预处理操作。本系统通过计算神经网络对不同用例的度量值，确定该用例对提升网络泛化能力的贡献，以此为标准从海量测试样本中挑选出高价值用例，从而有效地减少模型优化过程中所耗费的数据标注成本。此外，相比于使用完整数据集或随机挑选，使用高优先级的样本数据能够使训练的损失函数曲线更快地收敛。实验结果显示该方式下训练出的模型在测试集上的表现更加优秀。

为实现上述功能，系统需提供若干个功能需求。模型编码转换功能用来解决多种深度学习框架的兼容性问题。考虑到集群在执行任务节点时需拉取镜像并创建容器，因此不适宜将若干个深度学习框架库均集成到镜像中。原因之一为不同框架的版本更新会导致镜像的频繁更新，原因之二为过于臃肿的镜像文

件会拖累容器起停的效率，降低系统的吞吐量。数据集预处理功能用来解决用户如何在私有数据集上编写并执行自定义预处理操作的问题。考虑到数据集采集形式、编码格式的多样性，所对应的预处理方法也纷繁芜杂，因此需提供完备的接口以供用户自由定义多种预处理操作。度量算法功能是本系统的核心功能，通过计算基尼不纯度来实现不同数据样本之间优先级的度量。度量报告生成功能从多角度对度量结果进行分析，帮助用户检查数据预处理过程与特征工程是否完备。计算引擎功能支撑整个系统的运行，通过采用若干云原生技术，使得系统具备弹性可伸缩性，提升系统资源利用效率，保障系统平稳运行。

3.2 系统需求分析

3.2.1 功能性需求

本系统的需求分析主要分为 5 个部分，即模型编码转换、数据集预处理、度量算法、度量报告生成及计算引擎。具体如表 3-1所示。

表 3-1: 功能需求列表

需求 ID	需求名称	需求描述
R1	模型编码转换	将用户提交的模型文件转换为 ONNX 编码文件。如果转换发生错误，将返回错误提示信息。
R2	数据集预处理	将用户提交的数据集文件，如 URL、压缩包等，转换为 npy 编码格式。同时在数据集上作用用户指定的预处理操作。
R3	度量算法	根据基尼不纯度的计算公式，得到每个数据样本的度量标准值，并以此作为样本排序标准。
R4	度量报告生成	用户可以看到测试用例度量结果，包括各测试用例的基尼值、排序后的测试用例名列表。
R5	计算引擎	负责资源分配及回收、镜像与配置文件托管、容器创建与销毁、算力评估与调度、日志记录。

在创建度量任务时，用户除需要提交模型与数据集文件外，还需定义预处理操作。系统会根据领域特定语言的语法解析该操作并生成对应的 Python 代码，以供系统调用。

在度量任务执行完毕后，用户可以查看关于此次任务的分析报告，其中包括模型、数据集概述、多种图表以及部分数据样本的可视化结果等，帮助用户分析数据预处理及特征工程的缺陷。

3.2.2 非功能性需求

非功能性测试是系统需求设计中不可或缺的一部分，包含了系统可用性、可靠性、易用性等，将会对系统的架构设计和技术选型产生直接的影响。

表 3-2: 非功能需求列表

安全性	系统应该保证全局只有唯一的入口。
	系统应保证数据在传输过程中进行加密保护，不造成用户数据的泄漏。
健壮性	系统内部发生故障时，应该有相应的处理机制，不能导致服务崩溃。
可靠性	当系统发生故障时，系统应能够在 1 小时内恢复。
可扩展性	系统架构设计需保留扩展点，以便后续需求增加时简单修改即可使用。
可维护性	系统应使用日志记录系统。对于创建任务、节点执行结果等关键部分，均有相应日志记录在案，以供查询。
性能	系统的承载能力与机器数量呈线性相关。
	系统在单机下能够支撑 10~20 个度量任务的并发执行。

本系统是面向神经网络的测试用例度量系统，目的是依赖于数据集和模型的统计信息，寻找能够使模型决策出现错误的测试样例。由于系统的使用用户在提交任务后，并不需要实时地与系统进行交互，因此本系统属于批处理范畴。本系统应能够支撑大规模数据集的度量工作，并支持多任务并发执行。本系统的具体非功能性需求如表3-2所示。

3.2.3 系统用例图

根据以上功能性需求和非功能性需求的描述与分析，可以得到系统用例图。本系统的用户主要是具备深度学习领域知识的软件工程师，如图3-1所示为系统用例图。

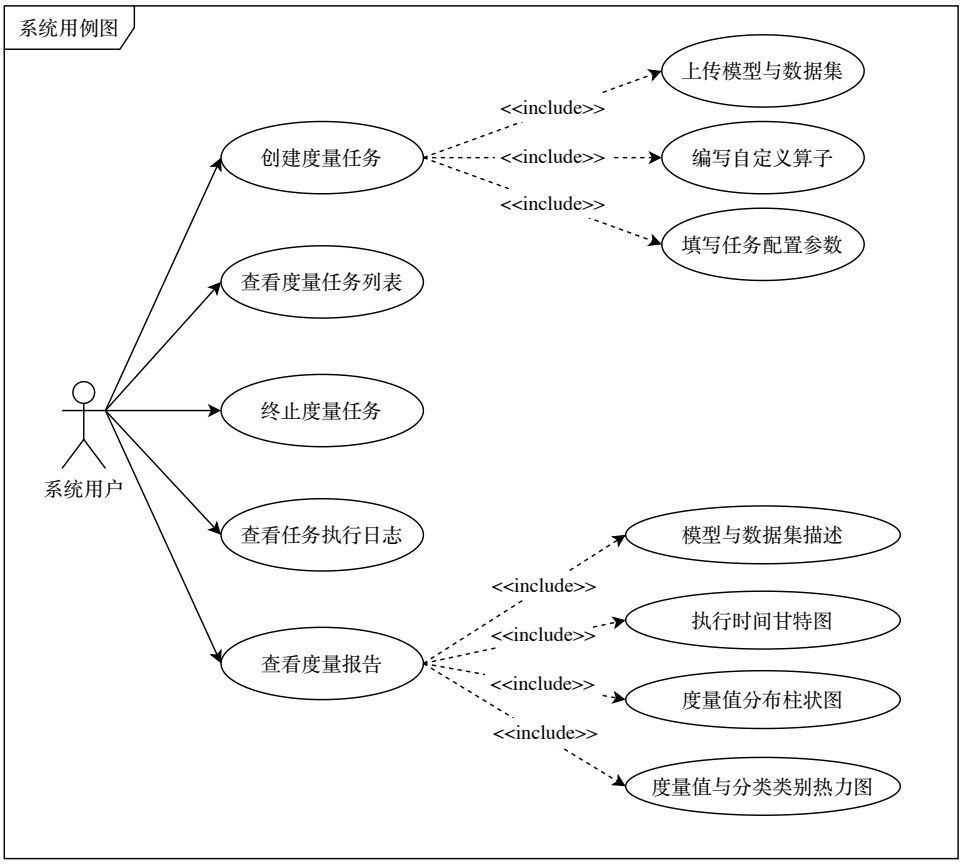

图 3-1: 系统用例图

本系统主要包含 5 个用例，分别为创建度量任务、查看度量任务列表、终止度量任务、查看执行日志以及查看度量报告。

3.2.4 系统用例描述

本小节将就上述 5 个用例进行详细的描述，包含以下部分：参与者、触发条件、前置条件、后置条件、正常流程、扩展流程。

创建度量任务功能，用户在登录系统后，点击创建度量任务按钮，会进入到创建度量任务的界面。用户需要上传度量所用的数据集与模型。若用户使用了原始数据集，则需要自定义预处理算子框中填入相应的预处理实现。通过系统提供的领域特定语言，用户可以根据示例编写出相应的预处理逻辑。考虑到部分预处理算子的常用性，系统提供了一部分常见的预处理算子，包括归一化、中心化、灰度化等。用户还需要配置一些额外的参数，包括度量算法、模型编码格式、数据集编码格式等。在相应信息填写完毕后，用户点击确认创建按钮，即可向系统后台提交度量任务。创建度量任务功能用例如表 3-3所示。

表 3-3: 创建度量任务用例描述

ID	UC1	用例名称	创建度量任务
参与者	系统用户		
触发条件	需要创建新的度量任务		
前置条件	用户已被认证并具备该操作权限		
后置条件	无		
正常流程	1. 用户点击侧边栏创建度量任务选项并跳转至指定页面; 2. 用户填写度量任务的名称; 3. 用户上传需要度量的模型文件与数据集文件; 4. 用户选择具体的度量算法; 5. 用户选择是否需要自定义预处理算子; 6. 用户点击确认创建按钮以提交请求;		
扩展流程	6a. 用户需要自定义预处理算子; 1. 在弹出的文本框内按照指定的接口编写预处理实现;		

表 3-4: 查看度量任务列表用例描述

ID	UC2	用例名称	查看度量任务列表
参与者	系统用户		
触发条件	用户需要查看已创建的度量任务		
前置条件	用户已被认证并具备该操作权限		
后置条件	无		
正常流程	1. 用户点击侧边栏创建度量任务选项并跳转至指定页面; 2. 用户完成度量任务的创建; 3. 用户点击侧边栏中查看度量任务选项并跳转至指定页面; 4. 用户可以在度量任务列表中查看到已经创建的度量任务;		
扩展流程	无		

查看度量任务列表功能，点击左侧边栏的查看度量任务选项，进入到度量任务列表界面。所有属于该用户的度量任务都将以表格项的形式展现。在单个表项中，用户可以获取该度量任务的相关信息，包括度量任务 ID、度量任务名称、模型名称、数据集名称、度量算法、任务执行状态、任务创建时间等。在表格最后一列，系统提供查看度量任务执行日志、终止度量任务、查看度量报

告的选项。在度量任务的执行完毕后，末列将出现查看度量报告的选项。通过该选项，用户可以查看到该度量任务所对应的度量分析报告。为方便用户快速查找相关表项，系统在度量任务列表上方提供搜索框。用户可以在搜索框中键入度量任务的名称进行查询。查看度量任务列表功能用例如表3-4所示。

终止度量任务功能，用户在提交度量任务后，如果因配置信息填写错误而需要取消，则可以通过点击度量任务列表界面末列的终止任务按钮，即可向系统后台发出终止该度量任务的请求。当 Airflow 接收到请求并成功终止任务后，系统会更新该表项中执行状态为已停止，说明度量任务已经被取消。已完成的度量任务无法被终止。终止度量任务功能用例如表3-5所示。

表 3-5: 终止度量任务用例描述

ID	UC3	用例名称	终止度量任务
参与者	系统用户		
触发条件	因操作错误等原因，用户需要终止已经创建的度量任务		
前置条件	用户已被认证并具备该操作权限		
后置条件	系统后台销毁对应 ID 的度量任务执行实例		
正常流程	1. 用户点击侧边栏中查看度量任务选项并跳转至指定页面； 2. 用户通过搜索框等方式查找到需要终止的度量任务表项； 3. 在该表项的操作列，用户点击终止任务按钮并确认，即可终止待执行或执行中的度量任务；		
扩展流程	无		

查看度量任务执行日志功能，用户在完成度量任务创建后，若希望查看该任务的具体执行情况或在任务执行出错时查找原因，可先在查找度量任务列表中找到对应度量任务的表项，并在操作栏点击执行日志按钮，即可跳转到该任务的详情界面。在该界面中，用户可以点击对应有向无环图中操作节点，以查看具体节点的执行日志。查看度量任务执行日志用例如表3-6所示。

查看度量报告功能，用户在度量任务完成后，可以点击查看度量任务列表中的度量报告按钮，即可查看到对应任务的度量报告；在度量分析报告中，给出了模型与数据集的相关分析、度量任务各节点的执行时间甘特图、度量标准值分布的柱状图、度量标准值与分类类别的热力图表示以及度量后前 10 个与末 10 个数据样本的可视化结果。上述图表均为响应式组件，用户可以通过鼠标拖拽等方式动态地检索图表细节。通过该分析报告，用户可以了解到神经网络

表 3-6: 查看度量任务执行日志用例描述

ID	UC4	用例名称	查看度量任务执行日志
参与者	系统用户		
触发条件	用户需要查看已创建的度量任务的执行情况		
前置条件	用户已被认证并具备该操作权限		
后置条件	系统返回对应任务 ID 的执行日志		
正常流程	1. 用户点击侧边栏中查看度量任务选项并跳转至指定页面; 2. 用户通过搜索框等方式查找到相应的度量任务表项; 3. 点击该表项操作栏中的执行日志按钮，跳转到任务详情页面，在日志文本框中将显示相应的执行日志信息;		
扩展流程	无		

对于哪些类别的样本分类表现较差，并通过可视化结果检查数据特征工程的效果。样本度量结果展示功能用例如表 3-7所示。

表 3-7: 查看度量报告用例描述

ID	UC5	用例名称	查看度量报告
参与者	系统用户		
触发条件	用户需要查看已完成的度量任务分析报告		
前置条件	用户已被认证并具备该操作权限		
后置条件	系统返回对应任务 ID 的分析报告		
正常流程	1. 用户点击侧边栏中查看度量任务选项并跳转至指定页面; 2. 用户通过搜索框等方式查找到相应的度量任务表项; 3. 用户需要等待该度量任务执行完成; 4. 点击该表项操作栏中的度量报告按钮，将跳转到度量报告分析页面;		
扩展流程	无		

本系统在接收到用户提交的模型和数据文件后，会根据配置信息生成有向无环图，并申请计算资源来执行度量任务。本系统自动化程度高，能够合理配置和管理资源，提高了系统运维效率和任务执行的可靠性。

3.3 系统总体设计

在本节中，将按照上述的功能和非功能需求进行系统概要设计。在需求分析的前提下，对软件体系结构建模，划分并设计系统各模块。具体来说，本节将首先描述系统总体架构设计，然后给出包括场景视图、逻辑视图、过程视图、物理部署视图、开发视图这五个不同视角的系统架构视图，并以此为依据划分出主要模块。

3.3.1 系统总体架构设计

本节内容从工程实现角度设计系统整体架构，面向深度学习的测试样本度量系统的整体架构设计见图 3-2。系统主要由前端系统、后端系统、计算引擎及外部存储四部分组成。

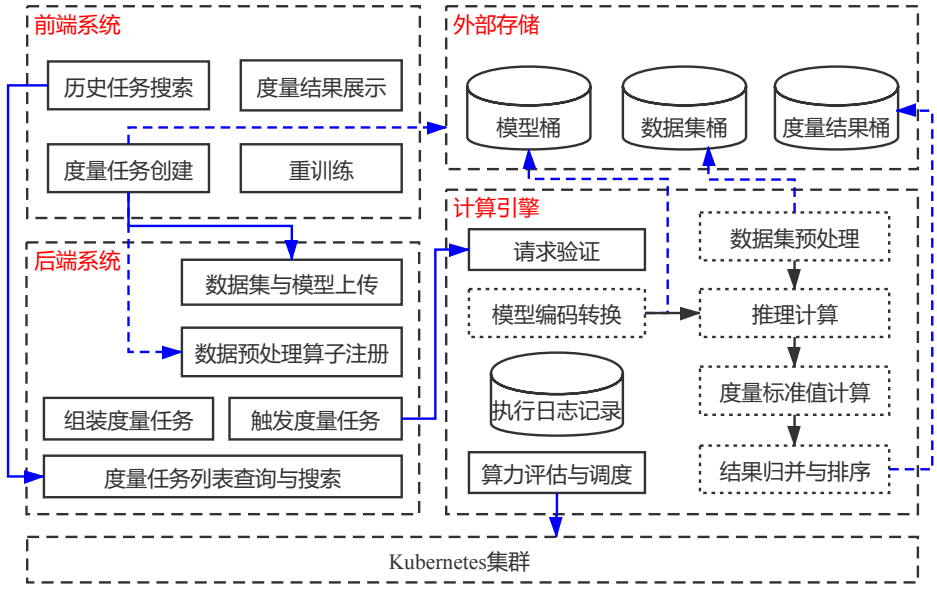

图 3-2: 系统架构图

前端系统基于 Vue.js 完成搭建，并实现前后端分离，有利于系统不同模块的高内聚低耦合。前端系统主要负责登录验证、度量任务创建、历史度量任务查询与搜索、度量结果展示以及重训练任务创建。

后端服务采用 Flask 框架进行搭建。考虑到在深度学习领域 Python 语言被广泛使用，本系统采用基于 Python 语言编写的 Flask 框架进行服务端的编码。Flask 框架由 Armin Ronacher 编写，优点在于简单快速、配置灵活、与外部第三

方组件耦合度低等。后端主要负责模型与数据集的上传下载、度量任务记录的保存与查询、触发引擎的运行等。服务层以 RESTful API 的形式与计算引擎进行通信。此外，权限管理、度量结果的加工与可视化构建均在服务端来实现。

计算引擎是本系统的核心部分，基于 Airflow 来实现。与度量任务相关的业务逻辑在模块化设计后，抽象为有向无环图中的节点，并以插件化的方式定义为 Airflow 的算子。根据度量任务中的配置信息，按照依赖关系将算子组合，形成一张具象的有向无环图。在资源管理方面，通过向 Kubernetes 集群申请所需的计算资源来完成对有向无环图的执行，保证资源用时申请、闲时释放，进一步地提高了系统物理资源的利用效率，水平扩展也使得系统具备线性可扩展性。

外部存储使用云端对象存储系统，以保证关键数据的读写效率和安全性。模型桶用来存储原始用户模型文件以及经过编码转换的模型文件；数据集桶用来存储原始用户数据集以及经预处理并压缩后的数据集文件；度量结果桶以度量结果表中的主键为名，存储对应任务的计算结果，包括具体度量值、神经网络的输出向量、样本集优先级排序后的下标值。

本系统采用容器化的方式来部署，相比于物理部署，容器消除了开发环境与生产环境的差异，保证了应用程序生命周期的环境一致性。软件工程师可拉取镜像直接系统开发与调试，运维工程师可通过镜像进行测试与发布，这极大地简化了持续集成和持续发布的流程。

3.3.2 4+1 视图

本小节将从“4+1 视图”角度出发描述本系统的总体架构设计。“4+1 视图”是由 Philippe Kruchten 在 1995 年提出，目的是从软件不同使用者的角度出发，对软件进行全面、综合、系统地刻画。“4+1 视图”包含逻辑视图、过程视图、开发视图、物理视图和场景视图。各视图的实际意义如表 3-8所示。

