南京大學

研 究 生 毕 业 论 文
（申请工程硕士学位）

论　文　题　目 基于差分测试的深度学习算子稳定性分析技术

作　者　姓　名 吕军

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，房春荣 助理研究员

2021 年 **5** 月 **18** 日

学　　号：**MF1932133**

论文答辩日期：**xxxx** 年 **xx** 月 **xx** 日

指 导 教 师：　　（签字）

Deep learning framework operator stability test technology based on differential testing

by

Jun Lv

Supervised by

Professor Zhenyu Chen, Assistant Researcher Chunrong Fang

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 18, 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：基于差分测试的深度学习算子稳定性分析技术

工程硕士（软件工程领域）专业 2019 级硕士生姓名：吕军

指导教师（姓名、职称）：陈振宇 教授，房春荣 助理研究员

摘　要

近年来，随着深度学习技术在自动驾驶、图像识别等领域不断产生突破性成果，研究人员逐步加强对深度学习框架的重视，涌现了大量深度学习框架。但是由于深度学习框架不断更新以及运行硬件环境的多样性，使得深度学习算子具有多种实现形式和运行环境，存在运算不稳定现象，并且这种不稳定因素也会存在于使用这些算子搭建的深度学习模型中。因此如何分析深度学习算子稳定性成为目前一个亟需研究解决的问题。

基于上述背景，结合算子测试和差分测试的特征，本文提出基于差分测试的深度学习算子稳定性分析技术，并给出相应系统实现。系统使用导向型模糊测试方案进行开发设计，导向目标为特定差分评估值。系统核心模块由算子与种子管理、种子队列管理、变异策略、差分测试模块组成。算子与种子管理模块负责算子和种子的增删改查功能，并实时同步文件资源和相应数据库记录；种子队列管理模块将种子按优先级排序，并使用模拟退火算法淘汰部分种子，防止种子队列包含大量无效、冗余种子；变异策略模块负责从扰动函数列表选取部分扰动函数，并对选取种子进行扰动变异生成测试用例；差分测试模块基于测试用例进行差分测试，记录执行过程中触发的差分异常，并将差分测试结果反馈给种子队列管理模块。

本文使用本系统分析评估 TensorFlow1.9.0、PyTorch1.3.1、Caffe1.0 框架中常见算子在CPU 和GPU 运行模式下的稳定性。结果表明 conv2d、avg_pooling、sigmoid、tanh、softmax 算子在不同框架、平台运算结果存在差异，出现差分异常现象，不具备良好的稳定性；max_pooling、relu、leaky relu 算子在不同框架、平台运算结果稳定一致，具有良好的稳定性。此外，算子在 GPU 运行模式下的稳定性略优于 CPU。这表明本系统具备深度学习算子稳定性分析能力。

关键词：深度学习算子，差分测试，模糊测试，变异策略，模拟退火算法

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Deep learning framework operator stability test technology based on differential testing

SPECIALIZATION: Software Engineering

POSTGRADUATE: Jun Lv

MENTOR: Professor Zhenyu Chen, Assistant Researcher Chunrong Fang

Abstract

In recent years, as deep learning technology has continuously produced breakthrough results in areas such as autonomous driving and image recognition, researchers have gradually increased their attention to deep learning frameworks, and a large number of deep learning frameworks have emerged. However, due to the continuous update of the deep learning framework and the diversity of operating hardware environments, deep learning operators have multiple implementation forms and operating environments, and there is a phenomenon of instability in operations, and this unstable factor will also exist in the deep learning model built using these operators. Therefore, how to analyze the stability of deep learning operators has become a problem that needs to be studied and solved urgently.

Based on the above background, combined with the characteristics of operator testing and differential testing, this paper proposes a deep learning operator stability analysis technique based on differential testing, and gives the corresponding system implementation. The system uses an oriented fuzzing test plan for development and design, and the oriented target is a specific differential evaluation value. The core module of the system consists of operator and seed management, seed queue management, mutation strategy, and differential testing modules. The operator and seed management module is responsible for the addition, deletion, modification, and searching of operators and seeds, and real-time synchronization of file resources and corresponding database records; the seed queue management module sorts seeds by priority, and uses simulated annealing algorithm to eliminate some seeds to prevent seed queues Contains a large number of invalid and redundant seeds; the mutation strategy module is responsible for selecting some disturbance functions from the disturbance function list, and

generating test cases for the selected seeds; the differential testing module performs differential testing based on the test cases and records the differential exceptions triggered during execution , And feedback the differential testing result to the seed queue management module.

This article uses this system to analyze and evaluate the stability of common operators in the TensorFlow1.9.0, PyTorch1.3.1, and Caffe1.0 frameworks in the CPU and GPU operating modes. The results show that conv2d, avg_pooling, sigmoid, tanh, and softmax operators have different calculation results in different frameworks and platforms, which do not have good stability; max_pooling, relu, leaky relu operators are stable and consistent in different frameworks and platforms, with good stability. In addition, the stability of the operator in the GPU operating mode is slightly better than that of the CPU. This shows that the system has the ability to analyze the stability of deep learning operators.

keywords: Deep Learning Operator, Differential Testing, Fuzzing, Mutation Strategy, Simulated Annealing Algorithm

目　　录

插图清单

附表清单

第一章　引言

1.1　项目背景

八十年代初期，反向传播算法的提出让相应神经网络模型的预测准确率大幅提高，极大激发机器学习的发展。九十年代，支持向量机、Boosting 的陆续提出使浅层深度神经网络模型的预测准确率达到新的高度。2006 年 Geoffrey Hinton 发表的两篇文章 [1, 2] 是深度学习领域具有里程碑意义的事件，使得深度学习无论是在学术界还是企业界都得到大量运用。

在深度学习起步阶段，模型构建者往往要反复编写同一模块，产生大量重复工作。为了高效复用这些模块，研究人员编写相应深度学习框架对常用模块进行统一管理，这些模块经过不断发展，最终以算子的形式存在于深度学习框架中。高效稳定的深度学习框架算子可以缩短模型训练时间并提高预测结果准确性，目前，国外较为主流的深度学习框架有 Tensorflow、Pytorch、Caffe [3]，国内有华为的深度学习框架 MindSpore、百度的 PaddlePaddle、腾讯优图开源深度学习框架 ncnn 以及阿里巴巴深度学习框架 X-Deep Learning [4]。这些成熟的，经过大量实践验证的深度学习框架成为了模型搭建工作的首选。然而在少部分场景下，为满足个性化需求，深度学习工作人员会使用自研深度学习框架。主流深度学习框架算子稳定性在科研、产业界得到反复证实，但是对于部分自研深度学习框架算子稳定性却难以得到保障。深度学习框架算子稳定性是极其重要和更为基础的属性，如果深度学习框架算子稳定性得不到确立，则通过这些算子搭建的深度学习模型的稳定性也难以得到保障。目前关于深度学习算子稳定性相关工作较少，并且由于算子运算的复杂性，通过理论证明算子稳定性的方式几乎不可行。针对这一现象，深度学习研究人员进行大量探索工作，其中基于现有稳定深度学习框架算子的差分测试技术是算子稳定性分析极具潜力的一个发展方向 [5]。

当前，算子稳定性分析的主要难点有：1) 算子输入张量维度复杂，数值区间跨度大，测试用例难以覆盖算子输入空间，并且目前算子测试用例基于规则或语义的变异困难，测试用例的生成缺乏导向性。2）算子稳定性测试尚缺乏

一套成熟稳定的评估方法，目前已有的基于路径覆盖的测试方法并不适用于算子稳定性测试。3）深度学习算子是特定抽象数学定义的具体体现，内部实现有一定的复杂性，很难通过静态分析的方式得到算子的输出结果。4）大部分针对算子的测试工作都是从模型测试入手，这类模型一般是已经训练好的模型，其测试目标仍是以发现模型输出结果是否发生较大误差为主，当发现模型输出结果产生较大误差时，进行反向推导，从而发现相应算子输出是否产生较大误差，这种方式具有较高的复杂度，可行性较低 [6]。5）深度学习算子的实现及运行环境具有多样性，并且不同深度学习框架的部分算子调用接口存在差异，这对算子稳定性分析技术的通用性提出了极大的挑战。

1.2 国内外研究现状

1.2.1 差分测试研究现状

由于深度学习模型、算子的复杂性，传统基于路径覆盖的测试技术难以得到运用。差分测试技术因只需关注待测模块的输入输出格式而无需关注待测目标的内部实现，被广泛运用在深度学习模型、算子测试领域 [7]。

Yuting Chen、Ting Su 教授等人提出覆盖导向差分测试技术发现 Java 虚拟机在不同实现平台执行结果不一致问题。为了扰动生成更多了 Java 类文件进行差分测试，Yuting Chen、Ting Su 教授等人提出了 129 种扰动函数，其中 123 种扰动函数从语法层面对 Java 文件进行扰动变异，剩余 6 种扰动函数基于 Jimple 文件扰动变异生成新 Java 文件。129 种扰动函数中只包含部分高效扰动函数，使用高效扰动函数才能更有效发现 Java 虚拟机执行不一致的问题，基于此 Yuting Chen、Ting Su 教授等人通过 MCMC 采样方法以更高概率接受使用高效扰动函数进行扰动变异。Java 类文件生成后，使用差分测试检测 Java 虚拟机之间执行结果不一致问题。与随机选取扰动函数相比，通过 MCMC 采用的方法将能触发 Java 虚拟机执行差异 Java 文件比例从 1.7％提高到 11.9％ [8]。

Jianmin Guo 等人提出了模糊测试框架 DLFuzz。DLFuzz 抛弃了仅对深度学习模型本身进行测试和手动标记的方法，以最大化神经元覆盖为导向生成测试用例 [9]。DLFuzz 通过对已有测试用例进行微小扰动生成对抗样本，基于对抗样本触发深度学习模型的异常行为。在编写变异函数过程中，DLFuzz 首先定义了变异的幅度，保证原始输入和突变输入的预测结果一致，接着通过微小变异提高测试用例所产生的神经元覆盖率，最后通过差分测试自动识别模

型的异常行为，并根据差分评估结果更新种子集合。为了评估 DLFuzz 的有效性，Jianmin Guo 等人使用深度学习领域主流的两个数据集 MNIST 和 ImageNet 进行实证研究。其中有一个实验采用与 DeepXplore 使用完全相同的数据集。较 DeepXplore 而言，DLFuzz 在不采用额外工作的前提下得到了十分相近的实验效果，并且，在扰动减小 79.56%-96.77% 的情况下，产生的对抗输入增加 35%-58.46%，神经元覆盖率提高 1.10%-5.59%。在时间消耗方面，平均节省了 20.11% 的时间消耗。

1.2.2 深度学习算子测试研究现状

深度学习框架的提出极大提高模型搭建人员的工作效率，被广泛采用在各个领域。算子作为深度学习框架的基本组成单元，承担者核心计算功能，是深度学习框架不可或缺的一部分 [10]。算子的稳定性直接影响了深度学习框架的稳定性。由于算子运算的复杂性，传统基于路径覆盖的测试评估技术难以运用到算子测试中，因此，研究人员试图通过多种测试的方法分析算子稳定性。

Pengfei Wang 等人结合深度学习算子测试和模糊测试的特点提出了导向型模糊测试方案。导向型模糊测试技术由常规模糊测试技术演化而来，工作流程与传统模糊测试类似：通过初始种子生成种子队列；根据种子优先级从种子队列选取种子，并输入到变异策略模块；变异策略模块采用确定性策略或随机性策略选取扰动函数，基于选取的扰动函数对种子进行变异生成测试用例；最后通过评估测试用例运行结果与导向目标之间的偏离值决定是否将新生成的测试用例更新至种子队列。导向型模糊测试核心技术包括：导向目标、输入优化、种子优先级管理、能量函数、变异策略、数据流分析。在进行深度学习算子测试任务之前，导向型模糊测试会设定导向目标，导向目标的评估方式一般为距离、概率等其他可度量的属性。其中导向目标选取后导向型模糊测试会分为两个阶段执行，前一个阶段让模糊测试产生更多的测试用例，后一个阶段使生成的种子更接近所设定的目标 [11]。导向型模糊测试方案化解了深度学习算子难以进行路径覆盖测试的难题，丰富深度学习算子测试的途径并极大提高深度学习算子测试效率。

1.2.3 变异策略研究现状

变异策略决定了测试用例的生成效率和质量，如果采用变异策略制定的不合理，则变异幅度可能过大或者反向变异。合理高效的变异策略是测试领域的测试领域的研究热点。

为解决求解复杂优化问题时差分优化算法参数设置繁琐、收敛效果不佳的问题，Bi Xiaojun 等人提出了一种基于新变异策略的动态自适应差分进化算法(p-ADE) [12] 。新变异策略基于目标个体的历史最优解和种群全局最优解确立变异方向，从而提高变异的针对性。同时为了提高变异策略的稳定性以及加快变异策略的收敛速度，新变异策略提出了一种动态自适应参数调整方法，动态调节个体在变异过程中的变异幅度，以达到快速稳定收敛的效果。最终实验结果表明 p-ADE 在鲁棒性、收敛速度和精度较多种主流 DE 优化策略均有明显优势。部分并行模糊测试方法由于测试用例生成缺少统一有效管理，导致测试用例生成缺乏导向性并生成大量冗余测试用例。Zou Yanyan 等人为解决这一问题，提出了一种以变异策略为基本粒度进行并行模糊测试的方案，该方案通过周期性同步不同并行模糊测试中测试用例并动态调整变异策略的实施比例的方式，极大减少测试用例冗余并提高综合覆盖率 [13]。

1.3 本文主要工作

本文首先介绍了深度学习算子稳定性分析技术的背景、研究现状、目标和意义，并描述深度学习算子稳定性分析的特点，以及基于差分测试进行深度学习算子稳定性分析的技术可行性。然后本技术进行了功能型需求分析和非功能型需求分析，并绘制相关用例图。随后结合需求分析，详细介绍了系统各个模块的详细设计以及实现，并提供相关架构图和关键代码实现。最后，基于本系统进行了多组实验分析，并统计不同算子对应差分异常测试用例数及生成趋势，进而验证本技术的有效性。基于差分测试的深度学习算子稳定性分析技术主要特点有：

1）使用导向型模糊测试方案，解决算子测试难以进行路径覆盖的问题，并将差分评估值作为导向型模糊测试的导性目标，解决导向型模糊测试中导向目标选取问题。

2）使用能量函数随机淘汰部分最优种子，增加其他种子被选取的可能性，从而防止种子选取出现局部最优现象。

3）技术实现具有良好拓展性，用户可便捷安装自研深度学习框架，以及通过可视化方式管理算子文件；算子支持热加载，极大提高算子测试效率。

4）系统采用模块化设计，并进行了模块化开发，使得系统可以便捷部署到不同的物理主机上，极大提供系统的测试任务负载能力。

1.4 本文组织结构

本文共分为六个章节，组织结构如下。

第一章为引言，介绍目前国内外深度学习框架和算子稳定性测试发展现状，分析算子特点、重要性，以及算子测试的难点。结合上述情况，本文提出了基于差分测试的深度学习算子稳定性分析技术，并介绍了国内外发展现状。

第二章为相关技术概述，介绍了本技术和实现所采用的各项工具和技术，主要包括深度学习框架、深度学习算子库、模糊测试、Django 以及 Docker。

第三章为需求分析与概要设计，本章节先对系工具进行了整体概述，简要叙述了工具的功能性需求和非功能需求，接着从用例、系统架构、逻辑视图、开发视图、进程视图、物理视图多个不同视角，对技术进行了细分和描述，并绘制相关图表。章节最后介绍工具所涉及的关键数据库实体对象。

第四章为详细设计与实现，主要包含了本工具各个功能模块的详细设计、流程示意图、关键代码、算法以及相应公式。

第五章为测试与有效性分析，描述测试目的与测试环境，并分别进行功能性和非功能性测试，最后对本系统的有效性展开分析验证。

第六章为总结与展望，简要汇总本技术的主要思路和核心模块、技术实现效果以及相应实验结论，并指出目前所发现工具的不足之处，展望未来改进方向。

第二章　相关技术概述

2.1　深度学习框架

在过去的一段时间，深度学习领域出现了大量的算法和实用程序，其中部分算法和程序在自动驾驶、图像识别、医疗诊断领域得到突破性运用，深度学习框架在里面扮演了越来越重要的作用 [14]。深度学习框架是模型搭建的效率工具，尤其是 TensorFlow、PyTorch 的提出，极大的提高模型搭建人员的工作效率，使得深度学习相关工作人员能够在各个领域高效编写深度学习模型。

深度学习框架的发展主要经历以下几个阶段。21 世纪初，神经网络的概念已经出现，并且也出现了部分工具用来描述和搭建神经网络模型，例如 Torch、OpenNN，但是这些工具提出的目的并不是特意用来搭建神经网络模型的，并且这些工具大量 api 并不能直接用来搭建神经网络模型，同时也缺乏 GPU 的支持，使用者需要做出大量额外的工作。2012 年多伦多大学 Alex Krizhevsky 等人提出一种全新的深度神经网络架构 AlexNet，AlexNet 在 ImageNet 数据集上精度达到 SOTA 标准 [15]。这一突破性进展激起了深度神经网络的热潮，在这个时期 Caffe、Theano 应运而生，使得 CNN、RNN、LSTM 模型的搭建工作十分方便，同时，这些模型支持多 GPU 训练，极大减少模型的训练时间。2015-2016 年，大型科技公司的加入是这一时期的重要特点，这一时期谷歌也开源了 TensorFlow 框架，并成为深度学习领域最流行的深度学习框架之一；FackBook 发布 PyTorch 框架，使用的是 Python API；微软研究院研发了 CNTK 框架。何凯明等人提出了 ResNet 模型在 ImageNet 数据集上的准确率创历史新高 [16]。2019-2020 年，深度学习框架经过不断发展，最终划分为两大板块，一个是 TensorFlow，另一个是 PyTorch，占据了深度学习框架领域 95% 的的份额 [17, 18]。接下来就深度学习框架 TensorFlow、PyTorch 展开叙述。

2.1.1　TensorFlow

TensorFlow 是由谷歌人工智能团队开发和维护基于数据流的深度学习框架，具有多层级结构，可以部署在服务器甚至普通个人电脑上，同时

也支持 CUDA 高性能数值运算，被大量运用在人工智能科学研究领域。TensorFlow 提供多种编程语言的 API 接口，常见的有：C、C++、Java、Go、Python。TensorFlow 很重要的一个概念是张量 (Tensor)，张量是 TensorFlow 和基本数据处理单位，内部构成为一个多维数组 [19]。TensorFlow 的主要具有以下几个特征：

（1）灵活性：TensorFlow 计算模型采用数据流图形式，节点代表算子运算，有向边代表节点之间关系，用户可自定义数据流图，可以直观的绘制、搭建神经网络模型，张量计算过程就是在数据流图中流动的过程。 [20] 此外用户还可以编写自定义库。

（2）可移植性：TensorFlow 框架中 API 高度统一，搭建好的模型可以同时在服务器、普通 PC、移动设备、Docker 以及云端设备运行，不需要或者仅需改动极少量代码就可以实现迁移部署。此外，TensorFlow 可以指定张量运算环境 (GPU 或 CPU)。

（3）自动求微分：TensorFlow 提供自动微分 API(tf.GradientTape)，可以根据函数的输入变量计算对应的导数 [21]。tf.GradientTape 会把所有操作记录在磁带上，然后基于磁带和每次操作对应的导数，结合反向微分法计算磁带中函数的导数。

