南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论　文　题　目　基于规则变异的深度学习框架自动化测试系统

作　者　姓　名　何云

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

2022 年 **5** 月 **23** 日

学　　号：**MF20320062**

论文答辩日期：**2022** 年 **05** 月 **20** 日

指 导 教 师：　　（签字）

Automated Testing System of Deep Learning Framework Based on Rule Mutation

by

Yun He

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 23, 2022

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：基于规则变异的深度学习框架自动化测试系统

工程硕士（软件工程领域）专业 2020 级硕士生姓名：何云

指导教师（姓名、职称）：陈振宇 教授，房春荣 助理研究员

摘　要

深度学习技术发展迅速，在学术界和工业界得到了广泛的应用。深度学习框架作为构建深度神经网络模型的基础软件，将封装完成的接口提供给开发者调用。现有工作已经证实框架中存在缺陷，其一旦引入模型可能会造成严重的后果。因此本文设计并实现了一个基于规则变异的深度学习框架自动化测试系统，旨在通过语法规则生成深度神经网络模型，并在生成后对其采用多种变异方法进行变异，将其作为测试数据实现深度学习框架的自动化测试和缺陷分析，以发现更多深度学习框架中存在的潜在缺陷。

本论文的主要工作是搭建了一个深度学习框架的自动化测试平台，将整个系统划分为环境管理模块、生成任务管理模块、变异任务管理模块、缺陷分析模块和编译器测试模块。在本文中，环境管理模块负责创建和切换虚拟环境，确保模型的生成和变异任务能够顺利执行；生成任务管理模块负责迭代生成不同深度学习框架下的深度神经网络模型结构，其生成过程依赖于构建好的语法规则，并在生成时进行安全检查，确保生成模型的动态可用性；变异任务管理模块将生成的深度神经网络模型作为种子模型进行迭代变异，在每一次迭代中选择最优变异模型作为下一次迭代的种子模型，以尽可能多地触发深度学习框架中的缺陷；缺陷分析模块对具体生成任务中产生的日志文件进行自动分析，最终提供缺陷分析结果；编译器测试模块将会随机生成大量用于测试深度学习编译器的程序，以此保障深度神经网络模型的安全部署和运行。

该系统最终构建为 Web 应用，用户可以通过浏览器的方式访问系统，并借助本系统实现模型的生成、变异和编译器测试，最终对任务执行时产生的中间数据和日志文件进行分析，自动化实现缺陷分类。该系统能够提高开发和测试深度学习框架的效率和质量，具有较高的实用性。

关键词： 深度学习框架；深度神经网络模型；规则变异；自动化测试

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Automated Testing System of Deep Learning Framework Based on Rule Mutation

SPECIALIZATION: Software Engineering

POSTGRADUATE: Yun He

MENTOR: Professor Zhenyu Chen, Assistant Researcher Chunrong Fang

Abstract

Deep learning technology has developed rapidly. It has been widely used in academia and industry. As the basic software for building deep neural network models, the deep learning framework provides the encapsulated interface for developers to call. Existing work has confirmed that there are defects in the framework. Once the defects are introduced into the model, they may cause serious consequences. Especially in security areas. Therefore, this paper designs and implements a deep learning framework automated testing system based on rule mutation, which aims to generate a large number of deep neural network models automatically by using the mutation method after generation through grammar rules, and implement the deep learning framework as test data. Batch testing and automated bug analysis to discover more potential bugs in deep learning frameworks.

The main work of this paper is to build an automated testing platform for deep learning framework. It completes the generation, training, reasoning, mutation, and analysis of deep neural network models. This paper first introduces the research status, including deep learning framework testing and deep neural network model generation. Then, the technology involved in this system is expounded in detail. The whole system is divided into the environment management module, generation task management module, mutation task management module, defect analysis module, and compiler testing module. In this paper, the environment management module is responsible for creating and switching virtual environments. Each virtual environment maintains its deep learning framework to ensure that the model generation and mutation tasks can be executed smoothly. The generation task management module is responsible for iteratively

generating different deep neural network models. The generation process depends on the constructed grammar rules. Security checks are carried out during generation to ensure the dynamic availability of the generated model. The mutation task management module uses the deep neural network model trained in the generation task as the seed model. According to the mutation parameter mutation model set by the user, the optimal mutation model is selected as the seed model of the next iteration. It aims to trigger as many defects in the deep learning framework as possible in each iteration. The defect analysis module automatically analyzes the cause of the defect to generate a log report. The compiler test module will randomly generate a large number of programs for testing the deep learning compiler. And it will compare the output of the program execution under different versions. The results are used to locate defects to ensure the safe deployment and operation of deep neural network models.

The system is finally constructed as a web application. Users can access the system through a browser. They can build a test environment, and use this system to realize the automatic iterative generation of models, model mutation, and compiler testing. Finally, they will get an analysis report about defect classification. The system can improve the efficiency and quality of development to test deep learning frameworks. It has high practicability.

keywords: Deep learning framework, deep neural networks, rule mutation, automated testing

目　　录

插图清单

附表清单

第一章　引言

1.1　选题的背景和意义

伴随着计算机运算能力的快速发展，深度学习技术正逐渐成为工业界和学术界的研究重点。2006 年 Geoffrey Hinton 等人 [1] 对深度学习领域进行了研究，并第一次发表关于神经网络的论文，标志着深度神经网络时代的到来。

目前，深度学习技术在工业界和学术界得到了广泛应用，如人脸识别 [2]、自动驾驶 [3]、语音识别 [4] 等领域，并取得了不错的成绩。除此之外，深度学习技术深入到了安全攸关领域，包括航空航天、军事活动等。但是，由于深度学习技术自身的脆弱性，深度学习系统中存在的缺陷可能会引发一系列安全事故。例如，2018 年在美国某个城市发生的一起关于自动驾驶的事故：一辆使用自动驾驶模式的汽车由于没有正确识别横穿马路的行人，致使行人被撞身亡。因此，为了确保深度学习技术能够得到安全应用，越来越多的研究者认为有必要对深度学习系统进行全面的测试，从而提高其质量。

深度学习框架在深度学习系统的广泛应用中发挥着至关重要的作用。深度学习框架封装了模型组件的实现和层，并以接口的形式暴露给开发者。深度学习框架的出现降低了深度学习系统的开发难度，让开发者可以将更多的精力投入到优化系统的逻辑结构上。作为基础软件，深度学习框架加速和简化了深度学习系统的开发和部署。

现有的工作已经发现，由于深度神经网络模型的构建基于深度学习框架，框架本身存在的缺陷会被引入到模型中（如断言错误、形状不一致等），因此需要关注深度学习框架中存在的缺陷。与传统的软件测试不同，在对深度学习框架进行测试时，需要考虑框架的构成。深度学习框架自顶向下包含三个部分，分别为库、算子和编译器。这三个部分中任何一个存在缺陷均会降低深度学习框架的质量，因此对其进行全面的测试可以有效避免缺陷被引入到深度学习系统中。

深度神经网络模型作为深度学习框架的测试数据，其触发缺陷的可能性会随着测试数据的不同而不同。因此，希望进一步扩大触发深度学习框架中缺陷

的可能性，尽可能多地生成能够触发深度学习框架缺陷的测试数据，从而开展对深度学习框架的有效测试。

现阶段，对于深度学习框架的测试研究存在两方面的问题:

一方面，目前的主要研究仅针对少量现有的深度神经网络模型进行变异，并将其作为测试输入进行测试。该测试方法容易受到模型数量和架构的约束，其测试结果具有一定的局限性。

另一方面，目前的主要研究只关注了深度学习框架中顶层库的部分，极少关注编译器部分。然而编译器作为框架的基础，支撑了深度神经网络模型的运行。

因此，本文对基于规则变异的深度学习框架自动化测试开展研究，旨在自动化生成深度神经网络模型，并将其作为测试输入进行框架测试。通过覆盖深度学习框架测试的全流程，实现自动化地环境搭建与切换、测试数据的生成与推理、缺陷的分析与报告生成，最终以工程化的形式构建高效率的深度学习框架自动化测试系统。

1.2 国内外研究现状及分析

深度神经网络模型的构建依赖于深度学习框架，输入训练数据通过主动学习生成模型 [5]。开发者无需了解算子实现的具体细节 [6]，直接调用深度学习框架提供的算子接口即可构建深度神经网络模型 [7]。随着深度学习技术在学术界和工业界的广泛应用 [8]，深度学习框架作为构建深度神经网络模型的重要组成部分，对深度学习框架的测试提出了更高的要求。

目前对于深度学习框架的测试主要集中在两方面，一方面是验证框架本身是否存在缺陷，另一方面是如何采用有效的方法提高触发缺陷的可能性。

首先，针对深度学习框架本身是否存在缺陷，国内外已经开展了一系列研究。Pham 等人 [9] 研究通过输入深度神经网络模型来切换深度学习框架，并采用差分测试方法触发缺陷。该方法被命名为 CRADLE，能够比较预期输出，并对缺陷进行筛选，以此捕获缺陷。Guo 等人 [10] 对深度神经网络模型的开发和部署进行了实证研究，发现模型性能与深度学习框架的差异密切相关。Zhang 等人 [11] 基于 Tensorflow 研究了深度学习框架中存在的缺陷。Nargiz 等人 [12] 对三个框架进行了研究，并分类总结了其中存在的缺陷。

其次，在如何采用有效方法提高触发缺陷可能性方面，目前已有研究者深

入研究了深度神经网络模型的生成。通过开展通过开展差分测试，对已有的模型进行变异操作。Ma 等人 [13] 提出了 DeepMutation，基于已经训练好的模型，通过设计蜕变关系，添加或删除原模型中的某些层，以生成新的模型。

Wang 等人 [14] 进一步提出了通用性更强的蜕变关系。Guo 等人 [15] 针对 25 个常用接口，分析其参数范围，基于已有模型的源码，随机生成结构相似、参数不同的模型。

Wang 等人 [16] 提出了一种启发式模型生成算法来引导模型生成，进而得到有利于放大框架缺陷的模型。Liu 等人 [17] 针对 Tensorflow 框架中的缺陷进行了初步的探索，总结了其中的缺陷类型。

虽然上述工作在深度学习框架测试方面已经有了较大突破，但仍然存在以下三方面的问题：

第一，深度学习框架的测试需要数量庞大的深度神经网络模型。在进行传统软件测试时，测试人员需要考虑不同的程序执行路径，并针对不同情况和边界条件设计测试用例。相应地，在对深度学习框架进行测试时，同样需要大量深度神经网络模型。然而，深度神经网络模型需要复杂的构建和训练等过程，而这一过程极其耗时。此外，由于需要尽可能多地触发深度学习框架中的缺陷，深度神经网络模型作为测试输入应尽可能覆盖深度学习框架的调用接口。因此，如何生成数量庞大和覆盖面广的深度神经网络模型是一个研究重点。

第二，分析深度神经网络模型的非预期行为的产生原因也是一大挑战。不同于传统软件测试，深度学习系统在训练和使用时具有随机性。随机性一方面表现在，开发人员在运行深度学习系统时，无法判断系统的运行情况；另一方面，由于深度学习系统涉及各种运算，在运行时难免出现误差等问题。上述问题提高了深度学习框架的测试难度。

第三，上述研究仍然无法实现深度学习框架的自动化测试，其测试数据的生成和框架缺陷的批量分析仍然无法满足深度学习框架测试的需求，因此本文对其进行了优化，从而提高框架测试的效率和质量。

综上所述，如何自动化地搭建与切换环境、生成深度神经网络模型作为测试数据、框架缺陷批量分析等诸多问题亟待解决。本文将结合神经网络结构自动搜索，研究生成深度神经网络模型和捕获框架缺陷的方法，针对深度学习框架测试，构建基于规则变异的深度学习框架自动化测试系统，以提高测试效率和质量。

1.3 本文的工作

本文旨在实现深度学习框架的自动化测试，提高框架的质量，从而保证深度学习智能软件的可靠性。将深度神经网络模型引入深度学习智能软件可以帮助人类做出决策或判断，然而深度学习框架是构建深度神经网络模型的重要组成部分，框架中隐藏的缺陷很可能被引入到深度神经网络模型中，造成一系列不可预测的后果。

因此，为了保证深度学习智能软件的安全运行，需要对深度学习框架进行全面测试，从而发现其中隐藏的缺陷。然而，由于缺乏大量深度神经网络模型作为测试数据，可能会导致深度学习框架的测试不充分，难以发掘尽可能多的缺陷。此外，由于深度学习框架的测试环境难以快速搭建，测试和缺陷分析过程繁琐，对深度学习框架的测试也提出了很大的挑战。

综上所述，本文旨在构建一套高效可用的深度学习框架自动化测试系统。本文支持创建和切换环境，不同环境下维护了不同的深度学习框架及版本，从而便于快速测试深度学习框架。同时，本文遵循编程语言的上下文无关语法，将其运用到深度学习框架的测试数据生成中。为了能够充分利用生成的测试数据，从而尽可能多地触发深度学习框架中的缺陷，即提高对于深度学习框架的测试覆盖率，需要对生成的测试数据进行变异。在每一次迭代过程中，计算不同数据下变异算子触发缺陷的概率，并与上一次迭代的概率进行比较，采用启发式的缺陷触发算法使得测试效率和质量更高。此外，本文在自动化生成和变异测试数据之后，支持归纳总结任务执行时产生的中间数据和日志信息，将分析结果反馈给开发人员。

本文具体工作如下：首先，将介绍深度学习技术以及目前流行的深度学习框架，并阐述深度神经网络模型生成技术和框架测试技术。然后，对系统进行需求分析和设计，将整个系统分为五个模块，涵盖了环境创建、模型生成和变异、训练和推理等主要步骤。同时考虑到需要尽可能多地触发框架中存在的缺陷，采用启发式的缺陷触发算法，生成能够放大框架缺陷的模型，从而提高测试数据的质量。接下来，将具体阐述系统的详细设计与实现过程。最后，对系统进行测试，确保基于规则变异的深度学习框架自动化测试系统能够高效且稳定的运行。

1.4 本文的组织结构

深度学习框架能够作为基础软件构建深度神经网络模型，从而帮助深度神经网络在不同的领域中得到应用，并做出重要决策。然而，现阶段缺乏自动化测试系统对深度学习框架进行高效、全方面的测试。

本文将会设计一款规则变异的方案，从而生成架构不一、数量庞大的深度神经网络模型，并基于该方案实现了一个基于规则变异的深度学习框架自动化测试系统。

本文组织结构如下：

第一章引言，阐述了深度学习框架自动化测试系统的项目背景及意义，并介绍了国内外目前的研究现状、相关技术发展的主要研究现状，同时说明了本文的主要研究目标以及为了实现基于规则变异的深度学习框架自动化测试系统所需要做的主要工作。

第二章技术综述，介绍本系统在实现过程中所涉及的深度学习技术与开源库和框架，包括深度学习技术、深度神经网络模型生成技术、深度学习框架测试技术和 Django 技术等。

第三章系统需求分析与概要设计，提出了基于规则变异的深度学习框架自动化测试系统在实现过程中所需要满足的基本需求，并从功能的角度将系统划分为了五个模块，从而实现对系统多层面的设计。

第四章基于规则变异的深度神经网络模型生成技术，重点阐述其实现的整体架构。同时从该技术的两个重要部分出发，介绍其实现流程。

第五章系统详细设计及实现，将在需求分析的基础上，重点阐述基于规则变异的深度学习框架自动化测试系统中，每一个功能模块的详细实现方法以及实现过程。

第六章系统测试与分析，基于第三章、第四章中的需求分析和详细设计及实现，对本系统的功能需求和非功能需求设计测试用例并验证，确保系统的完整性、可靠性和有效性。

第七章总结与展望，总结了基于规则变异的深度学习框架自动化测试系统的设计与开发，并对其进行总结，指出值得改进的方向。

第二章　技术综述

2.1　深度学习

人工智能是计算机科学中的一个重要分支，可以模拟人类意识和思维，并进行信息处理。机器学习技术作为人工智能的研究热点，主要研究怎样使用计算机来模拟人类学习活动，以获取新的知识或技能，重新组织已有的知识结构使之不断提升自身的性能。