场景视图又被称为用例视图，通过联系其他四个视图，作为最重要的需求抽象而存在，在 UML 中与用例图相对应。因此本系统的场景视图如图 3-1所示。

本系统的逻辑视图如图 3-3所示。逻辑视图涉及最终用户，关注系统所提供的功能，主要通过 UML 总类图进行表述。

TaskCreator 用来响应用户创建度量任务的请求，并根据 TaskDao 中定义的属性创建度量任务实例。TaskQuerier 用来响应用户对已提交度量任务的查

表 3-8: 4+1 视图意义

逻辑视图	用来描述系统提供给终端用户的功能，通常使用 UML 图的类图和状态图来描述。
过程视图	重点关注系统的动力学部分，用以解释系统如何运行以及各部分组件如何交互，并考虑系统的可靠性、可扩展性、性能等。通常使用时序图、活动图来描述。
开发视图	从程序员的角度对系统进行描述与设计，通常使用 UML 的包图进行描述。
物理视图	从系统工程师的角度来描述系统，根据部署决策进行设计，用来展示系统开发后的软硬件部署的情况。
场景视图	用来描述系统用户的使用场景，用来验证架构设计以及架构原型，通常使用系统用例图来描述。

询需求，包括度量任务的创建时间、执行状态、度量结果等。ResultFetcher 提供从外部存储系统中拉取相应度量结果的信息，用以辅助度量报告的生成。CptEngService 是整个度量系统的入口处，服务端将以 RESTful API 的方式与引擎进行交互。引擎包含 4 个关键部分，分别是模型编码转换、数据集预处理、推理计算、结果归并与排序。ModelConverterOp 继承自 Airflow 中的 BaseOperator，以算子的形式参与到有向无环图的构建过程。ModelConverterHook 是负责模型编码转换的具体实现，它根据任务配置中的框架信息将转换过程转发到第三方处理句柄中，本系统目前支持 TensorFlow、PyTorch、Paddle 三种深度学习框架。DataPreprocessorOp 负责数据集拉取与处理后的写入工作，还包括解析用户自定义的预处理方法。DataPreprocessorHook 封装了对数据集预处理的具体实现，按照配置中的信息依次作用多种预处理算子，并将结果以二进制压缩的格式存储在 OSS 中。InductionOp 负责从模型推理计算的准备工作，包括拉取模型与数据集、创建运行时环境 ONNXContext 等。InductionHook 负责实际的推理计算，并运行基于基尼不纯度的度量算法，并记录相关计算的执行日志信息。MergeAndSortOp 创建 PySpark 执行环境，并拉取所需数据。MergeAndSortHook 基于 PySpark 的上下文环境，将分批的推理结果进行合并与排序，并将最终结果写入到对象存储系统中的度量结果桶中。

本系统的过程视图如图 3-4所示。过程视图的涉众为系统设计人员以及集成人员，主要关注系统的非功能需求，例如鲁棒性、可用性、性能等，旨在解

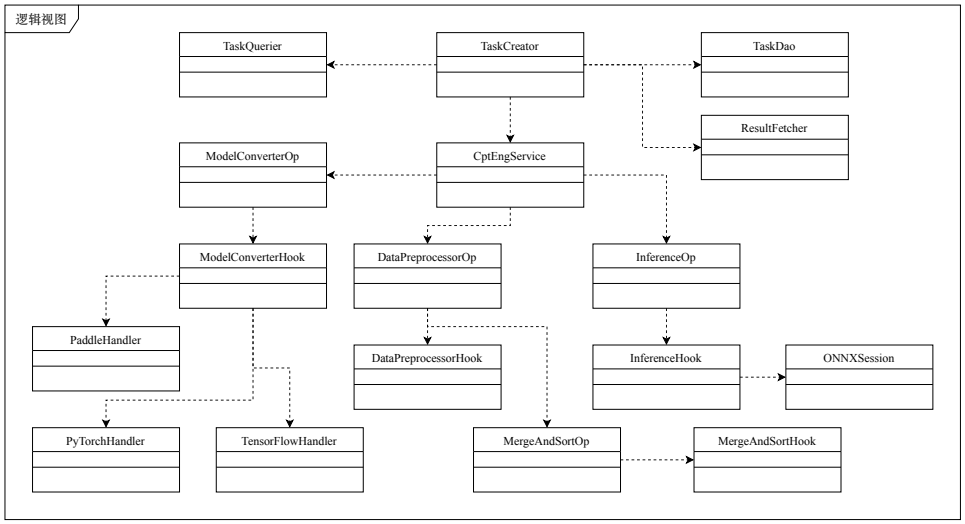

图 3-3: 逻辑视图

决系统并发、通信、同步等方面的问题。

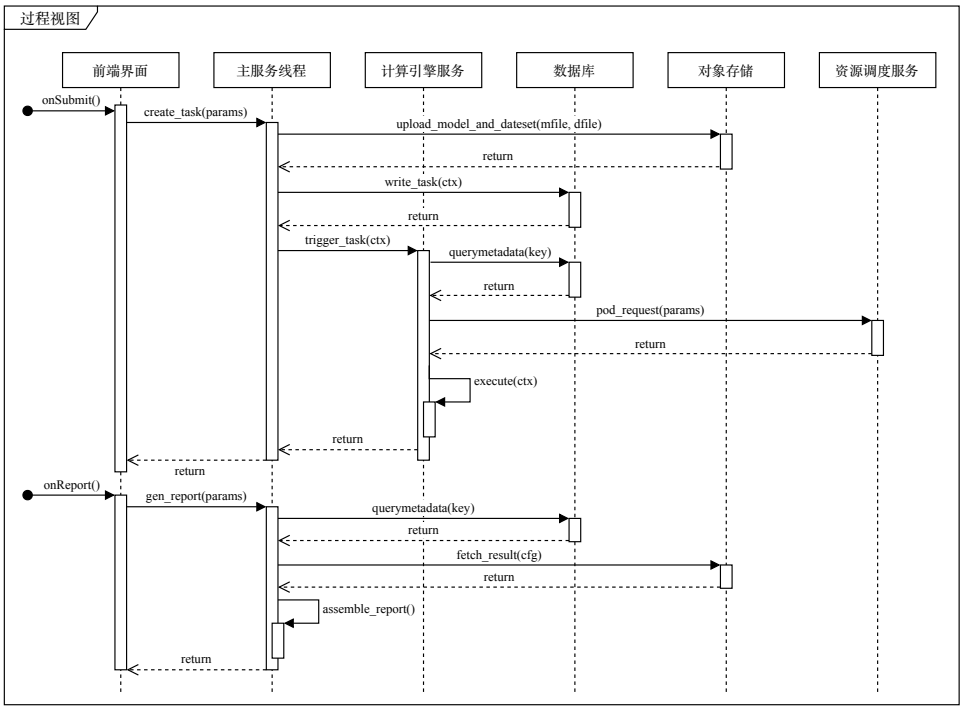

图 3-4: 过程视图

前端通过与后端服务进程通信完成度量任务创建、查询度量任务详情、度量任务检索、度量报告生成等功能。后端服务进程与计算引擎之间采用异步调用，原因是本系统属于批处理系统，任务执行时间较长，不适宜长时间进行阻塞等待。在创建度量任务时，用户所提供的数据集与模型文件将被写入到

外部存储系统中。此后，具体的度量任务参数将以表单的方式提交到后端进行处理，并在数据库中追加一条新任务的记录。后端通过 RESTful API 的方式来触发计算引擎。计算引擎需要根据任务的负担量，对算力进行评估，并从 Kubernetes 集群中申请相应的资源以实施计算。在生成度量报告时，后端服务会首先查询数据库，获得该任务所对应的度量结果 ID，并从外部存储系统中进行拉取。接着，后端相应的控制器会对数据进行再加工，为前端的页面展示作好铺垫。最后，基于相应的绘图库，在前端展示出完整的度量报告。

本系统的开发视图如图3-5所示。开发视图的涉众为开发人员和产品经理，主要关注代码结构组织、代码可重用性、可移植性，侧重于描述系统模块的组织结构和开发环境结构。

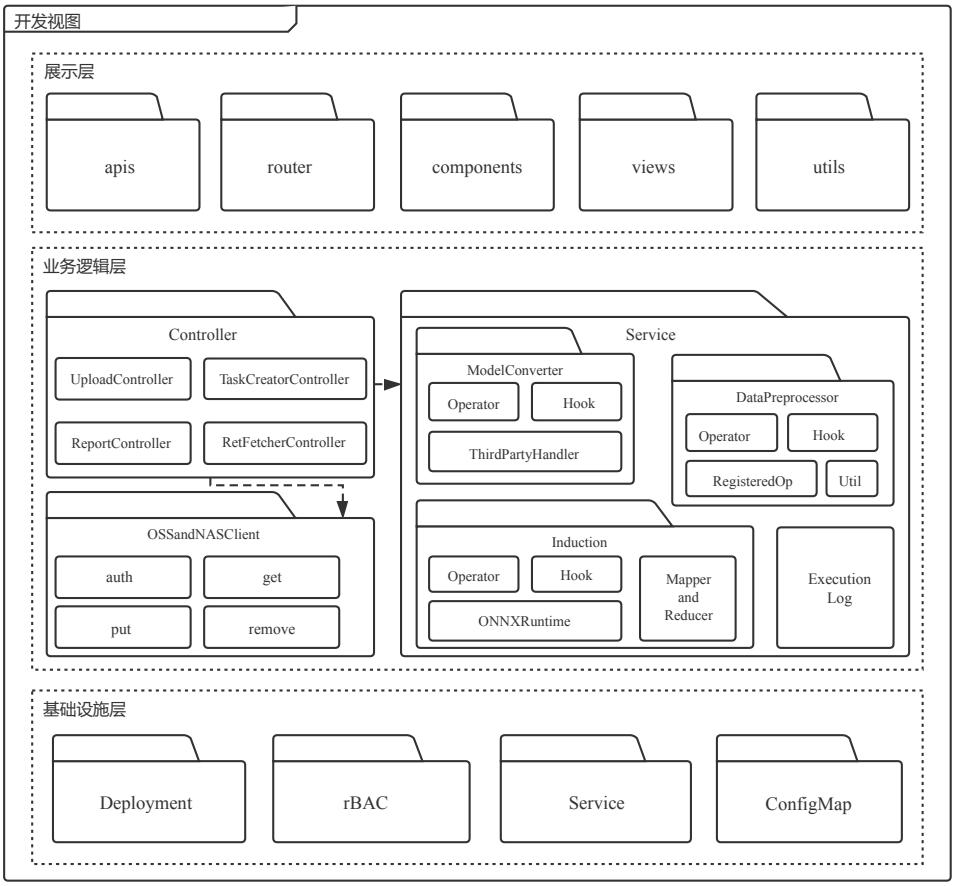

图 3-5: 开发视图

本系统的展示层包含 apis 接口封装、router 路由配置、components 容器组件、views 视图组件和 utils 的通用方法。业务逻辑层是对系统功能的划分。Controller 是业务控制器，用来负责控制业务的流程，通过调用其他服务的接

口来实现用户在前端的请求。UploadController 负责将用户上传的数据集与模型写入 NAS 系统。TaskCreatorController 负责响应用户创建度量任务的请求。RetFetcherController 负责响应对度量任务列表、度量任务详情的查询请求。ReportController 负责生成对应度量任务的分析报告。Service 是本系统的计算引擎服务。ModelConverter 负责将模型编码格式转化为开放神经网络交换格式。DataPreprocessor 负责对数据集进行指定的预处理操作。Induction 负责模型的推理计算以及度量算法的实施。在度量期间的关键操作都将写入到执行日志中。基础设施层是利用 Kubernetes 对底层物理资源的封装，通过部署、服务、鉴权等配置文件保证整个集群平稳地运行。

本系统的物理视图如图3-6所示。物理视图的涉众主要为系统的运维人员，它通过节点信息来描述软件到硬件的映射，主要关注系统的可伸缩性与性能。

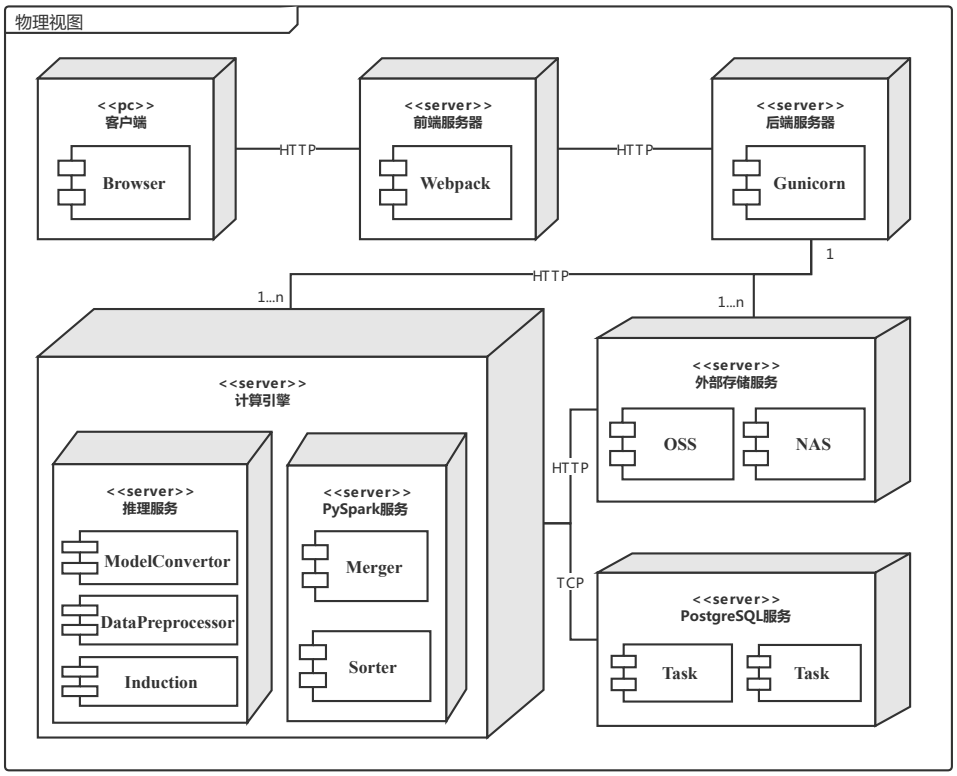

图 3-6: 物理视图

在本系统中，用户通过浏览器与本系统进行交互，前端代码经由 Webpack 打包后部署到前端服务器，并通过 axios 实现负载均衡与反向代理，从而实现前后端分离。用户的请求将以 HTTP 的形式转发到后端。对于用户的创建度量

任务请求，后端服务首先将用户提交的模型与数据集上传至外部存储服务并利用相应的 RESTful API 接口触发引擎。引擎在完成度量计算后，会将结果写入到对象存储系统的结果桶中，同时更新数据库中对应任务行的状态字段。引擎与数据库之间采用 TCP 的方式进行通信。引擎内的 PySpark 服务用以解决大规模样本度量时的吞吐量低、承载能力差的问题。

3.3.3 系统模块划分

面向神经网络的测试用例度量系统主要模块划分如图3-7所示，主要分为以下几个模块：模型编码转换模块、数据预处理模块、度量算法模块、度量报告生成模块以及计算引擎模块。

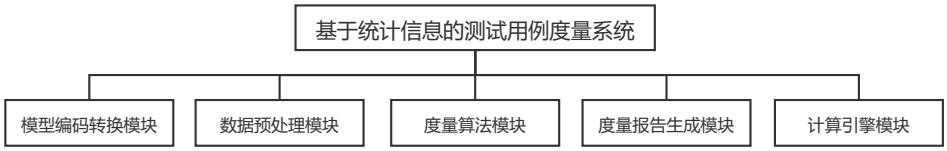

图 3-7: 系统模块划分图

模型编码转换模块以主流框架对 ONNX 编码提供的导出接口为基础，封装了模型验证、编码格式转换，封装了针对动态图、静态图两种不同结构的操作，实现了组件化使用。该模块使得系统在度量算法运行时无需考虑外部框架的库依赖并且消除了不同框架计算精度对度量结果的影响，同时缩小了生产环境镜像的体积以及解决了由于第三方框架频繁更新导致的镜像不稳定性问题。

数据预处理模块主要负责对用户提交的多源数据集进行解析，并作用用户预先注册的预处理变换算子，保证需要经受度量的数据集文件与模型训练时所用的数据的预处理相同。为保证用户能够简单、充分地表达预处理逻辑，系统提供了领域特定语言接口并给出示例。用户可以根据实际需求利用该接口快速编写预处理逻辑，系统在任务执行时会按照该逻辑修改抽象语法树并逆向生成预处理代码，以供调用。

度量算法模块基于 NumPy 数学库，封装了有关模型推理、信息熵、基尼值等度量标准的计算方法。考虑到深度学习通常面对海量的数据文件，因此度量算法会借助外部排序、MapReduce 等常用大数据处理算法，以保证系统整体的承载能力和计算效率。该模块最终的输出包括每样本的基尼值、每样本排序后的下标以及 TopK 的下标列表，用来辅助度量报告信息的生成。

度量报告生成模块基于 echarts 实现，通过利用度量任务的计算结果给出模型在该数据集上的推理情况，包括基尼值分布柱状图、基尼值与样本类别关系的热力图、前 10 个与后 10 个样本的可视化结果等，给予用户对该数据集和模型一个综合、全面的了解。

计算引擎模块是整个系统能够正常运行并保证横向可扩展性的关键所在。该模块基于 Apache Airflow 和 Kubernetes 实现，用来负责度量任务所对应有向无环图的执行、底层计算资源的弹性申请与释放、定时任务的管控、与存储系统的读写同步等任务。借助于虚拟化技术和容器编排引擎的水平扩展机制，资源调度可以做到忙时申请、闲时释放，最大程度地保证了物理资源的合理使用。在数据读取方面，本系统通常会面对两类数据：配置信息、工作流图结点间传递的元数据以及模型、数据集文件等海量数据。针对前者，本系统分别将配置信息等有关任务执行的元数据存储在关系型数据库中，以保证其可靠性，将节点的中间计算结果等热点数据存储在缓存中，以方便各节点的快速读取和写入；针对模型、数据集等大型文件，用户在提交后经过编码转换、预处理等操作后将被写入 NAS 中。度量结果将写入对象存储系统中。由于涉及多种存储引擎，数据的读取和写入会变得冗长复杂，因此引擎对涉及不同存储引擎的操作进行封装，屏蔽内部复杂调用，提供了干净统一的读写接口。

3.4 面向神经网络的测试用例度量系统各模块设计

3.4.1 模型编码转换模块

模型编码转换模块的概念类图如图3-8所示。模型编码转换模块的主要职责类是 ModelConverterHook 类，它继承自 Airflow 的 BaseHook 类，借助于 ONNXRuntime 的接口，封装了主流深度学习框架编码格式向 ONNX 格式的复杂转换逻辑。