（4）多语言支持：TensorFlow 官方目前对 Python、JavaScript、C++、Java 都有良好的支持。

（5）完善的文档：TensorFlow 官网提供了完善的文档介绍和 API 使用方法，高水平的研发团队，活跃的 TensorFlow 社区使 TensorFLow 成为目前最活跃的深度学习框架。

2.1.2 PyTorch

PyTorch 是基于 Torch 的开源深度学习库，底层实现采用 C++ 编写，支持 GPU 加速运算，目前由 Facebook 人工智能团队开发维护 [22]。PyTorch 支持动态计算图，计算图在运算过程中支持动态改变，这点与 TensorFlow 的静态计算图有所不同。PyTorch 的主要特点有：

（1）可基于 Numpy 初始化张量进行计算，并且具有 GPU 加速特性。

（2）能构建具有自动微分功能的深度神经网络 [23]。

（3）PyTorch 设计遵循最小封装原则，层次清晰，一般不存在跨层调用，例如 tensor 需要转化成 variable 才能被 module 使用。

（4）得益于 PyTorch 先进的设计理念和精简的功能，在许多训练场景中 Pytorch 的性能比 TensorFlow 更好。

（5）Pytorch 接口的实现具有一致性，不会因版本的变化而发生大的改变，并且接口命名方式易于理解，非常容易上手。

（6）PyTorch 在提供内容齐全的官方文档和指南同时还维护一个活跃的社区。

2.2 深度学习算子库

深度神经网络一般由输入层、隐藏层、输出层组成，常见的神经网络类型有卷积神经网络、循环神经网络、生成对抗网络、深度强化学习。深度神经网络模型通过大量训练学习被训练数据的特征，训练后模型可用于预测类似数据的行为或输出。在深度神经网络训练的过程中，深度神经网络算子库承担基础运算操作，算子内部逻辑不会因训练过程的进行而发生改变，算子在深度神经网络中具有原子性，作为深度神经网络中相对独立的单位，其作用不可或缺。在深度神经网络中，卷积神经网络由于简单高效，在自动驾驶、医疗、图像识别领域得到广泛运用 [24, 25]。CNN 主要有卷积算子 (conv1d、conv2d、conv3d 分别代表了一维卷积算子、二维卷积算子、三维卷积算子 [26]), 池化算子 (avg_pool、max_pool), 激活函数 (relu、tanh), 分类算子 (softmax) 组成 [27, 28]。卷积算子中，一维卷积运用在自然语言处理领域和序列模型中；二维卷积运用在图像识别、处理领域；三维卷积目前运用较少，可以用来处理相对复杂多维空间问题，例如医疗和视频处理等较为复杂的多维空间领域 [29, 30]。卷积算子非常适合用于描述和计算矩阵之间的线性变换，与传统矩阵乘法算法相比，需要的浮点运算较少。深度学习模型中包含大量的卷积浮点运算运算，卷积算子的准确性对于提高深度模型预测结果的可信度至关重要。

深度神经网络中池化算子是一类常用的算子，主要功能是进行样本特征压缩。深度学习模型池化算子主要包括 max_pooing 和 avg_pooling [31]，max_pooing 取给定池化核内的最大值，并将该值赋予池化核内的所有元素，通过保留主要特征的方式提高模型的泛化能力；avg_pooling 取给定池化核内元素的平均值，并将该值赋予池化核内的所有元素，从而提升模型的泛化能力。

相比包含大量乘法运算的卷积算子，深度学习框架激活函数的计算量相对少很多，其运算稳定性甚至可以通过形式化的方式验证，常见的激活函数

有：ReLU、Sigmoid、tanh。其中 Sigmoid 也被称作 S 型生长曲线，输出范围为 (0,1)，ReLU 激活函数是目前常用的激活函数，可以有效抑制深度学习模型训练梯度爆炸问题，

深度学习框架包含搭建深度神经网络模型必要的算子，同时，每种深度学习框架对算子的实现以及组织方式存在差异，例如 TensorFlow 和 PyTorch 中卷积算子输入张量的通道顺序不相一致，TensorFlow 为 NHWC，PyTorch 为 NCHW，其中 N 代表 batch, H 代表 height, W 代表 width, C 代表 channels。同一类型算子在不同深度学习框架中不仅输入接口存在不一致，输出格式、内部实现、组织方式也存在差异。当张量输入维度较大时，算子内部会进行大量浮点运算，由于计算机中浮点数本身是一种近似表示和浮点数溢出问题，算子内部浮点数运算有出错的的可能性。此外不同的硬件平台 (CPU、GPU) 浮点数表示、处理方式也存在差异 [32]。深度神经网络算子库的差异和多样性是是算子测试的核心难点之一。同时由于算子的这些特点和算子运算的复杂性，现有的深度神经网络算子测试方法大部分采用差分测试的方法，并在算子的输入输出接口增加相关的业务层，以发现算子不同实现方式的差异。

2.3 差分测试

差分测试是将相同测试用例输入到两个功能相同实现形式不全一致的程序、函数中，然后对输出进行差分分析，分析本来应该一致的输出是否存在偏差，进而判断程序、函数是否存在输出异常行为的一种软件测试技术 [33]。通过差分测试只能判断两个待测程序、函数存在异常，并不能直接判定具体是哪些待测程序、函数运算结果偏离真实值。

差分测试 (Differential testing) 因其简单易于实施，应用领域也越来越广泛 [34]。差分测试常见实施方法为：根据测试目标选取两个输入相似、逻辑规格一致、输出应一致的程序或者函数；将相同输入数据分别输入到两个不同的程序或函数，观察最终结果或者中间关键过程是否存在差异，并进行记录分析 [35]。进行差分测试之前首先需要提供待测对象的输入输出接口，然后将输入数据输入到两个待测对象，得到两个对应输出，对两个输出进行差分，根据差分结果判断待测对象是否存在差分异常，并进行相应的归档更新操作。差分测试由于简单易行，发现漏洞能力强，在学术界和工业界得到大量运用，尤其是在工业界，工业界往往注重其工业成本，不少工业单位都建立了自研差分测

试平台。此外，由于深度学习算子内部实现的复杂性，越来越多的深度学习算子检测方法采用差分测试，差分测试方法深度学习算子测试领域也逐渐成熟。

差分测试的优点是简单高效，其缺点也隐含在其简单高效的优点当中：结果导向，往往只能捕捉到结果异常，难以捕捉到产生异常的细节；缺少过程管理，同一异常被捕捉到多次，异常类型分布不均；差分异常标准难以判定，大部分根据实践经验获得，缺少理论上的说服力 [36]。简单高效是高质量测试方法的特征，在追求简单高效的过程中也会产生一些新的问题。面对日新月异的新技术，如何整合各种技术方法的优势，提高测试效率的同时解决新引入的问题，是当前差分测试发展的一个难点，同时也是实现基于差分测试的深度学习算子稳定性分析技术的关键。

2.4 Django

Django 是 Adrian Holovaty 和 Simon Willison 基于 Python 编写的开源项目。由于 Django 采用 MVC 思想设计，使得编程人员使用少量的代码就可以编写项目，具有低耦合、敏捷开发、高可用、成本低、便捷部署的特点。Django 开发初期主要用于管理劳伦斯出版集团旗下旗下的新闻类网站，主要目的是简化数据库驱动的网站开发 [37]。Django 设计遵循敏捷开发思想及”Don’t Repeat Yourself” 法则，其组件具有可重用性和可插拔性。

Django 采用的 MTV 模式和 MVC 的思路是一致的，核心思想是使模块之间松耦合。其中 M 表示模型 (Model)，主要负责业务对象与数据库之间的映射关系；T 表示模板 (Template)，负责渲染前端 html 界面；V 表示视图 (View)，主要负责业务逻辑，充当 Model 和 Template 之间的媒介 [38]。此外 Django 还具有 URL 分发器，将页面请求转发至对应的视图，视图之后调用、组织对应的模型和模板。

Django 对主流关系型数据库有良好的支持，例如 PostgreSQL、MySQL、SQLite、Oracle，并且抽象各个数据库的调用接口，提供统一的调用 API。Django 内部集成对象关系模型 (ORM)，可以对面向对象实例和数据库数据条目进行映射和操作，ORM 在这之间充当桥梁的作用，用户在使用过程中可以通过操作类的方式对相应数据进行增删改查操作 [39]。

2.5 Docker

Docker 是用于开发、交付、运行应用的开放平台，它允许将应用从基础设施中提取出来，拆分成多个容器，进而简化软件部署的流程。与虚拟机不同，虚拟机是计算机硬件的虚拟化，而容器是操作系统层面的虚拟化，正是由于容器的标准化，使得 Docker 可以便捷地部署到不同的基础设施上 [40]。

Docker 通过 Linux 的命名空间以及 Linux 核心中的的资源分离机制创建容器，使得在单个 Linux 实体中可以创建多个容器，这也是多个 Docker 运行在 Linux 系统中的原因，虽然目前有 Windows 平台下的 Docker 软件，但其实是在 Windows 系统虚拟出的 Linux 内核中运行 [41]。Linux 的命名空间机制完全隔离了运行在不同 Docker 镜像中程序的系统环境，cgroup 隔离了系统资源。Docker 的三个基本概念分别是镜像、容器、仓库，Docker 镜像包含独立可运行的系统，通常基于 Linux 系统构建；Docker 容器基于 Docker 镜像创建，是由镜像实例化出来的实例，镜像是静态的，容器是动态的；仓库主要用来保存、管理镜像。

Docker 的主要应用场景有：应用服务器镜像封装及容器式部署；持续集成、发布自动化测试工具；后台服务多机部署，例如将数据库部署到其他主机上；运用在云计算领域，搭建平台即服务平台 PaaS(平台) [42]。通过 Docker 可以快速开发、集成、测试和部署应用程序，可以让程序无须关注具体的生产环境。Docker 主要的优点有：1）适用于敏捷开发，程序运行在标准化环境中，简化了软件开发的生命周期，非常适合持续集成和持续交互的工作流程。2）高度可移植性和可扩展性，Docker 可以在服务器、普通 PC、云服务或者混合环境中运行，并可以随时随地扩展或者拆除服务。3）充分利用计算资源，同一主机可以运行多个不同的容器，并且他们之间互不干扰，此外，还可以联合不同主机共同解决同一业务目标。

2.6 本章小结

本章主要概述了技术实现所有到的方法和工具，并对技术实现所需用到的核心方法和工具进行了重点阐述。首先讲述了深度学习框架的作用及不同深度学习框架之间的异同点。接着介绍目前常用的深度学习算子库，介绍每一类算子的主要作用及在不同深度学习框架中的差异。然后描述模糊测试的特点及主

要工作流程，并引出本文使用的导向型模糊测试方法。之后讲述差分测试的特点，以及深度学习算子测试为什么使用差分测试。最后分别讲述了技术实现所用到的 Django、Docker 工具，分析每个工具的作用场景，功能特点，以及核心工作原理。

第三章　需求分析与概要设计

3.1　系统整体概述

随着深度学习技术的快速发展，越来越多的行业和领域开始运用深度学习技术来解决传统方法难以解决的难题，例如医疗领域使用深度学习技术进行医疗诊断。随着深度学习技术的需求越来越广泛，人们对深度学习模型搭建的框架要求也越来越细致。常用的深度学习框架有 TensorFlow、Pytorch、PaddlePaddle，这些常用深度学习框架有时难以满足定制化的要求，不少科研机构、企业单位开始自研深度学习框架，并有部分投入使用后较长时期内没有发生质量问题。由于深度学习框架的多样性和算子运算的复杂性，如何分析自研深度学习算子稳定性成为行业内的重要课题。

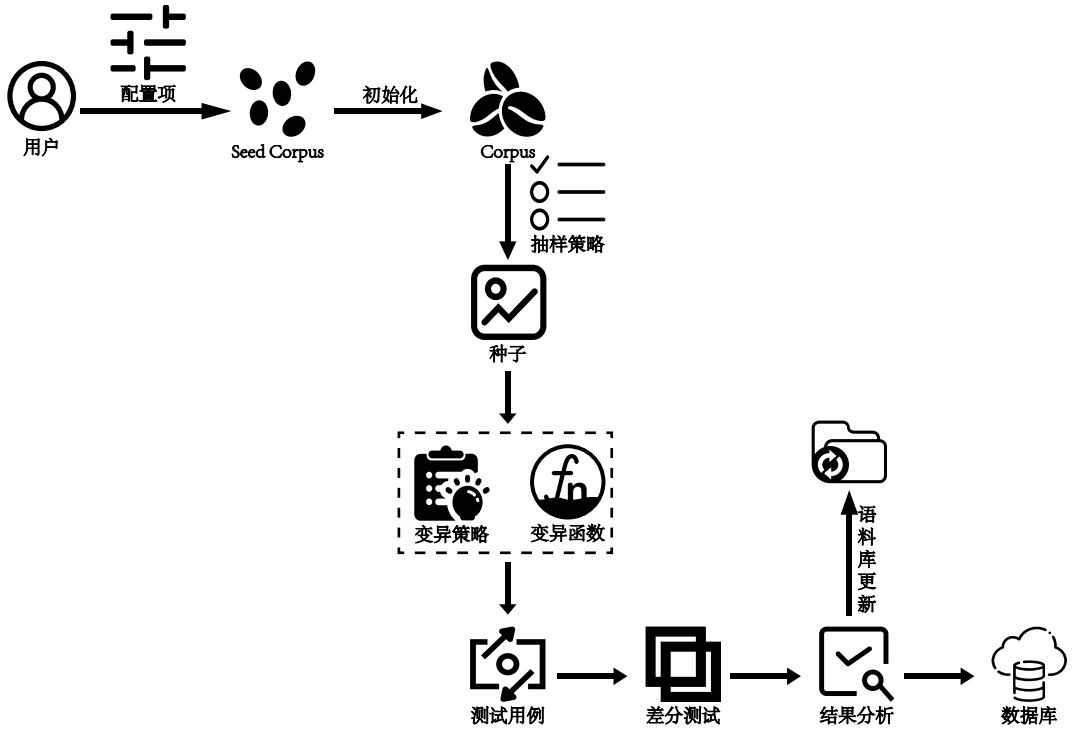

图 3-1: 深度学习算子稳定性分析技术工作流程图

为了解决这一课题，本文采用基于差分测试的深度学习算子稳定性分析技

术检测深度学习算子稳定性。图 3-1展示了该技术的工作流程。该技术技术实现可以根据用户配置自动化运行，用户可选配置项有：待测深度学习算子、待测深度学习框架及对应的运行模式 (GPU 或 CPU 运行模式)、模糊测试终止条件、种子集合。获取用户配置项之后系统就可以进行相应的初始化操作，并开始测试流程，本技术使用导向型模糊测试方法，导向目标为测试用例对应的差分值。系统在运行阶段会生成和记录一系列中间数据，主要包括差分测试结果详细信息，用户的操作记录以及服务器在运行过程中的状态信息：CPU 利用率、内存使用量、磁盘 IO 读写。

3.2 系统需求分析

3.2.1 系统涉众分析

系统所涉及的涉众如表 3-1所示，涉众表只包含设众需求方。测试需求方可以通过本系统提交算子测试任务，查看测试任务进度，暂停、停止正在进行的测试任务，下载已完成测试任务的报表，对测试任务进行分析等。深度学习模型开发人员可借助该系统实时、动态验证验证所研发深度学习框架的准确性，并对该系统进行定制化开发，以使测试目标更具针对性，测试结果更加准确。

表 3-1: 系统涉众分析结果

涉众名称	涉众特征与期望
测试需求方	作为测试需求方，用户期望可以本系统上传深度学习框架，自定义添加、修改算子，上传测试初始数据集以及在平台上登录注册账号。框架、算子、数据资源准备完整后，用户可以配置相应框架中的算子，并基于特定的数据集进行差分测试，测试详情会在数据库中持久化存储，测试任务完成后可通过系统接口统计触发差分异常测试用例总数及生成趋势，进而分析评估深度学习算子的稳定性。

3.2.2 功能性需求分析

根据系统的涉众分析，可进一步对系统进行功能性需求分析，该技术实现的的主要功能性需求如表 3-2所示。在进行算子测试任务之前，用户可在系统平台上添加常用的深度学习框架比如：TensorFlow、PyTorch 或者上传自研深度学习框架。上传深度学习框架之后，用户可基于已上传的深度学习框架编写算子调用的接口，在深度学习框架中已经实现大量常用的算子调用接口，此外，算子调用接口支持热添加、修改。系统自带相应的初始测试数据集，用户也可以在技术实现平台上新建数据集，并在数据集中进行更新操作。深度学习框架框架、算子调用接口、数据集准备好之后，用户就能进行算子测试任务。用户启动算子任务时需要配置相关运行参数，运行参数主要包括待测深度学习框架、待测算子、所用初始测试数据集。系统会根据用户配置的参数自动执行算子稳定性测试。测试任务队列通过线程池管理，用户可以随时暂停正在执行的任务，并持久化保存当时测试任务运行快照，之后，用户可基于此快照继续执行相对应的算子测试任务。在部分情形下，用户希望提前结束算子测试任务，系统集成了强制结束正在执行中任务的功能，并且任务执行过程中的部分结果也是具有指导意义的，也可以用于对算子进行稳定性评估，于是在强制结束任务的时候，任务执行的部分结果会被保留，并且会基于这部分数据生成对应的测试结果报表。算子测试任务完成之后，所有的中间结果会被持久化存储在数据库中，并且生成对应的算子测试结果报表，用户可自行下载测试结果报表。

表 3-2: 系统功能需求列表

需求 ID	需求名称	需求描述
R1	上传深度学习框架	用户可以通过 pip 命令安装第三方深度学习框架，或者将自研发深度学习框架通过可视化的方式添加至 python 对应虚拟环境 site-packges 中。
R2	热更新算子接口	用户可以在不重启服务器的情况下上传基于平台已有框架的算子调用接口。
R3	新建修改测试数据集	用户可新建数据集文件夹，并在数据集文件夹中添加、删除、查看操作。
R4	启动任务	用户上传算子接口、初始数据集后可自定义配置，并启动算子测试流程。

R5	查看任务状态	在任务列表页面用户可以实时查看任务状态，任务状态包括：运行中、挂起、强制结束、已完成。
R6	挂起任务	对于正在运行中的测试任务，用户可以随时对任务进行挂起操作，任务挂起后，挂起任务的运行环境快照会持久化保存。
R7	恢复任务	用户可基于挂起任务快照随时恢复挂起中的任务，并且继续执行的任务可以再次挂起。
R8	停止任务	对于正在执行的任务，用户可以随时终止，任务终止后，系统会根据阶段性测试结果生成报表。
R9	查看任务结果报表	测试任务完成或被强制结束后，用户可以查看测试详情及相应结果报表。

3.2.3 非功能性需求分析

系统需要为用户提供深度学习框架算子稳定性测试服务，在满足功能性需求的基础上对非功能性需求也有一定的要求。常见的非功能性需求有可用性、稳定性、可拓展性、安全性、性能、兼容性、伸缩性。结合应用场景和系统特性，系统的非功能性需求如表 3-3所示。服务器在运行过程中应确保高可用性，对常见的异常应有应急方案，尽量确保系统在每个时间段都可用。系统服务端和数据库部署在不同主机上，在缓解同一主机处理压力的同时，提高了整个系统的可拓展性。系统有严格的用户权限管理功能，对数据库请求接口都做了认证处理。在考虑到技术实现性能问题时，系统对测试任务请求做了异步处理，并且当服务器运算资源不足时会将任务放至等待队列，减少任务切换的性能损耗。由于 Docker 本身的作用是虚拟出操作系统，固系统具有良好的可拓展性性。服务器与数据库分离的设计可以便捷地进行再开发。同时由于目前技术水平的限制，本系统部署在 Docker 中无法提供算子在 GPU 环境下的稳定性分析服务，本系统提供的非 Docker 安装包可以在装有 NVIDIA 显卡的主机安装运行，并提供算子在 GPU 环境下的稳定性分析服务，并且为了提高易用性，本系统编写了非 Docker 安装包的一键安装脚本，