而深度学习是机器学习技术衍生的一个研究方向，随着大数据、云计算的发展，深度学习逐渐被广泛应用于解决工程应用和科学领域的复杂问题。在过去的几十年里，深度学习在生活中的应用随处可见，并取得了许多卓越的成就，如自然语言处理、语音识别、自动驾驶等。

深度学习由复杂的神经网络构成，基于图形处理器 GPU，通过大规模数据训练不断学习数据中存在的内在规律和表示层次，获得深层次的特征表示，从而做出决策或预测。

2.2　深度学习框架

深度学习从学术研究到应用的过程中会涉及不同的工具和步骤，由于环境配置、调参、迭代改进及测试等流程，可能会使工作效率降低，并且耗时延长。为了减少耗时并提高开发效率，工业界和学术界开发并完善了不同类型的基础且通用的深度学习框架，其中主流的框架包括 Tensorflow[18]、CNTK[19]、Torch[20]、Theano[21]、Caffe[22] 等。这些框架的出现降低了开发者们开发的难度，并且可以利用深度学习框架搭建网络或是对已有网络结构进行改进，从而应用于不同领域中。

然而由于架构设计和计算方式的不同，它们需要通过不同的编程语言来进一步实现，并且即使采用相同的深度学习训练算法 [23]，基于相同的计算范式也可能导致实现方式不同 [24]。因此，在训练数据、模型设计和训练配置相同的情况下，依然可能得出不同的训练性能 [25]，由此做出不同的决策。

由于框架的抽象级别不同，可以将其分为底层深度学习编译器和高层深度学习库。基于底层深度学习编译器和高层深度学习库，深度学习框架得以完善，使得深度神经网络模型能够快速地构建、部署和运行。

2.2.1 深度学习编译器

由于在不同的硬件上部署深度神经网络模型较为困难，因此对于深度学习编译器的研究逐渐成为热点，其出现在一定程度上缓解了部署困难的问题。目前，工业界和学术界已经提出了几个深度学习编译器，如 Tensorflow XLA 和 TVM[26]。深度学习编译器将不同深度学习框架下构建的深度神经网络模型作为输入，对其进行优化从而得到输出。

然而，由于编译器作为深度学习框架的底层，支撑了模型的正常运行，一旦编译器存在缺陷，并引入深度神经网络模型中，将会造成不可估量的后果。因此，本文将对深度学习编译器进行测试，从而更加精准和快速地定位缺陷。

2.2.2 深度学习库

深度学习库可以分为高层深度学习库和低层深度学习库。

低层深度学习库指的是深度学习模型的第一层抽象。开发者们可以通过框架所提供的接口来编写代码，从而构建深度神经网络模型，这大大减少了研发的耗时。由于灵活性、可移植性、多语言支持等特性，使得框架被广泛使用，TensorFlow 就属于这一类框架。

TensorFlow 是一个开源的基于数据流的深度学习框架 [18]，可以部署在服务器、普通个人电脑、移动设备，甚至云端设备上。同时，它支持 CUDA 高性能数值运算，因此在深度学习社区中备受关注和广泛使用。基于该框架，开发者们可以使用多种编程语言，轻松设计深度神经网络模型 [27]。

而高层深度学习库指的是封装低层的具体运算操作，通过高层接口（API）的方式方便用户使用，以此提高了可扩展性和易用性，方便开发者设计和训练模型。目前，广泛流行的高层深度学习库是 Keras[28]。

Keras 是由 Python 编程语言编写的深度学习框架，能够在 Tensorflow、Theano 等低层深度学习框架上运行。依靠这些低层深度学习库所提供的的底层运算能力的支持，可以方便开发者快速构建深度神经网络。通常情况下，Keras 并不算是广义的深度学习框架，而是以接口形式存在。开发者在开发深度神经

网络模型时，无论使用何种方式进行计算，都可以更加轻松地完成工作。

2.3 深度神经网络模型生成技术

深度神经网络由多层组成，一般包括输入层、隐藏层、输出层，每一层又由大量神经元组成。这些神经元通过链接连接，并在训练过程中，根据不同的输入数据进行特定的卷积和池化等特定权重的转换，最终得到相应的权重。尤其在某个深度神经网络模型中，尽管同一层可以被反复使用，但由于链路上权重控制的不同，同一类型的层的性能也有可能不同。

目前，常见的神经网络类型有循环神经网络 [29]、生成对抗网络 [30]、卷积神经网络 [31]、深度强化学习 [32]。通过大量训练数据特征的学习，深度神经网络模型在训练后可用于预测或做出决策。

2.3.1 模型变异

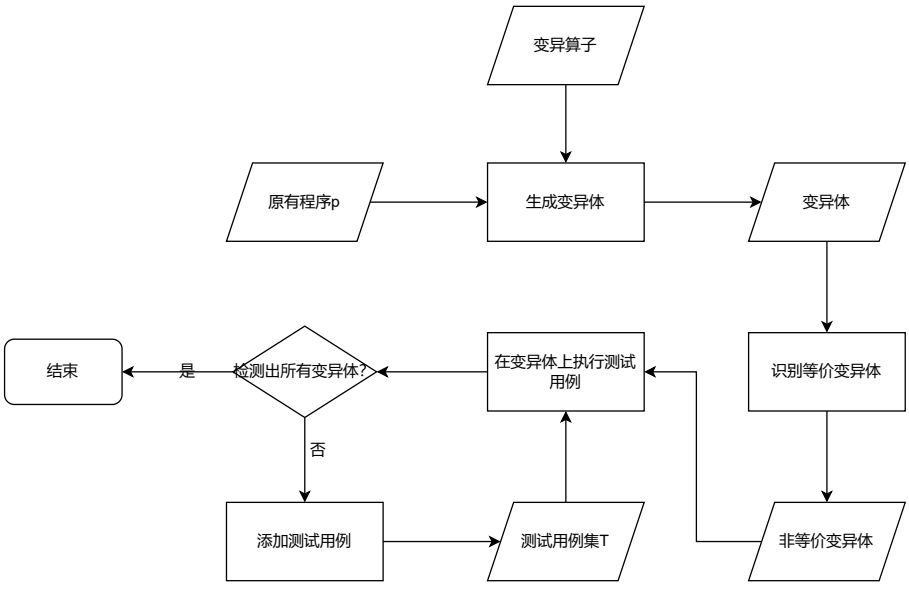

图 2-1: 变异测试流程图

变异测试，也称作变异分析，可以基于错误进行检测，并且提供变异分数来测试充分性。变异分数由变异体的总数和被杀死的变异体数量构成 [33]，将经过变异算子调整的程序或模型看作变异体。如果变异体的输出与原始程序的

不同，则变异体会被测试杀死 [34]。这种测试方式旨在找出有效的测试用例，发现程序或模型中存在的缺陷。变异测试的流程图可以用图 2-1表示。

通过引入变异测试，检测潜在的变异方式，动态调整深度神经网络，这种方法可以生成大量深度神经网络模型 [13]。通过模型变异的方式，可以极大程度地解决测试用例匮乏的问题，并改变固有的深度神经网络模型架构 [35]，生成更多样化的测试输入，尽可能充分地测试深度学习框架，触发其中存在的缺陷。

在生成深度神经网络模型的过程中，本文将针对深度神经网络中的参数、网络层等存在的蜕变关系，拟制定相应的变异方法，包括参数变异、架构变异等，以此生成大量有效的深度神经网络模型。如图 2-2所示为变异测试前后的模型内部结构变化，通过对其进行架构变异，产生新的深度神经网络结构。

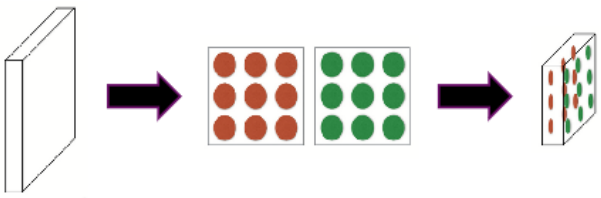

图 2-2: 变异测试前后模型内部结构

2.3.2 神经架构搜索

神经架构搜索（Neural Architecture Search，简称 NAS）是一种自动设计神经网络的技术，通过搜索可以产生大量深度神经网络模型 [36]。尽管开发者们已经人工设计了一些性能表现良好的深度神经网络架构，但是由于黑盒性质的设计原理，如果想要对原有网络架构进行优化，则需要大量专业理论，并对其进行反复实验，这种方式往往代价极高，也无法得到最佳的网络设置。因此，神经架构搜索的出现在一定程度上解决了这一问题，它将神经架构视为超参数，并对其进行优化，从而快速地自动生成最优网络。其实现过程如 2-3所示。

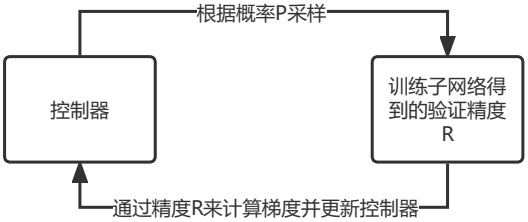

图 2-3: 神经架构搜索概述

如图2-4所示，NAS 主要由三个部分构成 [37]，分别是架构搜索空间、架构搜索方法和评估方法。根据给定的一个架构搜索空间，也就是定义了一组可以选择的候选网络架构集合，通过架构搜索方法在该集合中随机选取子网络，对该子网络进行性能评估，并将其反馈给下一轮搜索，最终训练得到性能表现良好的最优网络架构。

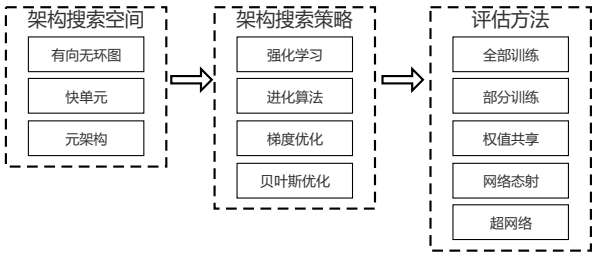

图 2-4: NAS 结构

其中，由于在架构搜索空间中定义了网络的深度、操作类型和相关超参数，因此最优网络架构的质量和搜索的效率由架构搜索空间的设计所决定。一旦架构搜索空间过大，则会降低时间效率和空间效率，反之如果过小又会影响神经架构的复杂度，导致其性能较差。所以，架构搜索空间设计的好坏至关重要。

基于架构搜索空间，架构搜索方法则定义了如何探索搜索空间，它将每次搜索到的网络架构与上一次搜索到的网络架构的相关参数进行共享，并进一步进行优化，从而得到最优模型参数进行性能评估，极大提高了架构搜索的效率。

神经架构搜索虽然在一些任务上表现突出，但是依然无法摆脱超参数的选择。此外，由于其迭代步骤繁琐以及性能评估效率较低，神经架构搜索往往存在效率低下的问题，因此难以在短时间内生成数量多且性能好的模型用于深度学习框架测试。

2.4 深度学习框架测试技术

目前，业内的研究人员通常使用深度学习框架的高级和低级库。开发者们可以使用高级库提供的接口来调用低级库所实现的深度学习算法来编写程序。尽管低级库提供了不同的编写方式和接口，但是通过高级库提供的接口，可以屏蔽低级库之间的的差异，保证一致性。

即使算法相同，不同的低级库也可能提供不同的实现，因此开发者们需要通过调用高级库来解决不一致的问题。目前，比较受欢迎的高级库有 Keras[28]，它可以运行于 TensorFlow、CNTK、Theano 和 MXNet[38] 四个低级库之上。由于这些库广泛使用，因此需要对其进行测试。

2.4.1 编译器测试技术

目前，业内的研究人员已经对编译器测试进行了研究，以此保证编译器的质量。Chen 等对其进行了实验性研究，比较了三种编译器测试技术的性能好坏，分别是随机差异测试 [39]、差异优化级别测试 [40]、等价模输入方法 [41]。下面对其进行简单介绍：

- 随机差异测试。该测试方法通过比较多个编译器来发现其中存在的缺陷。通常情况下，如果输入相同的测试用例可以得到一致的输出。反之，则可以判断所使用的编译器中的一个或多个可能存在缺陷。
- 差异优化级别测试。该测试方法不同于第一种的是可以设置不同优化级别参数来测试编译器。通常情况下，对同一个编译器进行测试，如果测试用例相同，使用不同的优化级别参数，应当得到一致的输出。反之，该编译器可能存在缺陷。
- 等价模输入方法。该测试方法会基于源测试用例生成等价测试用例用于测试，通过比较两个测试用例输入后的测试结果来判断编译器是否存在缺陷。

编译器测试的重点在于生成大量测试用例对其进行测试，即包含类型定义、类型转换、运算符操作等组成的结构和内容不同的小程序 [42]。通过大量生成的小程序基于上述提出的测试方法，可以最大程度地发现编译器中存在的缺陷，满足测试编译器的需要。

2.4.2 深度学习库测试技术

深度神经网络训练时，算子执行基础运算操作，具有原子性。同时算子作为相对独立的单位存在于深度神经网络中，其内部逻辑不会随训练的进行而改变。由于深度神经网络模型会进行大量卷积浮点运算，因此算子的准确性决定了模型预测结果的置信度。

深度学习框架中存在必要的算子，并且算子的实现和组织方式会由于框架的不同而不同，因此算子的多样性和差异性是深度学习库测试的挑战。目前对深度学习库进行测试时可以针对单个算子进行测试，也可以对算子进行组合测试 [43]，通过对运算符组合 [44]，最大程度地触发深度学习框架中存在的缺陷，满足测试深度学习框架的需要。

2.5 Vue

Vue.js 技术是一个轻量级的前端框架，为用户提供了多种指令和 API 等，能够高效快捷地搭建前端页面 [45]。同时，Vue.js 性能优越，可以快速渲染数据 [46]。该框架为数据渲染提供了简单的模板，降低了开发和维护的难度。

Vue 通过 view model 可以实现数据的双向绑定和监听 [47]，DOM 可以在 view model 层将数据传递到数据库，而数据可以在 view model 层通过数据双向绑定将数据从数据库传递到 DOM，从而实现 DOM 渲染。其渲染流程如图 2–5所示。

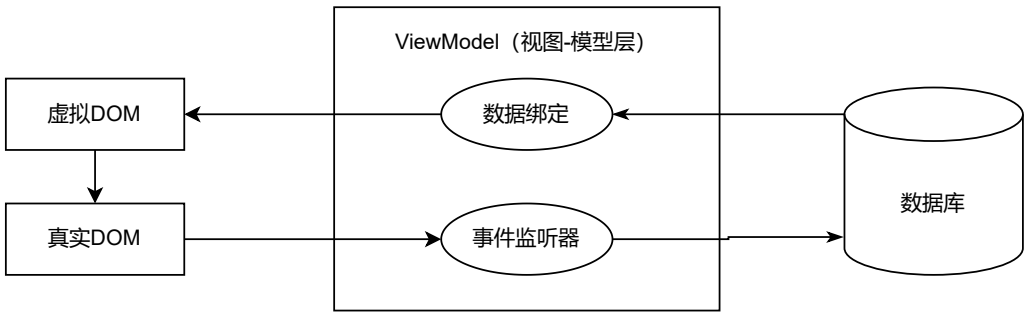

图 2–5: 渲染流程

2.6 Django

Django 是一个基于 Python 编程语言开发的 Web 开源框架。开发初期，Django 被用于开发劳伦斯集团下的新闻类内容管理系统 [48]。与其它 Web 框架相比，Django 的设计遵循了敏捷开发的思想，是一种大而全、可重用的框架，帮助开发者们快速开发 Web 网站。