具体来说，ModelConverterHook 根据配置信息中模型所使用的框架类型，将该任务转发到该框架所对应的处理函数。针对不同框架的处理函数在实现主要分为两类：针对静态图的转换和针对动态图的转换。ModelConverterOp 继承自 BaseOperator 类，是对整个转换过程更为抽象的封装，包括从存储系统读取需要转换的模型、记录转换过程中的部分元数据、将转换后的 ONNX 模型重新写入存储系统的过程。OSSHook 实现了如何从对象存储系统中读写数据、如何进行身份验证和权限验证等问题。BaseXcom 类是工作流图中结点的内部缓

存，用来存储部分管理信息，大小一般不超过 64KB。

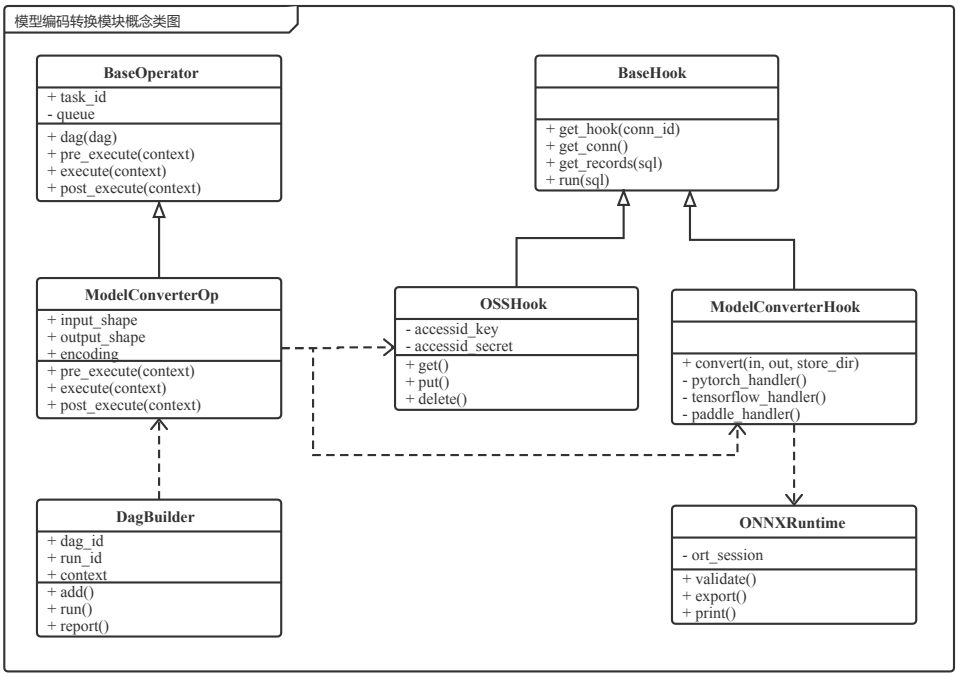

图 3-8: 模型编码转换模块概念类图

3.4.2 数据预处理模块

数据预处理模块的概念类图如图3-9所示。该模块的主要逻辑封装在 DataPreprocesserHook 类中，该类继承自 BaseHook 基类，基于 NumPy 数学库，根据用户提交的配置信息中的数据集类型作出相应的处理。对于知名数据集，系统将按照相应的文档说明完成对数据集的预处理；对于自定义数据集，系统将根据用户所指定的预处理算子，按照指定的顺序逐一地作用这些算子。系统提供地预处理算子包括归一化、灰度化等常用操作。DataPreprocessorOp 类可作为工作流图中的任务结点，它是对 DataPreprocessorHook 更为抽象地封装，包含了从存储系统读取与任务相关的数据集、记录转换过程中的部分元数据和日志、将预处理后的数据集重新写入存储系统的过程。

3.4.3 度量算法模块

度量算法模块的概念类图如图3-10所示。InferenceHook 类实现了如何对测试样本进行度量的核心算法，主要基于 NumPy 实现。InferenceHook 会首先创

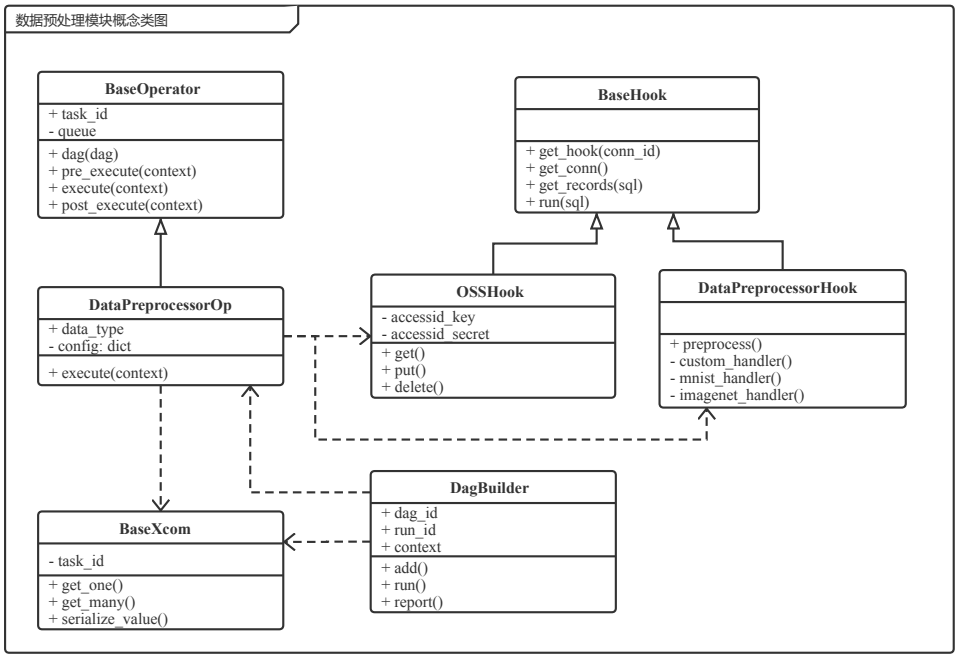

图 3-9: 数据预处理模块概念类图

建一个 ONNXSession，用来加载任务对应的模型和数据集文件。ONNXSession 会借助于 ONNXRuntime 完成模型在该数据集上的推理计算，得到每个样本的输出向量值。考虑到系统有时需要处理海量的样本数据，InferenceHook 需要根据当前容器的负载能力，适当地对数据集进行切片，采用分治的思想来完成推理计算。因此，ONNXSession 所计算得到的输出向量值需要按照文件的粒度持久化到磁盘介质中，以方便后续归并操作的实现。InferenceOp 继承自 BaseOperator，是对整个度量值计算的封装，包括了确定容器负载阈值、数据集切片大小、计算结果归并等过程，主要基于 PySpark 进行实现。与其他模块类似，计算检查点信息以及关键元数据信息将被写入 BaseXcom 中，以方便与其他工作流图结点进行共享。

3.4.4 度量报告生成模块

度量报告生成模块的概念类图如图3-11所示。该模块的主要职责类是 MetricReportBuilder 类，通过利用计算出的度量评价值生成相应的分析报表，包括每样本的度量值、所有样本的分布情况、前 K 个样本的可视化结果等。MetricReportBuilder 类利用 echarts 库来生成相应的曲线图和表格。DistributionUtil 用以辅助计算度量结果的数据分布规律。

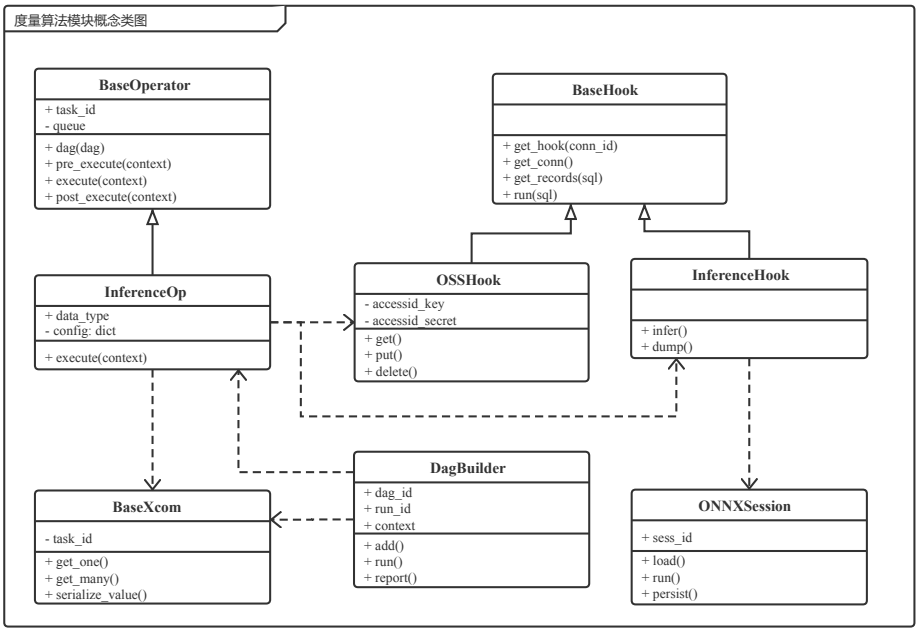

图 3-10: 度量算法模块概念类图

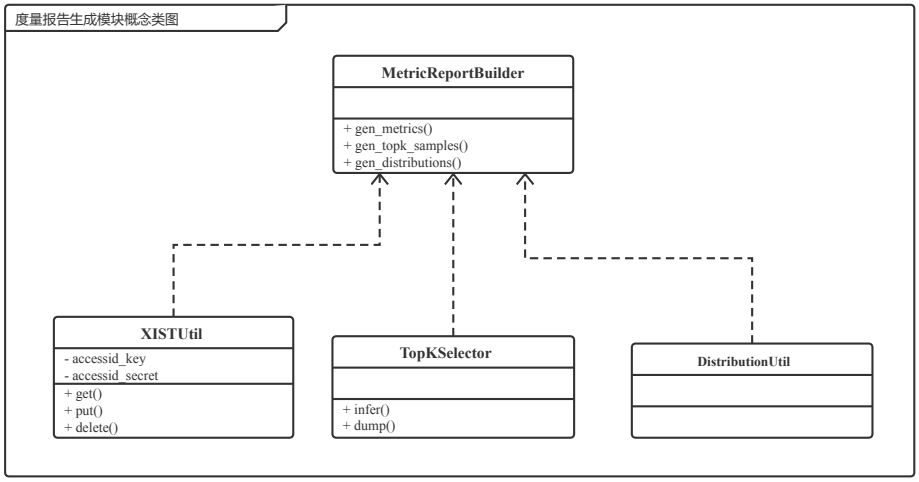

图 3-11: 度量报告生成模块概念类图

3.4.5 计算引擎模块

计算引擎模块的概念类图如图3-12所示。ScheduleJob 是整个模块的核心所在，负责整个系统所有任务的调度管控，以心跳包的机制保持与其他模块的通信连接。WebserverService 是 Airflow 的监控平台，赋予运维人员可视化的操作界面，提升运维效率。WebserverDeployment 是该服务在部署时的配置说明，用来在集群启动后分配容器运行 Webserver 服务。PostgreSQLService 是对关系型数据库服务的封装，目的是存储调度系统运行所需的管理数据。

PostgreSQLDeployment 同样是数据库服务的部署文件，用来制定容器所需的规格。PersistentVolumeClaim 是对外部存储的抽象，无论何种存储系统，都会以可持久化卷的方式挂载到对应的容器上，目的是方便容器内程序对外部存储的访问过程。RBACService 用来规范容器的访问权限以及操作权限。SchedulerDeployment 是调度器容器化的配置文件，它规定了所能调度的资源、包括副本数量、最大容器数等。SchduleJob 借助于 HPAService，能够实现容器的弹性化部署。当请求到来，调度系统会通过 HPA 创建并启动新容器，开始任务的执行；当任务执行结束，系统会自动销毁相关容器并归还资源。弹性化的架构保证了资源的合理使用，同时良好的横向可扩展性使得系统的承载能力可以随机器数量线性增长，为大规模的任务处理奠定了基础。

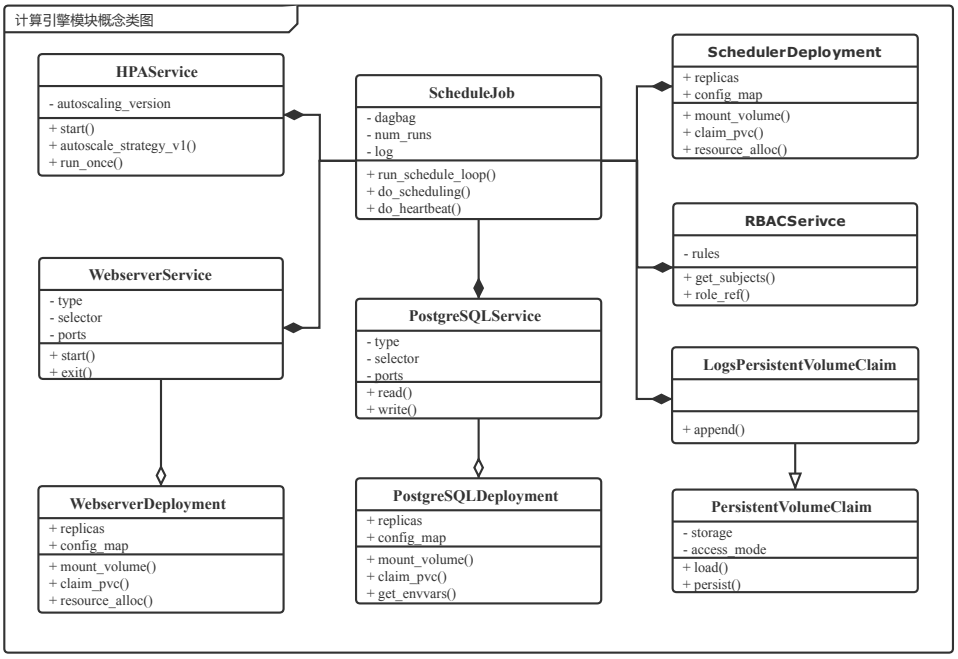

图 3-12: 计算引擎模块概念类图

3.5 数据库实体设计

为了保证数据存储的不可分割性、相对稳定等特性，需要对本系统进行数据持久化的设计，其中主要包含用户的登录与权限信息、度量任务的描述信息、数据集存储位置信息、模型编码以及度量结果。良好的数据库设计能够有效地提升系统响应速度、简化后端程序的处理流程，通过合理的设计外键约束，可以避免数据重复冗余，减少服务数据存储的压力。

通过对业务实体与实体之间的关系分析，可以得到本系统数据库的实体-联系图，如图3-13所示。实体-联系图展示了不同实体所拥有的属性以及实体与实体之间的关系，为单个数据库表的详细设计奠定了基础。

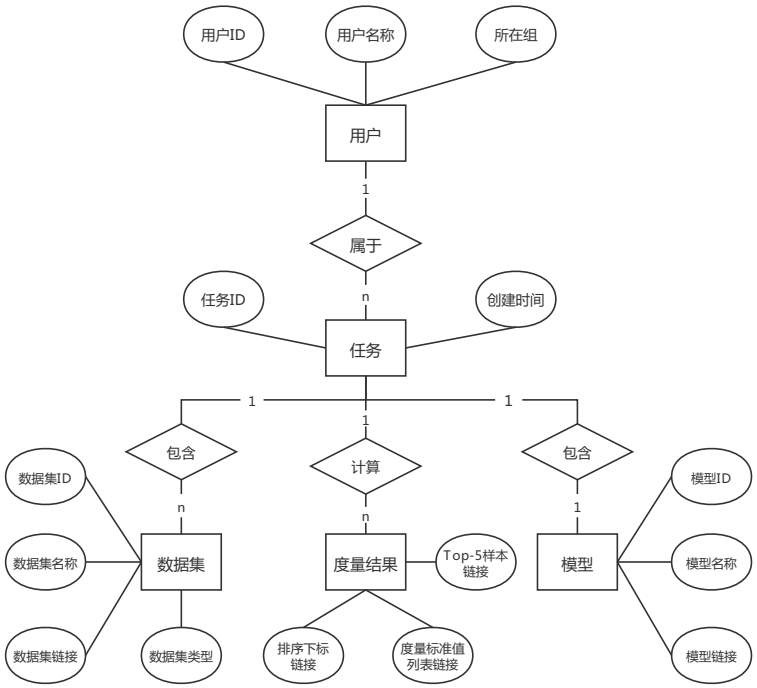

图 3-13: 系统数据库设计中的实体-联系图

表 3-9: 数据库 task 表

字段	类型	描述	备注
id	INTEGER	任务 ID	主键
name	VARCHAR	任务名称	
state	INTEGER	任务执行状态	
metrictype	INTEGER	度量标准	
creationtime	DATETIME	任务创建时间	
uid	INTERGER	用户 ID	外键
did	INTERGER	数据集 ID	外键
mid	INTERGER	模型 ID	外键
rid	INTEGER	计算结果 ID，关联结果表项	
dagrunid	VARCHAR	关联具体执行实例，以获取执行日志	

如表3-9所示为数据库 task 表字段说明。该表主要存储用户提交的度量任务的详细信息，主要包括任务创建时的名称、所选用度量算法、任务创建时间以及该任务与用户表、数据集表、模型表、度量结果表的对应关系。dagrunid 用来与 Airflow 的运行实例相对应，以方便查询相关任务的执行状态以及在任务出错时提供相关的日志信息。

表 3-10: 数据库 dataset 表

字段	类型	描述	备注
id	INTEGER	数据集 ID	主键
name	VARCHAR	数据集名称	
path	VARCHAR	数据集路径，本地路径或云端路径	不为 null
uid	INTEGER	该数据集所属用户	不为 null
state	INTEGER	该数据集是否经过度量	不为 null

如表3-10、3-11所示为数据库 dataset 表和 model 表的字段说明，主要描述了数据集文件和模型文件的相关信息，并指明了与用户的从属关系。在重训练模块中，用户只能在经过度量的数据集文件中选择训练数据，因此需要提供 state 字段来记录相应的状态。

表 3-11: 数据库 model 表

字段	类型	描述	备注
id	INTEGER	模型 ID	主键
name	VARCHAR	模型名称	
path	VARCHAR	模型路径，本地路径或云端路径	不为 null
uid	INTEGER	该模型所属用户	不为 null

如表3-12所示为数据库 result 表的字段说明。该表主要用来记录度量结果的相关信息，包括经过度量排序的样本下标列表、每个样本具体的度量值、前 10 个与末 10 个数据样本的下标，这用来在度量报告中以可视化的方式进行展示，最后 tid 描述与任务表的对应关系。