表 3-3: 系统非功能需求列表

需求名称	需求描述
可用性	系统应当具备 99.5% 的可用性，若因非人为因素例如断电、磁盘损坏导致技术实现平台出现故障不能正常运行，系统应该在故障修复完成技术实现启动后 10 分钟内继续执行在故障发生时的任务，生成故障日志并继续提供稳定可靠服务。
稳定性	系统提供稳定可靠的深度学习框架算子测试任务，技术实现部署在 Docker 容器中，Docker 运行在搭建 Ubuntu20.04 的服务主机上。由于物理主机、Ubuntu 系统、Docker 虚拟环境存在不稳定的因素，主机会出现断电或者物理损伤的情况，Ubuntu 系统不断更新可能导致不兼容的情况发生，技术实现应对常见异常情形编写对应的处理方案，使技术实现再遇到异常情形时能采取相应的措施，最大程度保证技术实现的稳定性。
可拓展性	技术实现应采用模块化开发思想，功能高度内聚，模块对外应接口风格应统一。模块不仅本身具有可扩展性，而且内部应该清除具体业务逻辑，使其能够与其它模块进行耦合。
安全性	系统应该具备完备严格的身份识别功能，确保系统相关数据安全可靠，系统的相关接口应该具备严格的鉴权功能，使其能够防范非法使用。
性能	在正常运行环境下，系统数据库接口读写请求响应时延应小于 200ms，技术实现前端请求响应时延应小于 300ms。
兼容性	系统可以兼容 Docker18.09.7 以后的版本。
伸缩性	系统应具备一定的负载均衡能力，能将测试任务动态分配到不同主机进行测试，并且主机发生崩溃时有相应的任务迁移方案。

3.3 系统用例分析

3.3.1 系统用例图

基于设涉众分析、功能性需求分析和非功能性需求分析得到如图 3-2所示的系统用例图，系统目前涉众只有算子稳定性测试需求方，共有八个核心用例：上传深度学习框架、热更新算子接口、新建修改测试数据集、启动任务、查看任务状态、挂起任务、恢复任务、停止任务。

3.3.2 系统用例描述

上传深度学习框架为系统基础性功能，其他需求都依托这一需求的实现而进行。测试需求方可以通过 pip 或 conda 安装已发布的深度学习框架或者通过可视化方式手动上传自研深度学习框架。上传深度学习框架详细用例描述如表 3-4所示。

表 3-4: 上传深度学习框架用例描述

ID	UC01	名称	上传深度学习框架
描述	用户上传自研或已发布深度学习框架		
参与者	系统用户		
触发条件	用户在模型上传界面点击模型上传功能按钮		
前置条件	用户登录系统，并进入模型上传界面		
后置条件	系统反馈模型上传的结果状态		
优先级	高		
正常流程	1. 用户登录系统 2. 用户进入模型上传界面 3. 用户选择模型上传方式 4. 若用户选择上传自研深度学习模型则需上传模型包，若用户安装已发布深度学习模型只需输入模型名称及对应的版本号 5. 用户点击提交按钮，提交模型上传请求		

扩展流程	1a. 用户输入错误的账号密码组合，系统重定向账号注册或找回密码界面 4a. 用户指定已发布深度学习框架版本不存在，系统返回可用的版本号 5a. 用户取消上传，则自动返回系统首页
特殊需求	若用户选定版本的已发布深度学习框架已安装则直接返回安装成功结果

在实际使用场景中，添加、修改、删除算子是经常性操作，若每次都算子进行修改都需要重启服务则测试效率会大幅度降低，因此系统支持算子热更新功能，用户可以在不重启系统服务的情况下更新算子调用接口。其详细用例描述如表 3-5所示。

表 3-5: 热更新算子接口

ID	UC02	名称	热更新算子接口
描述	用户可以在不重启服务的情况下添加、删除、修改算子接口		
参与者	系统用户		
触发条件	用户在算子管理界面		
前置条件	用户登录系统，并进入算子管理界面		
后置条件	系统反馈算子接口更新结果状态		
优先级	高		
正常流程	1. 用户登录系统 2. 用户进入算子管理界面 3. 用户选择算子更新方式 4. 若用户选择新增算子接口则用户需上传算子接口文件；若用户需修改算子则打开对应算子文件进行修改后再次上传；若用户选择删除算子接口，通过下拉列表或直接输入名称删除即可 5. 用户点击算子更新按钮，提交算子更新请求		

扩展流程	1a. 用户输入错误的账号密码组合，系统重定向账号注册或找回密码界面 4a. 若存在算子接口重名情况，则展示对应文件对比结果，让用户自行选择所需要保留的文件 5a. 用户取消算子更新操作，则自动返回系统首页
特殊需求	同一算子接口在同一时间内只允许被一个用户更改，若用户尝试更新正在修改中的文件则给用户进行相应提示，并拒绝修改请求

系统拥有测试数据集管理功能，同一集合文件在同一文件夹中。用户可以上传、查看及删除测试数据集，并且为了存储不同内容的同名文件，系统会被文件进行重命名并持久化存储映射关系。其详细用例描述如表 3-6所示。

表 3-6: 新建修改测试数据集用例描述

ID	UC03	名称	新建修改测试数据集
描述	用户可以在不重启服务的情况下添加、删除、修改测试数据集		
参与者	系统用户		
触发条件	用户在算子管理界面		
前置条件	用户登录系统，并进入算子管理界面		
后置条件	系统反馈算子接口更新结果状态		
优先级	高		
正常流程	1. 用户登录系统 2. 用户进入算子管理界面 3. 用户选择算子更新方式 4. 若用户选择新增算子接口则用户需上传算子接口文件；若用户需修改算子则打开对应算子文件进行修改后再次上传；若用户选择删除算子接口，通过下拉列表或直接输入名称删除即可 5. 用户点击算子更新按钮，提交算子更新请求		

扩展流程	1a. 用户输入错误的账号密码组合，系统重定向账号注册或找回密码界面 4a. 若存在算子接口重名情况，则展示对应文件对比结果，让用户自行选择所需要保留的文件 5a. 用户取消算子更新操作，则自动返回系统首页
特殊需求	同一算子接口在同一时间内只允许被一个用户更改，若用户尝试更新正在修改中的文件则给用户进行相应提示，并拒绝修改请求

当深度学习框架、算子、测试数据集上传完成之后用户就可以启动测试任务，进行测试任务之前需要对任务参数进行配置，主要的启动任务参数有待测深度学习框架及框架所运行的硬件环境、待测深度学习算子、初始测试数据集。基于这些主要配置参数系统就可以开始测试任务。其详细用例描述如表 3-7所示。

表 3-7: 启动任务用例描述

ID	UC04	名称	任务启动
描述	用户配置相关配置项启动算子测试任务		
参与者	系统用户		
触发条件	用户在新建测试任务界面		
前置条件	用户登录系统，并进入新建测试任务界面		
后置条件	系统反馈任务当前所处的状态		
优先级	高		
正常流程	1. 用户登录系统 2. 用户进入新建测试任务界面 3. 用户配置算子测试配置项 4. 用户点击任务提交按钮，提交新建测试任务请求		
扩展流程	1a. 用户输入错误的账号密码组合，系统重定向账号注册或找回密码界面 3a. 系统可以根据用户配置项生成任务详单，包括任务预计运行、结束时间 5a. 用户取消提交任务操作，则自动返回系统首页		

特殊需求	当用户配置参数不合理是，系统应该给予提示，如待测算子部分存在于待测深度学习框架

系统在任务提交后会处理一段时间，在这期间用户需要能够随时了解到系统目前所处的状态，常见的任务状态有运行中、挂起、强制结束、已完成，并且系统需要给出任务预计完成时间，其详细用例描述如表 3-8所示。

表 3-8: 查看任务状态用例描述

ID	UC05	名称	查看任务状态
描述	用户查看已提交任务的任务状态		
参与者	系统用户		
触发条件	用户在任务管理列表界面		
前置条件	用户登录系统，并进入任务管理列表界面		
后置条件	系统分页展现任务列表状态		
优先级	高		
正常流程	1. 用户登录系统 2. 用户进入任务列表管理界面 3. 用户分页查询或者关键字查询相应任务 4. 用户点击查询按钮，提交查询测试任务请求		
扩展流程	1a. 用户输入错误的账号密码组合，系统重定向账号注册或找回密码界面 3a. 当用户提交非法关键字时，系统应予以相应提示 5a. 用户取消查询任务操作，则中断查询过程		
特殊需求	当用户配置参数不合理是，系统应该给予提示，如待测算子部分存在于待测深度学习框架		

当系统被多用户使用时，可能会出现资源不足的场景，这是可以挂起优先级不高任务以腾出资源让新的测试任务运行。挂起任务的运行环境会以快照形式保存，并且可基于快照重新恢复运行。其详细用例描述如表 3-9所示。

表 3-9: 挂起任务用例描述

ID	UC06	名称	挂起任务

描述	用户挂起正在运行中的任务
参与者	系统用户
触发条件	用户在任务管理列表界面
前置条件	用户登录系统，并进入任务管理列表界面
后置条件	系统返回被挂起任务的状态
优先级	高
正常流程	1. 用户登录系统 2. 用户进入任务列表管理界面 3. 用户选中需要挂起的任务 4. 用户在选中任务操作栏点击挂起按钮，提交挂起任务请求
扩展流程	1a. 用户输入错误的账号密码组合，系统重定向账号注册或找回密码界面 3a. 用户可以延时启动挂起操作，计时时间结束后系统自动执行挂起操作
特殊需求	当被挂起任务有未处理完的必要流程时，系统应予以提示，确保系统处于稳定的状态

当系统资源较为充裕时，用户可以选择恢复挂起的任务。恢复挂起任务时首先需要找到挂起任务快照，找到快照后系统基于快照即可恢复任务挂起时对应的系统运行环境，接着被挂起任务就可以基于恢复后的环境继续运行，当任务继续运行时系统应重新计算任务完成预计所需要花费的时间，并将预计完成时间提示给用户。其详细用例描述如表 3-10所示。

表 3-10: 恢复任务用例描述

ID	UC07	名称	恢复任务
描述	用户恢复处于挂起状态的任务		
参与者	系统用户		
触发条件	用户在任务管理列表界面		
前置条件	用户登录系统，并进入任务管理列表界面		
后置条件	系统继续执行被挂起的任务，并返回被任务的执行状态		
优先级	高		

正常流程	1. 用户登录系统 2. 用户进入任务列表管理界面 3. 用户选中需要恢复运行的挂起的任务 4. 用户在对应任务操作栏点击继续执行按钮，提交恢复挂起任务请求
扩展流程	1a. 用户输入错误的账号密码组合，系统重定向账号注册或找回密码界面 4a. 当系统资源不足以恢复挂起任务请求时，系统应予以提示，并拒绝恢复任务请求
特殊需求	用户恢复挂起任务时，系统应提示任务完成预计所需要花费的时间

用户在提交任务之后，可能会因为参数配置不当、框架或算子需要更新等原因需要停止正在执行的任务，于是系统系统实现了这一需求的解决方案。用户在登录系统后可在任务管理列表界面停止正在执行的任务。其详细用例描述如表 3-11所示。

表 3-11: 停止任务用例描述

ID	UC08	名称	测试任务中断
描述	用户停止正在执行中的任务		
参与者	系统用户		
触发条件	用户在任务管理列表界面		
前置条件	用户登录系统，并进入任务管理列表界面		
后置条件	系统停止正在执行的任务，并返回被停止任务状态		
优先级	高		
正常流程	1. 用户登录系统 2. 用户进入任务列表管理界面 3. 用户选中需要停止运行的任务 4. 用户在对应任务操作栏点击停止运行按钮，提交停止任务运行请求		

扩展流程	1a. 用户输入错误的账号密码组合，系统重定向账号注册或找回密码界面 4a. 当被停止运行任务存在关键任务未完成时系统应给出提示，并拒绝用户的停止任务请求
特殊需求	运行中的任务被停止时，系统应生成任务当前的运行结果报表

系统执行测试任务过程中会持久化存储过程信息，任务运行结束后系统会基于这些过程信息进行分析，生成最终的任务运行结果报表。结果报表主要包括测试用例生成的总次数、有效的测试用例总数、每个变异算子参与生成有效测试用例的总数、任务的起始结束时间等。其详细用例描述如表 3-12所示。

表 3-12: 查看任务结果报表用例描述

ID	UC09	名称	查看任务结果报表
描述	用户查看运行完成任务的结果报表		
参与者	系统用户		
触发条件	用户在任务管理列表界面		
前置条件	用户登录系统，并进入任务管理列表界面		
后置条件	系统跳转至选取任务结果报表界面		
优先级	高		
正常流程	1. 用户登录系统 2. 用户进入任务列表管理界面 3. 用户选中需要查看运行结果报表的任务 4. 用户在对应任务操作栏点击查看运行结果报表按钮，提交查看请求		
扩展流程	1a. 用户输入错误的账号密码组合，系统重定向账号注册或找回密码界面 4a. 当选择任务的运行结果报表正在生成时，系统应该给出提示，让用户等待或者拒绝本次查询请求		
特殊需求	任务运行结果报表应至少包含三张图表，便于用户对任务运行结果进行可视化分析		

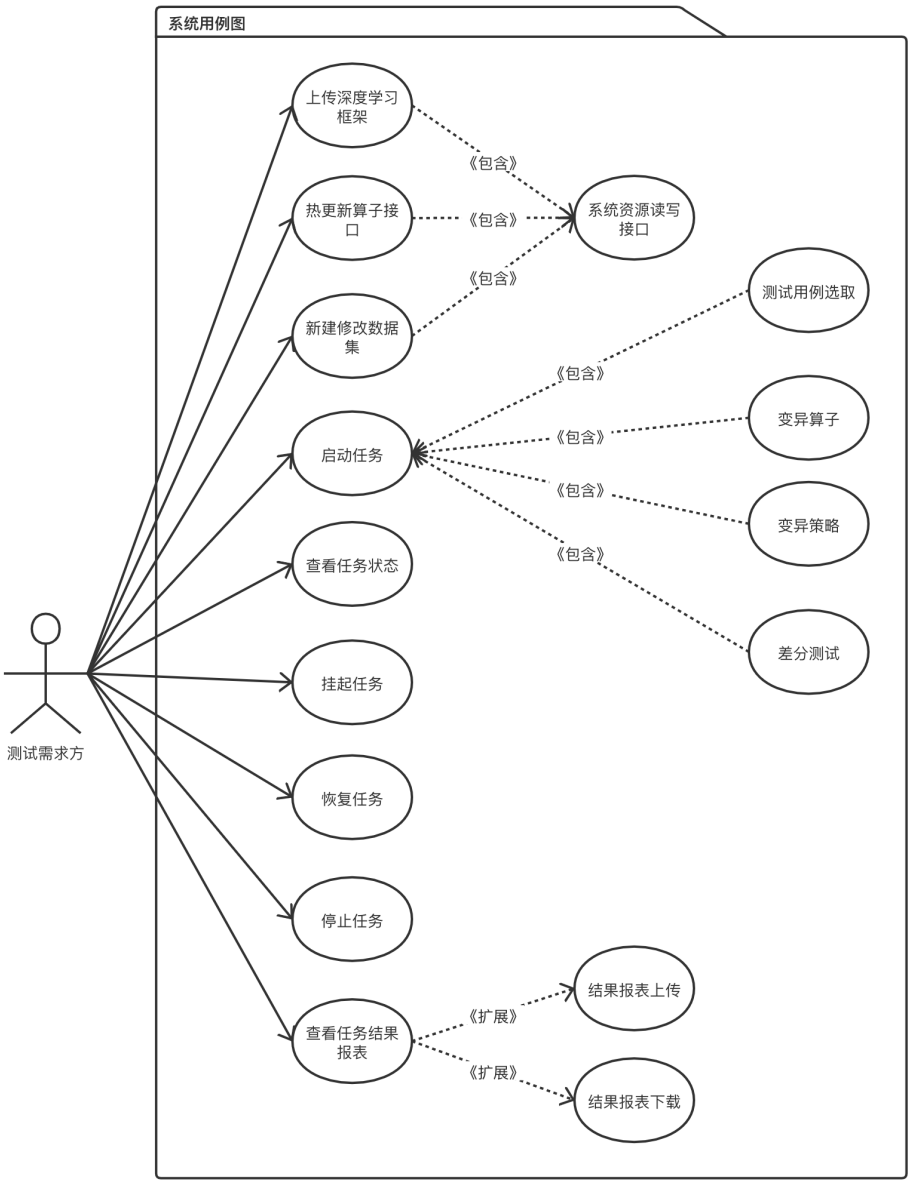

图 3-2: 系统用例图

3.4 系统总体设计

3.4.1 系统架构

系统系统的系统架构图如图 3-3所示。技术实现系统的底层服务主要由服务器、Ubuntu、Docker、Django 组成。服务器拥有 8 和 16 线程 CPU，11GB 的显卡，能够轻松满足中小量测试任务压力。Ubuntu 系统版本号为 20.04 LTS，

具有良好的稳定性。Docker 版本为 18.09，能够在绝大部分主机上流畅运行。Django 版本为 3.1.7，具有良好的可拓展性。系统的对外接口分为输入接口输出接口两类。输入接口主要包括深度学习框架输入接口、深度学习算子输入接口、初始测试数据集输入接口。用户可以通过对应接口上传已发布或自研深度学习框架，通过算子接口热更新深度学习算子，通过测试数据接口更新系统测试用例集合。输出接口主要包括测试结果报表接口和数据库调用 API 接口。用户通过测试结果报表接口生成测试结果，通过数据库 API 接口增删改查数据库中的记录。系统基础管理部分由用户管理、框架管理、算子管理、种子管理组成；算子测试部分由输入处理、扰动算子、导向型模糊测试、差分评估组成。