目前，Django 采用的设计模式是分层设计模式 [49]，即 MTV。其中，M 表示 Model 层，主要的任务是操纵数据库；T 表示 Template 层，主要的任务是与前端进行交互；V 表示 View 层，主要的任务是处理业务逻辑。此外，如果开发者们需要添加新的功能，Django 也提供了大量内置函数，帮助开发者们快速开发。

2.7 本章小结

本章主要介绍了技术实现时所涉及的方法和工具，并对其进行了具体阐述。首先介绍了深度学习，并进一步阐述深度学习框架的基础知识，对其类别进行了阐述。然后介绍了生成深度神经网络模型的两大主流技术，分别是模型变异和神经架构搜索。最后介绍了深度学习框架测试所涉及的相关技术，通过借鉴编译器测试技术和深度学习库测试技术，进而分析深度学习框架测试的可行性。本章为本文提供了深厚的技术和理论基础。

第三章　深度学习框架自动化测试系统需求分析与设计

3.1　系统总体规划

本系统主要针对的是需要测试和开发深度学习框架的人员，如图3-1所示的是深度学习框架自动化测试系统流程示意图。该系统主要涉及五个层次上的工作，这些层自底向上分别是基础层、编译器层、框架层、规则层和应用层。

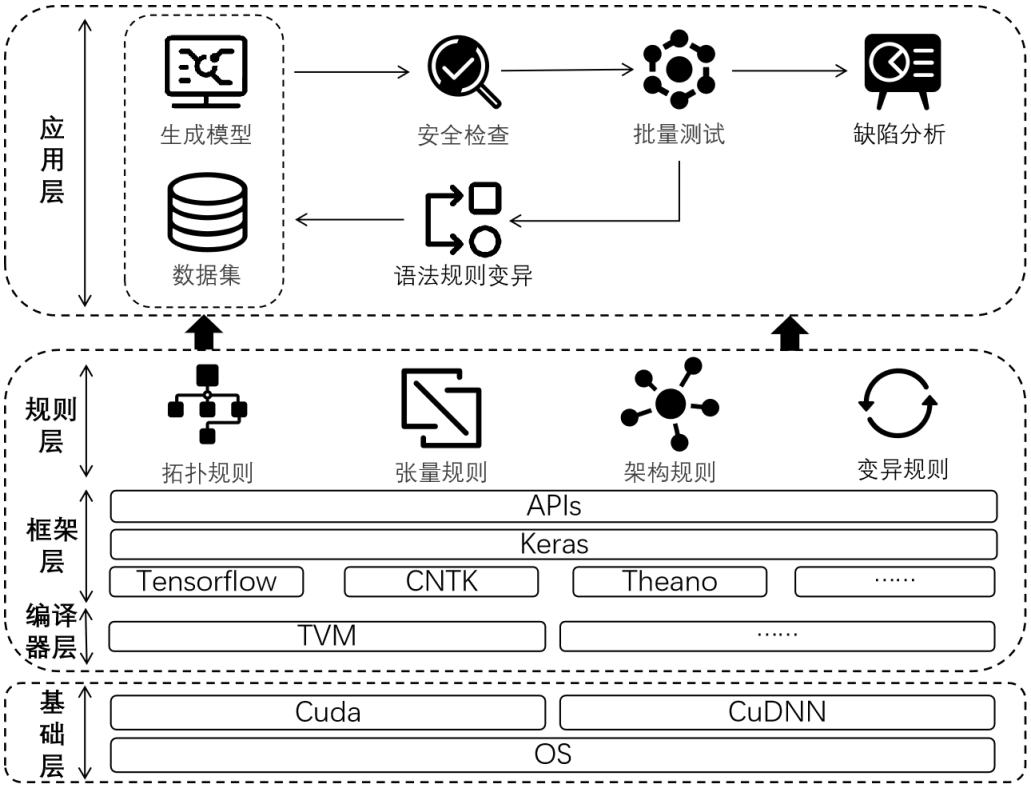

图 3-1: 深度学习框架自动化测试系统流程示意图

基础层主要进行计算机底层运算，它探讨如何使得计算机高效运算。基于框架层，基础层将框架层实现的抽象算法进一步转换为最低级的计算的组合，

从而提高计算效率，为算法奠定基础。

编译器层主要实现深度神经网络模型的部署。基于深度学习编译器，可以支撑深度神经网络的正常运行，从而提高其部署的效率和质量，为框架的运用奠定基础。

框架层中包含了多个深度学习框架，包括高层深度学习框架和低层深度学习框架。这些框架屏蔽了使用时存在的差异，开发者们只需通过调用深度学习框架所提供的的统一接口来使用基础算子，不仅提高了开发效率，也降低了开发和使用的难度。在该系统中，会对主流的深度学习框架进行测试，以发现其中存在的缺陷。

规则层定义了拓扑规则、张量规则、架构规则和变异规则。拓扑规则定义了层与层之间的连接方式，张量规则定义了每个层的相关参数，架构规则定义了深度神经网络模型的结构。基于这些规则，可以随机生成深度神经网络，以尽可能多地生成合理、有效的网络，保证测试顺利进行。

基于以上四个层，本系统的主要工作集中在应用层。在这一层中，开发人员首先配置生成模型的控制参数，将深度神经网络模型的基本架构和深度学习框架结合，随机生成符合语法规则的深度神经网络。接下来，开发人员选择训练数据，对模型进行训练，并在训练过程中进行安全检查、批量测试和捕获输出，迭代式地生成深度神经网络模型。同时，开发人员可以通过设置变异参数对训练好的模型进行变异，以生成架构不一、数量庞大的深度神经网络模型。最后，自动化分析任务执行时生成的日志文件，将触发的问题分类得到深度学习框架缺陷。此外，本系统支持对编译器进行自动化测试，通过随机生成大量调用编译器接口的程序，从而实现高效测试。

本系统计划由环境管理模块、生成任务管理模块、变异任务管理模块、缺陷分析模块和编译器测试模块五部分组成：通过环境管理模块来创建和切换虚拟环境，每个虚拟环境下维护了各自的深度学习框架，确保模型的生成和变异任务顺利执行，同时降低了人工配置环境的成本；生成任务管理模块指的是根据用户的模型配置信息，迭代训练不同深度学习框架下的深度神经网络模型，同时可以针对训练完成的模型进行推理，以获取模型的质量；变异任务管理将生成任务中训练完成的深度神经网络模型作为种子模型，根据用户设定的变异参数变异模型，在每一次迭代中选择最优变异模型作为下一次迭代的种子模型，以尽可能多地触发深度学习框架中的缺陷；缺陷分析模块对具体生成任务中产生的日志报告进行自动分析，对缺陷成因进行自动归类，最终将结果反馈

给用户；编译器测试模块随机生成用于测试的程序，该程序将尽可能多地覆盖接口，通过比较同一编译器不同版本的输出结果来判断编译器是否存在缺陷。通过五大模块的协同工作，最终实现了一个覆盖全流程的自动化测试系统，解决了测试人员测试框架的繁琐问题。本章将分析本系统的需求，明确系统边界，进行需求分析和系统设计。

3.2 深度学习框架自动化测试系统需求分析

3.2.1 功能性需求

由于使用情况的不同，深度神经网络的结构和大小不同。通过对结构和大小的调整，可以提高模型的效率和准确度。深度神经网络的输入可以是图片的像素、音频的振幅等，是一套表征网络加以分析处理的信息值。

深度神经网络模型依托于深度学习框架，具体表现在模型中的算子。因此，深度神经网络模型的质量与其使用的深度学习框架的质量息息相关，框架中存在的潜在缺陷不易被察觉，模型一旦引入缺陷，可能会造成深刻的影响。基于该原因，针对深度学习框架的研究刻不容缓。

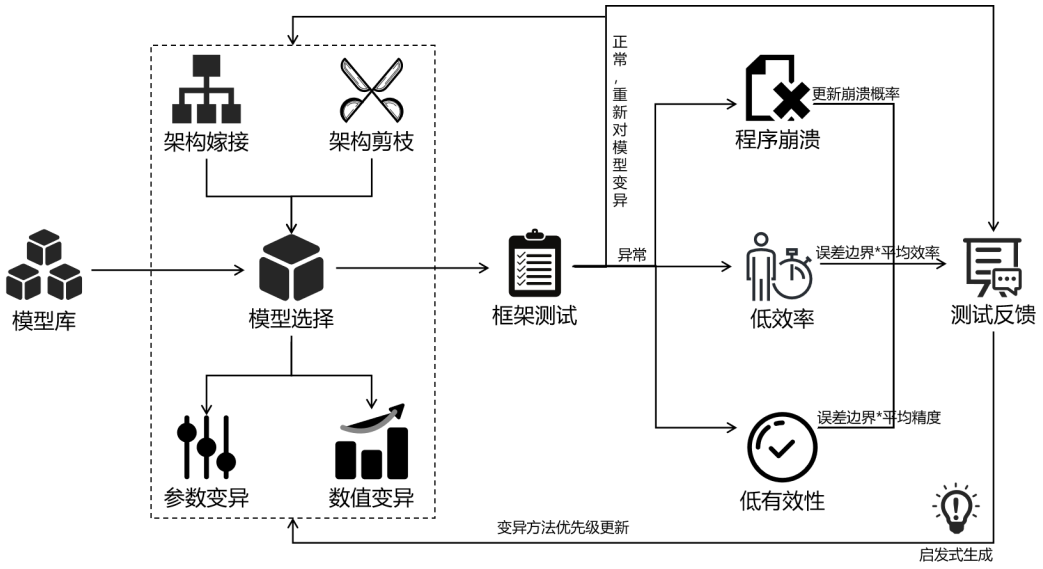

图 3-2: 基于规则变异的深度学习框架自动化测试流程

测试深度学习框架时，可以将深度神经网络模型作为特殊的测试输入。目前，由于被测深度神经网络模型的数量和质量存在较大的差异，因此需要对深

度学习框架的自动化测试开展研究，以实现自动化的深度神经网络模型生成和验证工作，从而提高深度学习框架的质量。

如图3-2所示，本系统分为环境管理模块、生成任务管理模块、变异任务管理模块、缺陷分析模块和编译器测试模块共五个模块来实现基于规则变异的深度学习框架自动化测试系统。其中，生成任务管理模块、变异任务管理模块和缺陷分析模块主要针对深度学习库进行测试，编译器测试模块主要针对深度学习编译器进行测试,环境管理模块则是支撑其它四个模块顺利运行，保障系统的测试环境。

在环境管理模块，系统主要负责虚拟环境的创建和维护。深度神经网络模型作为测试数据，需要基于不同的深度学习框架构建而成。为了能够隔离不同的深度学习框架及版本，防止不同版本的深度学习框架出现兼容性问题，系统支持自动化创建虚拟环境，并根据用户上传的依赖安装文件配置环境。通过不同环境的创建，用户可以根据需要选择待测框架，并将环境切换至指定框架下，进一步执行生成和变异任务。环境管理模块的功能需求如表3-1所示。

表 3-1: 环境管理功能性需求列表

需求 ID	需求名称	需求描述
R1	创建环境	在本系统中，用户可以在环境配置页面上传依赖安装文件，其中包括框架及版本号。系统将自动创建一个虚拟环境，并在该环境下安装依赖。
R2	查看环境信息列表	在本系统中，用户可以在环境配置页面查看自己创建的虚拟环境信息和该环境下的依赖包及版本号。

在生成任务管理模块，用户通过配置深度神经网络的生成参数，系统自动化地生成基于规则实现的深度神经网络模型，并在每一次迭代生成过程中，校验新生成的层是否合规，如果不合规，则回滚这一层的生成，从而保证生成模型的安全性，确保生成的深度神经网络模型动态可用。同时，用户可以根据需求和系统资源的分配实现生成任务的终止和删除。生成任务管理模块的功能需求如表3-2所示。

表 3-2: 生成任务管理功能性需求列表

需求 ID	需求名称	需求描述
R1	创建生成任务	在本系统中，用户在模型生成页面上传生成配置文件，其中包括生成环境、数据集、模式等相关参数。系统将自动创建一个生成任务，用户可以在生成列表中查看和进行后续操作。
R2	生成模型	在本系统中，根据用户的参数配置，从而控制系统批量生成、训练或推理模型。
R3	终止生成	在本系统中，支持用户终止生成任务，以释放系统资源。
R4	删除生成任务	在本系统中，支持用户删除指定生成任务，包括生成的模型、生成的文件和配置文件。
R5	查看生成任务列表	在本系统中，支持用户查看已经创建的生成任务信息。

在变异任务管理模块，系统将用户配置的参数作为模型变异参数，实现深度神经网络模型的变异和缺陷定位。在用户创建的每一批次的变异任务中，系统基于变异算法和因子迭代生成新的变异模型，同时对比不同框架下的输出，判断新的变异模型能否最大程度触发深度学习框架中的缺陷，反复迭代变异，最终完成变异任务。变异任务管理模块的功能需求如表3-3所示。

表 3-3: 变异任务管理功能性需求列表

需求 ID	需求名称	需求描述
R1	创建变异任务	在本系统中，支持用户上传配置文件为变异过程提供参数，系统将根据配置文件自动创建变异任务，用于后续变异任务的执行和缺陷定位。
R2	模型变异	在本系统中，用户可以选中指定变异任务执行模型变异。

R3	定位缺陷	在本系统中，模型在不同深度学习框架下执行变异任务后，系统支持通过比较输出来判断和定位深度学习框架中的缺陷。
R4	删除变异任务	在本系统中，支持用户删除指定变异任务，包括生成的模型、变异的文件和配置文件。
R5	查看变异任务列表	在本系统中，用户可以在模型变异页面查看自己创建的变异任务和该任务的执行状态。

在缺陷分析模块，系统将对深度神经网络模型生成过程中产生的数据和日志文件进行结果分析，并生成报告。此外，基于执行结果，进一步分析待测框架生成时触发的问题成因，同时归纳总结，将其反馈给用户，用户最终可以在系统中查看到具体深度学习框架中存在的缺陷类型和出现次数。缺陷分析模块的功能需求如表3-4所示。

表 3-4: 缺陷分析功能性需求列表

需求 ID	**需求名称**	**需求描述**
R1	查看框架缺陷列表	在本系统中，用户可以在数据分析页面查看缺陷分析信息，包括具体缺陷和缺陷出现的模型等。
R2	分析缺陷	在本系统中，系统遍历生成任务中产生的数据和日志文件，并对其进行分析和归纳总结。

在编译器测试模块，用户通过配置测试程序的参数，系统支持自动化地生成调用接口和参数不同的程序，同时将其运行于同一编译器的不同版本下，通过比较输出结果来定位编译器中存在的缺陷。通过该模块的实现，可以快速生成用于测试深度学习编译器的程序，便于后续定位和报告缺陷，从而提高深度学习编译器的测试效率和质量，支撑深度神经网络模型的运行。编译器测试模块的功能需求如表3-5所示。

表 3-5: 编译器测试功能性需求列表

需求 ID	需求名称	需求描述
R1	查看测试列表	在本系统中，用户可以在测试任务页面查看编译器测试任务的信息，包括每个任务的生成测试程序数量、测试状态和分析状态等。
R2	创建测试任务	在本系统中，支持用户配置任务信息，从而随机生成用于测试编译器的程序。
R3	测试编译器	在本系统中，基于同一编译器下不同的版本，批量运行生成的程序。
R4	分析日志	在本系统中，测试程序在不同版本的编译器下运行后，系统支持通过比较输出来判断和定位编译器中的缺陷。