通过上述的若干张数据表，能够有效地对系统数据存储的整个过程进行描述，同时在表设计过程中，遵循 BCNF 范式，使得数据表之间的冗余度达到最小，结构清晰，更新逻辑简单。

表 3-12: 数据库 result 表

字段	类型	描述	备注
id	INTEGER	模型 ID	主键
sortedindices	VARCHAR	描述度量排序后的数据集下标列表	
metricvalue	VARCHAR	描述样本集具体度量值的文件	
top10samples	VARCHAR	描述排序后前 10 个样本的路径	
last10samples	VARCHAR	描述排序后末 10 个样本的路径	
tid	INTEGER	任务 ID，指明该结果关联的任务	外键

3.6 本章小结

本章节内容主要介绍了面向神经网络的测试用例度量系统的总体设计。首先，介绍了该系统的功能性需求和非功能性需求，以此形成了系统用例图，并对每一个用例进行了详实的描述。接下来，在需求分析的基础上，对系统进行整体的架构设计，从逻辑视图、过程视图、开发视图和物理视图 4 个角度出发，给出系统的模块划分及其描述。最后，对系统划分出的模型编码转换模块、数据预处理模块、度量算法模块、度量报告生成模块、计算引擎模块给出概要设计，包括模块结构设计、概念类图、时序图、数据流图等详细文档制品，为下一章系统的详细设计与实现打下坚实的基础。

第四章　详细设计与实现

本章阐述基于基尼不纯度的测试用例度量系统中关键模块的详细设计，并介绍每个模块的主要功能以及关键代码实现，同时展示系统部分实现界面。首先，将介绍模型转换编码模块以及数据预处理模块的构建与实现，然后介绍度量算法原理以及代码实现，最后介绍用于支撑整个系统的调度模块以及数据存储模块的详细设计与实现。

4.1　模型编码转换模块设计

4.1.1　模型编码转换模块的详细设计

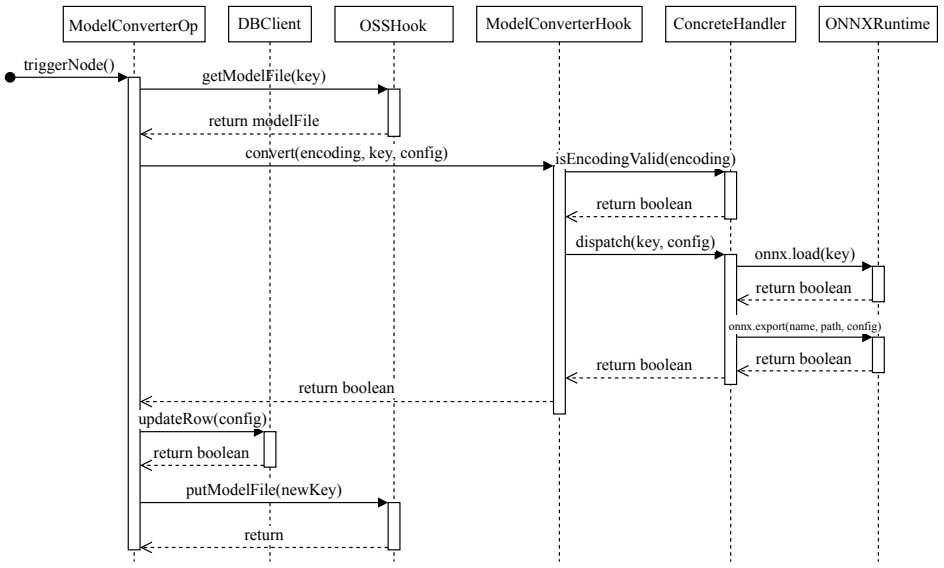

图 4-1: 模型编码转换模块的时序图

如图4-1所示是模型编码转换模块的时序图。用户在创建度量任务并提交相应的数据之后，系统会根据用户的配置文件来创建相应的工作流图。若需要进行用户模型文件编码格式的转换，则系统会在工作流图中增加模型编码转换节点。该节点对应的实现是 ModelConverter 类，首先，系统会通

过外部存储系统提供的 getModelFile 方法获得存储在外部文件系统上的模型文件。如果文件不存在，系统将给出错误提示信息并终止工作流图的执行。ModelConverterHook 类负责实现模型的转换工作，系统会调用该类的 convert 方法进行转换。ModelConverterHook 会检查模型现有的编码格式是否有效，根据其具体编码格式转发到对应的深度学习框架处理类 ConcreteHandler。ConcreteHandler 类会使用指定的深度学习框架来加载用户的模型文件，并导出计算图。ONNXRuntime 会加载相应的计算图并导出为 ONNX 文件。在此过程中，需显示地指定 ONNX 操作符的兼容版本以及可变模板参数。最后，转换的结果将被 DataPreprocessor 再次写入外部存储系统中，以方便其他节点的使用。

4.1.2 模型编码转换模块的实现

为了获取需要进行转换的模型文件，本系统首先需要从数据库中获取相应模型文件的名称，并根据用户提供的地址信息，从相应的存储系统中拉取所需的模型文件。利用 ModelConverterHook 类的 convert 方法完成对模型编码的转换工作后，再次通过相应的接口将结果写回到存储系统中，同时将转换后的文件名称更新到数据库表项对应字段中。具体的代码实现如图 4-2所示。

```
class ModelConverterOp(BaseOperator):
  def execute(self, context):
    config = context['dag_run'].conf

    # retrieve from OSS
    oss = OSSClient(config)
    oss.get(config['model_bucket'], self.key, config['store_dir'])
    hook = ModelConverterHook(0)
    newKey = hook.convert(self.model_lib, self.key,
                          self.input_dim, self.output_dim,
                          config['store_dir'])
    context['task_instance'].xcom_push(key='model', value=newKey)

    # update DB info
    db = DBClient("172.19.0.2", "gproj")
    row = db.sess.query(db.table)
                 .filter_by(id=config['id']).first()
    row.converted_model_name = newKey
    db.sess.commit()

    # write back into OSS
    oss.put(config['model_bucket'], newKey,
            os.path.join(config['store_dir'], newKey))
```

图 4-2: 模型编码转换模块的部分实现

ModelConverterHook 类负责具体的转换过程。首先，通过根据模型的编码

格式作函数转发，将不同框架下产生的模型转发到对应框架的处理函数中。这里以 PyTorch 框架为例。PyTorch 的主要特点是基于动态计算图，只有在运行期才能完全确定模型结构，因此实现难度更大。在开始转换之前，首先需要确定模型的具体结构，这通过 JIT 技术来实现。具体来说，通过利用 JIT 中的 trace 函数，它接受一个模块及一组数据实例作为输入，模拟计算该输入的结果，并在此过程中跟踪记录下所有控制流路径。TorchScript 以该记录结果为基础，推导出所有可能的执行路径，以 script 的形式进行记录。这样，就可通过该 script 文件重建出模型的图结构。最后，通过调用 ONNX 的相应接口完成转换工作。具体的代码实现如图 4-3所示。

```
class ModelConverterHook(BaseHook):
  @staticmethod
  def _pytorch_handler(key, input_shape, output_shape, store_dir):
    model_path = os.path.join(store_dir, key)
    model = torch.jit.load(model_path)
    model.eval()
    x = torch.randn(1, *input_shape)
    y = torch.randn(1, *output_shape)
    onnx_key = onnx_key[:onnx_key.index('.')] + '.onnx'
    torch.onnx.export(model, x, os.path.join(store_dir, onnx_key),
                      opset_version=11,
                      input=['input'], output=['output'],
                      dynamic_axes={'input': {0:'batch_size'},
                                    'output': {0:'batch_size'}})
    return onnx_key
```

图 4-3: PyTorch 模型格式转换 ONNX 格式的部分实现

针对 TensorFlow 和 Paddle 的转换过程，与上述类似，不再重复给出。考虑到 TensorFlow 和 Paddle 的模型均基于静态图进行构建，因此在实际模型编码中已经嵌入了具体结构信息，因此转换过程会更加简单。

4.2 数据预处理模块设计

4.2.1 数据预处理模块的详细设计

数据预处理模块的时序图如图 4-4所示。在系统根据用户度量任务生成的工作流图中，会包含一个数据预处理节点，它的职责是将用户提交的原始数据转换为可供模型进行推理计算的形式。该节点功能对应的类是 DataPreprocessorOp。与模型编码转换格式类似，该类首先会通过外部存储服务的接口获取到用户的数据集文件。此后，通过调用 DataPreprocessorHook 类的

preprocess 方法开始数据集的预处理。DataPreprocessorHook 类封装了整个数据预处理的过程。如果用户选择的是知名数据集，那么系统会自动调用内置的操作算子以完成数据预处理工作；如果用户提交了私有数据集，那么用户还需要利用系统提供的领域特定语言编写相应的预处理操作。针对这部分的用户输入，系统会调用 parseUInput 方法进行解析，并对模板代码所对应的抽象语法树中相关节点进行修改。之后，系统通过调用 instantiateOp 方式反解析出该抽象语法树对应的 Python 代码并保存，以供 doProcess 方法调用。doProcess 方法首先加载数据集并验证格式，然后作用用户自定义的预处理操作。在 preprocess 方法完成数据集的处理后，系统会通过 DBClient 类将转换后的数据集文件名称持久化到数据库中，接着将处理后的数据集以新文件名键值写入存储系统中。

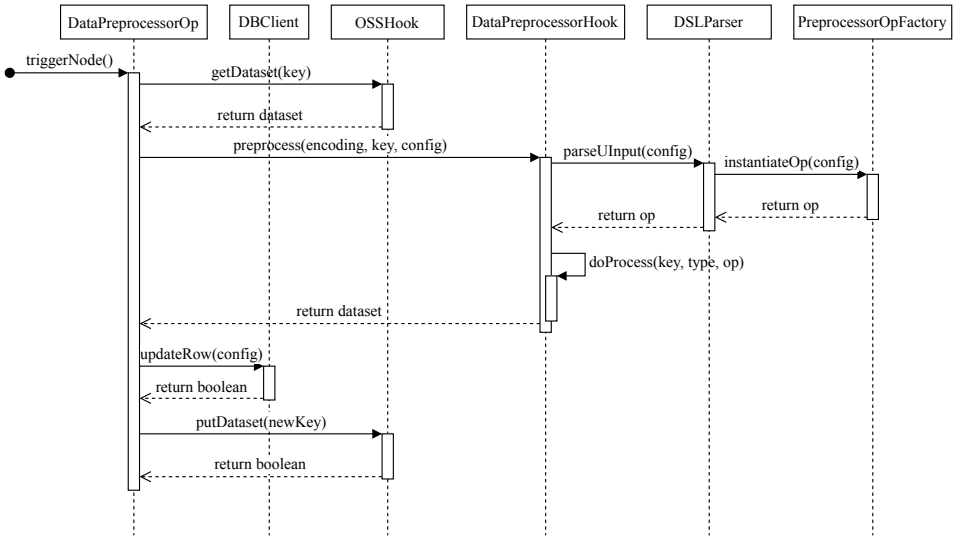

图 4-4: 数据预处理模块的时序图

4.2.2 数据预处理模块的实现

系统首先从外部存储中获取与度量任务相关的数据集文件，并将预处理前与预处理后的数据集文件名称以键值对的形式缓存到 Xcom 中，以方便其他后续节点的使用。预处理后的数据集文件将被回写到外部存储系统中，目的是为了在推理计算中再次使用。该部分具体的代码实现如图4-5所示。

对于一部分的常用数据集，系统已经按照相应数据集的说明，内置了标准的预处理过程，这些数据集包括手写体识别 MNIST、CIFAR、IMAGENET 以及德国交通标志数据集 GTSRB。

```
class DataPreprocessorOp(BaseOperator):
  def execute(self, context):
    config = context['dag_run'].conf

    oss_hook = OSSHook(config)
    oss_hook.get(config['data_bucket'], self.key,
                 config['store_dir'])

    hook = DataPreprocessorHook(0)
    newKey = hook.preprocess(self.key, self.data_type,
                             config['store_dir'])
    context['task_instance'].xcom_push(key='data', value=newKey)

    oss_hook.put(config['data_bucket'], newKey,
                 os.path.join(config['store_dir'], newKey))
```

图 4-5: DataPreprocessorOp 类的代码实现

对于此范围之外的数据集或用户私有数据集，用户在创建度量任务时，可以通过自定义预处理算子的方式将其注册到系统之中。系统会根据用户提交的预处理算子定义，将其加载到对应类下，并产生出对应的 Python 代码，并将以单独的模块包导入到系统之中。该部分具体的代码实现如图4-6所示。

```
class PreprocessorOpFactory():
  def __init__(self):
    self.uCode = None
    self.sCode = None

  def parse(self, config):
    self.uCode = config['user_code']
    tree = ast.parse(self.uCode)
    nodeList = inOrderTravesal(tree)
    self.sCode = assemble(nodeList)

  def unparse(self, config):
    with open(config['user_op_name'] + '.py', 'w') as f:
      gencode = astunparse.unparse(self.sCode)
      f.write(gencode)
```

图 4-6: 生成自定义预处理算子的代码实现

4.3 度量算法模块设计

4.3.1 度量算法模块的详细设计

度量算法模块是本系统的核心组成部分。算法的主要思想是通过基尼不纯度来衡量神经网络对输入数据进行决策时的确定程度。当某些输入数据位于决

策边界上时，神经网络模型判决概率呈现为均等化，说明对于该输入数据，神经网络对于其属于哪一类别并没有足够的信心。因此，该样本对提升模型的性能有较大的帮助。本测试用例度量算法的伪代码如算法 1所示。

算法 1: 测试用例度量算法

```
1  PrioritizeInputs (M, D)
       input  : 模型 M 以及数据集 D
       output: 按度量标准值排序后的下标列表及每个样本的度量值列表
2      G* ← empty list;
3      I* ← empty list;
4      foreach data item d_i ∈ D do
5          v⃗ ← M.forward(d_i);
6          g_i ← GetGiniImpurity(v⃗);
7          G*.append(g_i);
8      end
9      I* ← argsort(G*);
10     return G*, I*;
11 GetGiniImpurity (v⃗)
       input  : 神经网络的输出向量 v⃗
       output: 该数据样本的基尼不纯度
12     S* ← 0;
13     v⃗ ← Softmax(v⃗);
14     foreach element e ∈ v⃗ do
15         S* ← S* + e^2;
16     end
17     return 1 − S*;
```

具体来说，对于每一个样本输入，首先利用神经模型计算出它的输出结果，并利用 Softmax 函数进行归一化。接着，根据基尼不纯度的计算公式，得到该样本的基尼不纯度值，并以此作为后续排序的标准。对输入数据集的每一个样本重复如上操作，并将结果存储在列表 G^* 中。最后，利用 argsort 对 G^* 进行排序，并返回排序结果和基尼不纯度值的列表 G^*。

如图 4-7所示为度量算法模块的时序图。InferenceOp 类封装了上述度量算法的具体实现。首先通过 OSSHook 类获取存储在外部系统上的经过预处理的数据集文件和编码格式转换后的模型文件，以便后续计算需要。InferenceHook 类通过创建执行器来初始化执行环境。GeneralExecutor 是对 pyspark 框架的封装，通过 MapReduce 的编程模型完成对度量任务的计算，由 GeneralExecutor 创建的 MPJob 完成了对任务的划分。GeneralExecutor 的内部实现是一个典型的生产者-消费者队列，对于每个任务单元，执行线程都需要通过 ONNXRuntime 创

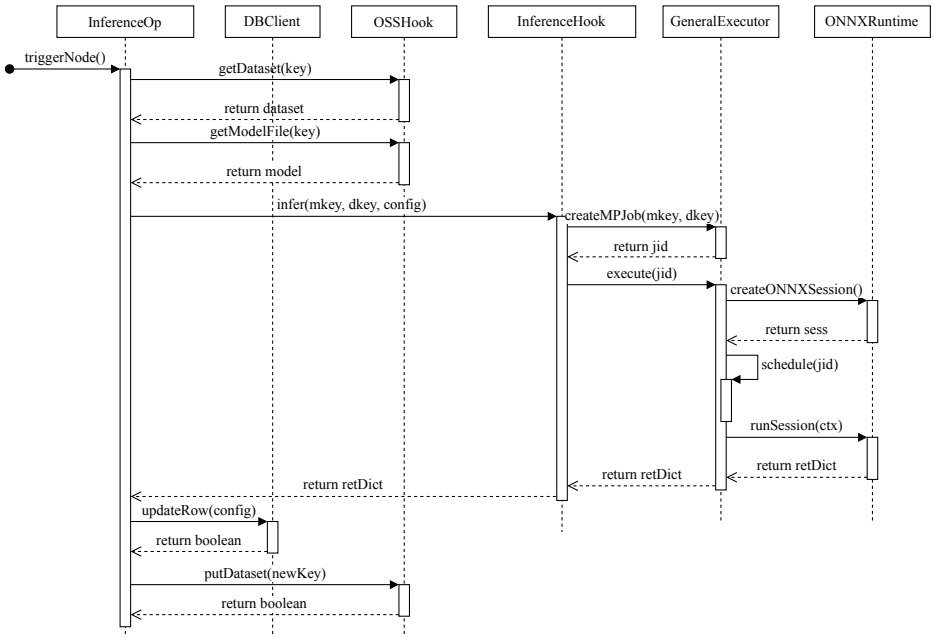

图 4-7: 度量算法模块的时序图

建相应的上下文环境，并完成相关的计算。当任务执行完成后，InferenceOp 类负责将执行的结果写入存储系统中，以方便后端程序返回给用户。

4.3.2 度量算法模块的实现

如图4-8所示为 InferenceOp 类的算法实现。首先查询数据库，获取到本次任务中所使用的模型与数据集文件。接着，利用云存储客户端从云端拉取相应的文件到本地。然后，通过调用 InferenceHook 类完成度量算法相关的计算。最终，通过存储系统的客户端将相应的度量结果写入到对应的数据桶中。在此过程中，由于有向无环图的每个任务都需从数据库中读取一些元数据，因此对于这部分热点数据，可直接写入系统提供的 NoSQL 缓存中，这能够较大地提升整个系统的运行效率。