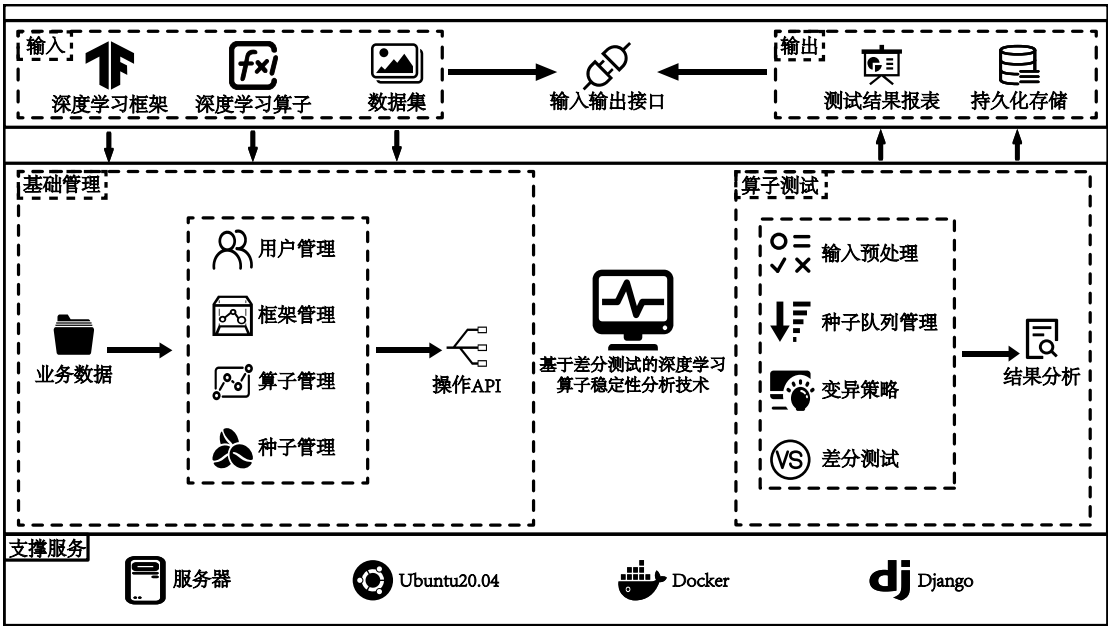

图 3-3: 系统架构图

3.4.2 4+1 视图

在总体结构的基础上结合 Kruchten 教授提出的“4+1”视图方法给出了系统系统在不同视角的逻辑架构设计 [43]。“4+1”视图包括逻辑视图、开发视图、进程视图、物理视图，即描述软件架构的五个不同视角。在第 3.3 章节的用例描述分析中对场景视图进行了详细的阐述，在本章节不再重复描述。在这一章节将重点从逻辑视图、开发视图、进程视图、物理部署视图对系统系统的逻辑架构设计展开相对应的描述。

(1) 逻辑视图

逻辑视图是面向最终用户的一种逻辑架构，通过直观、系统性的方式向最终用户展现系统的功能性服务、非功能性服务以及系统的逻辑组织架构。逻辑视图往往会分解成一系列功能抽象，系统系统逻辑视图对应如图 3-4所示。

逻辑视图应用层主要包括 web 后台、文件传输、Docker 部署。web 后台面向用户提供可视化操作服务，主要包括用户登录、框架及算子操作功能、算子测试功能；文件传输用来传输管理大型文件，算子管理涉及到大量文件的增删改查工作，高效的文件传输服务能提高系统效率；Docker 部署主要是通过编写 Dockerfile 实现 Docker 镜像的更新、提交、发布，最终达到系统一键式迁移运行的目的。逻辑视图转发层主要包括测试任务分发、限流及负载均衡功能。由于测试任务属于较长期任务需要耗费大量资源，单主机处理能力有限，测试任务提交时需根据各个主机的负载情况分发至系统资源较为充足的服务器；当服务集群资源不足以处理用户提交的任务请求时，系统会限制用户发起测试任务的数量直至集群资源充足系统。逻辑视图控制层主要由接口请求 POST 和 GET 组成，提供一系列功能完善，逻辑功能划分清晰的网络请求 API 接口。逻辑视图业务层包括用户、测试数据管理功能以及用例抽样、种子扰动变异、差分评估组成。用户管理实现用户登录注册以及权限控制功能；测试用例主要用来维护初始测试语料集；测试用例抽样通过策略或者随机抽样的方式抽取测试用例进行扰动变异；扰动变异服务由一系列扰动算子及扰动策略组成，用于对种子进行扰动生成测试用例；差分评估通过差分检测的方法检测待测算子是否存在差分异常，进而分析算子是否存在运算不稳定的情况。逻辑视图持久层主要用于持久化存储系统相关配置信息、用户信息、算子测试中间过程及结果数据。逻辑视图数据层主要包括数据库容灾方案，本系统采用主从复制的方式提高数据存储的可靠性，进一步降低数据丢失的可能性。

(2) 开发视图

开发视图适用方是具有专业技术水平的开发人员，重点在于展现和梳理系统模块的作用以及相互之间的调用关系。系统系统开发视图如图 3-5所示。

系统系统采用的是 MTV 模型，MTV 模型本质上是 MVC 模型，其中 M 表示 Model、T 表示 Template、V 表示 View。系统开发的外围接口包括 Web 浏览器、文本资源管理器以及数据库调用接口。Web 浏览器通过调用系统外围接口

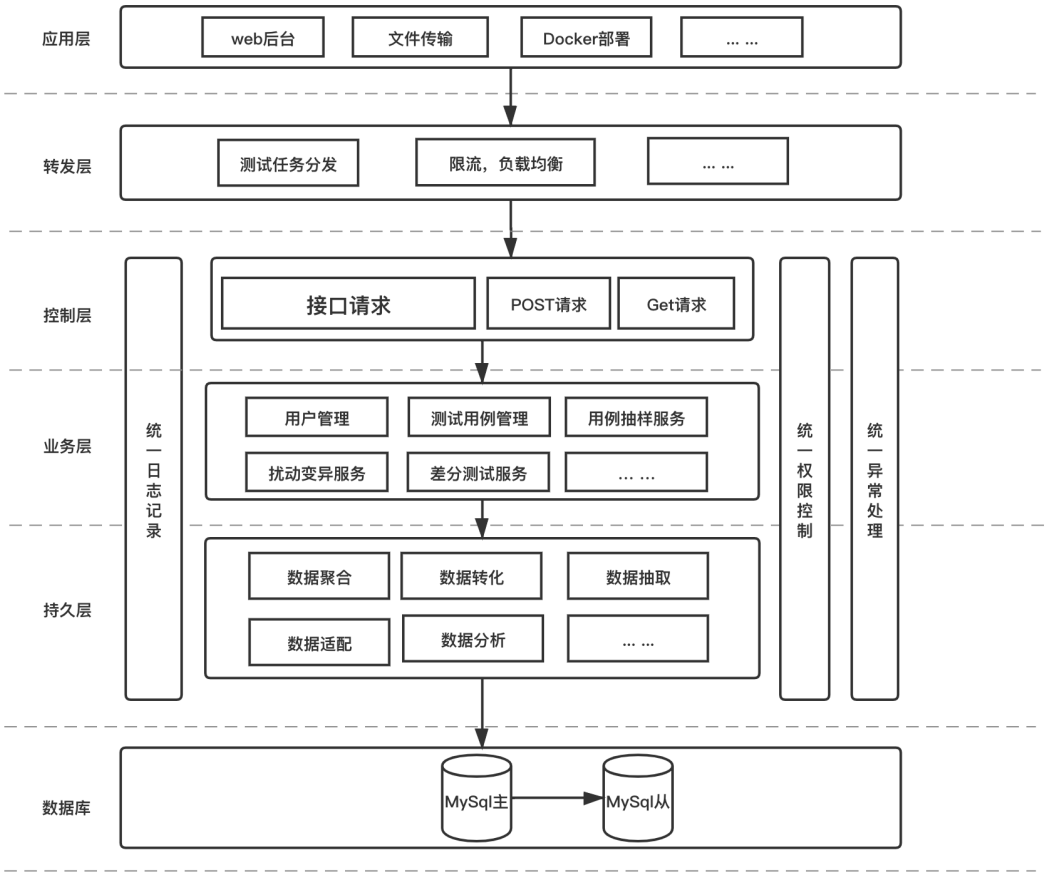

图 3-4: 系统逻辑视图

向用户展现系统界面；文本资源管理器用于管理系统的文本文件，例如自研深度学习框架的算子实现；数据库调用接口主要是用于查找修改数据库中的数据条目。在 Django 框架中 URL 路由转发主要通过项目根目录项目文件夹中 urls.py 文件实现；逻辑处理功能是基于 Django 子程序 views.py 中的函数实现，views.py 中函数主要用于处理逻辑业务以及组织前端界面；Model 数据实体负责和数据库中的数据表建立映射关系，其映射关系建立方式为：通过在 Django 子 app 文件夹中 models.py 编写相应的数据类实体，然后基于 models.py 文件执行 makemigrations 命令和 migrate 命令即可在对应数据库中生成相应的数据表；View 展现页面 Django 的实现与传统 MVC 架构稍有不同，View 界面资源存放在项目根目录 templates 文件夹中，里面主要为 html 类型文件，其中 css、js 相关文件不能够存放在 templates 文件夹中，需存放在项目根目录 static 文件夹中。Django 框架项目部署在 Docker 镜像容器中，可自由迁移部署到不同主

机中。系统系统 MySQL 采用主从复制 (一主一从) 实现数据库集群的高可用，目前 CentOS 默认使用 MariaDB，在实际使用中 MariaDB 和 MySQL 兼容性良好，固可根据实际情况自由选择 MariaDB 或者 MySQL 数据库。

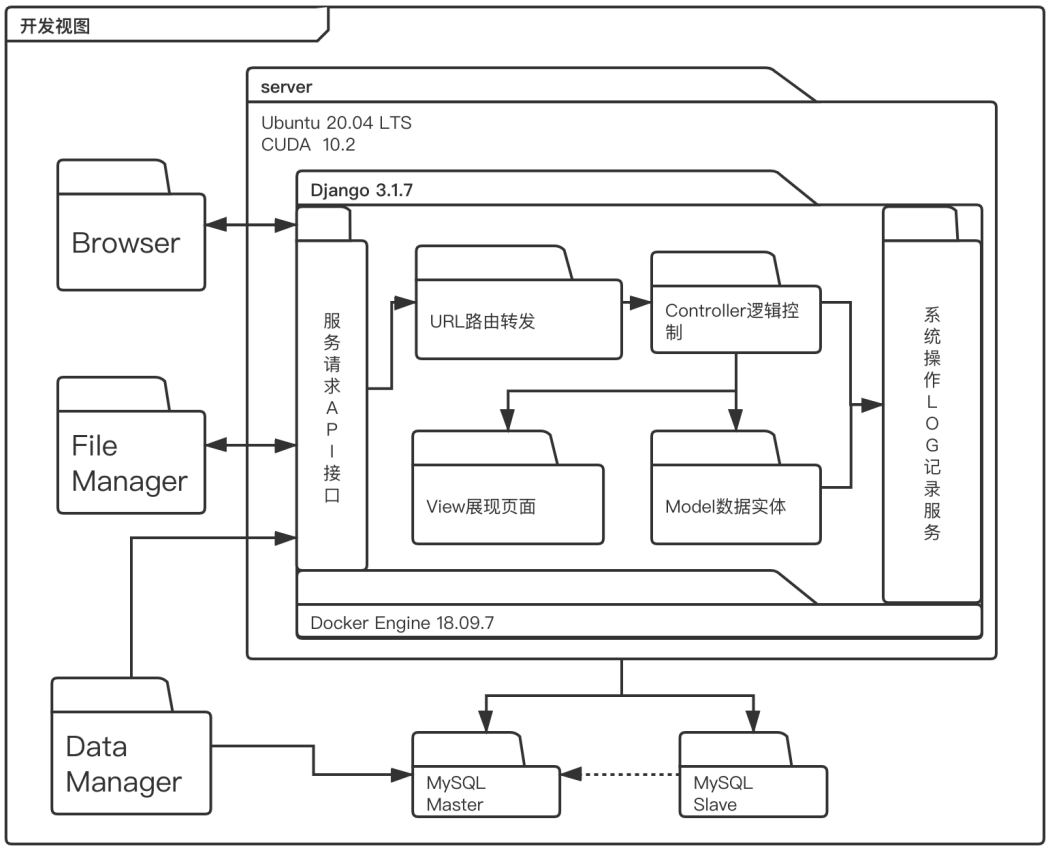

图 3-5: 系统开发视图

(3) 进程视图

进程视图从技术实现系统处理请求的运行过程描绘并发和同步的特性，主要是展示进行线程并发同步以及通讯问题。系统系统进程视图如图 3-6所示。

系统系统启动时首先会启动主线程，用于接收、管理用户请求，由于测试任务需要耗费大量时间，所以主线程采用线程池的方式启动、管理测试任务，线程池的默认上限为 CPU 的核心数，若系统活跃线程数超过 CPU 核心数则很有可能造成大量的 CPU 上下文切换。当线程池中的测试任务启动后，系统首先通过抽样策略对种子进行抽样，种子取好之后会通过变异函数对种子进行微小扰动生成对抗样本，最后差分测试线程基于生成的测试用例进行差分测试，这

一过程反复进行直至达到差分测试的终止条件，期间差分结果会反馈给任务管理线程进行分析并调用数据库接口进行持久化存储。为了实现用户对运行中线程的可控性，用户可向主线程发送指令，主线程转发这一命令指令至任务管理线程，然后任务管理线程对运行中测试任务进行控制。

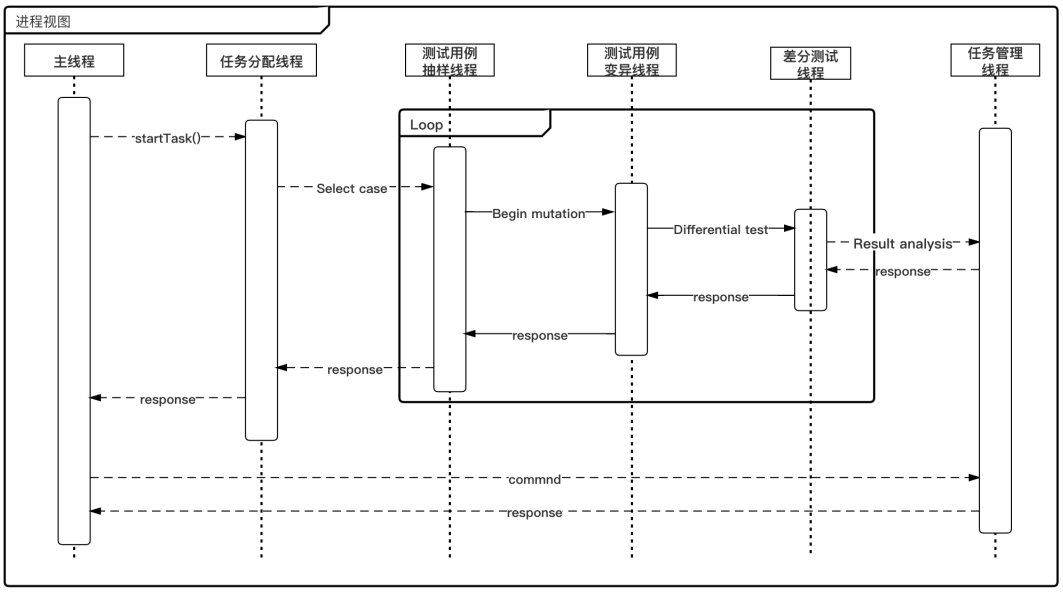

图 3-6: 系统进程视图

(4) 物理视图

物理视图是基于软件部署的角度出发，用来描述系统服务与物理硬件之间的映射关系，目的在于梳理系统逻辑，指导开发人员和运维人员安装部署系统服务。系统系统物理视图如图 3-7所示。

用户可通过前端界面访问本系统，常用可用的浏览器有：Chrome、Firefox。系统使用 Django 架构开发，服务器在收到用户任务请求后经过简单数据处理转发至主测试服务器。系统核心系统服务通过 Docker 镜像安装运行在不同物理主机上，Docker 内部封装的业务主要包括用户管理、资源管理、算子测试任务服务，这些服务通过 django.db.backends.mysql 通讯组件与 MySQL 数据库进行通讯交互。MySQL 服务器可部署在任一具有网络通讯功能的物理主机上，在测试环境下数据库服务器安装在腾讯云主机。为了缓解主测试服务器的任务压力，技术实现设计了一系列从测试服务器用来处理主测试服务器的测试任务。从服务通过 Docker 容器镜像安装运行在不同的物理主机上。通过

Docker 技术能很好屏蔽各物理主机硬件之间的差异，实现无差别安装运行。考虑到网络因素，系统使用阿里云容器镜像服务托管 Docker 镜像文件。

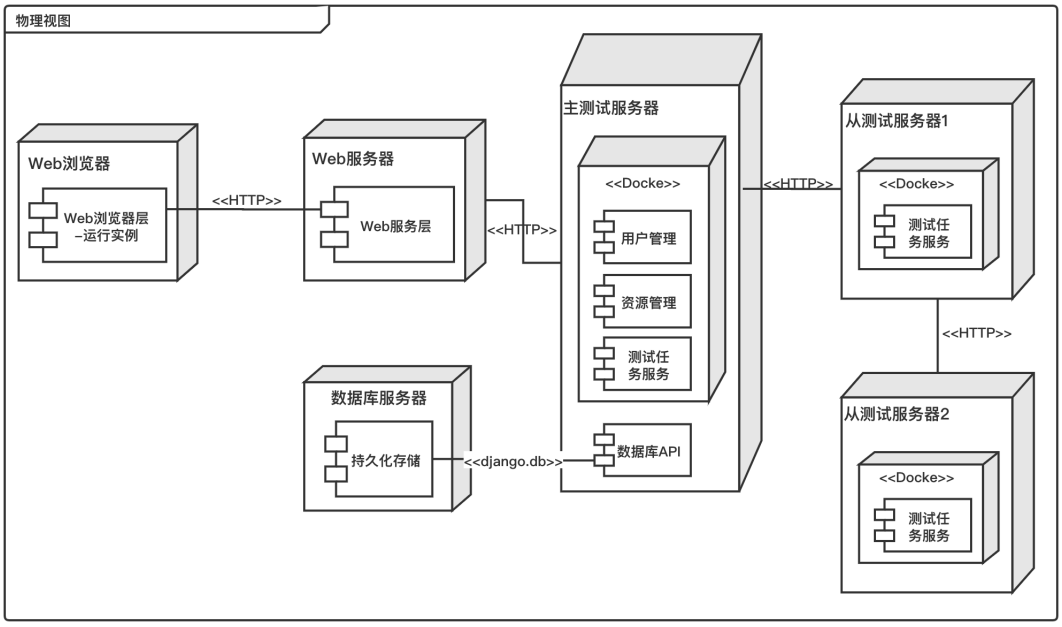

图 3-7: 系统物理视图

3.4.3 测试任务分配流程计

算子测试任务需要在较长一段时间内持续使用大量的系统资源，同一服务主机处理算子任务的并发量有限，故需要对算子测试任务进行统一分配和调度。当技术实现系统收到用户提交的测试任务时，系统首先会把测试任务添加至线程池。单个服务主机的最大并发线程数为服务器 CPU 核心数，并发数超过 CPU 线程数时会频繁的发生大量的 CPU 上下文切换造成性能损耗，故当线程池容纳量已满时，将测试任务添加至任务等待队列。线程池添加新的测试任务时会基于任务优先级对线程池中的任务进行排序。主测试服务器会不断从线程池中选取测试任务，选取测试任务后当主测试服务器并发线程数小于 CPU 核心数则在主测试服务器运行测试任务，否则将测试任务提交至从测试服务器运行。从测试服务器在收到测试任务之后，同样也会先检查线程并发数是否超过 CPU 核心数，若 CPU 核心数大于当时任务并发数则执行测试任务，否则将测试任务提交至任务等待队列，无论主测试服务器还是从测试服务器在执行完任务后都会将测试结果持久化存储在数据库中。当线程池执行完任务后会给任务