五个模块相互依存和协作，共同实现了基于规则变异的深度学习框架自动化测试系统。同时每个模块具有相对独立性，承担相应的职责，从而提高了系统的内聚性和可扩展性。

3.2.2 非功能性需求

如表3-6所示为基于规则变异的深度学习框架自动化测试系统的非功能性需求列表。非功能性需求主要从时间特性、并发性、可靠性、安全性和可扩展性几个方面进行描述，从而提高开发效率，保障软件系统运行质量。

表 3-6: 自动化测试系统的非功能需求

类型	非功能需求描述
时间特性	用户进行页面跳转操作，延迟时间不超过 2s；测试报告生成后，生成报告响应时间不超过 3s
并发性	支持至少 10 位用户并发访问同一系统接口
可靠性	故障可追溯，发生单一故障时不影响其他模块使用
安全性	本系统仅允许系统内用户使用，未鉴权或已过期用户无法使用功能
可拓展性	系统新增类似功能可复用

3.2.3 用例设计

根据背景和目标分析，基于规则变异的深度学习框架自动化测试系统主要针对深度学习框架的测试人员，即对于自动化测试深度学习框架有需求的用户，这些用户可以在语法规则的基础上，实现模型生成和模型变异等基本操作。根据上文所述的系统功能需求，本系统用例图如图3-3所示。用例图共涉及到18个用例，分别是创建环境、查看环境信息列表、创建生成任务、生成模型、终止生成、删除生成任务、查看生成任务列表、创建变异任务、模型变异、定位缺陷、删除变异任务、查看变异任务列表、查看缺陷分析列表、缺陷分析、查看编译器测试列表、创建测试任务、测试编译器和分析日志。

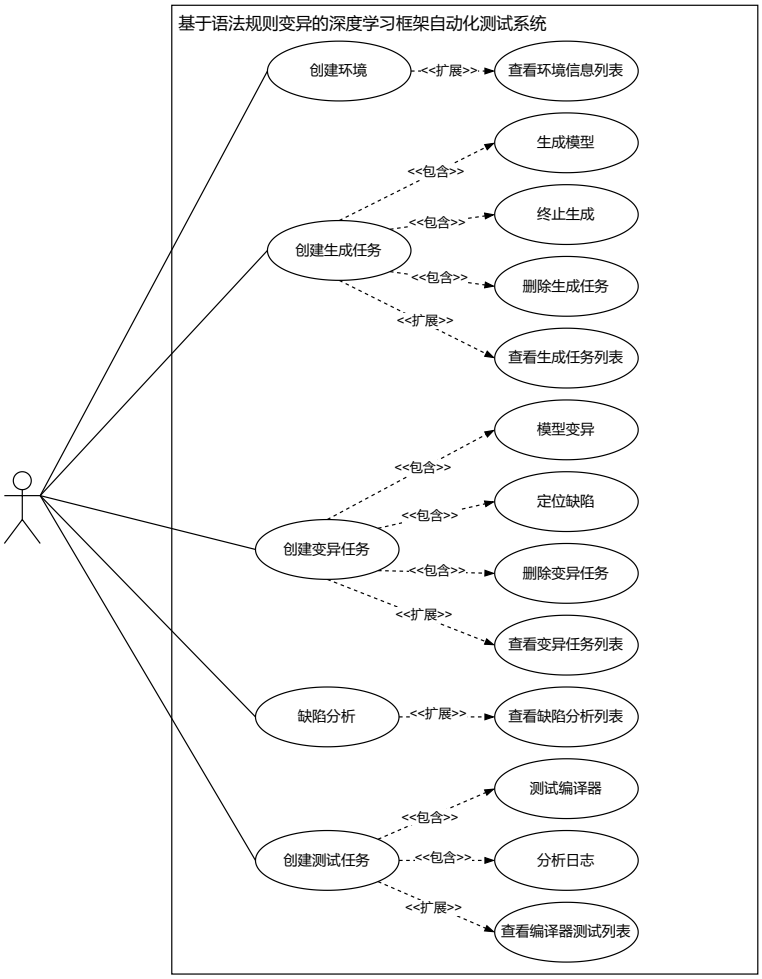

图 3-3: 深度学习框架自动化测试系统用例图

用户登陆系统后，可通过系统对虚拟环境进行管理，即创建和查看虚拟环境，并在该虚拟环境下安装相关依赖；由于不同深度学习框架的算子接口不

同，本系统维护了一套语法规则，包括拓扑规则、张量规则和架构规则，基于该规则可以构建有效的深度神经网络模型；通过用户自定义的模型生成参数，系统可以批量生成深度神经网络模型，并在生成过程中进行安全检查，保证模型的静态可用性；系统支持将成功生成的模型进行训练或推理，同时保存产生的中间数据和日志文件，从而检验模型的有效性。

为了能够更大程度地触发深度学习框架中隐藏的缺陷，系统可以执行模型变异任务。系统将生成任务中训练好的模型作为种子模型，同时分析用户上传的配置文件设置模型变异的参数，对模型进行自定义的变异任务，并记录中间数据和检测是否出现错误。此外，由于框架的不同，系统支持不同框架下同一模型变异后对比其决策，从而定位框架的缺陷。

最后，基于生成任务和变异任务执行中产生的结果和日志文件，系统将对其进行自动化分析，并将归纳总结得到的缺陷分析结果反馈给用户。用户通过反馈信息可以查看缺陷内容和对应缺陷出现的模型，从而实现深度学习框架的自动化测试。

此外，为了能够实现深度学习编译器的自动化测试，本系统支持生成用于测试编译器的程序，该程序将随机调用编译器提供的接口，同时尽可能多地覆盖接口，从而最大程度触发编译器中存在的缺陷。通过比较同一编译器下不同版本的输出来确认是否存在缺陷，并生成缺陷分析报告。

表3-7描述了创建环境用例，在本用例中，用户进入系统后，可以上传依赖安装文件，实现自动化环境创建和安装深度学习框架。首先用户上传依赖安装文件，文件中说明需要安装的深度学习框架和对应版本号。上传成功后，系统可以自动创建一个空的虚拟环境，同时在该环境下安装依赖包，并将环境信息保存至数据库中。此外，系统对于上传的依赖安装文件的个数和类型等也有一些限制。目前系统已经可以支持 txt 类型文件，文件个数限制在 1 个以内。

表 3-7: 创建环境用例描述

ID	UC01	名称	创建环境
描述	用户进行创建环境的操作		
参与者	系统用户		
触发条件	用户点击环境配置页面的创建虚拟环境按钮		
前置条件	用户需已有依赖安装文件		
后置条件	用户能够在环境配置息页面检索到该虚拟环境		

优先级	高
正常流程	1. 用户点击创建虚拟环境按钮 2. 系统显示文件方式上传 3. 用户通过选择或拖拽本地依赖安装文件完成文件方式上传 4. 系统显示环境创建成功
扩展流程	3a. 用户上传的文件不是 txt 类型
特殊需求	系统只支持单次上传一个依赖安装文件

表3-8描述了查看环境信息列表用例。在本用例中，用户进入系统后可以在环境配置页面查看创建完成的环境信息列表，包括虚拟环境名称和该环境下的深度学习框架及版本号。用户可以通过查看环境信息列表，查看已有的深度学习框架及版本。

表 3-8: 查看环境信息列表用例描述

ID	UC02	名称	查看环境信息列表
描述	用户查看环境信息		
参与者	系统用户		
触发条件	用户进入到环境配置页面		
前置条件	存在已经创建完成的虚拟环境		
后置条件	无		
优先级	中		
正常流程	1. 用户进入到环境配置页面 2. 系统显示虚拟环境相关信息		
扩展流程	无		
特殊需求	无		

表3-9描述了创建生成任务用例，在本用例中，用户进入系统后，可以上传模型生成的参数配置文件。首先用户上传参数配置文件，文件中说明任务名称、数据集、算法、生成批次等相关参数。接下来，系统将会把参数配置文件放至指定目录下，同时分析该文件，并将生成信息保存至数据库中。此外，系统对于上传的参数配置文件的个数和类型等也有一些限制。目前系统已经可以支持 conf 类型文件，文件个数限制在 1 个以内。

表 3-9: 创建生成任务用例描述

ID	UC03	名称	创建生成任务
描述	用户进行创建生成任务的操作		
参与者	系统用户		
触发条件	用户点击模型生成页面的创建生成任务按钮		
前置条件	用户需已有参数配置文件		
后置条件	用户能够在模型生成页面检索到该生成任务		
优先级	高		
正常流程	1. 用户点击创建生成任务按钮 2. 系统显示文件方式上传 3. 用户通过选择或拖拽参数配置文件完成文件方式上传 4. 系统显示生成任务创建成功		
扩展流程	3a. 用户上传的文件不是 conf 类型		
特殊需求	系统只支持单次上传一个参数配置文件		

表3-10描述了生成模型用例。在本用例中，系统可以自动生成深度神经网络模型。在每次迭代生成深度神经网络时，用户通过配置参数，控制模型的生成算法、输入数据集、生成大小等过程控制参数。用户配置成功后，系统会首先切换虚拟环境，再根据控制参数来生成深度神经网络模型，同时在生成过程中对其进行安全检查，如果检查发现新生成的模型不合规，则立即回滚，反之继续迭代生成深度神经网络，保证生成模型的动态可用性。生成任务执行完毕后，系统会反馈用户生成的结果。

表 3-10: 生成模型用例描述

ID	UC04	名称	生成模型
描述	用户触发生成模型		
参与者	系统用户		
触发条件	用户进入到模型生成页面		
前置条件	用户已成功创建生成任务		
后置条件	系统反馈生成结果		
优先级	高		

正常流程	1. 用户进入到模型生成页面 2. 系统反馈可选的生成任务 3. 用户选择需要生成的任务 4. 系统自动化生成模型 5. 用户查看生成结果
扩展流程	3a. 用户点击取消按钮，返回模型生成页面 4a. 生成任务被终止，系统重新跳回到步骤 1
特殊需求	无

表3-11描述了终止生成用例。在本用例中，用户可以管理深度学习模型的生成任务，即终止正在执行的生成任务。用户可以根据需求选择需要被终止的生成任务，系统后台将会终止相应进程。系统终止成功后，将会更新生成状态，并反馈给用户。

表 3-11: 终止生成用例描述

ID	UC05	名称	终止生成
描述	用户终止生成任务的操作		
参与者	系统用户		
触发条件	用户进入到模型生成页面		
前置条件	系统正在执行生成任务		
后置条件	系统反馈终止信息		
优先级	高		
正常流程	1. 用户进入到模型生成页面 2. 系统为用户反馈所有生成任务 3. 用户根据需求选择需要终止的生成任务，并提交请求 4. 系统后台终止进行 5. 用户查看终止结果		
扩展流程	3a. 用户点击取消按钮，返回模型生成页面 4a. 终止操作异常，系统反馈终止失败信息		
特殊需求	无		

表3-12描述了删除生成任务用例。在本用例中，用户可以选择删除生成任

务，选择删除后，系统将会把对应任务目录下的参数配置文件、生成的数据和数据库中的信息删除，并将删除结果反馈给用户。

表 3-12: 删除生成任务用例描述

ID	UC06	名称	删除生成任务
描述	用户删除生成任务的操作		
参与者	系统用户		
触发条件	用户进入到模型生成页面		
前置条件	系统已经创建生成任务		
后置条件	无		
优先级	中		
正常流程	1. 用户进入到模型生成页面 2. 系统显示所有生成任务 3. 用户选择需要删除的生成任务并删除 4. 系统返回删除结果		
扩展流程	用户点击取消按钮，系统重新调回步骤 2		
特殊需求	无		

表3-13描述了查看生成任务列表用例。在本用例中，系统可以查看已成功创建的生成任务的信息，同时可以执行生成、终止和删除操作。系统会将生成任务的具体信息反馈给用户，包括任务名称、环境名称、数据集等。用户在查看生成任务列表后，可以选择对应任务进行后续操作。目前可以支持生成、终止和删除操作。

表 3-13: 查看生成任务列表用例描述

ID	UC07	名称	查看生成任务列表
描述	用户查看生成任务		
参与者	系统用户		
触发条件	用户进入到模型生成页面		
前置条件	系统已成功创建生成任务		
后置条件	无		
优先级	中		

正常流程	1. 用户进入到模型生成页面 2. 系统反馈所有生成任务信息，并提供操作入口
扩展流程	2a. 用户点击生成任务对应的操作
特殊需求	无

表3-14描述了创建变异任务用例，在本用例中，用户进入系统后，可以上传模型变异的参数配置文件。首先用户上传参数配置文件，文件中说明模型个数、变异方式、迭代次数等相关参数。接下来，系统将会把参数配置文件放至指定目录下，同时分析该文件，并将变异信息保存至数据库中。此外，系统对于上传的参数配置文件的个数和类型等也有一些限制。目前系统已经可以支持conf类型文件，文件个数限制在1个以内。

表 3-14: 创建变异任务用例描述

ID	UC08	名称	创建变异任务
描述	用户进行创建变异任务的操作		
参与者	系统用户		
触发条件	用户点击模型变异页面的创建变异任务按钮		
前置条件	用户需已有参数配置文件		
后置条件	用户能够在模型变异页面检索到该变异任务		
优先级	高		
正常流程	1. 用户点击创建变异任务按钮 2. 系统显示文件方式上传 3. 用户通过选择或拖拽参数配置文件完成文件方式上传 4. 系统显示变异任务创建成功		
扩展流程	3a. 用户上传的文件不是conf类型		
特殊需求	系统只支持单次上传一个参数配置文件		

表3-15描述了模型变异用例。在本用例中，用户可以触发系统的模型变异任务，以尽可能多地触发深度学习框架中存在的缺陷。系统会首先切换虚拟环境，然后根据用户上传的参数配置文件控制每一次迭代中的变异算法、变异因子、模型地址等过程控制参数，最终生成指定个数的变异模型。任务成功执行后，系统会返回给用户任务执行结果。

表 3-15: 模型变异用例描述

ID	UC09	名称	模型变异
描述	用户触发模型变异的操作		
参与者	系统用户		
触发条件	用户进入到模型变异页面		
前置条件	系统已经执行过生成任务		
后置条件	系统反馈变异结果		
优先级	高		
正常流程	1. 用户进入到模型变异页面 2. 系统显示所有变异任务 3. 用户选择变异任务并执行 4. 系统返回变异任务执行结果		
扩展流程	无		
特殊需求	无		

表3-16描述了定位缺陷用例。在本用例中，系统可以对已经完成变异的模型进行检测，通过对比不同框架下同一模型的输出结果来判断框架中存在的缺陷。系统会自动执行定位缺陷，并生成缺陷文件等分析文档。

表 3-16: 定位缺陷用例描述

ID	UC10	名称	定位缺陷
描述	用户触发定位缺陷的操作		
参与者	系统用户		
触发条件	用户进入到模型变异页面		
前置条件	系统已经执行过模型变异任务		
后置条件	系统生成缺陷分析文档		
优先级	高		
正常流程	1. 用户进入到模型变异页面 2. 系统显示所有变异任务 3. 用户选择变异任务并执行 4. 系统生成缺陷分析文档		

扩展流程	无
特殊需求	无

表3-17描述了删除变异任务用例。在本用例中，用户可以选择删除变异任务，选择删除后，系统将会把对应任务目录下的参数配置文件、生成的数据和数据库中的信息删除，并将删除结果反馈给用户。

表 3-17: 删除变异任务用例描述

ID	UC11	名称	删除变异任务
描述	用户删除变异任务的操作		
参与者	系统用户		
触发条件	用户进入到模型变异页面		
前置条件	系统已成功创建变异任务		
后置条件	无		
优先级	中		
正常流程	1. 用户进入到模型变异页面 2. 系统显示所有变异任务 3. 用户选择需要删除的变异任务并点击删除按钮 4. 系统返回删除结果		
扩展流程	用户点击取消按钮，系统重新调回步骤 2		
特殊需求	无		