如图4-9所示为测试用例度量算法的具体实现。代码首先从集群存储系统中读取对应的数据集文件，创建 ONNX 的运行时环境并加载经过编码转换后的模型文件。接着，指定该 ONNX 上下文的输入数据并触发运行。在 ONNX 运行时完成相应的计算后，对输出向量作用 Softmax 函数，得到模型判决的概率分布。此后，根据基尼不纯度的计算公式，完成度量标准值的计算。考虑到最终只需进行结果排序，因此可将该过程简化为计算该样本的信息熵并指定从

```
class InferenceOp(BaseOperator):
  def execute(self, context):
    config = context['dag_run'].conf
    ...
    hook = InferenceHook(0)
    output, scores, indices = hook.infer(model_path, data_path)

    db = DBClient("172.19.0.2", "gproj")
    row = db.sess.query(db.table)
                .filter_by(id=config['id']).first()
    ...
    newKey = hashlib.sha256(
              mixed_name.encode('UTF-8')).hexdigest()
    row.metrics_result_name = newKey
    db.sess.commit()

    local_path = os.path.join(config['store_dir'], newKey)
    np.savez(local_path, output, scores, indices)
    oss.put(config['metrics_bucket'], newKey, local_path)
```

图 4-8: InferenceOp 类的代码实现

小到大排序即可。InferenceHook 会将计算结果返回给 InferenceOp 类，后者将会把度量结果写入 OSS 存储系统中，为后端读取相关数据并生成度量报告作铺垫。

```
class InferenceHook(BaseHook):
  def infer(self, model_path, data_path):
    X = np.load(data_path)
    X = X.astype(np.float32)
    ort_sess = ort.InferenceSession(model_path)
    ort_inputs = {ort_sess.get_inputs()[0].name: X}
    ort_outputs = ort_sess.run(None, ort_inputs)

    scores = np.exp(ort_outputs) /
             np.sum(np.exp(ort_outputs), axis=1, keepdims=True)
    scores = np.sum(scores**2, axis=1)
    indices = np.argsort(scores)

    return scores, indices
```

图 4-9: 基于基尼不纯度的测试用例度量算法实现

4.4 度量报告生成模块设计

4.4.1 度量报告生成模块的详细设计

如图4-10所示为度量报告生成模块的时序图。ReportHelper 类负责为前端的页面渲染提供相应的数据支持，这包括模型的元数据信息、数据集元数据信

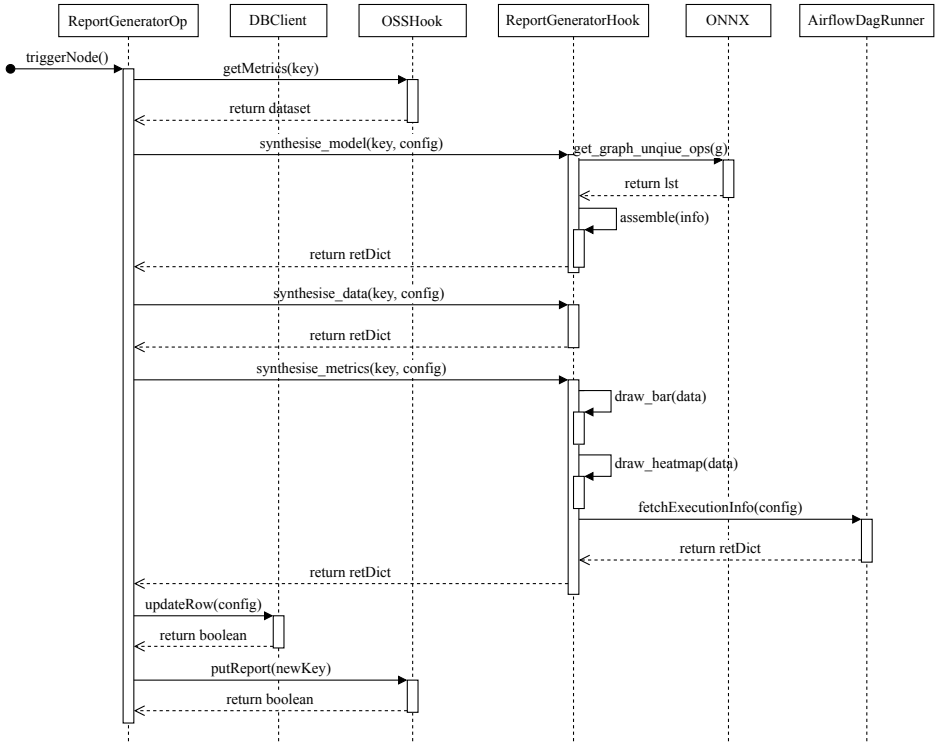

图 4-10: 度量报告生成模块的时序图

息、度量任务的执行日志以及度量结果等。ResultFetcher 负责从相应的中间件拉取信息，包括需要从对象存储系统中获取存储的数据集管理数据，通过创建 ONNXRuntime 的上下文环境来加载模型并获取结构等信息。AirflowDagRun 是每次度量任务运行的实例表示，系统可以从中获取相应的执行日志。在获取完所需的数据后，ResultHelper 对这些数据的加工处理，由前端生成对应图表展示给用户。

4.4.2 度量报告生成模块的实现

最终的度量报告包含以下几个部分：模型信息概述、数据集信息概述、度量任务执行时间的甘特图、数据集样本度量值的柱状图表示、数据集样本度量值与模型分类关系的热力图表示。模型信息概述展示了模型的大小、所使用的训练框架、模型层数、参数总量等。数据集信息概述用来展示数据集的一些元数据信息，这包括数据集大小、输入维度、分类目标数等。有关度量任务执行时间的甘特图展示了对应有向无环图每个节点的执行时间。数据集样本度量值的柱状图展现了该数据集中不同度量值的分布情况。数据集样本度量值与模型

分类关系的热力图形象地展示了对于该模型来说，何种类别的样本在判决时的概率最不确定，以方便用户在模型调优时重点关注属于该类别的样本数据。

模型信息概述的实现代码如图4-11所示。系统先利用 ONNXRuntime 库创建运行上下文环境，然后加载经过编码转换后的模型文件，并利用 model.graph 结构体获取到输入维度信息、参数数量、输出维度信息等，例如获取到当前模型的带有运算符的总层数可以通过 len(model.graph.node) 实现。类似地，可通过对 model.graph 进行循环遍历并计数得到模型的总参数量。

```
def synthesise_model(self, file_path, info):
    model = onnx.load(file_path)
    info['model_name'] = os.path.basename(file_path)
    info['model_size'] = round(model.ByteSize() / 1024, 2)
    info['model_lib'] =
        model.producer_name + ' ' + model.producer_version
    info['model_levels'] = len(model.graph.node)
    info['model_ops'] =
        ReportGeneratorHook._get_graph_unique_ops(model.graph)
```

图 4-11: 模型信息概述的代码实现

数据集信息概述的代码实现如图4-12所示。通过加载经预处理、压缩后的数据集文件，并 NumPy 数学库可以计算得到该数据集的一些信息，其中包括总样本个数，每个样本的特征维度，数据集的分类总数等。此外，在数据集信息概述中，还将随机挑选出若干个样本，以可视化的形式展现给用户。

```
def synthesise_data(self, file_path, info):
  X = np.load(file_path).astype(np.float32)
  pname = os.path.basename(file_path)
  info['data_name'] = pname[pname.find('-')+1:]
  info['data_counts'] = X.shape[0]
  info['data_dim'] = X.shape[1:]
  info['data_size'] = round(X.nbytes / 1024, 2) # KB
  info['data_mean'] = np.mean(X)
  info['data_std'] = np.std(X)
```

图 4-12: 数据集信息概述的代码实现

度量任务执行时间的甘特图的代码实现如图4-14所示。在 Airflow 完成度量任务的执行后，可以借助 airflow.dagrun 接口获取到最近一次该有向无环图每个节点的执行日志。通过从该结构体中提取执行时间信息，经过换算后，返回前端以完成甘特图的绘制。

数据集样本度量值的柱状图的代码实现如图4-14所示。首先获取每个样本的基尼不纯度值，然后通过 NumPy 中 ndarray 的运算广播机制，以向量化地方

```
def successCallback(context):
  oss = OSSClient(context['dag_run'].conf)
  config = context['dag_run'].conf
  stateHelper(config['id'], 2) # 2 -> SUCCESS

  execinfo, tmplst = {}, []
  for task_id in context['dag'].task_ids:
    ti = get_task_instance(context['dag'].dag_id,
                           task_id, context['execution_date'])
    if ti.end_date is None:
      ti.end_date = datetime.now()
      ti.duration = ti.end_date.timestamp() -
                    ti.start_date.timestamp()
    ele = {
      'id': task_id,
      'start_date': int(1000 * ti.start_date.timestamp()),
      'end_date': int(1000 * ti.end_date.timestamp()),
      'duration': int(1000 * ti.duration)
    }
    tmplst.append(ele)

  execinfo['time'] = tmplst
  new_key = str(config['id']) + '__report__sply' + '.pkl'
  with open(os.path.join(config['dir'], new_key), 'wb') as f:
    pickle.dump(execinfo, f)
  oss.put(config['metrics_bucket'], new_key,
          os.path.join(config['store_dir'], new_key))
```

图 4-13: 度量任务执行时间甘特图的代码实现

式统计出度量值在 0.1 区间长度下的样本个数。最后，将该结果返回给前端以完成柱状图的绘制。

```
@staticmethod
def _bar_helper(metrics):
  m = 1 - metrics
  barLst = []
  for i in range(10):
    d = i / 10.0
    cur = m[(m >= d) & (m < d + 0.1)]
    barLst.append(len(cur))
  return barLst
```

图 4-14: 数据集样本度量值柱状图的代码实现

数据集样本度量值与模型分类关系的热力图的代码实现如图4-15所示。首先循环遍历数据集中的每个样本，从 output 数组中获取到神经网络在该样本输入下的输出向量。接着，利用 argmax 函数求得神经网络对该样本的判决结果并作记录。然后，从 metrics 数组中获取到该样本的基尼不纯度值，并判断其位于哪个 0.1 长度的区间内并做记录。为了保证可视化结果的友好性，该过程剔除了基尼不纯度处于 0.9 1 之间的数据样本。原因是神经网络对于大部分数据的判

决信心都较高，这部分数据并不能反映出神经网络模型所存在的问题，也没有意义作为训练数据在重训练过程中使用，因此可以进行剔除。以上述的信息作为横纵坐标，在二维矩阵中寻址到该单元的计数器并加 1。当循环结束时，再次剔除计数器为 0 的单元信息，就可得到样本度量值与模型分类关系的热力图数据。

```
@staticmethod
def _heatmap_helper(output, metrics):
  m = 1 - metrics
  R = np.zeros((10, 10), dtype=np.int32)
  N, C = output.shape
  for i in range(N):
    xp = np.argmax(output[i])
    yp = int(m[i] * 10)
    if yp != 0:
      R[yp][xp] += 1
  heatMapLst = []
  for i in range(1, 10):
    for j in range(10):
      if R[i][j] != 0:
        heatMapLst.append([i, j, int(R[i][j])])
  return heatMapLst
```

图 4-15: 数据集样本度量值与模型分类关系的热力图表示

4.5 计算引擎模块设计

4.5.1 计算引擎模块的详细设计

计算引擎模块主要负责 Airflow 中有向无环图的执行工作。在传统的 Airflow 应用架构中，通常使用基于 Celery 编写 CeleryExecutor 作为执行器，如图 2-4所示。Celery 作为一种分布式任务队列，通过消息机制进行通信，以消息队列中的 Broker 作为客户端，Worker 负责进行任务的执行。Celery 着重于处理实时操作，因此适用于工作量小、吞吐量低但响应时间要求低的任务。

Celery 架构的优势在于通过提前配置 Worker 资源，在突发流量到来时，能够最大化地减少系统的响应时延，保证系统正常运行，并在不同 Worker 之间实现负载均衡。此外，当其中的某个 Worker 节点出现故障宕机时，调度器由于没有接受该节点的心跳包，会自动将该任务调度到其他节点进行执行。Celery 架构的缺点在于当系统开始运行后，无法根据具体的任务数量对 Worker 节点的数量以及不同 Worker 所使用的 CPU 或内存资源进行精细化调整，会造成系统中部分资源闲置，无法得到有效地利用。

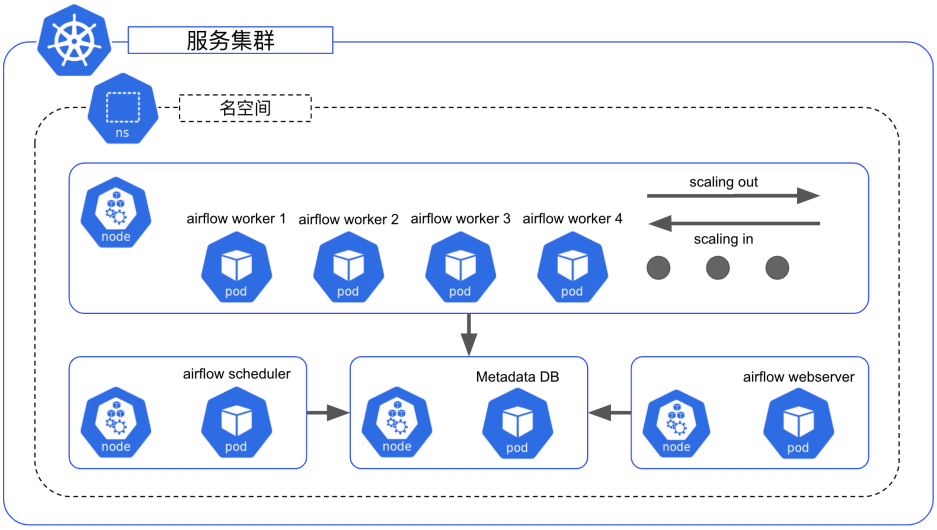

图 4-16: 基于 Kubernetes 的 Airflow 架构设计

考虑到 Celery 架构下动态资源管理问题无法得到解决，Airflow 在 1.10 版本开始支持使用 Kubernetes 容器编排引擎，可在 DAG 文件中指定 KubernetesExecutor 进行使用。基于 Kubernetes 的应用架构如图 4-16所示，在系统负载为零时，集群中将只存在运行 Airflow Webserver 与 Airflow Scheduler 的容器，当任务到来时，集群将为每一个任务实例生成对应的容器执行环境，并在该容器中完成任务的执行。因此，当计算资源足够的前提下，系统的可承载能力随机器数量线性增长，即系统具有良好的线性可扩展性。

除 KubernetesExecutor 之外，Airflow 还提供了 KubernetesPodOperator 的运算符，用以定义有向无环图中单个节点的业务逻辑。相比于前者，KubernetesPodOperator 为有向无环图中的单个节点创建一个容器，用户通过将相应的操作和所需的资源封装在镜像中，当执行到该节点时，会由 Kubernetes 拉取镜像创建容器环境并执行预先定义的指令。KubernetesPodOperator 的优点在于将具体业务逻辑与调度逻辑相解耦，用户只需集中精力修改镜像中的逻辑代码，而无需对与 Airflow 相关的代码进行大量修改。例如，在图 4-17所示的节点定义中，当用户更新业务逻辑后，只需在节点定义中将“v1”修改为“v2”，无需其他任务的改动。

KubernetesPodOperator 的缺点在于当有向无环图中的节点过多时，会到集群在执行任务实例时分配大量的容器，造成暂时性地资源紧张，同时也会增加集群中 Master 的管理负担，导致其性能降低。另外，在一张工作流图中，通常不同节点之间会彼此复用一些代码，KubernetesPodOperator 所带来的强制隔离

```
task_1 = KubernetesPodOperator(
  image="prioritize_task/v1", # only this needs to be changed
  name="Prioritize Task",
  task_id="PT_01",
  cmd=["python", "metrics.py", "arg1", "arg2"]
)
```

图 4-17: 基于 K8sPodOperator 的更新示例

性会导致一些相同的代码片段散布在若干镜像中，给项目增加维护难度。

综合考虑，在本系统中将选用 KubernetesExecutor 进行架构设计，这能够保证整个系统具有良好的线性扩展性，同时提高整个系统的资源利用率。从应用场景上看，用户并不会要求度量任务实时返回度量结果，因此本系统属于批处理系统类别，没有实时性方面的性能需求，因此集群中在动态扩容或缩容时产生的额外开销并不会对用户体验产生影响。

4.5.2 计算引擎模块的实现

为了实现基于 Kubernetes 的弹性 Airflow 架构，首先需要构建出创建容器时所使用的镜像文件，本系统选用 Dockerfile 来编写相应镜像文件。具体代码如图**??**所示。对于项目中所使用到的第三方库，如 onnxruntime、numpy 等，可通过 PYTHON_DEPS 参数进行引入。

在完成镜像文件创建后，下一步需要对集群进行配置。本系统中利用 ConfigMap 来完成集群中环境变量的配置，具体配置文件如图4-18所示。在该配置中，首先定义了 Airflow 所使用的执行器为 Kubernetes，然后指定了在 Airflow 在运行时所需用到的数据库相关信息，接着指定了 Airflow 运行时的参数，包括镜像地址、镜像版本、工作流图定义所在卷、日志卷等。该配置文件会在集群新创建容器时进行加载。

由于对应每个任务实例的容器在执行完成后都会被销毁，因此需要创建相应的存储对象来记录容器运行过程的日志以及最终计算结果。本系统中利用 Kubernetes 中的卷机制来满足这一需求。本系统需要创建两个卷，其中一个用来存储日志信息，另一个用来存储用户定义的工作流任务。容器会直接将计算结果写入到外部存储系统中，因此无需创建相应的卷。对于日志卷，本系统通过 PersistentVolumeClaim 机制来实现，具体代码如图4-19所示。其中 metadata.name 与集群环境变量中的 AIRFLOW__KUBERNETES__LOGS_VOLUME_CLAIM 变量相对应；accessModes

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: airflow-envvars-configmap
data:
  EXECUTOR: Kubernetes
  POSTGRES_HOST: postgres
  POSTGRES_DB: airflow
  POSTGRES_PORT: "5432"
  AIRFLOW__KUBERNETES__WORKER_CONTAINER_TAG: "1.10.10"
  AIRFLOW__KUBERNETES__DAGS_VOLUME_HOST: /mnt/airflow/dags
  AIRFLOW__KUBERNETES__LOGS_VOLUME_CLAIM: airflow-logs-pvc
  AIRFLOW__KUBERNETES__ENV_FROM_CONFIGMAP_REF: airflow-envvars-
configmap
```

图 4-18: Airflow 集群环境变量配置文件

决定了卷的访问形式，即是否需要采取并发控制。对于工作流图所使用的卷，可通过类似的方法完成配置。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: airflow-logs-pvc
  labels:
    app: airflow-k8s
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 512Mi
```