等待队列发送唤醒信号，等待队列收到唤醒信号后会从队列中取出测试任务提交运行，当等待队列不存在任务时则不处理唤醒信号。

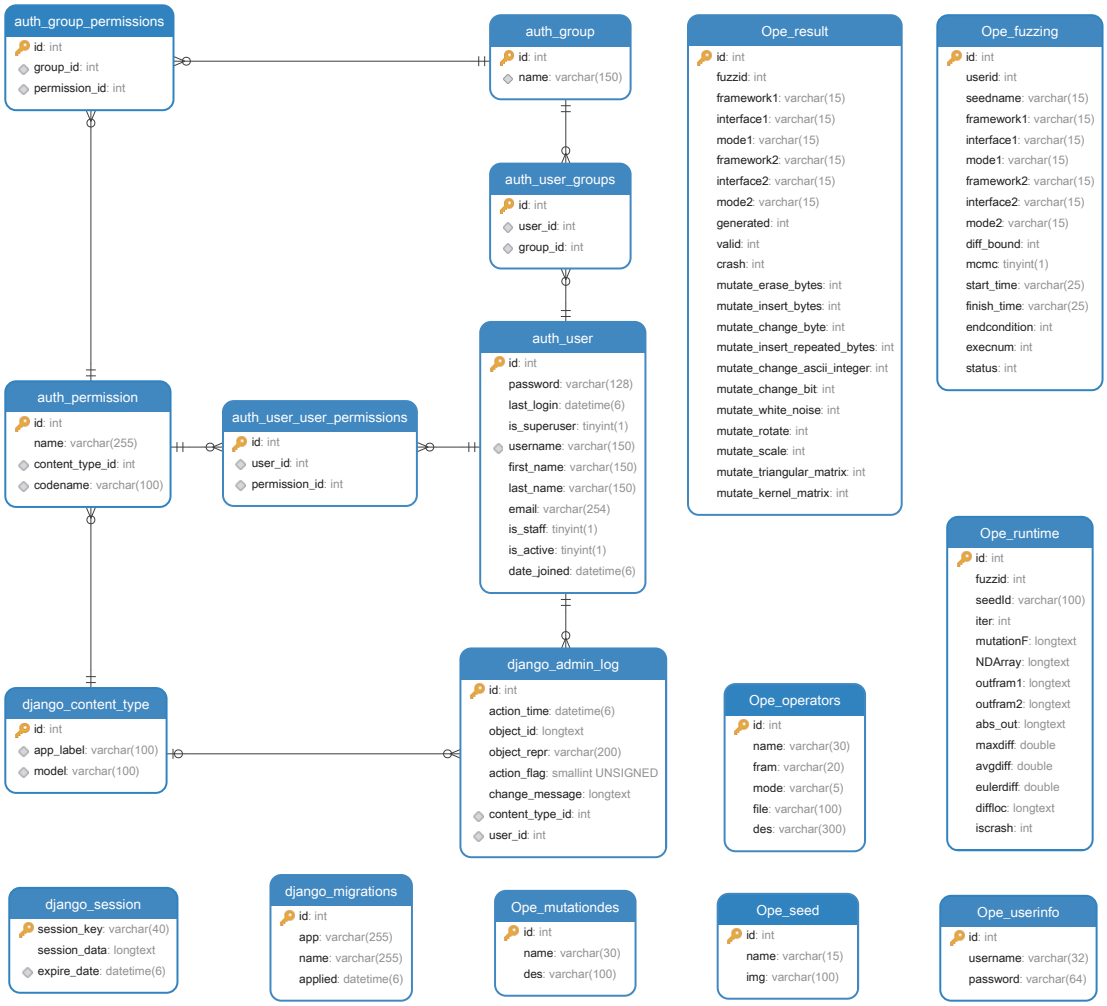

图 3-8: 数据库表设计物理模型视图

3.5 数据库实体设计

系统的用户信息，测试任务过程信息以及分析结果分析都需要持久化存储在数据库中，数据库可是使用 MySQL 或者 MariaDB，两种数据库连接及使用并基本无差别。在技术实现过程中使用 django.db.backends.mysql 引擎与数据库建立连接并进行数据交互。数据库中业务直接相关数据表有：Ope_result、Ope_fuzzing、Ope_runtime、Ope_operators、Ope_mutationdes、Ope_userinfo，同时考虑到测试任务运行过程中存在大量读写请求，业务相关数据表全部取消外

键，以提高数据库读写效率。数据库对应的表设计物理视图如图 3-8所示。

Ope_result 是算子测试结果记录表，存储一次测试记录对应的测试结果，该表主要用于记录测试记录的标识信息以及对应的测试结果汇总分析数据，其详细设计如表 3-13所示。

表 3-13: Ope_result 表字段详细设计

字段	含义	类型	描述
id	测试结果 ID	int	用于唯一标识测试记录结果
fuzzid	对应测试记录 ID	int	用于和测试记录建立映射关系
framework1	待测深度学习框架	string	深度学习框架名称例如 TensorFlow、PyTorch 或自研深度学习框架
interface1	待测算子	string	表示进行差分测试的算子名称例如 conv2d、sigmoid 等
mode1	待测算子运行环境	string	标识深度学习框架是在 CPU 运行还是在 GPU 运行
framework2	对比深度学习框架	string	深度学习框架名称例如 TensorFlow、PyTorch 或自研深度学习框架
interface2	对比算子	string	表示进行差分测试的算子名称例如 conv2d、sigmoid 等
mode2	对比算子运行环境	string	标识深度学习框架是在 CPU 运行还是在 GPU 运行
generated	生成测试用例数	int	表示算子测试过程中生成的测试用例总数
valid	有效测试用例数	int	表示算子测试过程中有效生成用例数
crash	差分异常用例数	int	表示引起算子差分异常用例数
erase_byte	字节擦除扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目

insert_byte	字节插入扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目
change_byte	字节改变扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目
repeat_byte	字节重复扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目
erase_bit	比特擦除扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目
insert_bit	比特插入扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目
change_bit	比特翻转扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目
repeat_bit	比特重复扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目
change_ascii	ascii 改变扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目
white_noise	白噪声扰动函数	int	表示在触发差分异常测试用例中该扰动函数参与的数目

Ope_fuzzing 表示算子测试启动参数记录表，存储了一次算子测试用户所配置的相关参数，主要包括框架算子信息、测试启动结束时间、运行次数终止条件等信息。其详细设计如表 3-14所示。

表 3-14: Ope_fuzzing 表字段详细设计

字段	含义	类型	描述
id	测试记录 ID	int	用于唯一标识测试记录
userid	测试任务发起人	int	表示该测试任务发起用户
seedname	初始 Corpus	string	表示测试任务使用的初始 Corpus
framework1	待测深度学习框架	string	深度学习框架名称例如 TensorFlow、PyTorch 或自研深度学习框架

interface1	待测算子	string	表示进行差分测试的算子名称例如 conv2d、sigmoid 等
mode1	待测算子运行环境	string	标识深度学习框架是在 CPU 运行还是在 GPU 运行
framework2	对比深度学习框架	string	深度学习框架名称例如 TensorFlow、PyTorch 或自研深度学习框架
interface2	对比算子	string	表示进行差分测试的算子名称例如 conv2d、sigmoid 等
mode2	对比算子运行环境	string	标识深度学习框架是在 CPU 运行还是在 GPU 运行
diff_bound	差分阈值	int	表示差分测试对应的差分阈值
mcmc	是否采用 mcmc 策略	int	表示测试任务是否使用 mcmc 策略
start_time	任务开始时间	string	记录测试任务的开始时间
end_time	任务结束时间	string	记录测试任务的结束时间
endcondition	目标差分异常数	int	算子测试差分异常目标总数
execnum	测试用例目标数	int	测试用例目标生成总数
status	测试任务状态	int	表示测试任务的运行状态，0: 运行 1: 完成 2: 挂起 3: 异常

Ope_runtime 为算子测试过程记录表，记录每个测试用例的测试结果详情，从多个维度数值化度量单个测试用例对应的差分评估结果信息。其详细设计如表 3-15所示。

表 3-15: Ope_runtime 表字段详细设计

字段	含义	类型	描述
id	测试结果 ID	int	用于唯一标识测试记录结果
fuzzid	对应测试记录 ID	int	用于和测试记录建立映射关系

seedId	初始 CorpusID	string	表示测试任务使用的初始 Corpus
iter	迭代序号	int	算子测试过程迭代次数编号
mutationF	变异函数	string	本次迭代对抗样本生成使用的扰动函数列表
NDArray	多维矩阵	text	Numpy 多维矩阵形式输入
outfram1	多维矩阵	text	Numpy 多维矩阵形式待测算子输出
outfram2	多维矩阵	text	Numpy 多维矩阵形式对比算子输出
abs_out	多维矩阵	text	算子输出张量之间元素差分绝对值
maxdiff	最大差分值	double	算子输出张量之间差分结果元素最大值
avgdiff	最大差分值	double	算子输出张量之间差分结果元素平均值
eulerdiff	最大差分值	double	算子输出张量之间欧氏距离
diffloc	差分详情	text	差分异常对应的坐标、具体输出及差值
iscrash	差分结果类型	int	差分异常类型。-6: 待测框架和对比框架算子输出空值异常 -4: 对比框架算子输出空值异常 -2: 待测框架算子输出空值异常 -1: 输入空值异常 0: 正常 1: 最大差分异常 2: 平均差分异常 4: 欧式差分异常 3: 最大及平均差分异常 5: 最大及欧式差分异常 6: 平均及欧式差分异常 7: 最大、平均以及欧式差分异常

Ope_operators 为算子描述表，表示深度学习算子相关信息，主要信息有:

算子名称、所在框架、运行的硬件环境、对应文件名及相关描述信息。其详细设计如表 3-16所示。

表 3-16: Ope_operators 表字段详细设计

字段	含义	类型	描述
id	算子 ID	int	用于唯一标识算子
name	算子名称	string	表示算子的描述名称
fram	深度学习框架	string	算子所在深度学习框架
mode	算子运行模式	int	表示算子是在 CPU 还是在 GPU 环境下运行
file	算子对应文件	string	算子所对应的文件名称
des	描述	string	算子描述信息

Ope_mutationdes 变异函数描述。基于用户视角向用户描述变异算子的相关定义及变异效果。其详细设计如表 3-17所示。

表 3-17: Ope_mutationdes 表字段详细设计

字段	含义	类型	描述
id	变异函数 ID	int	用于唯一标识变异函数
name	变异函数名称	string	表示变异函数名称
des	变异函数描述	string	表示变异函数的描述信息

Ope_seed 为模糊测试种子描述表，用来标识模糊测试种子的名称以及存储路径。模糊测试种子物理存储表现形式为文件系统中的文件夹，文件夹包含若干种子资源。其详细设计如表 3-18所示。

表 3-18: Ope_mutationdes 表字段详细设计

字段	含义	类型	描述
id	测试数据集合 ID	int	用于唯一标识测试数据集合
name	测试数据集合名称	string	表示测试数据集合名称
loc	测试数据集合存储位置	string	表示测试数据集合的物理位置

3.6 本章小结

本章节重点描述本技术的需求分析。章节开始对该技术实现系统进行了涉众分析，以及简要分析了功能性需求和非功能性需求，并在文字分析的基础上结合用例图以及用例描述表对重点需求进行了多维度描述。接着，描述了系统总体设计：首先介绍系统架构图，对系统架构展开分析描述，然后分别对技术实现的逻辑视图、开发视图、进程视图以及物理视图进行了细致分析并绘制相关图形描述。最后，对算子测试任务的分配逻辑进行了说明分析，并重点描述算子测试任务在不同测试服务器中的流转规则。

第四章　详细设计与实现

本系统基于导向型模糊测试方案开发实现，核心关键流程为：制定导向目标、输入优化、种子优先级定义、种子淘汰逻辑、变异策略以及结果分析，图 4-1表示对应的流程示意图。基于此本文将本系统核心模块划分为：算子与种子管理模块、种子队列管理模块、变异策略模块、差分测试模块。其中输入优化、种子优先级定义、种子淘汰逻辑功能由系统种子队列管理模块实现；变异策略功能由变异策略模块实现；结果分析功能由差分测试模块实现。

本章节按模块划分，并通过文字描述、流程图或界面、关键代码对每个模块进行了详细论述。算子与种子管理模块介绍算子与种子文件的上传、下载、查找、删除功能以及资源文件和数据库记录同步的方式；种子队列管理模块重点分析种子优先级定义方式、种子抽取策略以及使用模拟退火算法淘汰种子的具体步骤；变异策略模块则主要描述进行种子变异时扰动函数列表的选取方式，以及扰动幅度的的动态调整方法；差分测试模块则主要介绍差分评估方法以及具体的差分异常类型。

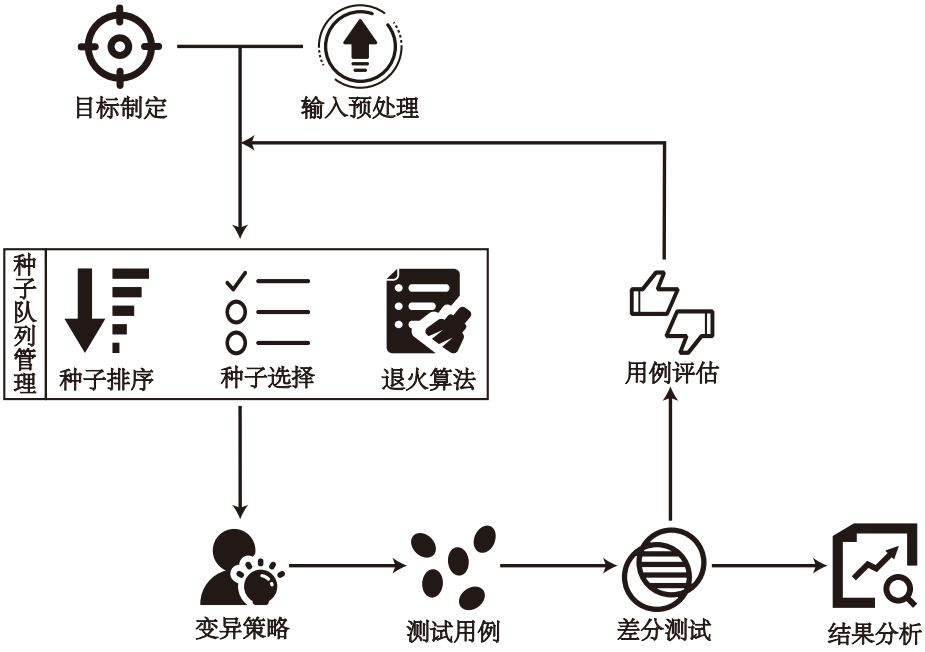

图 4-1: 导向型模糊测试方案流程示意图

4.1 算子与种子管理模块

本系统算子和种子管理模块操作形式一致，具体业务有所差异。用户可通过管理文件系统文件夹的形式对种子进行管理操作。用户在上传算子调用接口之前，需要在系统通过 conda、pip 或者直接上传 site-packages 的方式安装深度学习框架，深度学习框架安装完成之后就可以上传相应地算子接口，其对应的流程逻辑示意图如图 4-2所示。

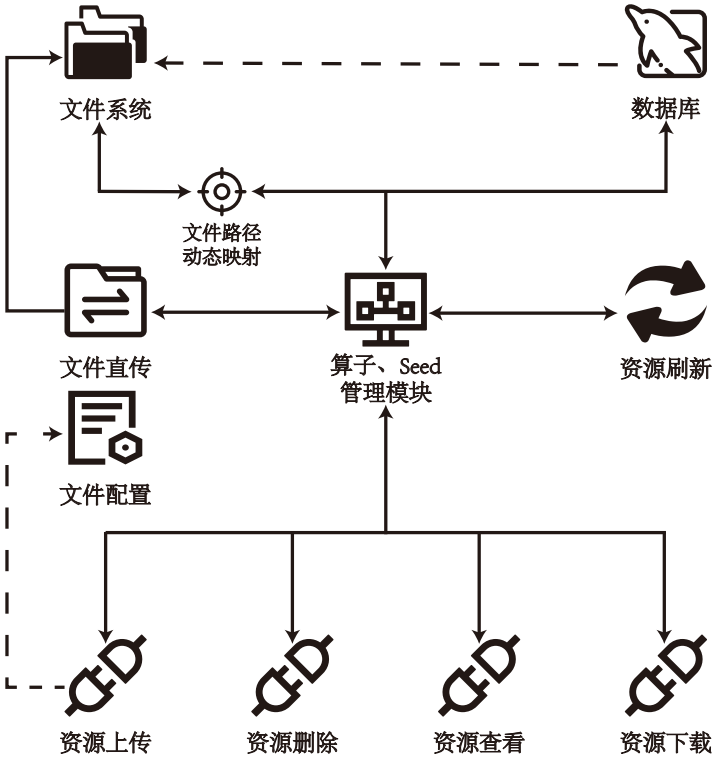

图 4-2: 算子与种子管理模块流程示意图

算子管理模块为用户提供算子上传、删除、查看以及下载接口。用户上传算子接口文件需要编写配置文件，配置文件主要用于描述算子名称、所在框架以及存储路径，图 4-3表示 Sigmoid 算子的配置文件。算子管理模块收到用户算子上传请求后会基于用户配置文件生成文件存储的物理位置，并将相应信息持久化存储到数据库中。此外，用户可直接在文件系统上传算子文件，算子文件上传到文件系统之后，可通过算子管理模块的刷新操作将信息同步到数据库，进一步提高了算子上传灵活性。算子上传、查看、删除模块通过算子 ID 从数据库中查找文件存储路径从而从文件系统中提取算子文件进行下一步操

作，同时文件系统和数据库中信息会保持同步，删除数据库中条目是对应的资源文件也会被删除。本系统通过类热加载机制实现算子接口文件的热加载：即在不重启系统的情况下实现算子接口文件的加载调用，对应的关键代码实现如图 4-4所示。

```
{
    "id": "0001",
    "name": "Sigmoid",
    "mode": "GPU",
    "belongFramWork": "TensorFlow1.9.0",
    "owner": "user01",
    "des": "Sigmoid常作为为深度神经网络模型中的激活函数，其映射范围为(0,1)",
    "fileinfo": {
      "name": "Sigmoid.py",
      "location": "/mnt/data/Ope/"
    }
  }
```

图 4-3: 算子配置 json 格式文件截图

```
1.  import importlib
2.      """
3.      动态解析加载算子
4.      :param fram1: 算子 1
5.      :param fram2: 算子 2
6.      :return: 对应的两个算子
7.      """
8.      def obtain_operator(self, fram1='computer.pytorch.conv1_gpu',
9.                          fram2='computer.tensorflow.conv1_gpu'):
10.         # 通过文件名实现类加载
11.         self.fram1 = importlib.import_module('computer.%s.%s_%s' % (
                        self.framework1, self.interface1, self.mode1))
12.         self.fram2 = importlib.import_module('computer.%s.%s_%s' % (
                        self.framework2, self.interface2, self.mode2))
```

图 4-4: 算子动态热加载核心实现代码

深度学习框架算子稳定性测试工具算子管理模块具体代码实现主要有视图、数据模型、逻辑处理组成。视图部分表单可通过 form.as_p 命令自动化方式生成，此外，由于 Ajax 异步请求正常情况下无法进行资源下载操作，于是采用通过 Dom 调用隐藏 From 表单的方式进行资源下载操作；数据模型部分需要动态生成算子文件存储路径，文件存储路径根据处理后的文件名动态生成，其详细实现如图 4-5所示；逻辑部分首先提取用户表单中的数据，对数据进行预处理，接着利用正则表达式对文件名以及文件存储路径进行匹配验证，拒绝不规范的命名文件，然后将文件上传到文件系统并同步到数据库，其对应核心实现如图 4-6所示。

```python
def user_directory_path(instance, name):
    if isinstance(instance, Operators):
        return os.path.join(instance.fram, name)
    if isinstance(instance, Seed):
        return os.path.join('seeds', instance.name, name)
class Operators(models.Model):
    name = models.CharField(default='', max_length=30)
    fram = models.CharField(default='', max_length=20)
    mode = models.CharField(default='', max_length=5)
    # 调用路径生成函数，动态生成文件存储路径
    file = models.FileField(upload_to=user_directory_path, default='')
    des = models.CharField(default='', max_length=300)
```