表3-18描述了查看变异任务列表用例。在本用例中，系统可以查看变异任务的信息，包括模型个数、变异方法、测试大小等相关信息。同时系统也可以执行开始、分析和删除操作。用户在查看变异任务列表后，可以选择执行和分析相应的变异任务，同时也可以选择删除不需要的变异任务。

表 3-18: 查看变异任务列表用例描述

ID	UC12	名称	查看变异任务列表
描述	用户查看变异任务		
参与者	系统用户		

触发条件	用户进入到模型变异页面
前置条件	系统已成功创建变异任务
后置条件	无
优先级	中
正常流程	1. 用户进入到模型变异页面 2. 系统反馈所有变异任务信息，并提供其它操作入口
扩展流程	2a. 用户点击变异任务对应的相关操作
特殊需求	无

表3-19描述了查看缺陷分析列表用例。在本用例中，用户可以查看具体深度学习框架中存在的缺陷，同时也可以对某一生成任务进行缺陷分析，从而获取分析结果，包括其模型信息、框架信息和缺陷定位信息等。

表 3-19: 查看缺陷分析列表用例描述

ID	UC13	名称	查看缺陷分析列表
描述	用户查看缺陷分析		
参与者	系统用户		
触发条件	用户进入到数据分析页面		
前置条件	用户已执行完毕生成任务		
后置条件	系统反馈缺陷分析结果		
优先级	中		
正常流程	1. 用户进入到数据分析页面 2. 系统反馈所有生成完毕的任务，并提供分析操作入口		
扩展流程	2a. 用户点击生成任务对应的分析操作		
特殊需求	无		

表3-20描述了缺陷分析用例。在本用例中，用户可以获取已完成生成的深度神经网络模型的过程数据和日志文件。系统会遍历信息，筛选和归纳总结存在问题的数据和模型，最终得到缺陷分析结果，将其反馈给用户。用户可以获取具体版本的深度学习框架和对应缺陷分析，从而进行后续上报和修复工作，提高深度学习框架的质量。

表 3-20: 缺陷分析用例描述

ID	UC14	名称	缺陷分析
描述	用户触发缺陷分析的操作		
参与者	系统用户		
触发条件	用户进入到数据分析页面		
前置条件	用户已有生成任务的信息		
后置条件	系统反馈缺陷分析结果		
优先级	高		
正常流程	1. 用户进入到数据分析页面 2. 系统返回已完成生成的生成任务 3. 用户根据需要选择生成任务进行缺陷分析 4. 系统后台执行分析操作 5. 用户查看分析结果		
扩展流程	3a. 用户点击取消按钮，返回缺陷分析页面 4a. 分析失败，系统反馈分析失败信息		
特殊需求	无		

表3-21描述了查看编译器测试列表用例。在本用例中，用户可以获取测试任务的信息，包括测试程序生成数量、测试状态和分析状态等。

表 3-21: 查看编译器测试列表用例描述

ID	UC15	名称	查看编译器测试列表
描述	用户查看测试任务		
参与者	系统用户		
触发条件	用户进入到测试任务页面		
前置条件	用户已有测试任务的信息		
后置条件	无		
优先级	中		
正常流程	1. 用户进入到测试任务页面 2. 系统反馈所有测试任务信息，并提供测试和分析操作入口		
扩展流程	2a. 用户点击测试任务对应的操作		

特殊需求	无

表3-22描述了创建测试任务用例，在本用例中，用户进入系统后，可以创建测试任务。首先用户配置任务信息，包括任务名称和生成数量。接下来，系统将会把测试任务信息保存至数据库中。

表 3-22: 创建测试任务用例描述

ID	UC16	名称	创建测试任务
描述	用户进行创建测试任务的操作		
参与者	系统用户		
触发条件	用户进入到创建任务页面		
前置条件	用户需要创建任务		
后置条件	用户能在测试任务页面检索到该测试任务		
优先级	高		
正常流程	1. 用户进入到创建任务页面 2. 系统显示配置表单 3. 用户填写任务信息并提交 4. 系统显示测试任务创建成功		
扩展流程	无		
特殊需求	无		

表3-23描述了测试编译器用例。在本用例中，用户可以测试已自动生成的编译器程序。本系统已经生成了大量用于测试编译器的程序，因此，需要验证所生成的程序的质量和测试能力，以分析编译器中存在的缺陷。

表 3-23: 测试编译器用例描述

ID	UC17	名称	测试编译器
描述	用户进行测试编译器的操作		
参与者	系统用户		
触发条件	用户进入到测试任务页面		
前置条件	系统已完成生成测试程序任务		

后置条件	系统反馈测试状态
优先级	高
正常流程	1. 用户进入到测试任务页面 2. 系统显示可选的测试任务 3. 用户选择需要测试的测试任务 4. 系统执行测试任务 5. 用户查看测试状态
扩展流程	3a. 用户点击取消按钮，返回测试任务页面
特殊需求	无

表3-24描述了分析日志用例。在本用例中，用户可以获取编译器测试的结果。系统将分析两个版本编译器下程序运行所产生的日志文件，最终生成报告。

表 3-24: 分析日志用例描述

ID	UC18	名称	分析日志
描述	用户触发分析日志的操作		
参与者	系统用户		
触发条件	用户进入到测试任务页面		
前置条件	用户已测试生成的程序		
后置条件	系统反馈分析结果		
优先级	高		
正常流程	1. 用户进入到测试任务页面 2. 系统返回创建成功的测试任务 3. 用户根据需要选择测试任务进行日志分析 4. 系统后台执行分析操作 5. 用户查看分析结果		
扩展流程	3a. 用户点击取消按钮，返回测试任务页面		
特殊需求	无		

3.3 深度学习框架自动化测试系统设计

3.3.1 系统架构设计

本系统采用 C/S 架构，以前后端分离的开发思想实现系统的总体架构。其中，前端采用 Vue 框架搭建，结合 Element-UI 桌面端组件库，以自底向上的方式构建应用，实现与用户交互的 Web 端。后端采用 Django 框架进行搭建，依据 MTV 架构模式（数据存取层-表现层-业务逻辑层），通过 WSGI 服务器进行部署，实现应用服务器与底层代码交互的路径配置。如图3-4所示，基于自顶向下的分层式架构设计，将本系统分为四层，分别是用户交互层、系统后端层、数据计算层和数据存储层。

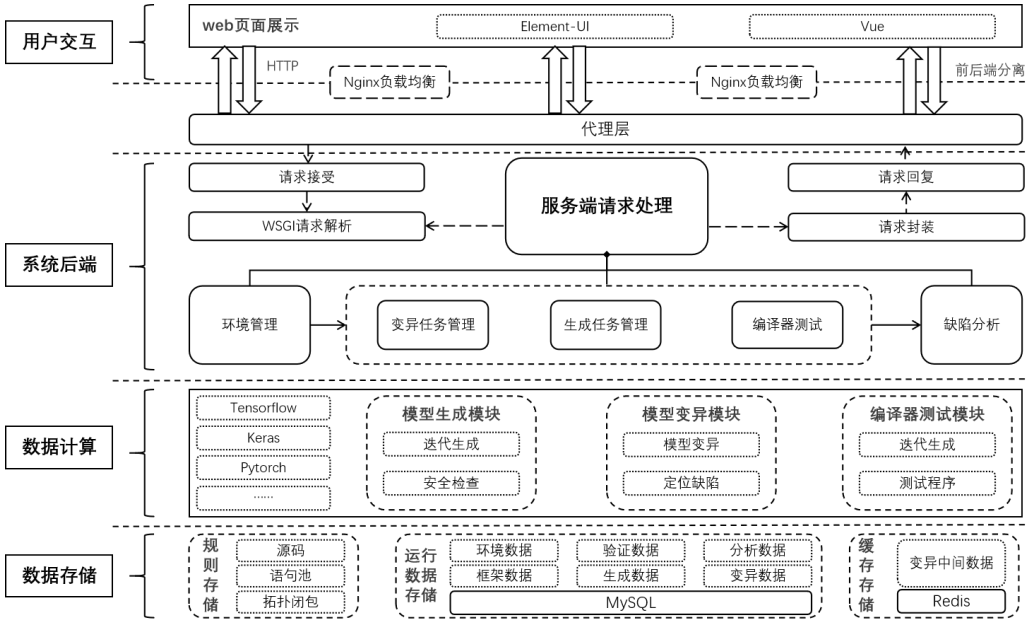

图 3-4: 深度学习框架自动化测试系统框架图

在系统的后端架构设计中，总共分为了环境管理、生成任务管理、变异任务管理、缺陷分析和编译器测试五个模块。系统通过环境管理模块来创建和切换虚拟环境，每个虚拟环境下维护了各自的深度学习框架，确保模型的生成和变异任务能够顺利执行；基于上下文无关语法，生成能够用于测试深度学习框架的测试数据。其中上下文无关语法是由起始符、终止符、非终止符和偏序规则组成的形式化语言的语法四元组，结合生成的语句池和拓扑偏序关系，实现深度神经网络模型迭代生成。为了验证生成模型的质量和有效性，系统支持批

量训练和推理模型，从而触发深度学习框架中存在的缺陷；为了提高测试的质量和广泛性，系统支持对生成的测试数据进行变异，根据用户的自定义变异参数，实现启发式的变异引导策略；缺陷分析模块将对生成任务中产生的中间数据和日志文件进行分析，从而评价模型的生成任务，进一步获取测试所触发的框架缺陷，并将其反馈给用户；而编译器测试模块则是对深度学习编译器进行测试，系统将随机生成大量程序，通过对比不同版本编译器的输出内容来定位缺陷，从而评价编译器的质量。

在本系统中，五个模块相互依赖，协同完成了基于规则变异的深度学习框架自动化测试的技术路线，结合前端 Vue 框架和后端 Django 框架，最终实现了高效可用的深度学习框架自动化测试系统。凭借着 Python 编程语言强大的数据计算能力，本系统将在数据计算层执行所有计算操作，作为本系统架构中至关重要的环节。同时，系统执行时需要将产生的关键数据存储在数据库和指定目录下，其中生成模型的结构文档、接口参数规则等数据将以文件形式存储在指定目录下，而分析结果、模型数据等数据将存放在关系型数据库中。

3.3.2 4+1 视图模型

软件架构由系统抽象构件组成，实现了一个结构、行为和属性的高级抽象，为软件系统的设计提供指导意见。软件架构通过对软件架构建模，解决了如何表示软件架构的问题。为了刻画架构，采用构件、连接件等表示系统不同的行为和内容。

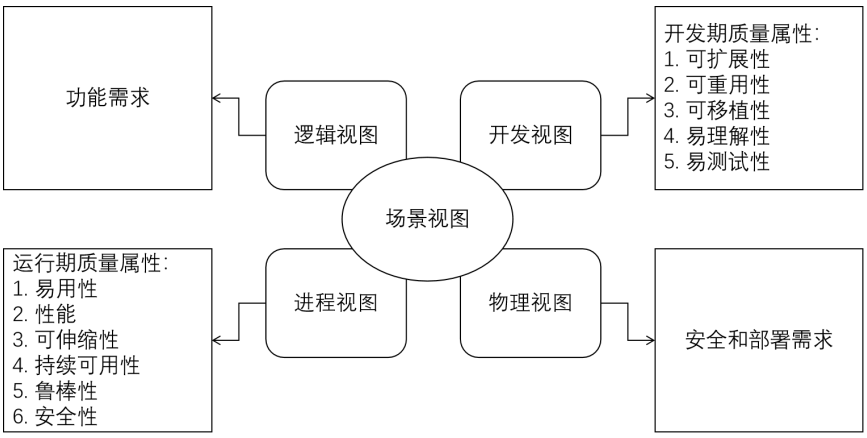

图 3-5: 4+1 视图模型

在软件工程领域，为了抽象描述整个系统，通常使用如图3-5所示的

“4+1”视图方法 [50] 基于不同视角设计逻辑架构。该视图方法将软件系统架构分为五个视图，分别是逻辑视图、进程视图、物理视图、开发视图和场景视图。

逻辑视图通过设计对象模型，抽象分解系统的功能。根据上文的用例设计以及系统架构设计，本文将对系统的逻辑设计进行具体阐述。系统的逻辑视图如图3-6所示，基于面向对象的设计方法，对系统整体逻辑进行对象模型构建和内部动态协作关系的描述。

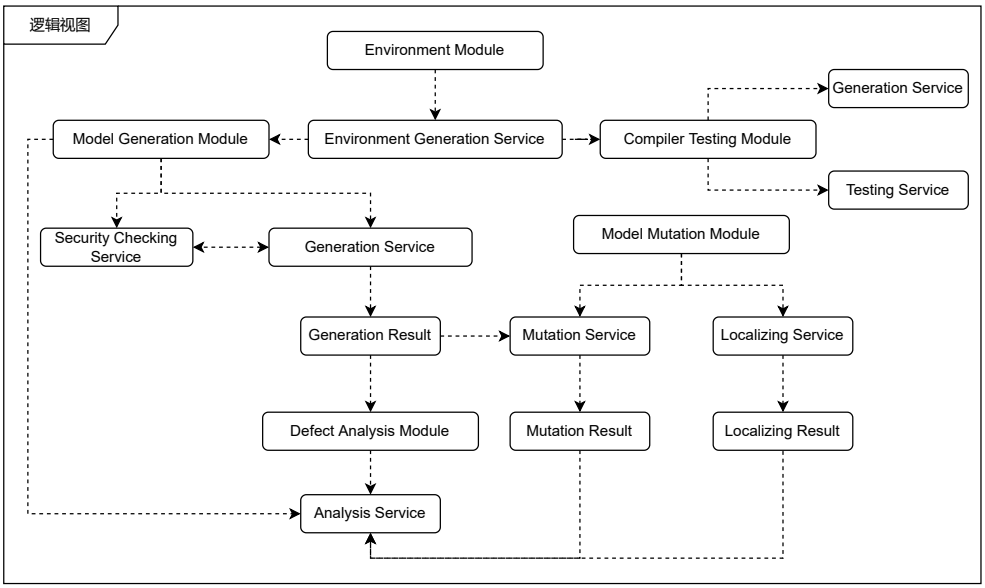

图 3-6: 深度学习框架自动化测试系统逻辑视图

逻辑视图的核心在于描述系统的功能需求，即系统提供给最终用户的深度学习框架的自动化测试服务。为了完成该功能，系统从逻辑层面分解为环境管理模块（Environment Module）、模型生成模块（Model Generation Module）、模型变异模块（Model Mutation Module）、缺陷分析模块（Defect Analysis Module）和编译器测试模块（Compiler Testing Module）。这种分解不仅完成了功能分析，而且标识了整个系统的各个组成部分。通过逻辑视图的设计，本系统基于抽象、封装和继承，最终构建了具有高内聚、低耦合特点的自动化测试系统。

具体而言，Environment Module 负责创建虚拟环境和安装深度学习框架，通过 Environment Generation Service 自动化创建虚拟环境和在该环境下安装依赖包，以隔离不同虚拟环境执行任务；Model Generation Module 负责自动化生成深度神经网络模型，通过 Generation Service 和 Security Checking Service 相互

协作迭代生成用于测试深度学习框架的测试数据；Model Mutation Module 负责对 Generation Task 生成的测试数据进行变异，以更大程度触发深度学习框架的缺陷，通过 Mutation Service 和 Localizing Service 可以实现模型的变异，并定位变异后模型触发的框架缺陷，最终生成 Mutation Result 和 Localizing Result；Defect Analysis Module 负责数据和日志文件的自动化分析，基于 Model Generation Module 和 Model Mutation Module 产生的中间数据和日志文件，实现自动化地分析和缺陷分类；Compiler Testing Module 负责完成编译器测试任务，通过 Generation Service 随机生成用于测试的程序，接着通过 Testing Service 验证生成的程序，从而实现自动化编译器测试。