图 4-19: 基于 PVC 机制的日志卷实现

在完成上述基础设施方面的配置后，需要创建集群中分担不同任务的角色，这在本系统中对应了 3 个不同的服务，包括负责提供平台管控的 Airflow Webserver、负责任务调度的 Airflow Schduler 以及用于中间件运行状态管理的 PostgreSQL 数据库。在 Kubernetes 中，Pod 是一个或多个容器的组合，这些容器共享存储、网络和名空间。Pod 同时也是集群最小的可部署单元。ReplicaSet 用来管理 Pod，同时控制当前 Pod 的数量。Deployment 用来管理 Pod 以及 ReplicaSet，可用于实现热更新与回滚、弹性扩容与缩容。Service 定义了一个服务的访问入口地址，外部调用者不会获取到具体 Pod 的 IP 地址，而是通过 Service 进行调用。Service 通过预先定义好的 Label Selector 进行转发同时在此过程中实现负载均衡。

在本系统中，由于 WebServer 与 PostgreSQL 数据库需要对外部开放接

口，需要定义相应的 Deployment 配置与 Service 配置。Airflow WebServer 的 Deployment 配置如图4-20所示。其中 containers 指定了使用的镜像文件及版本，resource 指明了 Pod 所需的资源用量，ports 指明了容器对外暴露的端口号，volumeMounts 指明了自定义卷的名称与访问路径。 Webserver 的 Service

```
apiVersion: apps/v1
kind: Deployment

spec:
  selector:
    matchLabels:
      app: airflow-webserver
    spec:
      containers:
      - name: airflow-webserver
        image: puckel/docker-airflow:1.10.10
        envFrom:
          ...
        resources:
          ...
        ports:
          - containerPort: 8080
```

图 4-20: Airflow Webserver 的 Deployment 配置文件

配置文件通过使用 selector 用来关联后端中哪个 Pod 负责接收对应的流量。NodePort 允许将该服务以指定的端口号暴露出去，以便外部访问。对于 Airflow 的 Scheduler 服务，因为它的主要职责是负责调度集群内的 Worker 来完成相应任务的执行，无需对外提供服务，所以无需对其定义相应的 Service 文件。考虑到 Scheduler 需要动态地创建容器来分配不同任务实例的执行，因此需要授予其相应的权限，如图4-21所示。通过授予 Scheduler 在 Pods 资源组上获取、查看、监听、创建、销毁的权限，Scheduler 能够在任务触发时自动分配相应的 pods 以完成任务的执行。

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: pods-permissions
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch", "create", "delete"]
```

图 4-21: Airflow Scheduler 的 rBAC 配置文件

在完成上述配置后，在命令行键入挂载存储卷命令，并利用 script-apply.sh 脚本完成集群的启动。在集群控制台界面，可以看到各服务的部署状态，如

图 4-22所示。从中可以看出，本系统的三个关键服务 webserver、scheduler、database 已经成功部署。

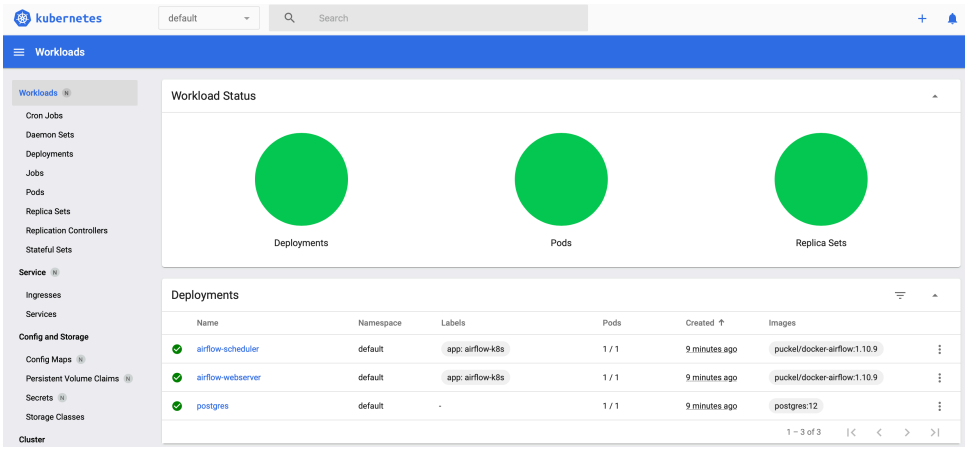

图 4-22: Airflow 集群服务的部署状态

通过 kubeadmin service 命令进入到 Airflow 的管理后台，点击 metrics 任务并手动触发。在有向无环图视图界面，系统展示出该度量任务中各节点间的依赖关系，如图 4-23所示。

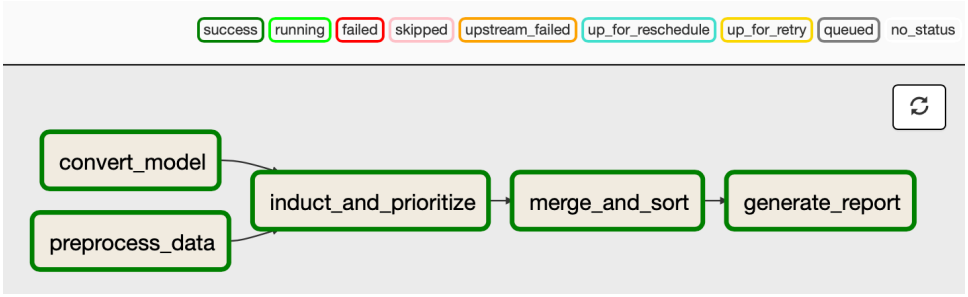

图 4-23: 根据度量任务建模的有向无环图

```
12m         Normal    Pulled              pod/metricsinductandprioritize-0ba59c71ed4948d5ae3167d3efab7cd7   Container image "puckel/docker
12m         Normal    Created             pod/metricsinductandprioritize-0ba59c71ed4948d5ae3167d3efab7cd7   Created container base
12m         Normal    Started             pod/metricsinductandprioritize-0ba59c71ed4948d5ae3167d3efab7cd7   Started container base
51m         Normal    Scheduled           pod/metricsinduction-3166556fdb814309885b5bdbb8203a21             Successfully assigned default/
```

图 4-24: 度量任务中推理节点的日志信息

在度量任务执行期间，可利用集群控制台界面查看 Pod 资源的使用情况。伴随着度量任务不同阶段的执行，Pod 的资源分配与回收也处于动态变化中。对于有向无环图中的每个任务节点，集群会为该节点创建环境并执行任务；当该节点任务完成后，集群负责回收对应的资源。利用 kubectl get events 能够更加清晰地展现这一过程，如图 4-24所示，该日志以度量任务中推理节点为例，展示了集群的弹性可扩展性。

4.6 本章小结

本章节主要对基于统计信息的测试用例系统各模块的详细设计与具体实现依次进行描述。首先对模型编码转换模块的时序图进行介绍，并从代码层面给出具体实现；其次，对数据预处理模块的时序图进行介绍，并展示了相关的实现细节；接着，介绍了本系统具体采用的测试用例度量算法，给出了相应的伪代码描述以及具体的工程实现；然后，对度量报告生成模块的设计进行了详细介绍，并给出详细的代码实现过程；最后，详细地介绍了支撑整个系统运行的计算引擎模块，包括架构选择、架构设计以及具体实现过程。对于系统的测试与验证部分，将在下一章中进行详细地介绍与说明。

第五章　系统测试与案例分析

本章将阐述系统测试与案例分析，首先介绍系统测试环境以及系统测试目标，然后分别介绍系统的功能性测试和非功能性测试过程。最后，通过实际的案例分析检验系统的有效性。

5.1　系统测试

5.1.1　测试环境

面向神经网络的测试用例度量系统的测试环境如表5-1所示，主要包括运行 Chrome 浏览器的用户客户端，其操作系统版本为 macOS 10.15、运行系统前后端服务的 ECS 服务器一以及运行计算引擎的 ECS 服务器集群。 由于系统部

表 5-1: 测试环境说明

设备名称	应用程序	版本
用户计算机（macOS 10.15）	Chrome 浏览器	Chrome 88.0
ECS 服务器 (Ubuntu 18.04 x86_64 4 核 CPU 8G 内存 500GB 硬盘)	Vue 前端服务	Vue.js 2.6.12
	Flask 后端服务	Flask 1.1.2
	Docker	Docker 20.10.6
ECS 服务器 (Ubuntu 18.04 x86_64 4 核 CPU 8G 内存 500GB 硬盘)	ONNX 运行时	ONNX 1.6.0
	PySpark 计算服务	Spark 1.6
	Airflow 调度平台	Airflow 1.10.10
	Kubernetes	Kubernetes 1.20

署采用了 Kubernetes 的容器编排技术，所以需要至少两台 ECS 机器来组建服务器集群。在测试环境中，主要包含以下几个服务：运行 Vue.js 框架的前端服务、基于 Flask 搭建的后端服务、基于 PySpark 与 Airflow 搭建的计算引擎，其中前后端服务以容器的方式部署在同一台 ECS 服务器上，计算引擎部署在由两台 ECS 机器组成的集群中。服务与服务之间通过 HTTP 的方式进行通信。所有

服务主机均使用阿里云的弹性云主机，其操作系统版本为 Ubuntu 18.04，CPU 核数为 4，内存容量为 8GB，并挂载 500GB 容量的云盘。服务器中所部属的相应软件版本如表 5-1所示。

5.1.2 功能测试

本节将根据功能性需求分析结果和用例评估结果，对系统的各项功能点进行测试，以保证系统的所提供功能能够满足规范以及预期结果。该过程主要为黑盒测试，用户无需关心任何程序结果或特性，只需要按照需求规格设计测试用例，检查系统功能是否能够按照设计正确地执行。

表 5-2: 创建度量任务测试用例

测试编号	TC1
对应用例编号	UC1
测试名称	创建度量任务
测试目标	系统服务端能否解析度量任务请求表单并触发引擎执行
前置条件	用户已经被认证并具有该操作权限
测试流程	1. 点击左边栏中“新建评估任务”标签； 2. 填写度量任务名称； 3. 上传需要度量的模型和数据文件； 4. 根据示例编写预处理操作逻辑； 5. 填写配置信息，如模型库、度量标准等，并提交；
预期结果	1. 页面中间弹出“新增度量任务”面板，用户可以根据需要填入相关的信息； 2. 在 Airflow 的管控平台，可以看到对应的工作流图被创建，并按照拓扑序进行执行； 3. 在数据库的 task 表中，新增一行关于该任务的记录； 4. 在模型桶中新增原始模型文件以及基于 ONNX 的模型文件，数据集桶中新增原始数据集文件以及预处理后的版本，在度量结果桶中新增该任务的结果文件；
测试结果	与预期结果相符

表 5-2为创建度量任务的详细测试用例设计。该用例需要验证创建度量任

务的链路是否正常工作，其中包括后台是否接收到用户上传的模型文件与数据集文件并写入对应数据桶中、后台是否能鲁棒地处理用户自定义的预处理逻辑以及后台是否能够正确解析户提交的表单信息并触发任务实例的执行。该用例还需验证如果出现模型不能正常转换的情况，系统应能够给出友好的错误提示，以方便用户查找错误原因。

表 5-3: 查看度量任务列表测试用例

测试编号	TC2
对应测试用例编号	UC2
测试名称	查看度量任务列表
测试目标	用户查看所有已经创建的度量任务
前置条件	用户已经被认证并具有该操作权限
测试流程	1. 点击左边栏中“新建度量任务”标签； 2. 按照相应要求填写信息并创建度量任务； 3. 点击左边栏中“查看度量任务”标签； 4. 在表格上方的搜索框，通过键入关键字来快速查找相应的度量任务表项；