图 4-5: 算子存储路径动态生成代码实现

```python
@csrf_exempt
def upope(request):
    if request.method == "GET":
        return render(request, "upope.html", {"form": form})
    else:
        # 文件名正则表达式匹配模式
        pattern = r'[a-zA-Z]+(\d+(\.\d+)?)?/[a-zA-Z][a-zA-Z0-
                  9]+(_[gc]pu\.py)$'
        # 文件存储路径正则表达式匹配模式
        fileloc = '%s/%s' % (data['fram'], data['file'].name)
        if re.match(pattern, fileloc):
            if getopebyfile(fileloc):
                return
            models.Operators(**data).save()
            return redirect("/upope/")
```

图 4-6: 算子管理模块逻辑处理部分核心代码实现

本系统种子管理模块与算子管理模块类似，同样由视图、数据模型、逻辑处理组成。视图部分可通过 form.as_p 命令自动化方式生成，使用 Dom 调用 Form 表单的形式实现资源下载操作；数据模型部分通过路径生成函数实现种子文件动态存储操作，代码实现如图 4-7所示；逻辑处理模块首先过滤 GET 请求，提取 Form 表单中种子提交数据，然后基于提取数据将文件上传到文件系统并同步到数据库，其对应核心代码实现如图 4-8所示。

通过算子管理模块和种子管理模块，本系统可实现算子、种子文件的上传、下载、查看以及删除操作。种子及算子管理模块提供文件直传接口，用户只需将文件通过 ftp 或者 sftp 方式将文件上传到系统，即可实现文件上传功能，

```python
# 路径生成函数，基于传入参数生成对应存储路径
def user_directory_path(instance, name):
    if isinstance(instance, Operators):
        return os.path.join(instance.fram, name)
    if isinstance(instance, Seed):
        return os.path.join('seeds', instance.name, name)

# Seed 映射模型，通过 user_directory_path 函数实现资源文件动态存储
class Seed(models.Model):
    name = models.CharField(default='', max_length=15)
    # 图片
    img = models.ImageField(upload_to=user_directory_path, default='')
```

图 4-7: 种子管理模块动态存储核心实现

```python
@csrf_exempt
def seedupload(request):
    if request.method == "GET":
        form = SeedForm()  # 初始化 form 对象
        return render(request, "seedupload.html", {"form": form})
    else:
        form = SeedForm(request.POST, request.FILES)  # 数据传给 form 对象
        if form.is_valid():
            data = form.cleaned_data
            models.Seed(**data).save()
            return redirect("/seedupload/")
        obj = dict()
        obj['error'] = '没有上传的文件'
        return HttpResponse(json.dumps(obj))
```

图 4-8: 种子管理模块逻辑处理核心实现

进一步提升系统的可用性，此外，用户还可手动刷新算子及种子资源文件，实现文件系统和数据库存储信息的手动同步操作。

4.2 种子队列管理模块

种子放入种子队列进行管理之前会进行预处理操作：将文件转化为多维数组形式，实现从文件到多维数组的转换。其对应的核心代码实现如图 4-9所示。generate_seed_corpus 函数遍历目标目录的所有文件，遍历过程中 CorpusElement 会实例化种子，并将实例化后的的种子添加至 seed corpus。

种子队列管理算法如 Algorithm 4.1所示。算法输入由初始数据集合构成，

```
1.  def generate_seed_corpus(corpus_dir):
2.      for path in to_import:
3.          inputs.append(import_testcase(path))
4.      seed_corpus = []
5.      for input in inputs:
6.          # 此处仅考虑在CPU 模式下的目标输出值即可
7.          input = input.astype(np.float64)
8.          new_element = CorpusElement(data=input, parent=None)
9.          seed_corpus.append(new_element)
10.     return seed_corpus
11. class CorpusElement(object):
12.     def __init__(self, data, output=None, coverage=None, parent=None,
                        count=0, find_time=0, speed=0):
13.     ...
```

图 4-9: 种子预处理核心代码

初始输入经过预处理生成初始种子队列。算法执行前先将输出集合置空，接着从种子队列选取种子进行扰动变异生成测试用例。基于生成的测试用例进行差分测试，得到相应地差分评估结果，根据差分评估结果决定是否把新生成的测试用例添加至种子队列。如果测试用例执行过程中触发系统崩溃则将测试用例会被添加至异常用例集合中。最后，本系统会根据测试用例的差分偏移距离动态调整种子在优先级队列的位置，同时为了防止种子队列不断无效扩增以及出现局部最优现象，种子队列会使用模拟退火算法统一管理。

种子队列管理模块由种子排序、种子选择以及退火算法组成，当种子选取之后会基于变异策略进行变异生成测试用例，之后测试用例进行差分测试得到相应的差分评估结果。种子队列根据差分评估结果有选择性的更新种子队列。

种子优先级管理功能通过优先级队列实现，种子在优先级队列中的排序依据为种子触发算子差分异常的概率，可通过统计种子变异运行结果获得，对应计算方法如公式 4-1所示。确立种子优先级之后可通过种子优先级自定义种子优先级队列，对应的代码实现如图 4-10所示。

$$\text{PriorityVal} = \mid \text{diffVal} - \text{targetVal} \mid * \frac{\text{successDiff Num}}{\text{totalGeneratedNum}} \tag{4-1}$$

种子模拟退火算法能有效防止测试用例生成产生局部最优现象，模拟退火算法逻辑如 Algorithm 4.2所示。算法首先指定种子初始能量以及单个种子最大扰动次数。然后从种子优先队列选取单个种子，并对种子进行扰动变异。之后通过评估函数计算种子扰动前后的评估变化值，若变化值小于 0 则继续扰动变

算法 4.1 种子队列管理算法

Input:
initInput - Initial input
targetDiffVal - Target differential test threshold
Output:
seedQueue - Optimal corpus that causes difference anomalies
crashInput - A set of crash input
report - Fuzzing report

```
1:  crashInput ← ∅
2:  seedQueue ← initInput
3:  while true do
4:      seed ← SELECT(seedQueue)
5:      seed′ ← MUTATION(seed)
6:      diffVal ← DIFFERENTIAL_TEST(seed′)
7:      if diffVal ≥ targetDiffVal then
8:          seedQueue ← seedQueue + seed′
9:      end if
10:     if Trigger crash then
11:         crashInput ← crashInput + seed′
12:     end if
13:     distance ← EVALUATE_SEED(seed′, diffVal)
14:     seedQueue ← SORT_INSERT(seedQueue, seed′, distance)
15: end while
```

异，否则以一定的概率选择继续或结束扰动变异，选取概率通过 Metropolis 准则计算，对应的计算方法如公式 4-2所示，其中 ΔT 表示评估变化值，T 表示初始评估值。

$$P_{\text{select}} = e^{-\frac{\Delta T}{T}} \tag{4-2}$$

4.3 变异策略模块

变异策略模块主要用于调控选取种子的变异方式以及变异幅度。变异方式主要有扰动函数组成，本文实现的扰动函数包括字节类扰动、比特类扰动以及其他扰动函数。其中字节类扰动包括：erase_bytes(字节擦除扰动)、insert_bytes(字节插入扰动)、change_bytes(字节改变扰动)、repeat_bytes(字节重复扰动)，字节擦除扰动实现如图 4-11所示; 比特类扰动包括：erase_bits(比特擦除扰动)、insert_bits(比特插入扰动)、change_bits(比特改变扰动)、repeat_bits(比特

```python
import heapq
class SeedPriorityQueue(object):
    def __init__(self):
        self._queue = []
        self._index = 0
    def push(self, seed, priority):
        """
        队列由 (priority, index, item) 形式组成
        priority heappush 默认最小堆，使用 '-' 调整为最大堆
        """
        heapq.heappush(self._queue, (-priority, self._index, seed))
        self._index += 1
    def pop(self):
        return heapq.heappop(self._queue)[-1]
    def qsize(self):
        return len(self._queue)
    def empty(self):
        return True if not self._queue else False
```

图 4-10: 种子优先队列实现类

重复扰动)，比特改变扰动实现如图 4-12所示；其他类扰动包括：white_noise(白噪声扰动)、change_ascii(ascii 码扰动)，白噪声扰动实现如图 4-13所示。

变异策略每次执行变异时会从扰动函数列表选取扰动函数对种子进行变异，本系统扰动函数选取方式基于 MCMC 采样方法实现。扰动函数列表通过排序数组存储，排序依据为扰动函数成功变异的概率，可通过扰动函数成功扰动次数除以扰动函数被选取的次数获得。扰动函数选取时首先会从扰动函数列表随机选取一个扰动函数，然后基于公式 4-3 [8] 概率性选择同意或拒绝将随机选取的扰动函数作为本次变异使用的扰动函数。其中 $k1$ 、$k2$ 分别表示当前、随机选择变异策略在排序数组的坐标为，p 为自定义参数，本系统根据实际使用效果设为 0.4。扰动函数选取好之后会对种子进行变异生成测试用例，并且本次选取扰动函数会作为下次变异策略执行的初始扰动函数。此外，变异幅度根据种子变异触发算子差分异常的概率动态调整，当种子触发差分异常概率更大时应该进行细粒度的变异，以发现更多差分异常。基于这一思路，编写了种子的变异幅度计算公式 4-4，其中 P(seed) 表示种子触发差分异常的概率，k 为比例系数。

$$P_{\text{select}} = \min\left(1, (1-p)^{k_2-k_1}\right) \tag{4-3}$$

算法 4.2 种子模拟退火算法

Input:

seedQueue - Seed priority queue

L - Maximum number of seed mutations

Output:

seedQueue′ - Optimal corpus that causes difference anomalies

while *true* **do**
  seed ← SELECT(*seedQueue*)
  for i in range(1, *L*+1) **do**
    seedTemp = *initTemp*
    seed′ ← MUTATION(*seed*)
    ΔTemp = Evaluate (seed′) – Evaluate(seed)
    seedTemp += (ΔTemp ≥ 0)? 1:-1
    if *seedTemp* ≥ 0 **then**
      Probability continues and updating seed
    end if
    if Evaluate (seed′) ≤ 0 **then**
      seedQueue += (seed′)
    end if
  end for
  Return *seedQueue*
end while

```python
def mutate_erase_bytes(data):
    """
    随机擦除字节
    :param data:
    :return:
    """
    num, location = list_to_byte(data)
    if len(num) == 0:
        return data
    idx = random.randrange(len(num))
    num = num[idx:random.randrange(idx, len(num))]
    data = byte_to_list(data, num, location)
    return data
```

图 4-11: 字节擦除扰动代码实现

本文变异策略分为确定性策略和随机性策略两种，确定性策略每次使用若干个确定的扰动函数进行变异，随机性策略每次进行扰动时会随机选取扰动函数进行变异。用户可自行选择使用确定性策略或者随机性策略。变异策略模块

```python
def mutate_change_bit(data):
    """
    随机改变 bit
    :param data:
    :return:
    """
    num, location = list_to_byte(data)
    num = bytearray(num)
    if len(num) > 0:
        idx = random.randrange(len(num))
        num[idx] ^= 1 << random.randrange(8)
        num = bytes(num)
        data = byte_to_list(data, num, location)
    return data
```

图 4-12: 比特改变扰动代码实现

```python
def mutate_white_noise(data, a_min=0.0, a_max=1.0, sigma=5):
    """
    添加白噪音
    :param data: 待变异数据
    :param a_min: 噪声下界
    :param a_max: 噪声上界
    :return: 变异后值
    """
    data_size = []
    data_size.append(len(data))
    element = data[0]
    data_size.append(len(element))
    element = element[0]
    data_size.append(len(element))
    noise = np.random.normal(size=data_size, scale=sigma)
    mutated_data = noise + data
    mutated_data = np.clip(mutated_data, a_min=a_min, a_max=a_
max)
    return mutated_data
```

图 4-13: 白噪声扰动代码实现

首先从种子队列管理模块接收选中的种子；之后随机选取若干扰动函数作为本次种子变异的扰动函数列表，并从扰动函数列表随机选取扰动函数对种子进行扰动变异生成初步测试用例；最后通过变异幅度公式动态调整初步测试用例生成最终测试用例。生成的测试用例作为差分测试的输入，差分测试的结果会反馈给变异策略，使得种子变异策略能够动态调整变异方式。

$$\text{Mutate Degree(seed)} = \frac{1}{P(\text{ seed })} * k \tag{4-4}$$

4.4 差分测试

差分阈值是深度深度算子差分测试的重要指标，在本技术中用户自定义方式指定差分阈值，还可以通过实验的方式指定差分阈值：通过对同一算子进行大量的差分测试，差分测试记录按差分结果数值从大到小排序，取排名前 2% 数据的平均值可作为差分阈值，其对应的计算逻辑如公式 4-5所示。实验获得的差分阈值只是一个参考依据，可根据实践经验进行动态调整。

$$\text{targetVal} = \frac{\sum_{\text{i}=a_1}^{a_n} \text{DiffVal (i)}}{n}, \quad a_k \subseteq \text{ Top 5\% DiffVal } (k = 1, 2 \ldots n) \tag{4-5}$$

差分测试模块主要用于对待测算子和对比算子进行差分评估，其对应的流程示意图如图 4-14所示。差分结果按差分评估方式不同可分为矩阵最大差分值、矩阵平均差分值、曼哈顿距离型差分值、欧式距离型差分值，其中矩阵最大差分值表示输出矩阵张量内部对应位置元素差值绝对值的最大值，矩阵平均差分值表示输出矩阵张量内部对应元素差值的平均值，曼哈顿距离型差分值表示输出矩阵张量之间的曼哈顿距离，欧式距离型差分值表示输出矩阵张量之间的欧式距离。其中矩阵曼哈顿距离和欧式距离计算的代码实现如图 4-15所示。本系统默认采用欧式距离最为测试用例的差分评估结果。

在差分测试过程中，一个测试用例可能产生一种或多种差分异常形式，为了便捷高效记录测试用例对应的差分异常类型，本技术方案使用二进制表示法表示异常类型，1 表示矩阵最大差分值异常，2 表示矩阵平均差分值异常，4 表示曼哈顿距离型差分值异常，8 表示欧式距离型差分值异常，当出发某种差分异常时在相应的二进制位置 1，默认置 0，图 4-16对该表示方法进行了进一步说明。通过这种方式表示的异常类型确定且无歧义。以便后续通过不同的差分评估角度分析算子稳定性。

差分测试结果持久化存储在数据库，测试任务成功运行完成后，用户可通过前端页面查看测试任务结果报表以及每个测试记录详情。测试任务结果报表由表格、饼状图、折线图组成，表格主要包括算子、框架、运行平台、生成测试用例数、有效测试用例数、触发差分异常用例数组成，饼状图用于统计各类

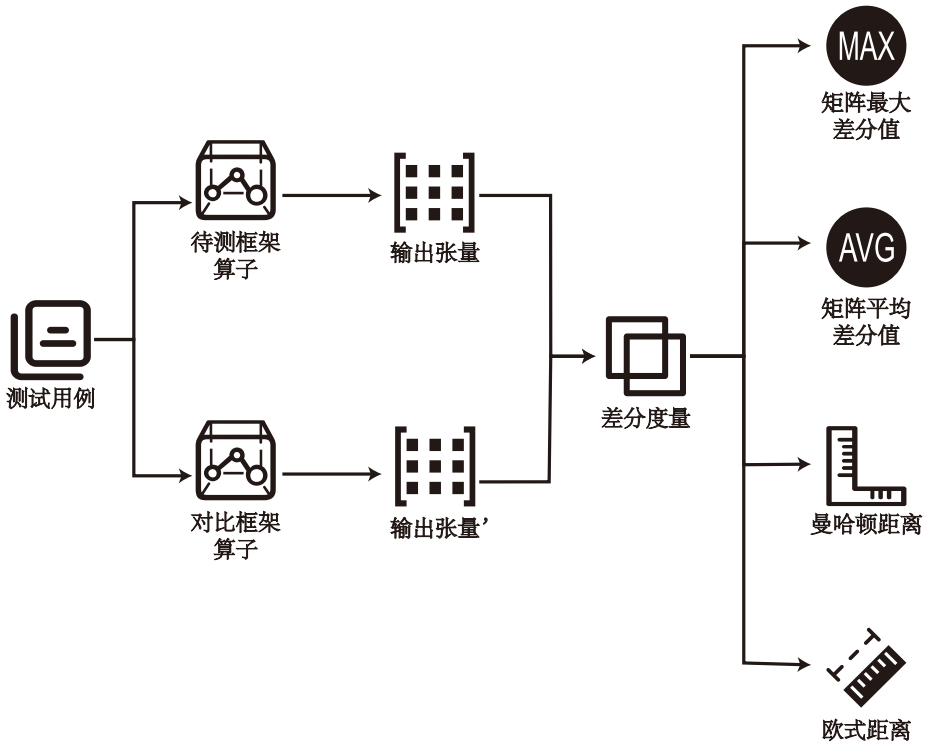

图 4-14: 差分评估流程示意图

```
1.  #曼哈顿距离
2.  def dis_mhd(Tensor1,Tensor2):
3.      if len(Tensor1) == len(Tensor2):
4.          return sum([abs(Tensor1[i]-
                    Tensor2[i]) for i in range(len(Tensor1))])
5.  #欧氏距离
6.  def dis_eu(Tensor1,Tensor2):
7.      if len(Tensor1) == len(Tensor2):
8.          return sum([(Tensor1[i] -
                    Tensor2[i])**2 for i in range(len(Tensor1))]) ** (1.0/2)
```

图 4-15: 矩阵之间曼哈顿距离、欧式距离计算代码实现

差分异常的占比情况，折线图用于描述触发欧式差分异常测试用例的生成趋势，其对应的界面表示如图 4-17所示。测试记录详情包含一个测试用例执行的具体信息，由输入、原框架输出、对比框架输出、所使用变异方法以及差异记录详情组成，其对应的界面图如图 4-18所示。