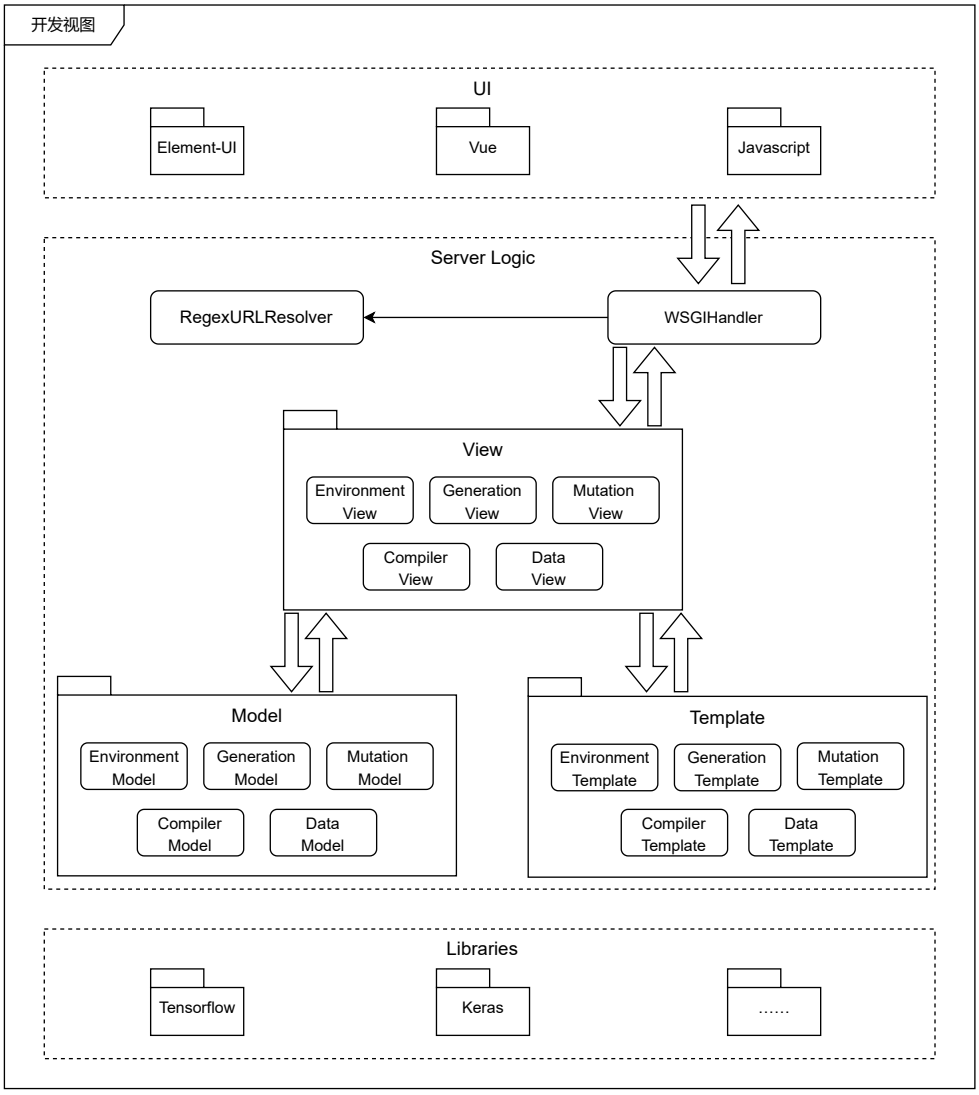

图 3-7: 深度学习框架自动化测试系统开发视图

本系统的开发视图如图3-7所示，其前端采用 Vue 框架，后端采用 Django 框架。将 Vue 框架和 Element-UI 结合，基于 JavaScript 编程语言，使得前端可以快速渲染和响应。依据 MTV 架构风格，将本系统后端分为 View、Model 和 Template 三个包，每个包都各自为五个模块建立了视图类、模型类和模板类。系统通过 WSGIHandler 与前端实现交互，并使用 RegexURLResolver 对 url 进行解析，建立 View 包与前端接口的映射关系。同时，本系统引入了 Tensorflow、Keras、Pytorch 等主流深度学习框架作为生成模型的依赖框架。

本系统的核心实体关系图如图3-8所示，包含环境、任务、变异、分析结果和编译器测试共五个实体，通过必要的数据将这些实体联系起来。其中，环境与任务之间存在多对多的关系，通过环境名称关联；任务与变异之间存在一对多的关系；任务与分析结果之间存在一对多的关系，通过任务名称关联。用户创建环境后，后端会对传入的依赖安装文件解析并执行安装命令，安装成功后，数据库中将会新增对应的环境记录。在创建环境记录成功后，可以在该环境下创建任务，任务执行后也可对生成的模型进行变异，并且该任务与分析结果一一对应。

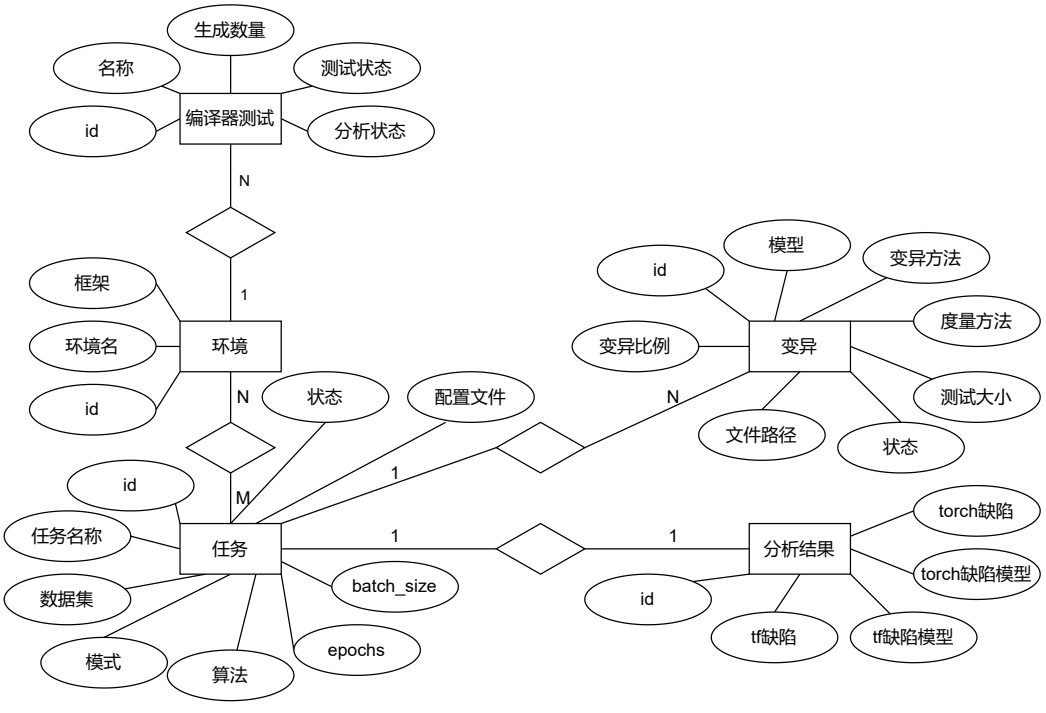

图 3-8: 深度学习框架自动化测试系统核心实体关系图

本系统对五个模块所涉及到的数据实体进行详细的表结构设计，下面将进行具体阐述。环境管理模块涉及到环境实体 Env，对应数据库中的 env 表。Env

表示由用户创建的虚拟环境，其中包含环境名称和该环境下的深度学习框架。其主要字段描述如表3-25所示。

表 3-25: env 视图字段详细设计

字段	含义	类型	描述
id	环境编号	integer	主键
env	环境名称	varchar	该环境的名称
framework	框架名称及版本号	varchar	该环境下的依赖包

生成任务管理模块涉及到任务实体 Task，对应数据库中的 task 表。Task 表示由用户创建的生成任务，其中包含任务名称、数据集、模式、算法等。其主要字段描述如表3-26所示。

表 3-26: task 视图字段详细设计

字段	含义	类型	描述
id	任务编号	integer	主键
task_name	任务名称	varchar	该任务的名称
dataset	数据集	varchar	该任务使用的数据集
mode	模式	varchar	训练或测试模式
alg	算法	varchar	DQN 或 DDQN
epochs	epochs	integer	
batch_size	batch_size	integer	
env	虚拟环境	varchar	该任务执行的虚拟环境
status	状态	varchar	该任务的执行状态
file	配置文件	varchar	该任务的配置文件

变异管理模块涉及到任务实体 Mutate，对应数据库中的 mutate 表。Mutate 表示由用户创建的变异任务，其中包含模型名称、变异方法、测试大小等。其主要字段描述如表3-27所示。

表 3-27: mutate 视图字段详细设计

字段	含义	类型	描述
id	变异任务编号	integer	主键
exps	模型名称	varchar	该任务涉及的模型
ops	变异方法	varchar	变异方法
metrics	度量方法	varchar	度量方法
test_size	测试大小	integer	测试大小
mutate_ratio	变异比例	varchar	变异比例
file	配置文件	varchar	配置文件名称
status	状态	varchar	执行状态

缺陷分析模块涉及到任务实体 Error，对应数据库中的 error 表。Error 表示数据分析的结果，其中包含缺陷内容、问题模型、任务名称等。其主要字段描述如表 6-1所示。

表 3-28: error 视图字段详细设计

字段	含义	类型	描述
id	缺陷编号	integer	主键
error_tf	tf 缺陷内容	varchar	tf 分析结果
model_tf	tf 问题模型	varchar	tf 缺陷对应的问题模型
error_torch	torch 缺陷内容	varchar	torch 分析结果
model_torch	torch 问题模型	varchar	torch 缺陷对应的问题模型
task_name	任务名称	varchar	分析对应的任务

表 3-29: compiler 视图字段详细设计

字段	含义	类型	描述
id	任务编号	integer	主键
name	任务名称	varchar	任务名称
size	生成数量	integer	生成的测试程序数量
status	测试状态	varchar	是否完成测试
analyze	分析状态	varchar	是否完成分析

编译器测试模块涉及到任务实体 Compiler，对应数据库中的 compiler 表。Compiler 表示编译器测试任务，其中包含任务名称、生成数量、测试状态等。其主要字段描述如表 3-29所示。

根据上述视图中描述的需求及设计，本节将从用户角度对深度学习框架自动化测试系统进行进程视图设计。从用户角度出发，基于四层系统架构，完整的进程视图分别实现了五个模块：环境管理模块、生成任务管理模块、变异任务管理模块、缺陷分析模块和编译器测试模块。如图 3-9所示，针对生成任务管理模块和变异任务管理模块进行进程视图设计。

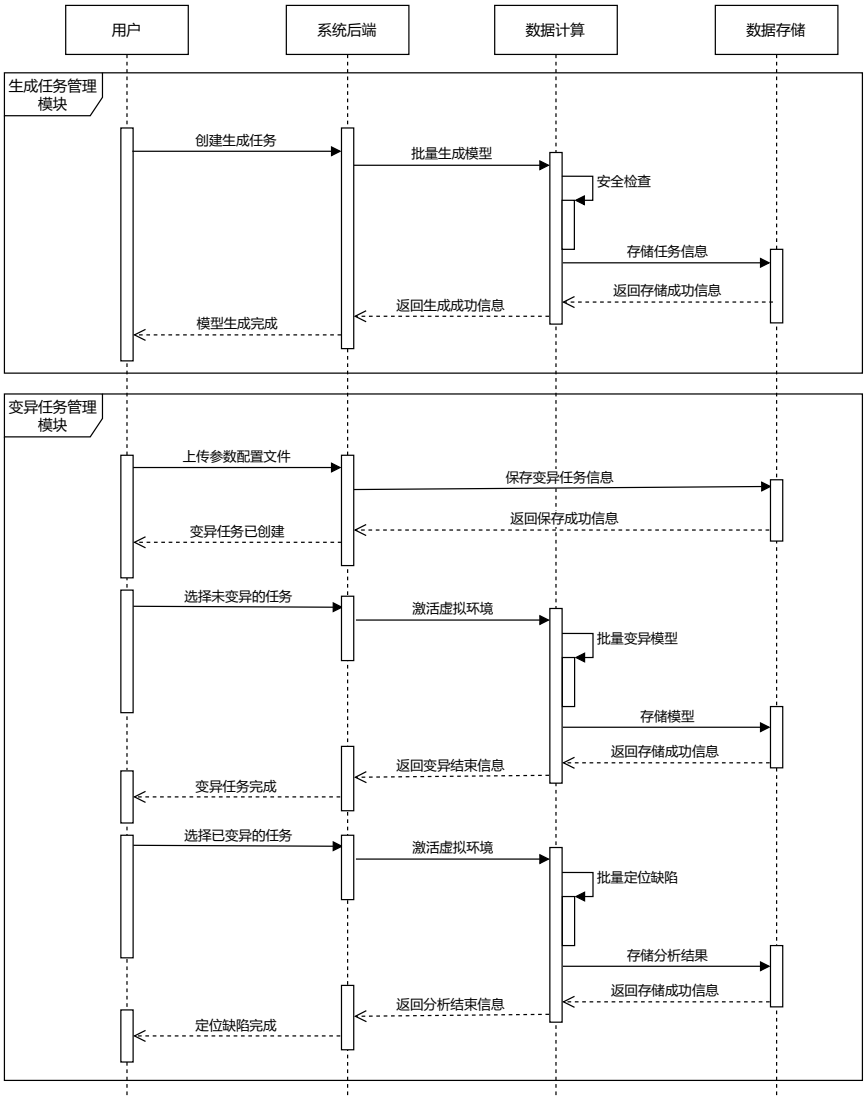

图 3-9: 任务管理进程视图

如图 3-10所示，针对环境管理模块和缺陷分析模块进行进程视图设计。在

环境管理模块中，系统支持用户上传依赖安装文件，并实现自动解析和环境配置。缺陷分析模块则是对日志文件进行分析，生成分析报告。

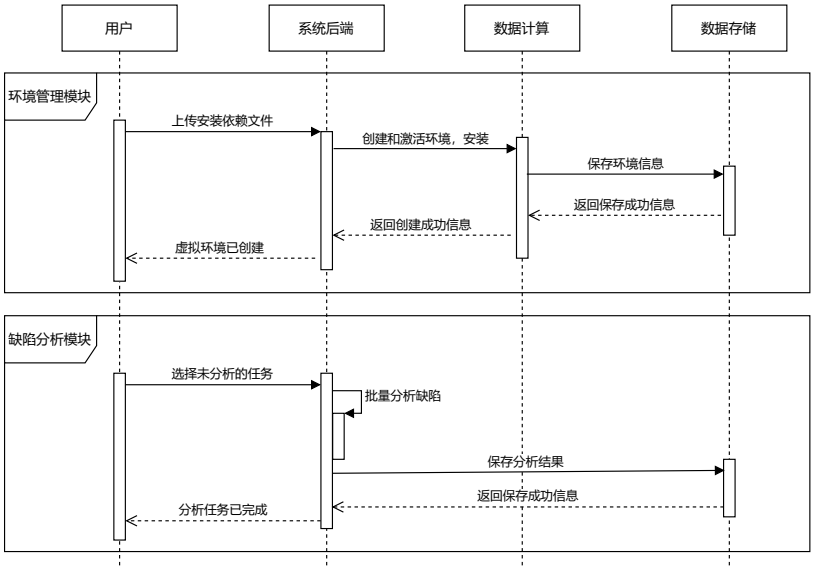

图 3-10: 环境管理和缺陷分析进程视图

如图3-11所示，针对编译器测试模块进行进程视图设计。该模块支持用户配置测试任务，同时执行和分析测试任务，最终将生成分析报告反馈给用户。

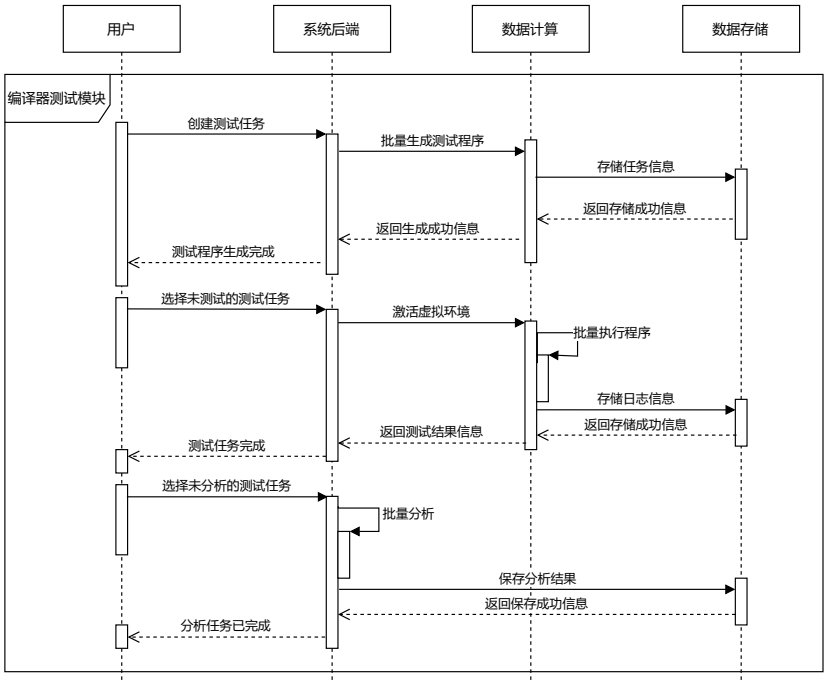

图 3-11: 编译器测试进程视图

通过前端，用户可以和每个模块实现交互操作，将用户的请求路由到后端中的路径进行处理；系统后端收到用户请求后，将会调用相应接口实现数据计算和存储操作；五个模块在执行后，首先将计算的结果持久化在存储层中，再将计算结果由后端返回给用户，防止结果在系统崩溃后由于没有及时存储，导致计算数据的丢失。