预期结果	1. 在“查看度量任务”页面中，以表格的方式展示出所有属于当前用户的度量任务； 2. 表格中每一行包含任务 ID、任务名称、模型名称、数据集名称、创建时间、执行状态等信息； 3. 当通过搜索框进行检索时，系统能返回包含相应关键字的度量任务表项；
测试结果	与预期结果相符

表5-3为查看度量任务列表的详细测试用例设计。该用例需要验证服务端是否能从数据库的 task 表中获取所有属于当前用户的度量任务。当度量任务的状态发生改变时，列表中对应的执行状态字段也需随之变化，例如当该度量任务的执行状态处于执行中时，末列的操作字段应包含查看日志的选项；当该度量任务的执行状态为已完成时，末列的操作字段应包含查看报告的选项。此外，该用例还需验证列表上方的搜索框是否正常工作，即是否能够在任务名称、模型名称、数据集名称这三个字段中进行模糊搜索。

表5-4为终止度量任务的详细测试用例设计。在用户提交度量任务后，若

发现度量任务的配置文件存在错误，用户可以在查看度量任务的页面中点击终止任务选项，以请求取消度量任务的执行。考虑到多层函数转发调用、中间件响应等因素影响，系统往往不能立即终止该任务，需要等待一定的时间以保证所有资源被释放、相应数据被写入、日志信息被记录等。当系统成功地终止度量任务后，该任务在度量任务列表中的执行状态字段将被更新为已终止。

表 5-4: 终止度量任务测试用例

测试编号	TC3
对应测试用例编号	UC3
测试名称	终止度量任务
测试目标	当用户提交创建度量任务的请求后，若发现配置信息填写错误，可以选择提前终止该任务的执行
前置条件	用户已经被认证并具有该操作权限，并且已经创建并提交相关的度量任务
测试流程	1. 点击左边栏中“查看度量任务”标签； 2. 通过搜索框检索需要进行操作的任务表项； 3. 若当前任务的执行状态为执行中，则点击该表项末列的终止按钮； 4. 页面提示该度量任务已被终止；
预期结果	1. 在 Airflow 的管控平台中，显示该次运行实例已经被终止； 2. 在 Kubernetes 的运维平台上，与该任务相关的资源均被回收； 3. 在数据库的 task 表中，对应当前运行实例行的执行状态字段被更新为已终止； 4. 在“查看度量任务”页面中，对应表项的执行状态字段被更新为已终止； 5. 查询对应日志信息，应有该运行实例被用户主动终止的相关记录；
测试结果	与预期结果相符

表5-5为查看度量任务执行日志的详细测试用例设计。当度量任务在执行过程中或执行完毕时，用户可以获取到当前度量任务的日志信息。具体来说，

通过点击查看度量任务页面中的查看日志选项，用户会跳转到日志查询界面。在该界面中，用户通过点击有向无环图上的节点，系统会展示对应节点的执行日志，这包括关键函数的调用、调试信息、节点执行结果等。当度量任务执行出现错误时，用户就可以利用该页面，点击有向无环图中的红色节点，以查看具体、详细的错误信息。该测试用例主要验证系统是否可以提取出各节点的执行日志信息以及报错信息。

表 5-5: 查看度量任务执行日志测试用例

测试编号	TC4
对应测试用例编号	UC4
测试名称	查看度量任务执行日志
测试目标	用户能够查看到当前任务执行实例所对应每个节点的具体日志信息
前置条件	用户已经被认证并具有该操作权限，并且已经创建并提交相关的度量任务
测试流程	1. 点击左边栏中“查看度量任务”标签； 2. 通过搜索框检索到需要进行操作的表项； 3. 点击该表项的末列查看日志按钮，跳转到日志查看界面； 4. 通过点击有向无环图上的具体节点，系统展示出对应节点的日志信息；
预期结果	1. 在日志查看页面，用户点击对应节点，系统能够显示对应节点的执行日志；
测试结果	与预期结果相符

表5-6为度量报告生成的详细测试用例设计。该用例主要验证系统能否为已经执行完毕的度量任务产生出对应的分析报告。在提交的度量任务完成后，用户通过在查看度量任务页面中点击查看报告选项，就能获得当前任务所对应的度量分析报告。完整的度量分析报告包括模型与数据集元数据概述、有向无环图各节点执行时间甘特图、数据集样本度量值分布的柱状图、数据集样本分类类别与度量值的热力图、优先级度量后前 10 位与末 10 位数据样本的可视化结果。此外，系统提供导出分析报告的功能，因此该用例还需测试是否能够完整、无误地导出度量分析报告。

表 5-6: 度量报告生成的测试用例

测试编号	TC5
对应测试用例编号	UC5
测试名称	度量报告生成
测试目标	在度量任务执行完成后，用户可以获得关于该次度量任务的分析报告。
前置条件	用户已经被认证并具有该操作权限，并且已经创建并提交相关的度量任务
测试流程	1. 点击左边栏中“查看度量任务”标签； 2. 通过搜索框检索到需要进行操作的表项； 3. 等待该表项的执行状态变更为已完成； 4. 点击该表项末列报告按钮，跳转度量报告界面； 5. 通过缩放、拖拽等方式动态调整度量报告中若干图表的展示信息； 6. 点击度量报告右上方“导出”按钮；
预期结果	1. 展示模型与数据集相关元数据的描述； 2. 以甘特图形式展示有向无环图各节点执行时间； 3. 以柱状图形式展示数据样本的度量值分布情况； 4. 以热力图形式展示样本类别与度量值的关系； 5. 以轮播形式展现度量排序结果中排名前 10 位与末 10 位样本的可视化结果； 6. 当用户点击导出按钮时，系统后台对报告及其资源进行打包并响应下载请求；
测试结果	与预期结果相符

5.1.3 性能测试

本部分内容阐述了利用开源压力测试程序 JMeter 对测试用例度量系统的关键接口进行压力测试。

接口压力测试的流程如图5-1所示。首先，需要确定系统待测接口以及具体的测试需求。然后，根据系统所用物理资源，设计压力测试方案并执行压力测试。最后，根据压力测试结果进行分析。若不满足既定的测试需求，需分析

具体原因并系统瓶颈作出修改，再进行压力测试；反之，终止测试流程并输出压力测试报告。

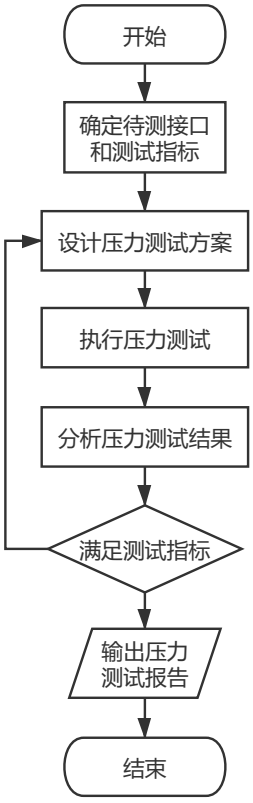

图 5-1: 系统压力测试流程图

本次压力测试的具体流程如下：

1. 确定待测接口和测试需求。本次压力测试的对象为上传模型及数据集文件的 POST 接口、创建度量任务的 POST 接口、查询度量任务的 GET 接口、获取度量报告的 GET 接口以及系统首页的 GET 接口。根据用户调研，接口请求响应时间要求在 2 秒以内。
2. 设计压力测试方案。设计以上被测接口的压力测试并发值为 120，即模拟每秒 120 个用户请求，压测时长为 60s。
3. 执行压力测试。本次压力测试使用 JMeter 完成。首先添加线程组，设置线程数为 120，Ramp-Up 时间为 60 秒；然后，添加 HTTP 请求，并编辑相应的请求接口以及参数；最后，添加结果监听器，用来记录此过程中产生的压力测试数据。
4. 分析压力测试结果。根据 JMeter 软件中结果监听器中的数据进行分析，若所有接口的响应时间小于 2 秒且请求异常数量为 0.00%，则说明系统被测接

口测试结果符合预期；反之，则说明接口并未达到预期，需要分析原因并修改代码，再次设计压力测试方案并执行，直到满足测试需求。

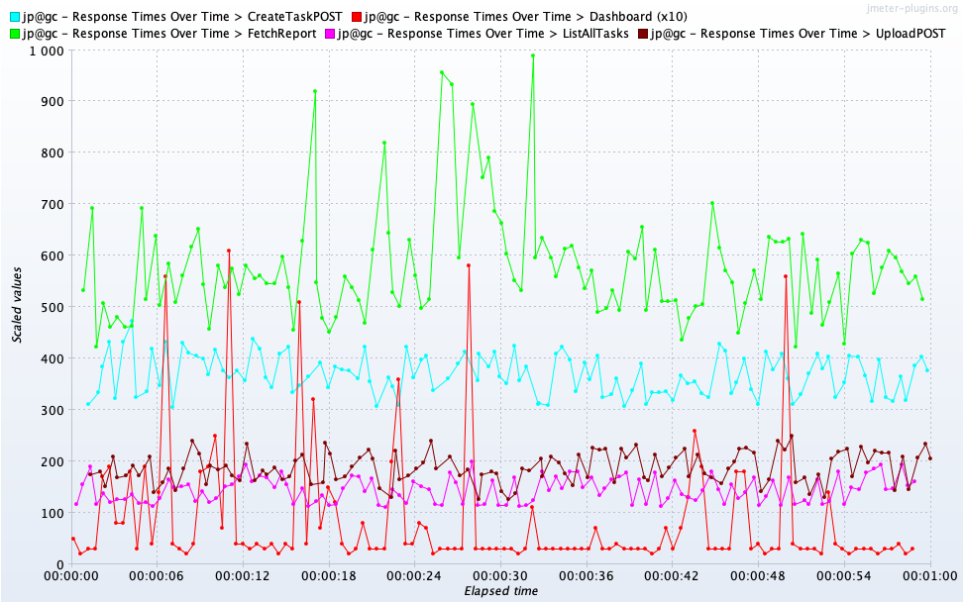

图 5-2: 系统关键接口压力测试结果图

测试用例度量系统的若干待测接口压力测试数据结果如图5-2所示。考虑系统被测接口中的系统首页 GET 接口、查询度量任务 GET 接口以及上传文件 POST 接口，其最大响应时间均小于 250 毫秒，平均响应时间为 113 毫秒且方差较小，较为稳定，符合预期。创建度量任务 POST 接口的平均响应时间为 369 毫秒，90% 的请求响应时间小于 422 毫秒，最大响应时间为 473 毫秒。考虑到该接口需要调用 Airflow 的 RESTful 接口，其内部存在一定时延，因此结果符合预期。FetchReport 接口的响应时间曲线波动较大，其平均响应时间为 581 毫秒、最大响应时间为 983 毫秒，90% 的请求响应时间小于 687 毫秒。原因在于系统在生成度量报告时，需从对象存储系统中即时下载度量结果以及可视化的图片数据，考虑到公共网络的传输质量，确实会产生一定的丢包与时延，因此测试结果符合预期。综上，所有被测接口的压力测试结果均符合要求。

5.2 案例分析

为验证本系统能够通过测试用例度量排序有效解决测试数据标注成本高、模型泛化能够力调优效率低的问题，本节选取了自动驾驶领域中交通标志识别的案例进行验证。在本案例中，用户已经设计并实现了一种识别模型，并且在

测试集上具有一定的准确度，进入模型性能调优阶段。在经历测试用例度量分析后，本系统将挑选出价值较高的测试用例，并使用该部分测试用例进行模型重训练，以进一步地提高模型的性能。

5.2.1 案例设计与执行

在成熟的自动驾驶系统中，通常包含若干个决策模块，例如车道线检测、交通标志识别、行为模仿、车辆检测与跟踪、多源传感器数据融合等。本案例中将以交通标志识别为例，验证基于统计信息的测试用例度量系统的有效性。

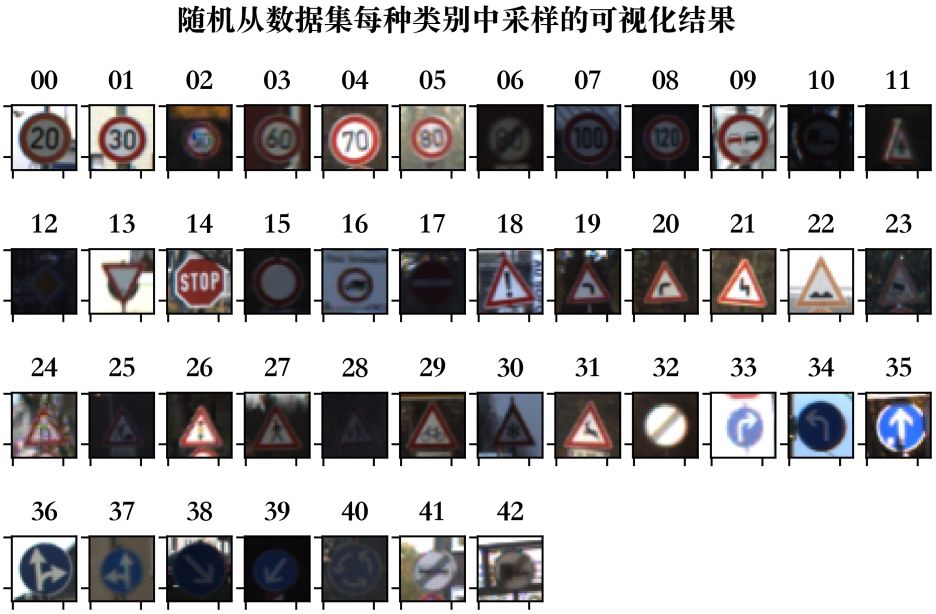

图 5-3: 从 43 类中随机采样的数据样本展示

本案例所使用的是德国交通标志数据集 GTSRB，它是一个多分类任务的数据集，广泛用于自动驾驶、车辆辅助系统的测试验证中，总共包含 43 个不同种类的 34 799 张训练图片以及 12 630 张测试图片，全部从真实世界中的场景中收集所得，如图 5-3所示。在该数据集中，提供了 csv 格式的标签文档，用以指明所需识别的交通标志在样本图片中的坐标以及该样本所对应的真实标签。

根据 LeCun 等人的描述，针对该数据集，需要进行如下预处理操作 [32]。首先，需要将每张图片从 RGB 空间变换到 YUV 空间，并舍弃除 Y 通道外的所有信息，以减少外部噪音对模型分类能力的干扰。接着，采用直方图均衡的方法对每张样本图片的对比度进行调整。最后，对每张样本图片进行标准化处理，即对每张图片减去数据集总体的均值并除以数据集总体的标准差。此外，

为最大程度地保证训练出模型的性能，可通过对训练样本每类别个数的可视化结果，对样本量较少的种类进行数据扩增，以保证不同类别的训练数据尽量均衡。原始数据集不同类别样本数的分布情况如图 5-4所示。

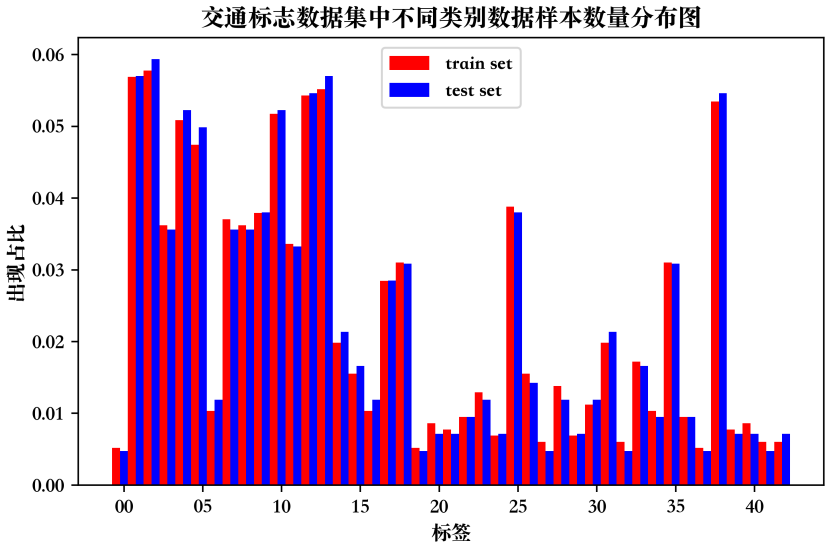

图 5-4: GTSRB 数据集不同类别样本数量分布情况

本案例将使用最为常见的卷积神经网络模型进行试验。该卷积神经网络模型一共包含 7 层，其中前 5 层用于卷积操作，末 2 层作为全连接层，卷积核的大小均为 3*3，步长均为 1。此外，卷积层的输出将进行池化计算并随机采样，以防止网络训练出现过拟合的现象。模型的大致结构如图 5-5所示。

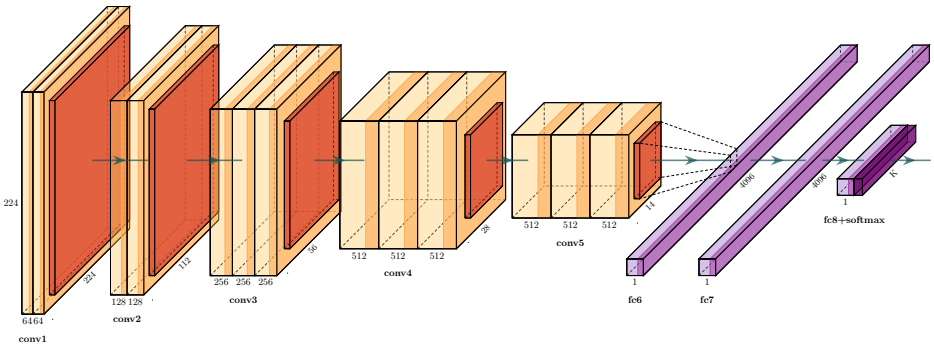

图 5-5: 卷积神经网络结构图

本案例用户应提前完成神经网络在该数据集上的训练并在测试集上取得良好的效果。例如，用户使用 TensorFlow 完成模型的搭建，并在每批次 2048 张

图片、每轮迭代 2000 批次、共 5 轮的参数下进行训练，每轮后在验证集上进行测试，结果如图 5-6所示。