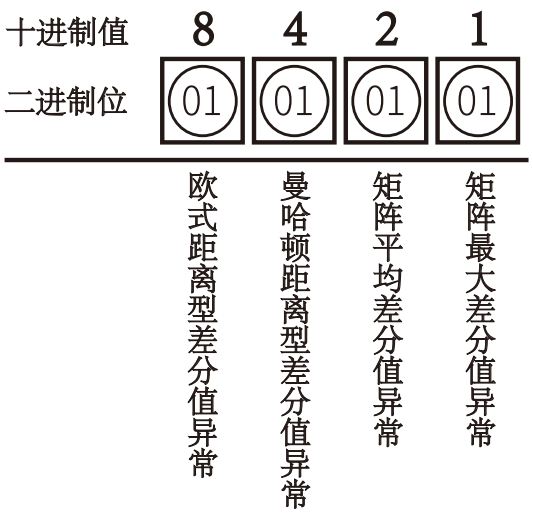

图 4-16: 差分异常类型二进制表示示意图

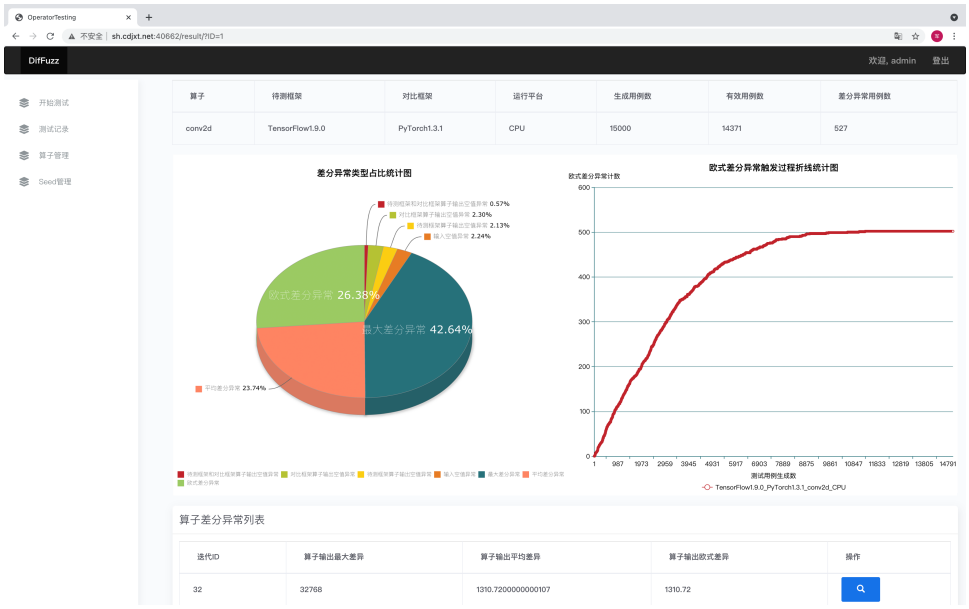

图 4-17: 算子测试任务结果详情界面图

4.5 本章小结

本章首先描述了本系统所采用的导向型模糊测试方案，并给出核心关键工作流程以及相应流程示意图。然后基于导向型模糊测试方案将本系统划分为四个核心模块，分别为算子与种子管理模块、种子队列管理模块、变异策略模块、差分测试模块，并描述各个流程与模块之间的对应关系。接着对各个核心模块的详细设计与实现进行了重点分析描述：首先通过流程示意图、核心代码

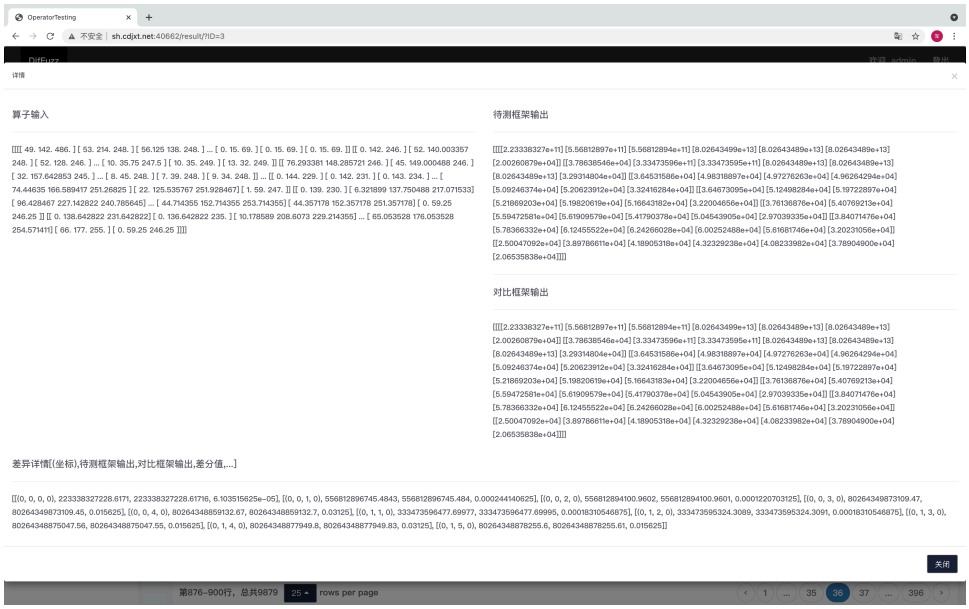

图 4-18: 单测试用例执行结果详情

以及用户交互界面的形式介绍了算子与种子管理模块的具体设计与实现；接着在以上形式的基础上，结合公式以及算法对种子队列管理模块、变异策略模块的详细设计与实现部分进行了细致、清晰的分析和描述；最后介绍差分测试模块，并介绍差分评估中几种典型差分异常类型以及相应的二进制表示。最后，本章给出了本系统相关运行界面截图。

第五章　测试与有效性分析

本章节在第二章系统需求分析与概要设计与第三章系统详细设计与实现的基础上，对本技术进行了测试及有效性分析，测试具体包括测试准备、功能性测试以及非功能性测试；有效性分析部分主要是分析验证本系统深度学习算子稳定性分析能力。

5.1　测试准备

5.1.1　测试目标

在已有的测试硬件、软件的基础上，通过设计并执行功能测试用例的方式对本技术的有效性进行测试，主要测试本技术核心模块外在表现是否达到预期需求目标。本系统的核心模块包括算子与种子管理模块、种子队列管理模块、变异策略模块、差分测试模块。同时，设计相关测试用例对本技术进行非功能性测试，进一步评估及提高技术的可用性、效率、可维护性以及可移植性。

5.1.2　测试环境

本技术测试时所使用的软硬件环境如表 5-1所示，所使用的物理主机安装 Ubuntu 稳定版操作系统并配备两个具备深度学习算子运算加速功能的英伟达显卡。Django 服务以及数据库服务运行在 Docker 构建的虚拟操作系统中，可以一键部署到不同物理主机上，在本技术方案中 Django 和数据库部署在不同的物理主机上。为了模拟出更加真实的运行环境，Docker 运行容器相应端口绑定至具有公网 IP 代理服务器的对应端口，用户可通过代理服务器公网 IP 访问本技术相关服务并进行系统测试。

表 5-1: 测试环境说明

软硬件环境	软硬件环境详情描述
服务主机	CPU: Intel® Core™ i7-9700K CPU @ 3.60GHz × 16 GPU: GEFORCE GTX 1080Ti(11GB) * 2 RAM: 骇客 DDR4 3600 64G 套条 (16G*4)
操作系统	Ubuntu 20.04.2 LTS
Python envs	anaconda3 version:5.2.0-Linux-x86_64
数据库	mysql version:8.0.23-0ubuntu0.20.04.1 for Linux on x86_64
Docker	docker version:20.10.5, build 55c4c88
浏览器	chrome version:89.0.4389.114(正式版本) x86_64
Django	django version:3.1.2(Python3.6.10)

5.2 功能性测试

本技术基于需求分析里描述的功能设计功能性测试用例，测试用例由编号、测试项目、输入/步骤、预期输出组成。通过比对测试用例的实际输出和预期输出的方式验证技术功能的正确性，并对在测试过程中遇到的各种缺陷进行及时修复和验证，确保功能模块都能正确的实现预期目标。功能性测试分别对用户登录控制、种子管理、算子管理、测试任务管理、测试结果分析模块进行测试，其中算子与种子管理模块主要包括算子上传、下载以及删除功能测试。

表 5-2表示用户登录控制测试用例，包括拦截用户非登录状态下的访问系统资源功能，用户退出系统后无法访问受控资源功能，以及进行权限细化，防止用户越权访问功能。经实际测试，对应的测试用例全部通过。

表 5-2: 用户登录控制测试用例

编号	测试项目	输入/步骤	预期输出
TC1-1	拦截非登录用户访问	1. 在浏览器中通过 GET 请求访问内部页面 2. 在 Postman 中通过 POST 请求获取数据	1. 系统拒绝访问，并给出提示 2. 跳转至登录注册页面

TC1-2	用户成功登出	1. 用户通过账号密码登录系统 2. 用户登录系统后进行登出操作 3. 登录登出后重新访问对应用户界面	1. 用户界面提示用户成功登出 2. 再次访问提示未登录并跳转登录界面
TC1-3	拒绝越权访问	1. 用户通过普通用户/管理员身份登录登录系统 2. 访问管理员/普通用户界面	1. 拒绝访问 2. 跳转至相应权限用户登录界面

表 5-3表示算子管理模块测试用例，该模块主要包括算子上传、算子删除、算子下载功能。算子上传时，数据库新增算子描述信息记录，文件系统存放算子文件；算子删除时，数据库删除对应算子描述信息记录。经实际测试，算子管理测试用例全部通过。

表 5-3: 算子管理模块测试用例

编号	测试项目	输入/步骤	预期输出
TC2-1	算子上传功能	1. 在算子管理界面点击算子上传按钮 2. 选中本地算子文件并点击确定上传按钮	1. 数据库新增对应算子描述信息记录 2. 文件系统新增对应算子文件
TC2-2	算子下载功能	1. 在算子管理界面查找对应算子 2. 点击算子对应操作栏的下载按钮	1. 对应算子文件下载到本地
TC2-3	算子删除功能	1. 在算子管理界面查找对应算子 2. 点击算子对应操作栏的删除按钮	1. 数据库删除对应算子描述信息记录 2. 文件系统删除对应算子文件

表 5-4表示种子管理模块测试用例，该模块包括种子上传、种子删除、种子下载功能。种子上传时，数据库新增种子描述信息记录，文件系统存放种子文件；种子删除时，数据库删除对应种子描述信息记录。经实际测试，种子管理测试用例全部通过。

表 5-4: 种子管理模块测试用例

编号	测试项目	输入/步骤	预期输出
TC3-1	种子上传功能	1. 在种子管理界面点击种子上传按钮 2. 选中本地种子文件并点击确定上传按钮	1. 数据库新增对应种子描述信息记录 2. 文件系统新增对应种子文件
TC3-2	种子下载功能	1. 在种子管理界面查找对应种子 2. 点击种子对应操作栏的下载按钮	1. 对应种子文件下载到本地
TC3-3	种子删除功能	1. 在种子管理界面查找对应种子 2. 点击种子对应操作栏的删除按钮	1. 数据库删除对应种子描述信息记录 2. 文件系统删除对应种子文件

表 5-5表示算子测试任务管理测试用例，具体测试用例主要包括新建算子测试任务、挂起算子测试任务、中断算子测试任务、恢复算子测试任务。由于算子测试耗时较长，算子列表页面算子运行状态采用懒加载策略，即需要用户手动刷新页面获取算子运行最新状态。经实际测试，算子测试任务模块对应的测试用例全部通过。

表 5-5: 算子测试任务管理测试用例

编号	测试项目	输入/步骤	预期输出
TC4-1	新建算子测试任务	1. 进入创建算子测试任务界面 2. 配置算子测试任务相关参数，并提交测试任务	1. 跳转至算子测试任务列表 2. 刷新算子测试任务运行状态
TC4-2	挂起算子测试任务	1. 进入算子测试任务列表 2. 点击算子测试任务对应操作栏中挂起按钮	1. 系统提示挂起成功 2. 算子测试任务快照文件存至文件系统
TC4-3	中断算子测试任务	1. 用户进入算子测试任务列表 2. 点击算子测试任务对应操作栏中中断按钮	1. 系统提示算子测试任务中断成功 自动刷新算子测试任务列表
TC4-4	恢复算子测试任务	1. 用户进入算子测试任务列表 2. 点击查看挂起算子测试任务列表 3. 在挂起任务对应操作栏中点击恢复任务按钮	1. 系统提示算子测试任务恢复成功 2. 自动刷子算子测试任务列表并定位至恢复任务

表 5-6表示算子测试结果分析测试用例，测试项目主要包括测试任务统计表格、变异函数变异分布饼状图、差分异常生成折线统计图、单测试用例执行详情查看。其中单测试用例执行详情会在前端页面以弹窗的方式展现。经实际测试，算子测试结果分析管理模块测试用例全部通过。

表 5-6: 算子测试结果分析测试用例

编号	测试项目	输入/步骤	预期输出
TC5-1	测试任务统计表格	1. 测试任务完成后自动统计数据并持久化存储 2. 点击测试任务对应操作栏查看详情按钮	1. 单个测试任务详情页面生成测试任务统计表格
TC5-2	变异函数变异分布饼状图	1. 进入算子测试任务列表界面 2. 点击测试任务对应操作栏查看详情按钮	1. 生成测试任务变异函数参与种子变异分布饼状图
TC5-3	差分异常生成折线统计图	1. 用户进入算子测试任务列表 2. 点击测试任务对应操作栏查看详情按钮	1. 生成差分异常增长折线图
TC5-4	查看单测试用例执行详情	1. 进入相应测试任务详情页面 2. 点击对应测试用例操作栏中的详情查看按钮	1. 弹出单测试用例测试详情窗口 2. 详情窗口包括测试输入、输出、差分详情等

5.3 非功能性测试

非功能性测试主要用于检测应用系统的非功能性特性，例如性能、可用性、可靠性，与功能性测试类似，非功能性测试同样有不可替代的作用，关系到系统用户的使用满意度。非功能性测试主要用来提升系统产品的性能、可用性以及效率，并且非功能性测试具有可度量特性。非功能需求中可靠性通过数据库可靠性存储实现，用户管理模块保证了系统了安全性，Django 采用模块化开发具有良好的可扩展性。

表 5-7: 系统界面响应时间统计表

测试页面	平均响应时间/ms	最大响应时间/ms	最小响应时间/ms
用户登录	1278	1924	615
算子上传	2074	2410	1435
算子查看	1437	1991	1022
算子下载	1840	2436	2044
种子上传	1615	2299	657
种子查看	1940	2167	1504
种子下载	1292	1869	1103
测试任务详情	1331	2256	1074
测试用例详情	1510	2184	1120

系统使用 Fiddler 工具对页面进行响应时间测试，测试页面主要包括用户登录、算子上传、算子查看、算子下载、种子上传、种子查看、种子下载、测试任务详情、测试用例详情，由于算子执行界面响应时间较长，故没有进行页面响应时间测试。页面平均、最大、最小响应时间通过统计同一页面执行 500 次的响应时间获得，对应的界面响应时间测试结果如表 5-7所示。并发量同样也是性能测试的重要指标，考虑到算子测试任务需要消耗大量的系统资源，本系统通过 Jmeter 进行了测试任务并发量为 50 的压力测试，压力测试覆盖所有核心接口，并且响应接口返回数据均正确，达到系统基本并发要求。

在算子稳定性分析技术使用过程中，系统主服务器 CPU 利用率、内存使用量、磁盘 IO 读、磁盘 IO 写情况如表 5-1所示。在没有测试任务时，主服务器各项指标均保持稳定低负载状态。当出现新的算子测试任务时 CPU 会以较快速度增长，然后逐渐下降至较低负载，这与算子测试任务启动时会进行大量资源初始化工作有关，系统总体负载呈现规律性波动。

5.4 有效性分析

为了评估本文深度学习算子稳定性分析技术的有效性，在有效性分析阶段分别对 conv2d、max_pooling、avg_pooling、relu、sigmoid、tanh、softmax、leaky relu 进行了算子稳定性测试，每组测试任务生成 15000 个测试用例，差分

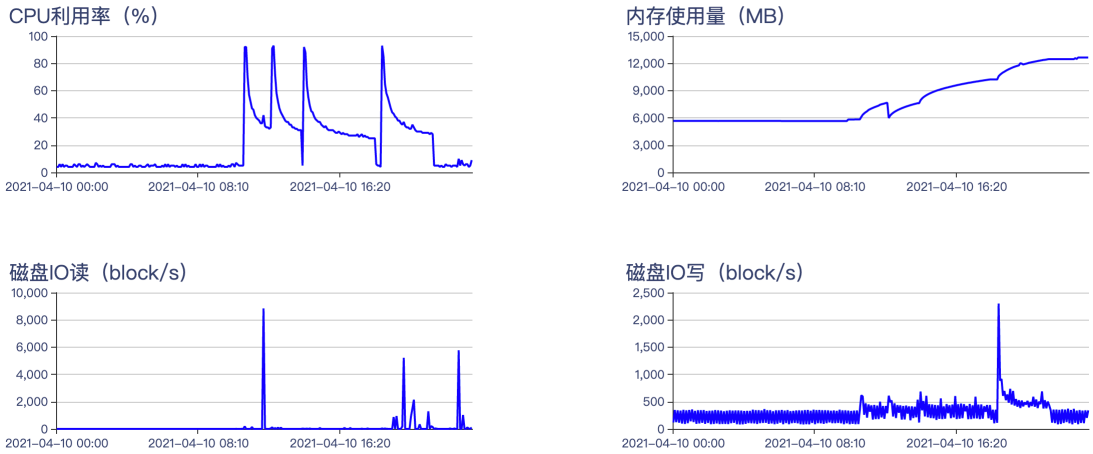

图 5-1: 主服务器负载监控情况统计表

测试设定的差分阈值为 $6 * 10^{-}5$，并重复进行五次，最终结果取平均值。其中 con2d 对应的测试详情如表 5-8所示。

表 5-8: 算子差分测试结果统计表

待测框架	对比框架	待测算子	运行模式	异常/生成
TensorFlow1.9.0	PyTorch1.3.1	conv2d	GPU	519/15000
	Caffe1.0	conv2d	GPU	497/15000
	PyTorch1.3.1	conv2d	CPU	543/15000
	Caffe1.0	conv2d	CPU	481/15000
PyTorch1.3.1	TensorFlow1.9.0	conv2d	GPU	472/15000
	Caffe1.0	conv2d	GPU	465/15000
	TensorFlow1.9.0	conv2d	CPU	436/15000
	Caffe1.0	conv2d	CPU	525/15000
Caffe1.0	TensorFlow1.9.0	conv2d	GPU	317/15000
	Caffe1.0	conv2d	GPU	476/15000
	TensorFlow1.9.0	conv2d	CPU	515/15000
	Caffe1.0	conv2d	CPU	439/15000

在算子测试任务进行过程中，由于触发差分异常的输入空间不断发掘，新生成测试用例触发差分异常的概率逐渐减少，差分异常测试用例增长出现收敛现象，其具体表现形式如图 5-2所示。图 5-2中红色折线未展现出明显的收敛状态，继续增加测试用例生成总数则可使对应曲线达到收敛效果。

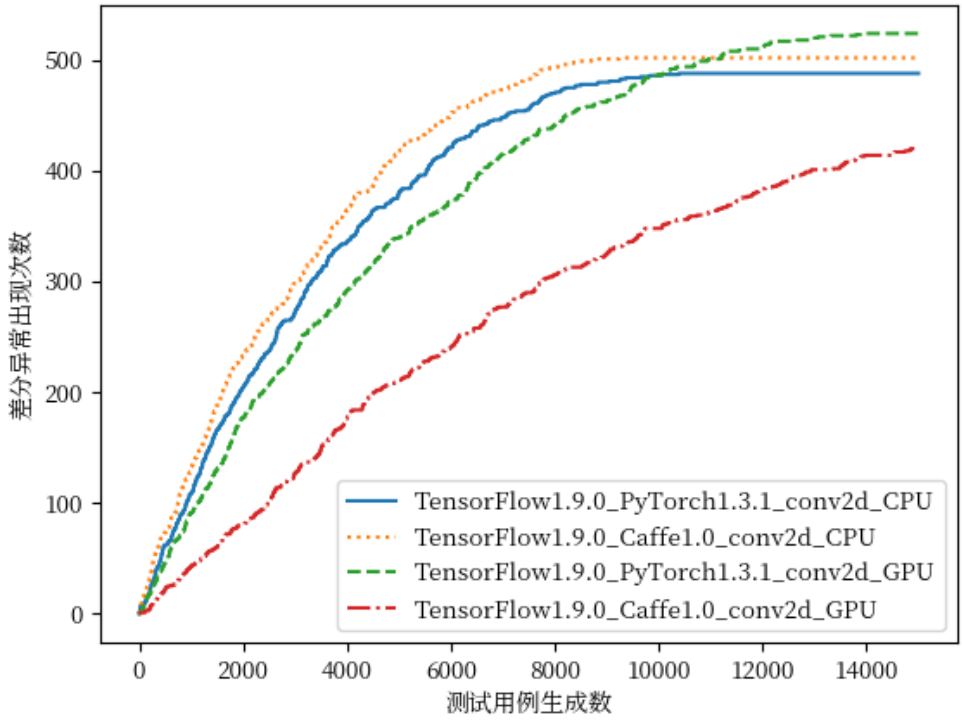

图 5-2: 差分异常触发过程折线统计图

图 5-3表示算子差分异常的统计结果，获取算子差分异常数时对 TensorFlow-PyTorch、TensorFlow-Caffe、PyTorch-Caffe 进行分组测试，每组测试共生成 15000 个测试用例，并重复进行 5 次，取其平均值作为算子差分异常数目。在生成 15000 个测试用例的情况下，通过分析图 5-3可以发现算子在 CPU 和 GPU 运行模式下差分异常数目会有所差异，CPU 模式下差分异常数目稍多于 GPU 模式运行下差分异常数目；每种算子的差分异常数目也出现较大差异，没有乘除法运行的 max_pooling、relu 算子就没有出现差分异常，leaky relu 由于只包含简单的乘法运算也没有出现差分异常，conv2d、avg_pooling、sigmoid、tanh、softmax 由于包含较复杂乘除法运算，大部分差分异常被本系统发掘，其中 conv2d 算子由于包含大量矩阵乘法运算，被捕获的差分异常数目明显多于其他算子。