系统整体的部署视图设计如图3-12所示，前后端分离分别在单独的节点部署。用户在浏览器中发出 HTTP 请求后，前端服务器将会接受到交互请求并进行处理；基于前端 Vue 框架和 Element-UI 库，前端页面简洁易用，方便用户操作；前端服务器将用户的 Resquet 发送至 WSGI 服务器，等待 WSGI 服务器处理后返回 Response；WSGI 服务器将解析用户的请求，并发送给相应的 WSGI 应用；WSGI 应用处理完各个模块的工作，将会发送 HTTP 请求存储文件数据和任务数据。

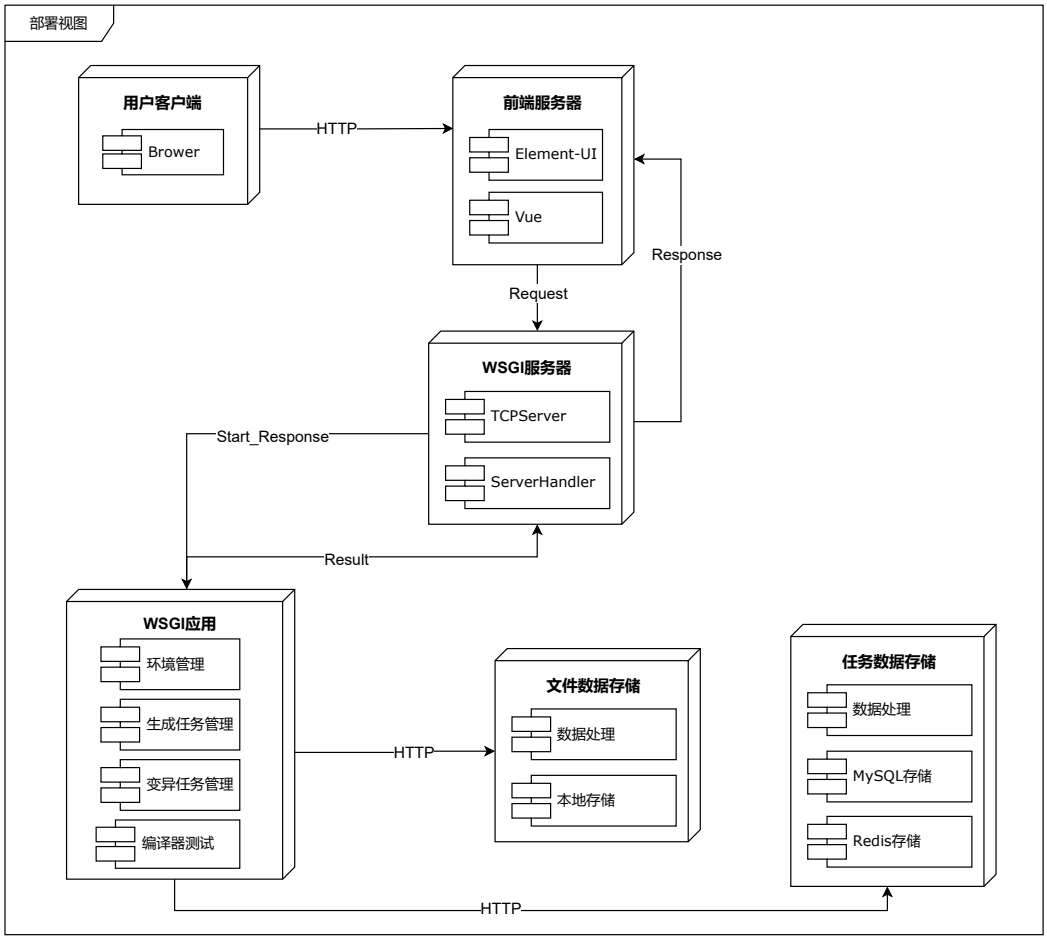

图 3-12: 深度学习框架自动化测试系统部署视图

基于 4+1 视图的分析可知，本文将深度学习框架测试系统划分为环境管理模块、生成任务管理模块、变异任务管理模块、缺陷分析模块和编译器测试模块。在这些模块中，环境管理模块维护了不同虚拟环境，每种虚拟环境下安装了不同版本的深度学习框架，用于支撑生成任务和变异任务的执行；生成任务管理模块支持生成不同框架下用于测试的深度神经网络模型，同时自动化地分配系统资源和验证生成模型的质量；变异任务管理模块会将生成任务管理模块中生成的模型作为输入进行变异，以触发更多不易发现的框架缺陷；缺陷分析模块将会对系统执行任务时产生的文件数据和任务数据进行分析，并自动化实现缺陷归类，作为用户与系统沟通的桥梁；编译器测试模块将会随机生成大量用于测试深度学习编译器的程序，通过比较不同版本下程序执行后的输出结果来定位缺陷。

3.4 本章小结

本章首先对基于规则变异的深度学习框架自动化测试系统进行总体规划，并分析其功能需求与非功能需求。在系统设计过程中，结合 4+1 视图，按照功能将该系统分为环境管理模块、生成任务管理模块、变异任务管理模块、缺陷分析模块和编译器测试模块 5 个部分，开展了系统的概要设计，对本系统进行了多方面的设计。

第四章　基于规则变异的深度神经网络模型生成技术

4.1　整体概述

本方法的整体框架如图4-1所示。支持不同框架下迭代生成更加多样的种子模型，将其作为测试数据进行框架测试，同时自动化地分配系统资源和验证生成模型的质量。

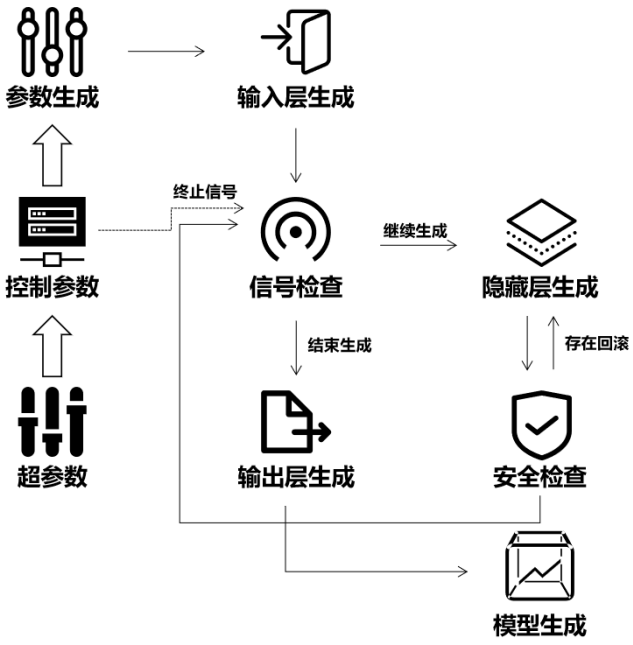

图 4-1: 整体架构

本技术主要由两个重要部分构成，分别是深度神经网络模型的迭代生成（4.2节）和安全检查（4.3节）。通过以上两个部分的实现，可以构建符合语法规则的深度神经网络模型，保证了其动态可用性。

4.2 深度神经网络模型的迭代生成

本技术支持不同框架下迭代生成深度神经网络模型。具体而言，根据上下文无关语法，即基于形式化语法四元组，由终结符、非终结符、规则和起始符组成，终结符对应了模型中的输出层，非终结符对应了模型中的隐藏层，规则对应模型中的拓扑关系，起始符对应模型的输入层。在拓扑规则、张量规则和架构规则的约束下，自动化地构建深度神经网络模型的结构，保证模型的静态可用性。

算法4.1中形式化地描述了基于规则变异的深度神经网络模型迭代生成的流程，主要包括三个部分，分别为参数初始化、数据集加载和模型生成三个部分。

算法 4.1 基于规则变异的深度神经网络模型生成

输入: *target* : 目标生成数量
DLF : 待测深度学习框架
dataPool : 模型生成所需的数据集集合
layerPools : 语句池
grammarRules : 语法规则
APIConfig : 接口参数配置项
table : 接口选择概率表
输出: *Models* : 生成的模型

```
1  while Size(Models) < N do
2      //1. 参数初始化
3      signal, M ← ControlParamInitialization()
4      M ← HyperParamInitialization(M)
5      M ← PackageImport(DLF)
6      //2. 数据集加载
7      dataset, dataProfile ← DatasetSelection(dataPool)
8      LP ← LayerPoolSelection(layerPools, dataset, DLF)
9      M ← DataTransform(dataset, dataProfile)
10     //3. 模型生成
11     foreach i from 0 to signal do
12         L ← LayerSelection(M, LP, table, grammarRules, i)
13         layer ← LayerGeneration(grammarL)
14         M ← LayerAdd(layer)
15     Models ← Models ∪ {M}
16 return Models
```

在参数初始化阶段，将会对超参数和控制参数进行初始化。所谓超参数指

的是训练迭代的次数、样本数和学习率。而控制参数能够控制模型架构的生成，包括终止信号，即隐藏层的数量。当参数初始化完成后，将会导入待测框架。

在数据集加载阶段，针对每一个生成的深度神经网络模型，系统都会从 *dataPool* 中选取一个数据集作为训练数据。选取的同时，会对数据集的输入形状、数据类型等进行设置，从而决定模型的层池类型、预处理方式等。

在模型生成阶段，将从 *layerPools* 中递归地选择需要添加的层。每当一个层被添加成功后，均会检查是否触发终止信号。当隐藏层的数量达到终止信号后，即表示模型生成完成，最终添加输出层，构建完整的深度神经网络模型。

4.3 安全检查

深度神经网络模型由若干层叠加而成，即深度学习框架提供的算子和接口调用语句。基于接口之间连接的规则，能够构建有效的深度神经网络模型。而测试深度学习框架需要生成架构不一、参数多样的深度神经网络模型。如果任意组合框架接口来构建深度神经网络模型的源码，可能会导致生成大量无效的模型。因此，本文通过对模型生成过程进行安全检查，从而构建符合语法规则的深度神经网络模型，保证模型的动态可用性。

如4-2所示，是实现安全检查的关键代码。生成深度神经网络模型时，会将大量的层添加至模型中，为了检查添加的层是否符合语法规则，需要进行检查。每添加新的一层，都需要检查新的一层是否可以添加。如果添加后的模型不合规，则会回滚添加操作。反之，则继续迭代生成隐藏层，直到触发终止信号，生成输出层。最终实现构建完整的深度神经网络模型，保证了生成的模型的动态可用性，提高了生成模型的质量，从而使得深度学习框架测试具有一定的有效性。

具体而言，首先将检查随机生成的模型结构是否符合规范，包含模型名称、框架等相关参数。接着，系统将开始逐层添加进行训练，如果训练过程中一些不合适的算子被加入，将在训练模式下屏蔽该算子，并回滚该轮添加。同时计算算子的输出维度，进而添加下一层。最后，添加优化器，生成最终模型。当然，在训练过程中，系统将检查是否存在 NAN/INF，如果存在，我们认为该问题可能由框架缺陷而引发，因此将会对其进行记录，之后进行缺陷分析。

```
def step_in_mode_train(self, action):
    cur_layer = self.get_op(action)
    if action in CNNEnvironment.train_mask:
        return self.get_one_hot_obv(action), -self.scale_factor, False
    if cur_layer is None:
        return self.get_one_hot_obv(action), -self.scale_factor, False
    self.model_cur_shape = self.get_shape_by_ops(cur_layer)
    next_obv = self.get_one_hot_obv(action)
    st_record = self.cur_state
    self.cur_state = action
    self.model_json_list.append(cur_layer)
    torch_dict, torch_add_condition = self.get_dict('PyTorch')
    tf_dict, tf_add_condition = self.get_dict('TensorFlow')
    assert torch_add_condition == tf_add_condition
    torch_model = ModelJSON(torch_dict)
    tf_model = ModelJSON(tf_dict)
    last_layer_idx = -1 - tf_add_condition
    tf_last_layer = tf_model.network.get_layer(index=last_layer_idx)
    torch_last_layer = self.get_layer_torch(last_layer_idx,
        torch_model.network)
    tf_last_layer_output_function =
        function([tf_model.network.layers[0].input],[tf_last_layer.output,
        tf_model.network.layers[-1].output])
    torch_last_layer_output_function = []
```

图 4-2: 安全检查关键代码

4.4 本章小结

本章主要介绍了基于规则变异的深度神经网络模型生成技术的整体架构。我们从该技术的两个重要部分出发，分别为深度神经网络模型的迭代生成和安全检查。针对深度神经网络模型的迭代生成，以算法实现为基础，具体阐述了

模型生成过程。针对安全检查，以关键代码为基础，具体阐述了如何校验模型生成的有效性。通过以上两个重要部分的实现，最终构建完整的深度神经网络模型。

第五章　深度学习框架自动化测试系统详细设计与实现

本章主要介绍基于规则变异的深度学习框架自动化测试系统的详细设计与实现。结合第三章中所做的的需求分析和概要设计，将本系统从逻辑上分为环境管理模块、生成任务管理模块、变异任务管理模块、缺陷分析模块和编译器测试模块，本章将会对这五个模块的设计与实现进行详细介绍。