```
INFO:tensorflow:Restoring parameters from /content/drive/MyDrive/checkpointer/
Train Accuracy = 0.964 - Validation Accuracy: 0.955
Performance on test set: 0.857
INFO:tensorflow:Restoring parameters from /content/drive/MyDrive/checkpointer/
Train Accuracy = 0.991 - Validation Accuracy: 0.986
Performance on test set: 0.912
INFO:tensorflow:Restoring parameters from /content/drive/MyDrive/checkpointer/
Train Accuracy = 0.996 - Validation Accuracy: 0.988
Performance on test set: 0.920
INFO:tensorflow:Restoring parameters from /content/drive/MyDrive/checkpointer/
Train Accuracy = 0.997 - Validation Accuracy: 0.990
Performance on test set: 0.925
INFO:tensorflow:Restoring parameters from /content/drive/MyDrive/checkpointer/
Train Accuracy = 0.998 - Validation Accuracy: 0.992
Performance on test set: 0.931
```

图 5-6: 不同验证点的 VGG 模型在测试集上的结果图

从图 5-6中可看出，训练集的准确率与验证集上的准确率已接近 1，测试集上的变现随迭代次数增加而提升越来越缓慢。因此进行更多轮的迭代很可能造成模型的过拟合，从而导致模型在测试集上的准确率进一步降低。用户为进一步地提升模型泛化能力，保障模型的推理质量，选择测试用例度量的方法，查找具体哪些测试用例容易产生错误，并利用一部分数据进行重训练，以进一步地提升模型的泛化能力。

当度量任务完成后，系统会首先以可视化地方式给用户呈现度量结果中前 10 个数据样本与后 10 个数据样本，如图 5-7所示。 前 10 个样本具有较高的基

图 5-7: 度量结果中前 10 个与后 10 个数据样本的可视化结果

尼不纯度值，说明神经网络在进行推理时，并没有非常确定的概率来判决该样本应归属为哪类；后 10 个样本具有很低的基尼不纯度值，表示神经网络模型对

该样本的分类非常确定。从人类主观认知地角度来看，后 10 个数据样本相比于前 10 个样本明显更容易进行识别与分类，前 10 个样本存在一些由于采集质量而导致的低像素问题，这类样本对于模型来说确实更难判决，因此度量结果具有一定的合理性。

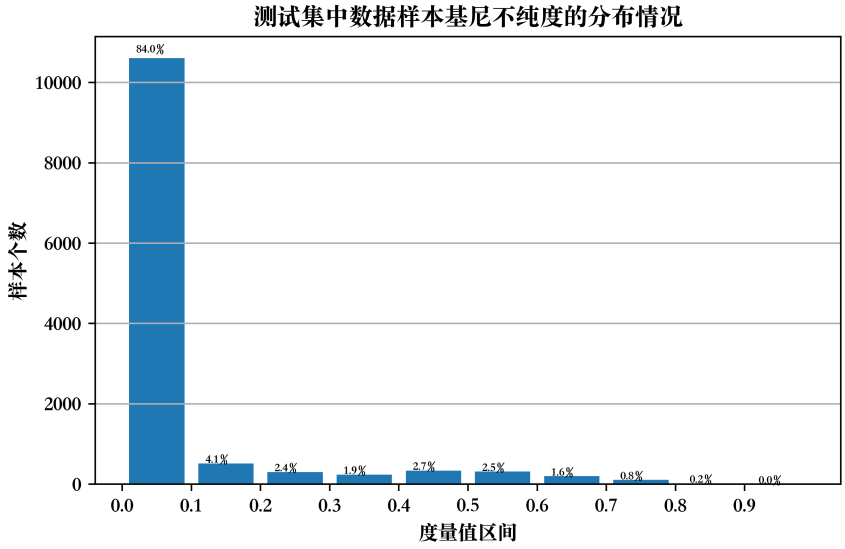

图 5-8: 度量结果中不同样本基尼不纯度值的分布情况

度量报告中还会给出该数据集中度量标准值的分布情况，通常以柱状图的方式呈现，如图 5-8所示。柱状图显示，有 84.0% 的样本数据具有极低的基尼不纯度，即说明神经网络对该部分数据分类的表现较好；有大概 5% 的数据样本其基尼不纯度值超过 0.5，意味着神经网络对该 5% 的数据分类效果较差。

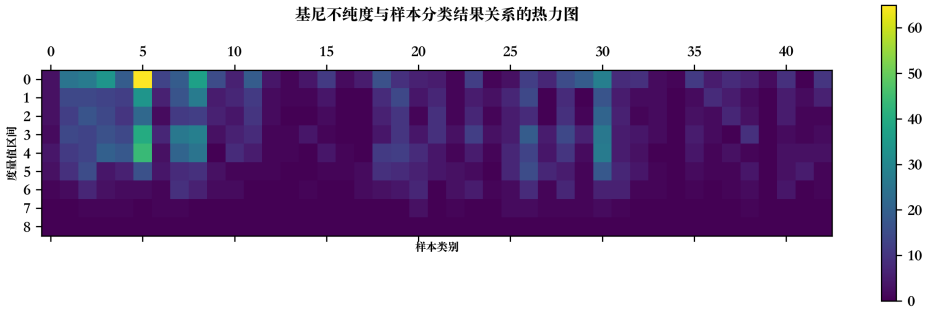

图 5-9: 基尼不纯度值与样本分类关系的热力图

此外，度量报告中还会提供基尼不纯度值与样本类别数的热力图表示，如图 5-9所示。从热力图中，可以看出基尼不纯度值偏高的样本主要分布在第 3

类、第 5 类、第 8 类以及第 30 类中，它们分别对应 60 英里限速、80 英里限速、120 英里限速、注意下雪交通标志，因此用户可以重点检查这部分数据，以查看是否在数据采集阶段受到外部因素的影响。

图 5-10: 度量分析报告界面图

通过分析度量任务执行日志并组合上述图表信息，本系统在任务执行完成后给出如图 5-10所示的度量分析报告。在该度量报告中，给出了度量任务每阶段执行耗时的甘特图、模型与数据集信息概述、度量后部分数据样本的可视化结果、数据集度量结果的分布情况等。用户可以根据此报告定位模型缺陷，例如特征工程是否完备、数据预处理操作是否合理等。为方便用户在本地调试使用，本报告能以 ZIP 的形式打包下载，其中包括每个数据样本在神经网络中的输出向量值、基尼不纯度值、所有数据样本排序后的下标值列表以及该报告的 HTML 版本。这部分数据可支撑用户在本地对模型进行修改、调试。

通过使用基于基尼不纯度的测试用例度量技术，用户可以对当前模型存在的一些问题有直观地了解，同时系统所生成的度量报告从多角度描述了数据集中所存在的一些问题。此外，利用优先级排序后的部分数据进行重训练，能够在损失函数曲线已经收敛的前提下进一步地提高模型的泛化能力，在测试集上取得更好的表现。

5.2.2 实验调查

为验证经优先级度量后的数据确实能对提升模型泛化能力以及特征工程改进有帮助，本节以上述案例为基础，选取优先级排序后前 10% 的样本，以 1% 为步长，对模型进行重训练，并与随机采样、NAC、SNAC 等算法进行对比，重训练后的模型性能曲线如图 5-11所示。实验结果说明采用基尼不纯度进行优先级排序后的样本确实能够有效地提高模型泛化能力。

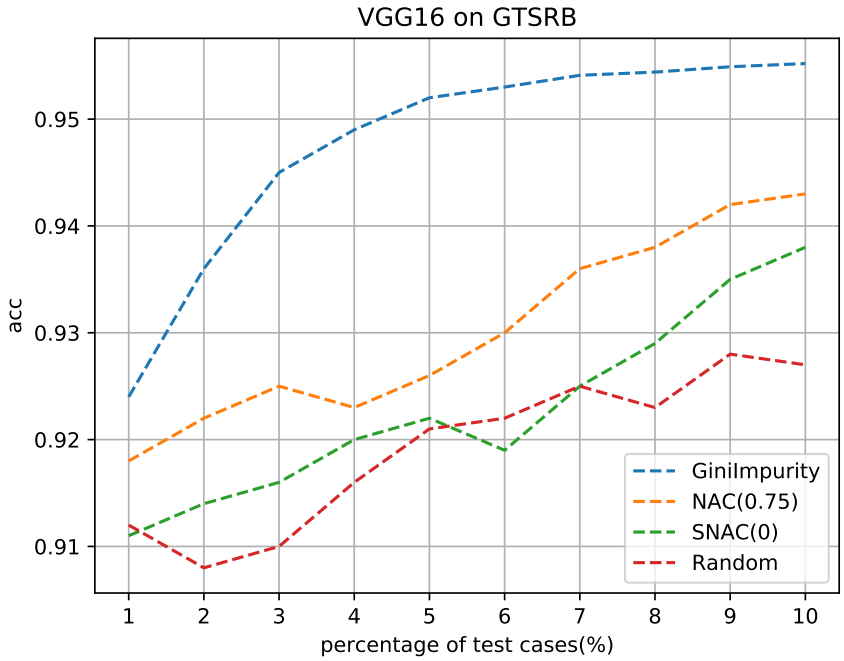

图 5-11: 不同度量算法下模型重训练后的性能曲线图

为调查基于基尼不纯度的测试用例度量技术对辅助开发人员的项目迭代效率与提升深度学习软件的作用与价值，本节还针对用户体验设计了一个问卷实验。实验中，安排 14 名程序员每人完成某个场景下的测试用例度量任务。该场景为自动驾驶中的交通标识符识别。对于该场景下的任务，参与人员被要求前一次不使用测试用例度量系统，后一次使用测试用例度量系统。在所有的程序员完成任务后，对他们进行简短的问卷调查。每个任务都提供了相应场景下的预训练模型、带有标签的训练数据集与未带有标签的测试集，参与者被要求尽可能地提高模型的泛化性能并找出数据集中质量较差的样本数据。

针对测试用例度量系统中的功能，对 14 名参与者提出以下三个问题：

第一问：基于基尼不纯度的测试用例度量算法在特征工程迭代时是否有

用。对于系统提供的度量报告分析功能，问卷结果显示，在该场景任务中，有 10 名程序员认为其非常有用，有 3 名程序员认为偶尔有用，有 1 名程序员认为没有用；

第二问：在提升模型性能时是否有必要提供经过优先级排序的样本列表。对于系统提供的优先级度量后数据样本列表，问卷结果显示，在该场景任务中，有 13 名程序员认为其非常有用，1 名程序员认为偶尔有用；

第三问：在数据质量评估过程中是否有必要提供度量后的分析报告。问卷结果显示，在该场景任务中，有 12 名程序员认为其非常有用，有 2 名程序员认为偶尔有用；

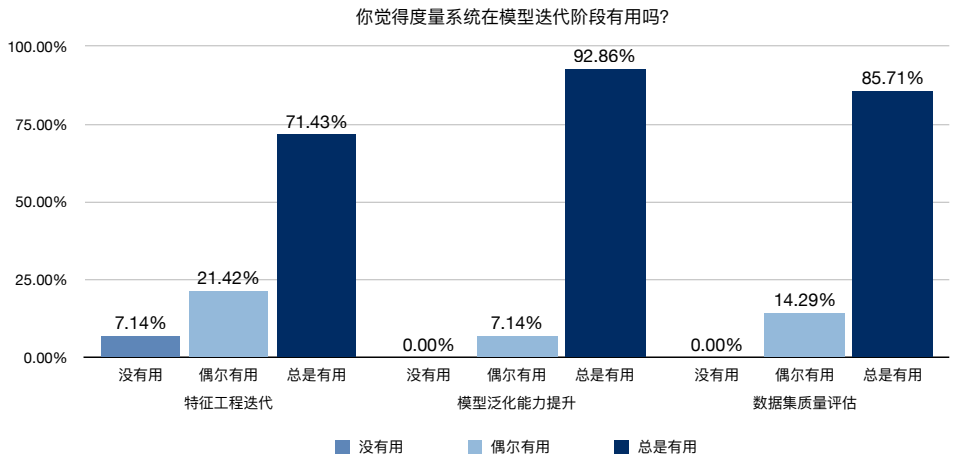

图 5-12: 问卷调查结果统计图

综合以上调查结果，如图5-12所示，基于基尼不纯度的测试用例度量系统在实际场景下对程序员的开发工作起到了比较强的辅助作用，充分提高了数据集预处理与模型改进的效率，减少了在新样本数据上花费的标注成本，提升了用户整体的使用体验。

5.3 本章小结

本章节主要介绍了系统测试，从用户角度设计并执行系统功能测试，对系统的测试用例度量算法接口进行了压力测试，功能测试结果和压力测试结果均符合预期的系统需求。通过分析实际的测试用例度量案例，证明了本系统的有效性和可靠性。

第六章　总结与展望

6.1　工作总结

在深度学习快速发展的背景下，越来越多的深度学习技术在日常生活中得到应用，包括人脸识别、信息推荐、自动驾驶等。但是，深度学习模型也会出现一些错误决策，这在自动驾驶等生命攸关场景下，会危害人类生命安全。因此，有必要对深度学习模型进行大量地测试以保证其安全性。在实际工程中可通过多种方式来采集到大量的无标签数据，开发团队雇用额外的人员来完成数据的标注工作，这将导致大量的人力成本开销。本论文实现了基于统计信息的测试用例度量系统，能够在大量的测试样本中挑选出一部分价值较高的样本，该部分样本相比于随机挑选的样本能够更有效地检测出模型中存在的问题，从而极大地减少数据集的标注成本。此外，通过利用优先级较高的数据对模型进行重训练，相比于随机采样，前者能更加有效地提升模型的泛化能力。

本系统使用 Vue.js 作为前端开发框架，以 ElementUI 完成页面布局与设计，使用 Webpack 进行模块的打包与管理。后端使用轻量级的 Flask 框架来处理用户的请求，使用 MySQL 数据库来满足度量任务的存储需求。核心度量算法被封装为计算引擎，使用 Airflow 调度中间件作为引擎的主体部分，相关业务逻辑以插件的方式嵌入到引擎中。模型统一采用 ONNX 的编码格式，以解决在推理过程中与过多第三方框架紧耦合的问题。通过利用 ONNXRuntime 的高度优化，使得推理计算的时延和开销做到最小。为了应对大规模的数据集度量任务，系统利用了 MapReduce 的编程模型对具体问题进行建模，并通过 PySpark 大数据处理框架完成相关实现。在物理资源管理方面，为了保证资源的高效使用以及保障系统的线性可扩展性，系统基于 Kubernetes 编排引擎，以容器为粒度对资源进行分配与回收。通过利用水平容器扩容机制，系统可以实现存在度量任务时分配相关资源进行计算，任务完成后回收相关资源。为了应对数据集、模型等大型文件，系统采用对象存储等云存储服务，进一步地提升了系统的承载能力，使其能够满足更加多元化的需求。

本文对本系统进行了功能性测试、非功能性测试以及自动驾驶中交通标志

识别的案例分析，测试结果与实际案例均验证了系统的有效性。目前本系统已经成功上线并应用于幕测人工智能基础平台。在用户的实际使用过程中，本系统能够有效地降低数据集的标注成本，检查数据预处理与特征工程的完备性，利用重训练可以快速地提高模型的泛化能力，为用户提供了良好的体验。

6.2 进一步展望

本文设计并实现了基于基尼不纯度的测试用例度量系统，用户可在拥有模型和数据集的基础上，通过本系统度量出新产生数据样本对提高模型性能的优先级排序，从而大幅度地减少测试数据的标注成本。通过利用标注后优先级高的测试样本对模型进行重训练，能够快速、有效地提升模型在验证集上的表现。当前本系统可以从以下几个方面进行改进：

1. 度量能力。本系统目前支持处理包括 TensorFlow、PyTorch、Paddle 主流的深度学习框架，对于一些在特定领域专用的框架，本系统暂未提供支持。随着通信网和嵌入式设备的不断发展，数据流量呈井喷式增长，因此需要随之提高本系统对超大规模数据集与模型的存储与分析能力。
2. 交互形式。在数据集预处理方面，为了能够最大程度地支持用户对预处理操作的定义与配置能力，本系统直接以领域特定语言为载体、用语句作为粒度，来满足用户自定义预处理操作的需求。但这也会导致用户的使用门槛提升。因此，未来可以在用户交互方面作出改进，降低系统使用门槛，服务有需求的用户。

综上，目前系统中仍存在这问题，具备很多可以改进的空间。通过优化测试用例的度量算法、系统对于大规模数据的承载能力、简化用户界面的交互方式，使得本系统能够进一步地提升数据集度量的准确性与合理性，扩大系统的应用场景，增强系统的负载能力，降低操作门槛及所需的领域知识，最终服务更多的用户，帮助其提高工程迭代效率。

致　　谢

研究生的生活充实而短暂，基于神经网络的测试用例度量技术是我在读研时的重点研究方向，非常有幸能在实验室中度过一段充实的科研生活。

感谢陈振宇老师在研究生期间对我学业的指导与帮助。在毕业设计中，陈老师从选题到最终实验，提供了很多建设性的意见，对论文写作与项目的设计开发实验都提供了很大的帮助。

感谢冯洋老师在论文修改与项目开发中提供的专业意见，并给予了很大的帮助。

感谢父母在我研究生学习阶段给予的经济支持与精神支持。

感谢研究生期间南京大学软件学院对我的教育与培养，让我在专业领域上不断充实自己，获得成长。

参考文献

[1] KER J, WANG L, RAO J, et al. Deep Learning Applications in Medical Image Analysis[J/OL]. IEEE Access, 2018, 6: 9375–9389. https://doi.org/10.1109/ACCESS.2017.2788044.

[2] STRUBELL E, GANESH A, MCCALLUM A. Energy and Policy Considerations for Deep Learning in NLP[C/OL] // KORHONEN A, TRAUM D R, MÀRQUEZ L. Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers. [S.l.]: Association for Computational Linguistics, 2019: 3645–3650. https://doi.org/10.18653/v1/p19-1355.

[3] UÇAR A, DEMIR Y, GÜZELIS C. Object recognition and detection with deep learning for autonomous driving applications[J/OL]. Simul., 2017, 93(9): 759–769. https://doi.org/10.1177/0037549717709932.

[4] PAN S J, YANG Q. A Survey on Transfer Learning[J/OL]. IEEE Trans. Knowl. Data Eng., 2010, 22(10): 1345–1359. https://doi.org/10.1109/TKDE.2009.191.

[5] WANG Y, YAO Q, KWOK J T, et al. Generalizing from a Few Examples: A Survey on Few-shot Learning[J/OL]. ACM Comput. Surv., 2020, 53(3): 63:1–63:34. https://doi.org/10.1145/3386252.

[6] ROTHERMEL G, UNTCH R H, CHU C, et al. Test Case Prioritization: An Empirical Study[C/OL] // 1999 International Conference on Software Maintenance, ICSM 1999, Oxford, England, UK, August 30 - September 3, 1999. [S.l.]: IEEE Computer Society, 1999: 179–188. https://doi.org/10.1109/ICSM.1999.792604.

[7] GOODFELLOW I J, SHLENS J, SZEGEDY C. Explaining and Harnessing Adversarial Examples[C/OL] // BENGIO Y, LECUN Y. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. 2015.
http://arxiv.org/abs/1412.6572.

[8] SZEGEDY C, ZAREMBA W, SUTSKEVER I, et al. Intriguing properties of neural networks[C/OL] // BENGIO Y, LECUN Y. 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings. 2014.
http://arxiv.org/abs/1312.6199.

[9] PAPERNOT N, MCDANIEL P D, JHA S, et al. The Limitations of Deep Learning in Adversarial Settings[C/OL] // IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016. [S.l.] : IEEE, 2016 : 372 – 387.
https://doi.org/10.1109/EuroSP.2016.36.

[10] SU J, VARGAS D V, SAKURAI K. One Pixel Attack for Fooling Deep Neural Networks[J/OL]. IEEE Trans. Evol. Comput., 2019, 23(5) : 828 – 841.
https://doi.org/10.1109/TEVC.2019.2890858.

[11] SUN Y, WU M, RUAN W, et al. Concolic testing for deep neural networks[C/OL] // HUCHARD M, KÄSTNER C, FRASER G. Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018. [S.l.] : ACM, 2018 : 109 – 119.
https://doi.org/10.1145/3238147.3238172.

[12] ZHAO Z, DUA D, SINGH S. Generating Natural Adversarial Examples[C/OL] // 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. [S.l.] : OpenReview.net, 2018.
https://openreview.net/forum?id=H1BLjgZCb.

[13] XIE C, TAN M, GONG B, et al. Adversarial Examples Improve Image Recognition[C/OL] // 2020 IEEE/CVF Conference on Computer Vision and Pattern

Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020. [S.l.]: IEEE, 2020: 816–825.
https://doi.org/10.1109/CVPR42600.2020.00090.

[14] WANG Z, CHEN L, XU B, et al. Cost-Cognizant Combinatorial Test Case Prioritization[J/OL]. Int. J. Softw. Eng. Knowl. Eng., 2011, 21(6): 829–854.
https://doi.org/10.1142/S0218194011005499.

[15] ROTHERMEL G, UNTCH R H, CHU C, et al. Prioritizing Test Cases For Regression Testing[J/OL]. IEEE Trans. Software Eng., 2001, 27(10): 929–948.
https://doi.org/10.1109/32.962562.

[16] SRIKANTH H, WILLIAMS L A, OSBORNE J A. System test case prioritization of new and regression test cases[C/OL] // 2005 International Symposium on Empirical Software Engineering (ISESE 2005), 17-18 November 2005, Noosa Heads, Australia. [S.l.]: IEEE Computer Society, 2005: 64–73.
https://doi.org/10.1109/ISESE.2005.1541815.

[17] JIA Y, HARMAN M. An Analysis and Survey of the Development of Mutation Testing[J/OL]. IEEE Trans. Software Eng., 2011, 37(5): 649–678.
https://doi.org/10.1109/TSE.2010.62.

[18] GOLOV N, RÖNNBÄCK L. Big Data normalization for massively parallel processing databases[J/OL]. Comput. Stand. Interfaces, 2017, 54: 86–93.
https://doi.org/10.1016/j.csi.2017.01.009.

[19] SHORTEN C, KHOSHGOFTAAR T M. A survey on Image Data Augmentation for Deep Learning[J/OL]. J. Big Data, 2019, 6: 60.
https://doi.org/10.1186/s40537-019-0197-0.

[20] TAYLOR L, NITSCHKE G. Improving Deep Learning with Generic Data Augmentation[C/OL] // IEEE Symposium Series on Computational Intelligence, SSCI 2018, Bangalore, India, November 18-21, 2018. [S.l.]: IEEE, 2018: 1542–1547.
https://doi.org/10.1109/SSCI.2018.8628742.

[21] LEMLEY J, BAZRAFKAN S, CORCORAN P. Smart Augmentation Learning an Optimal Data Augmentation Strategy[J/OL]. IEEE Access, 2017, 5: 5858–

5869.
https://doi.org/10.1109/ACCESS.2017.2696121.

[22] KOSAR T, LÓPEZ P E M, BARRIENTOS P A, et al. A preliminary study on various implementation approaches of domain-specific language[J/OL]. Inf. Softw. Technol., 2008, 50(5): 390 – 405.
https://doi.org/10.1016/j.infsof.2007.04.002.

[23] SHANNON C E. A mathematical theory of communication[J/OL]. Bell Syst. Tech. J., 1948, 27(3): 379 – 423.
https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

[24] RAILEANU L E, STOFFEL K. Theoretical Comparison between the Gini Index and Information Gain Criteria[C/OL] // International Symposium on Artificial Intelligence and Mathematics, AI&M 2002, Fort Lauderdale, Florida, USA, January 2-4, 2002. 2002.
http://rutcor.rutgers.edu/%7Eamai/aimath02/PAPERS/25.ps.

[25] HORGAN J R, LONDON S, LYU M R. Achieving Software Quality with Testing Coverage Measures[J/OL]. Computer, 1994, 27(9): 60 – 69.
https://doi.org/10.1109/2.312032.

[26] LI Z, MA X, XU C, et al. Structural coverage criteria for neural networks could be misleading[C/OL] // SARMA A, MURTA L. Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019, Montreal, QC, Canada, May 29-31, 2019. [S.l.]: IEEE / ACM, 2019: 89 – 92.
https://doi.org/10.1109/ICSE-NIER.2019.00031.

[27] OYALLON E. Building a Regular Decision Boundary with Deep Networks[C/OL] // 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017. [S.l.]: IEEE Computer Society, 2017: 1886 – 1894.
https://doi.org/10.1109/CVPR.2017.204.

[28] DEAN J, GHEMAWAT S. MapReduce: Simplified Data Processing on Large Clusters[C/OL] // BREWER E A, CHEN P. 6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004. [S.l.] : USENIX Association, 2004 : 137 – 150. http://www.usenix.org/events/osdi04/tech/dean.html.

[29] RICHARDSON M, DOMINGOS P M. The Intelligent surfer: Probabilistic Combination of Link and Content Information in PageRank[C/OL] // DIETTERICH T G, BECKER S, GHAHRAMANI Z. Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]. [S.l.] : MIT Press, 2001 : 1441 – 1448. https://proceedings.neurips.cc/paper/2001/hash/a501bebf79d570651ff601788ea9d16d-Abstract.html.

[30] NGUYEN G, DLUGOLINSKY S, BOBÁK M, et al. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey[J/OL]. Artif. Intell. Rev., 2019, 52(1) : 77 – 124. https://doi.org/10.1007/s10462-018-09679-z.

[31] BERNSTEIN D. Containers and Cloud: From LXC to Docker to Kubernetes[J/OL]. IEEE Cloud Comput., 2014, 1(3) : 81 – 84. https://doi.org/10.1109/MCC.2014.51.

[32] SERMANET P, LECUN Y. Traffic sign recognition with multi-scale Convolutional Networks[C/OL] // The 2011 International Joint Conference on Neural Networks, IJCNN 2011, San Jose, California, USA, July 31 - August 5, 2011. [S.l.] : IEEE, 2011 : 2809 – 2813. https://doi.org/10.1109/IJCNN.2011.6033589.