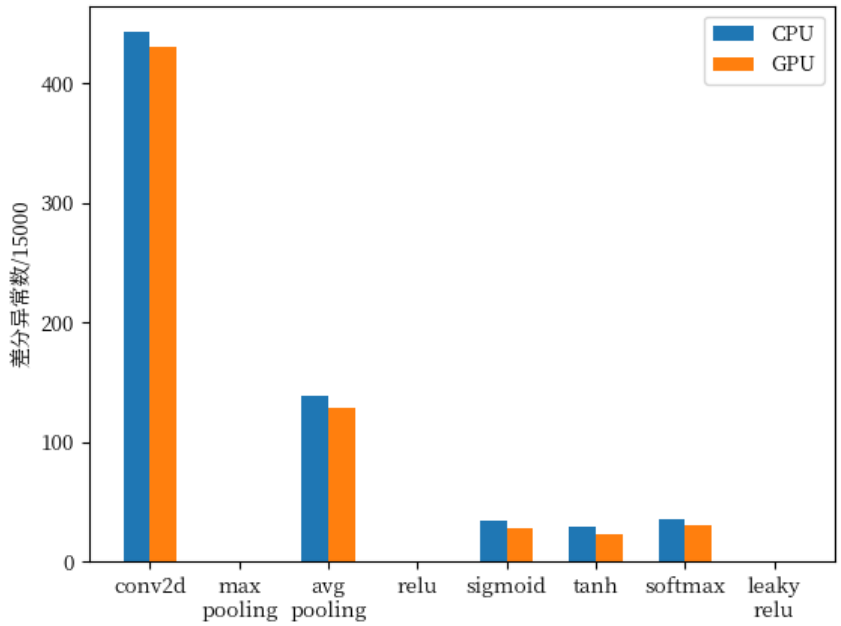

图 5-3: 算子差分异常总数柱状统计图

5.5 本章小结

本章首先简要介绍了本技术的测试目标和相应软硬件测试环境。接着分别介绍本技术系统的功能性测试和非功能性测试，功能性测试包括用户登录控制、算子管理、种子管理、测试任务管理、测试结果分析测试；非功能性测试主要包括技术系统界面响应时间、以及主服务器在处理测试任务时的资源负载情况。最后对本系统的有效性进行分析。使用本系统分析评估 TensorFlow1.9.0、PyTorch1.3.1、Caffe1.0 框架常见算子在 CPU/GPU 运行模式下的稳定性。每次分析评估共生成 15000 个测试用例，重复进行 5 次，最终结果取平均值，conv2d、avg_pooling、sigmoid、tanh、softmax 算子发现的差分异常数目在 GPU 模式下分别为：431、129、28、23、31，在 CPU 运行模式下分别为：443、139、34、29、36。max_pooling、relu、leaky relu 算子暂未发现差分异常。结果表明：conv2d、avg_pooling、sigmoid、tanh、softmax 出现运算结果不一致的情况，不具备良好的稳定性；max_pooling、relu、leaky relu 运算结果稳定一致稳定一致，具备良好的稳定性。此外，深度学习算子在 GPU 运行模式下的稳定性略优于 CPU 模式。

第六章　总结与展望

6.1　总结

深度学习算子是深度学习模型的基本组成单位，其正确性、稳定性是深度神经网络模型正确性、稳定性的基础。面向深度学习算子稳定性分析，本文提出了基于差分测试的深度学习算子稳定性分析技术，主要模块包括算子与种子管理模块、种子队列管理模块、变异策略模块、差分测试模块。最终系统通过 Docker 技术一键安装运行在多个物理主机上。最后通过实验分析，本技术系统能够发现以 conv2d、avg_pooling、sigmoid、tanh、softmax 为主的算子差分异常，并且差分异常测试用例的生成呈现出收敛趋势，为深度学习算子稳定性分析提供了依据。

本文基于项目背景、项目国内外研究现状对本系统的来源、基础以及目标进行整体性概述，并给出本文主要工作目标。在相关技术概述章节给介绍本技术实现所需要的关键技术：本技术系统采用模块化设计思想，使用 Docker 技术将不同模块安装运行在不同物理主机，不同主机之间不同 TCP 进行通讯；算子测试任务使用线程池进行管理，并且用户可对算子测试任务优先级进行配置，高优先级测试任务会优先执行；种子队列管理模块基于差分测试结果定义种子优先级，化解算子测试难以进行路径覆盖测试的难题；使用 Django 框架爱进行模块化开发，模块化的设计极大减少系统开发难度。在系统需求分析与概要设计章节，给出了系统的功能性以及非功能性需求，对系统进行模块化划分，通过 4+1 视图对系统设计进行了不同视角的描述，最后给出数据持久化存储相应表结构的实现。

在详细设计与实现章节分别对算子与种子管理模块、种子队列管理模块、变异策略模块、差分测试模块进行具体描述与说明，每个模块介绍时首先通过流程示意图描述系统的流程逻辑，接着通过核心代码或者伪代码的形式描述模块的核心实现思路以及细节，最后通过界面图的形式展现相应模块的实现效果并进行相应描述说明。

在测试与实验分析章节，首先介绍了主要测试目标，并具体说明了系统的

软硬件测试环境。接着分别介绍功能性和非功能性测试，功能性测试包括用户登录控制、算子管理、种子管理、算子测试任务管理以及算子测试结果分析；非功能性测试由系统界面响应时间、算子测试任务执行时系统资源负载情况组成。最后对本技术从差分异常测试用例生成总数以及生成趋势的视角进行试验分析，实验分析结果表明，本技术生成一定数量能触发深度学习算子差分异常的测试用例，并且触发差分异常测试用例的生成趋势呈现出收敛效果，即本技术能够对深度学习算子进行稳定性分析。

6.2 展望

本文成功设计实现基于差分测试的深度学习算子稳定性分析技术，并部署到物理主机上对外提供服务，为深度学习算子准确性稳定性分析提供量化依据，能够从算子层面为深度神经网络模型预测结果可信度提供保障。虽然技术实现取得初步成果，但仍然存在诸多可以优化改进的内容和方向。

1）技术系统目前仅支持对单一算子进行测试，难以检测同一算子或不同算子组合所带来的误差，测试结果具有片面性，未来可实现对多个算子进行组合测试，进一步提高算子测试结果的全面性和准确性。

2）测试用例重复性判别效率较低，需要逐一比较才能判断新生成测试用例是否已存在种子集合中，之后工作可对测试用例进行聚类操作，进一步缩减测试用例逐一比对范围。

3）种子队列管理模块中种子优先级计算方式单一，未来可丰富种子优先级计算方式，进一步提高种子排序结果准确性。

4）数据持久化存储方式稳定性较低，系统使用用单数据库，当数据库读写并发较多时会使数据库响应时间较长甚至出现读写异常，未来数据库可采用数据库集群的设计方案，进一步提高数据库抗并发能力。

致　　谢

在论文即将完成之际，我想对在我开发工具及编写论文过程中提供指导和帮助的人表示由衷的感谢！

在研究生两年的学习生活中，我首先要感谢我的导师房春荣老师，感谢房老师两年来在我读研期间的精心指导以及生活上的帮助。研一进入 iSE 之初，房老师就给我安排了深度学习框架算子稳定性测试相关的工作。房老师不仅会不定期给我推荐相关学习资料，在我工作过程中遇到困难时也会耐心给我解答，让我都良师益友有了更深刻的理解。通过两年的学习让我对常见深度学习框架的特点，使用场景，不同深度学习框架之间的差别有了更近一步的了解；也积累了深度学习框架算子相关的实践经历。更为重要的一点，房老师非常注重软件工程思维和写作能力的培养，通过研究生阶段的学习，我的软件工程思维和实践能力均有所提高，写作思维也更加清晰。在这里再次感谢房老师的指导和帮助。在我编写论文期间，房老师更是不厌其烦的给我指导写作思路以及注意事项，使得我的论文能够按质按量完成，能我体会到编写论文的快乐。

同时我还要感谢在我读研阶段的三位室友。他们是我研究生生涯的不可或缺的一部分，在学习上互相帮助，相互分享选课心得，提醒课堂重要节点时间；在生活上互帮互助，有空在宿舍聊家常，让我在学校也感到像在家一样的温馨；在择业上，相互分享各自的实习心得，从各自的特点出发，推荐相关公司和分享相关的面试经验。我还要感谢 iSE 实验室 2019 级的同学们，我们在一起往往能激发青春的活力，碰撞出思维的火花，给人带来不一样的亲切感与快乐。感恩。

在这里我要特别感谢我的家人，父母在我读研期间不仅对我经济上提供支持，还时常与我视频沟通，分享日常的生活和工作，在我遇到困难时精神上对我予以鼓励。家人的所思所想永远是会我生活的一部分，他们永远是我的精神支柱、动力源泉。他们给我的爱与理解是任何人都无法代替的。

最后，我要对各位审阅老师、评审专家、相关工作人员表示衷心的感谢，感谢百忙之中对我的论文答辩工作的帮助和支持！

参考文献

[1] HINTON G E, OSINDERO S, TEH Y W. A Fast Learning Algorithm for Deep Belief Nets[J]. Neural Comput., 2006, 18(7): 1527 – 1554.

[2] HINTON G E, SALAKHUTDINOV R R. Reducing the Dimensionality of Data with Neural Networks[J]. Science, 2006, 313(5786): 504 – 507.

[3] RAUBER J, ZIMMERMANN R, BETHGE M, et al. Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX[J]. J. Open Source Softw., 2020, 5(53): 2607.

[4] YU F. Research on the next-generation deep learning framework[J]. Big Data Research, 2020, 6(4): 0.

[5] KUMAR A, MISRA R K, SINGH D, et al. Testing A Multi-Operator based Differential Evolution Algorithm on the 100-Digit Challenge for Single Objective Numerical Optimization[C] // IEEE Congress on Evolutionary Computation, CEC 2019, Wellington, New Zealand, June 10-13, 2019. [S.l.]: IEEE, 2019: 34 – 40.

[6] HUANG Y, SHU T, DING Z. A Learn-to-Rank Method for Model-Based Regression Test Case Prioritization[J]. IEEE Access, 2021, 9: 16365 – 16382.

[7] PEI K, CAO Y, YANG J, et al. DeepXplore: automated whitebox testing of deep learning systems[J]. Commun. ACM, 2019, 62(11): 137 – 145.

[8] CHEN Y, SU T, SUN C, et al. Coverage-directed differential testing of JVM implementations[J], 2016: 85 – 99.

[9] GUO J, JIANG Y, ZHAO Y, et al. DLFuzz: Differential Fuzzing Testing of Deep Learning Systems[J]. CoRR, 2018, abs/1808.09413.

[10] GOTTSCHALK S. Differential Equation Based Framework for Deep Reinforcement Learning[D]. [S.l.]: Kaiserslautern University of Technology, Germany, 2021.

[11] WANG P, ZHOU X, LU K, et al. The Progress, Challenges, and Perspectives of Directed Greybox Fuzzing[J], 2021.

[12] BI XIAOJUN L G, JING X. Dynamic Adaptive Differential Evolution Based on Novel Mutation Strategy[J]. Journal of Computer Research and Development, 2012, 49(6): 1288.

[13] 邹燕燕, 邹维, 尹嘉伟, et al. 变异策略感知的并行模糊测试研究 [J]. 信息安全学报, 2020, 5(5): 1–16.

[14] JIN X, YAN Z, YIN G, et al. An Adaptive Motion Planning Technique for On-Road Autonomous Driving[J]. IEEE Access, 2021, 9: 2655–2664.

[15] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet classification with deep convolutional neural networks[J]. Commun. ACM, 2017, 60(6): 84–90.

[16] HE K, ZHANG X, REN S, et al. Deep Residual Learning for Image Recognition[C] // 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. [S.l.]: IEEE Computer Society, 2016: 770–778.

[17] DEVLIN J, CHANG M, LEE K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[C] // BURSTEIN J, DORAN C, SOLORIO T. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers). [S.l.]: Association for Computational Linguistics, 2019: 4171–4186.

[18] BROWN T B, MANN B, RYDER N, et al. Language Models are Few-Shot Learners[C] // LAROCHELLE H, RANZATO M, HADSELL R, et al. Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. 2020.

[19] NOVIKOV A, IZMAILOV P, KHRULKOV V, et al. Tensor Train Decomposition on TensorFlow (T3F)[J]. J. Mach. Learn. Res., 2020, 21: 30:1–30:7.

[20] de G. MATTHEWS A G, van der WILK M, NICKSON T, et al. GPflow: A Gaussian Process Library using TensorFlow[J]. J. Mach. Learn. Res., 2017, 18 : 40:1 – 40:6.

[21] ABADI M. TensorFlow: learning functions at scale[C] // GARRIGUE J, KELLER G, SUMII E. Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016. [S.l.] : ACM, 2016 : 1.

[22] BAUDART G, HIRZEL M, MANDEL L. Deep Probabilistic Programming Languages: A Qualitative Study[J]. CoRR, 2018, abs/1804.06458.

[23] WANG J, LIU Y, HU Y, et al. FaceX-Zoo: A PyTorch Toolbox for Face Recognition[J]. CoRR, 2021, abs/2101.04407.

[24] LAWRENCE S, GILES C L, TSOI A C, et al. Face recognition: a convolutional neural-network approach[J]. IEEE Trans. Neural Networks, 1997, 8(1) : 98 – 113.

[25] 万士宁. 基于卷积神经网络的人脸识别研究与实现 [D]. [S.l.] : 电子科技大学, 2016.

[26] SUZUKI J, ISOZAKI H, MAEDA E. Convolution Kernels with Feature Selection for Natural Language Processing Tasks[C] // SCOTT D, DAELEMANS W, WALKER M A. Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 21-26 July, 2004, Barcelona, Spain. [S.l.] : ACL, 2004 : 119 – 126.

[27] 何鹏程. 改进的卷积神经网络模型及其应用研究 [D]. [S.l.] : 大连理工大学, 2015.

[28] 田娟, 李英祥, 李彤岩. 激活函数在卷积神经网络中的对比研究 [J]. 计算机系统应用, , v.27(7) : 45 – 51.

[29] GUPTA S, AGRAWAL A, GOPALAKRISHNAN K, et al. Deep Learning with Limited Numerical Precision[J]. CoRR, 2015, abs/1502.02551.

[30] ZHANG C, BENGIO S, HARDT M, et al. Understanding deep learning requires rethinking generalization[C] // 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. [S.l.] : OpenReview.net, 2017.

[31] YU D, WANG H, CHEN P, et al. Mixed Pooling for Convolutional Neural Networks[C] // MIAO D, PEDRYCZ W, SLEZAK D, et al. Lecture Notes in Computer Science, Vol 8818 : Rough Sets and Knowledge Technology - 9th International Conference, RSKT 2014, Shanghai, China, October 24-26, 2014, Proceedings. [S.l.] : Springer, 2014 : 364 – 375.

[32] LAWRENCE J, MALMSTEN J, RYBKA A, et al. Comparing TensorFlow Deep Learning Performance Using CPUs, GPUs, Local PCs and Cloud[J], 2017.

[33] LIMA I, SILVA J, MIRANDA B, et al. Exposing bugs in JavaScript engines through test transplantation and differential testing[J]. Softw. Qual. J., 2021, 29(1) : 129 – 158.

[34] MCKEEMAN W M. Differential Testing for Software[J]. Digit. Tech. J., 1998, 10(1) : 100 – 107.

[35] EVANS R B, SAVOIA A. Differential testing: a new approach to change detection[C] // CRNKOVIC I, BERTOLINO A. Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007. [S.l.] : ACM, 2007 : 549 – 552.

[36] WISE S L, SOLAND J, BO Y. The (Non)Impact of Differential Test Taker Engagement on Aggregated Scores[J]. International Journal of Testing, 2020, 20(1) : 57 – 77.

[37] CHRISTIE M, MARRU S, ABEYSINGHE E, et al. An extensible Django-based web portal for Apache Airavata[C] // JACOBS G A, STEWART C A. PEARC '20: Practice and Experience in Advanced Research Computing, Portland, OR, USA, July 27-31, 2020. [S.l.] : ACM, 2020 : 160 – 167.

[38] GIANNOPOULOS L, DEGKLERI E, TSANAKAS P, et al. Pythia: Identifying Dangerous Data-flows in Django-based Applications[J], 2019 : 5:1 – 5:6.

[39] STEINHAUSER A, TUMA P. DjangoChecker: Applying Extended Taint Tracking and Server Side Parsing for Detection of Context-Sensitive XSS Flaws[J]. CoRR, 2020, abs/2005.06990.

[40] ZEROUALI A, MENS T, DECAN A, et al. A multi-dimensional analysis of technical lag in Debian-based Docker images[J]. Empir. Softw. Eng., 2021, 26(2) : 19.

[41] HENKEL J, SILVA D, TEIXEIRA L, et al. Shipwright: A Human-in-the-Loop System for Dockerfile Repair[J]. CoRR, 2021, abs/2103.02591.

[42] KWON S, LEE J. DIVDS: Docker Image Vulnerability Diagnostic System[J]. IEEE Access, 2020, 8 : 42666 – 42673.

[43] KRUCHTEN P. Architectural Blueprints: The 4+1 View Model of Software Architecture[J]. CoRR, 2020, abs/2006.04975.

简历与科研成果

基本信息

吕军，男，汉族，1995 年 6 月出生，江西省九江人。

教育背景

2019 年 9 月 — 2021 年 7 月 南京大学软件学院 硕士

2015 年 9 月 — 2019 年 7 月 天津大学软件学院 本科

攻读工程硕士学位期间参与的科研课题

1. 国家自然科学基金面上项目 “问题研究”
2. 房春荣，曹可凡，**吕军**，章许帆，沪胜浩，顾逸飞，“一种针对深度学习算子运算精度的差分模糊测试工具”，申请号：2020104872014，已受理。
3. 房春荣，曹可凡，**吕军**，章许帆，顾逸飞，沪胜浩，“一种基于 Fuzz 的人工神经网络测试方法”，申请号：2020104871647，已受理。
4. 房春荣，曹可凡，顾逸飞，章许帆，沪胜浩，**吕军**，“一种基于模糊测试的深度学习算子测试工具”，申请号：2020104871651，已受理。