5.1　环境管理模块

5.1.1　环境管理模块总体实现

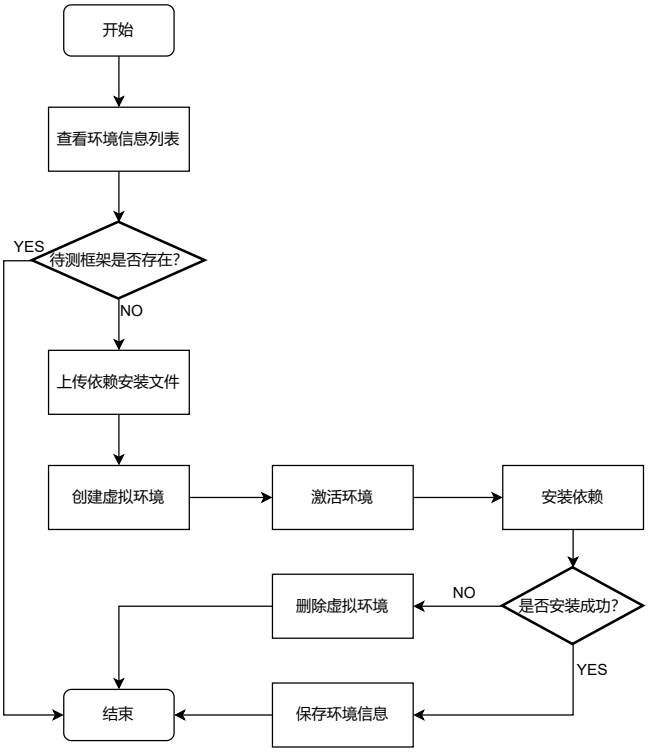

图 5-1: 环境管理模块流程图

在环境管理模块，系统主要负责完成根据用户上传的依赖安装文件，自动化执行虚拟环境的创建、激活和安装。具体而言，本模块需要完成在系统指定目录下创建一个空的虚拟环境，并安装用户指定版本的深度学习框架。虚拟环境的存在可以隔离不同任务的运行环境，在不同环境下，系统支持用户选择不同版本的待测框架进行测试，从而实现本系统测试的广泛性。

如图5-1所示，是本模块的实现流程图，主要完成了本系统中的环境创建和框架安装。用户可以查看环境信息列表，筛选所需待测框架及版本是否存在，如果不存在，支持用户上传依赖安装文件，该文件包含待测框架的名称及版本号。上传成功后，系统会为该框架创建一个新的虚拟环境，同时激活该环境，并在该环境下批量安装依赖包。系统创建成功后，会将该环境的信息和框架名称保存至数据库，方便用户查看环境信息列表。

环境管理模块的实现支撑了系统中生成任务和变异任务的执行，为任务的构建和运行提供了良好的运行环境，其模块的类图如5-2所示。在本模块中，系统支持用户上传依赖安装文件，让用户自定义待测框架。系统还可以自动化地创建虚拟环境和执行安装命令，减少用户配置环境的压力，提供测试效率。

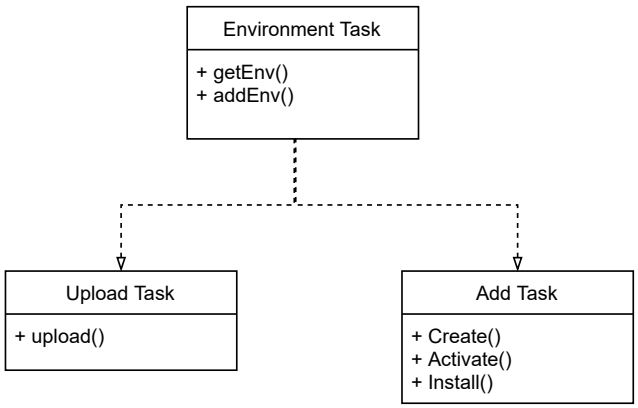

图 5-2: 环境管理模块类图

5.1.2 环境创建的实现

创建环境是为了隔离不同的虚拟环境，每个虚拟环境下安装了不同版本的深度学习框架，可以防止框架之间发生版本冲突，同时方便用户切换待测框架来执行执行任务，为模型的生成和变异奠定了基础。

如5-3所示，是本系统中实现环境创建的关键代码。由于环境创建需要用户熟悉各种命令行指令，同时会花费用户不必要的时间，因此本模块封装了命

令行指令，自动化地执行创建虚拟环境的命令，可以在一定程度上减少测试人员工作量和避免发生意外问题。

```
def create_venv(self):
    """创建虚拟环境包"""
    self.venv_dir = os.path.join("\HTesting\main\env", VENV_DIR)
    # 定位到虚拟环境./venv 当前位置下
    self.requirements_dir = os.path.join("\HTesting\main\env",
        REQUIREMENTS_DIR) # 文本位置
    # 定位到requirement.txt 文件 ./requirement 当前位置下
    if os.path.exists(self.venv_dir): # 判断当前目录下是否存在虚拟文件夹
        # 已经存在,删除虚拟文件夹
        shutil.rmtree(self.venv_dir) # 删除虚拟文件夹以及其下所有的文件
    print("Creating virtual environment..")
    subprocess.Popen(["python", "-m", "virtualenv", "-p",
        "D:\Anaconda3\python.exe", self.venv_dir],
                     stderr=subprocess.STDOUT).wait()
    self.pip = os.path.join(self.venv_dir, 'Scripts\pip.exe')
    # 定位到venv虚拟环境下pip包的位置
    self.python = os.path.join(self.venv_dir, 'Scripts\python.exe')
    self.activate = os.path.join(self.venv_dir, 'Scripts\activate')
```

图 5-3: 环境创建关键代码

5.1.3 框架安装的实现

框架安装是为了在不同的虚拟环境下，安装不同版本的深度学习框架，以应对不同的测试需求。框架安装成功后，用户可以随意切换环境，使用该环境下的框架构建和测试模型，从而解决测试环境搭建困难的问题，提高测试效率和质量。

如5-4所示，是本系统中安装框架的关键代码。为了能够实现用户安装任意版本的框架，本模块支持激活创建成功的虚拟环境，从而使用需要进行测试的深度学习框架及版本，同时根据用户上传的依赖安装文件，实现自动化框架

安装。本模块的实现能够提高切换环境的效率，并减少用户安装框架的时间，避免出现安装问题。

```
def  install_packages ( self ):
    print (" Installing  packages .. ")
    subprocess . Popen([ self . pip ,  " install ",  "-r",  self . requirements_dir ,"-i",
        "https :// pypi . tuna . tsinghua . edu . cn / simple" ],
            stderr =subprocess . STDOUT).wait()
    return  VENV_DIR
def  create_symlink ( self ):
    print ("Creating  ' activate  '.. ")
    subprocess . Popen(os . path . join ( self . venv_dir ,  ' Scripts ' ,
        ' activate . bat ' ),  stderr =subprocess . STDOUT).wait()
```

图 5-4: 框架安装关键代码

5.2 生成任务管理模块

5.2.1 生成任务管理模块总体实现

在生成任务管理模块，系统实现了基于语法规则的深度神经网络模型的迭代生成。基于上下文无关语法，对模型的架构形成语法映射，同时在拓扑规则、张量规则和架构规则的约束下，自动化地构建深度神经网络模型，同时保证模型的动态可用性。

如图5-5所示，是本模块的实现流程，主要完成了生成任务的创建与维护。用户首先需要配置生成模型的配置控制参数，包括数据集、训练迭代次数等。在每一次迭代生成过程中，系统会根据用户提交的配置控制参数，进行参数初始化、数据集加载等操作，进而迭代生成深度神经网络模型。

每进行一次迭代，系统首先会生成输入层；接着，系统会判断隐藏层的数量是否达到配置的数量，如果达到，则生成输出层，并保存生成的模型；如果没有达到，系统会根据拓扑规则，从中选择一个隐藏层添加，添加完成后，会计算和矫正张量的形状。最后，为了保证添加的有效性，系统会进行模型预训练。

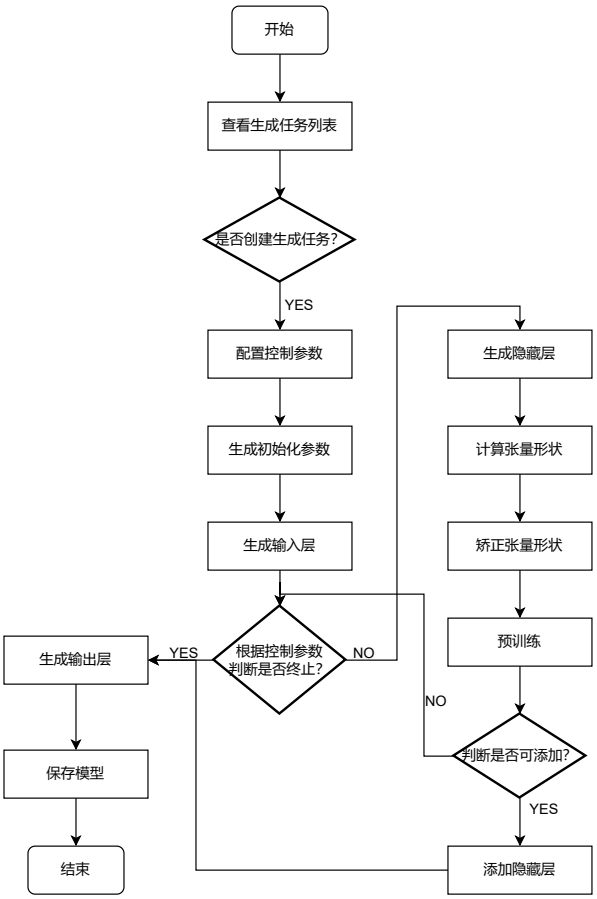

图 5-5: 生成任务管理模块流程图

生成任务管理模块基于用户配置的控制参数，迭代生成深度神经网络模型，其模块的类图如5-6所示。在本模块中，根据用户配置的参数，系统能够实现模型的自动化批量生成。具体而言，系统会在迭代生成的过程中，判断新生成的一层是否符合语法规则，如果不符合，将回滚生成操作。

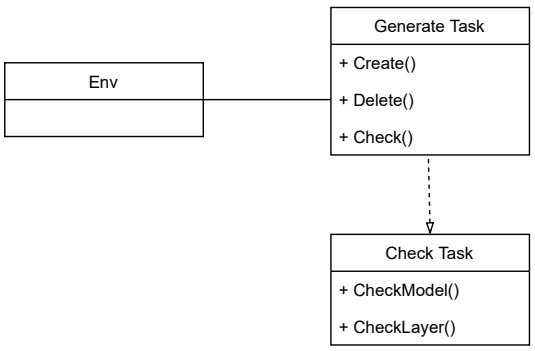

图 5-6: 生成任务管理模块类图

5.2.2 迭代生成的实现

```
{
  ”name”: ”ResNet18”,
  ”framework”: ”TensorFlow”,
  ”input_shape”: [224, 224, 3],
  ”network”: [
    {
      ”name”: ”Conv2d”,
      ”params”: {”in_channels”: 3, ”out_channels”: 64, ”kernel_size”: 7,
          ”stride”: 2, ”padding”: ”same”}
    },
    {
      ”name”: ”BatchNorm2d”,
      ”params”: {”num_features”: 64}
    },
    {
      ”name”: ”MaxPool2d”,
      ”params”: {”pool_size”: 3, ”stride”: 2, ”padding”: ”same”},
      ”branch_to”: ”residual_add_1”
    },
    {
      ”name”: ”Conv2d”,
      ”params”: {”in_channels”: 64, ”out_channels”: 64, ”kernel_size”: 3,
          ”padding”: ”same”}
    }
  ]
}
```

图 5-7: 迭代生成关键代码

如5-7所示，是本系统随机生成的模型结构的关键代码。在本模型中，模型结构存储了模型生成时可以被添加的层的详细信息，包括每一层的名称、参

数等，是生成模型的关键组成部分。由于不同的层的层内接口和约束范围不同，每一层的详细信息以层为单位，存储在不同的 json 文件中，每一个文件对应着一个深度神经网络模型结构。系统在生成该 json 文件后，将根据其中的网络结构训练深度神经网络模型。同时，在训练过程中，对其进行安全检查，从而保证生成模型的动态可用性。

5.2.3 终止生成的实现

如5-8所示，是本系统中实现终止生成任务的关键代码。由于生成任务的执行需要占用大量系统资源且耗时较长，因此本模块支持用户终止生成任务，从而释放系统资源。

在本模块中，为了终止所有正在执行的生成任务，系统会维护一个存储进程号的文件。首先，系统打开并遍历该文件；接着，系统会判断每一个进程号下的进程是否在执行，如果正在执行，则杀死该进程；最后，终止所有正在执行的生成任务，释放系统资源。

```
def kill (task_name):
    base="\HTesting\main"
    path = os.path.join(base, 'generation', 'preliminary_trials', 'dqn',
        task_name)
    f=open(os.path.join(path,'%s.txt'%task_name))
    for line in f:
        line = line.strip('\n')
        try:
            os.kill(int(line), signal.SIGTERM)
        except:
            continue
    f.close()
    print("kill success!")
```

图 5-8: 终止生成关键代码

5.3 变异任务管理模块

5.3.1 变异任务管理模块总体实现

本小节针对深度神经网络模型执行变异任务，实现更大程度的测试覆盖待测深度学习框架。本文采用模型级别的变异，不仅可以避免模型长时间训练，而且可以在模型中引入更细微的变化，有利于进一步定位缺陷。

与传统的测试输入不同，深度神经网络模型由多层叠加构成，同时每一层都包含大量的神经元和连接权重，因此传统的测试生成工具不能用于生成深度神经网络模型。此外，由于训练成本和训练数据限制，且现存的模型架构单一、数量有限，很难获得数量庞大的深度神经网络模型。

因此，本小节将生成任务执行后得到的深度神经网络模型作为变异种子，采用启发式策略来指导模型的变异，从而生成能够尽可能放大不同深度学习框架之间不一致性的模型。具体而言，在每一次迭代过程中，选取更有可能触发深度学习框架不一致性的模型作为种子模型，同时采用更有可能触发深度学习框架不一致性的变异策略进行下一次迭代。

种子模型指即将执行变异任务的深度神经网络模型，既可以是初始模型，也可以是执行过变异任务的模型。本文中的种子模型指经过生成任务训练完成的基于语法规则构建的深度神经网络模型，具有随机性和可用性。在变异任务执行过程中，通过对模型迭代变异，逐步增加模型在不同深度学习框架下的不一致程度，最终基于不一致性发现框架中存在的潜在缺陷。因此，变异任务会在执行过程中选择经过多次变异且表现不一致程度更大的模型作为种子模型，进入下一次迭代。

设 M 为待测深度学习模型，给定一组输入 $\{I_1, I_2, \ldots, I_n\}$ 和一组深度学习框架 $\{L_1, L_2, \ldots, L_m\}$，则模型 M 对所有输入在所有深度学习框架下对比累计的不一致性可如下表示：

$$ACC(M) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=j+1}^{m} D\_MAD_{G,O_{ji},O_{ki}}$$

设 M_s 和 M_m 分别是种子模型和变异模型，如果 $ACC(M_m)$ 比 $ACC(M_s)$ 大，则 M_s 向 M_m 变异时不一致性在变大。

如果种子模型很少被选中执行变异任务，那么应该增加被选为种子模型的机会。这种做法是为了提高变异的多样性，避免反复变异同一个模型。对于种

子模型 s_i，变异次数记为 c_i，则 s_i 对应的计算分数公式为：

$$score_i = \frac{1}{c_i + 1}$$

计算所得的分数越大，种子模型被选中执行变异任务的机会就越大。设 r 表示种子模型的数量，可以将 s_i 被选中的概率表示如下：

$$p_i = \frac{score_i}{\sum_{k=1}^{r} score_k}$$

算法 5.1 模型变异

输入: $Rules$: 变异规则列表
$Seeds$: 种子模型列表 [M_s]
N : 生成的模型数量
输出: $Models$: 生成的模型

17 $MU_a \leftarrow random(Rules)$
18 **while** $Size(Models) < N$ **do**
19 **foreach** i from 1 to $Size(Seeds)$ **do**
20 $Pro[i] \leftarrow calProb(Seeds[i])$
21 $r \leftarrow random(0, 1)$
22 $bound \leftarrow 0$
23 **foreach** i from 1 to $Size(Seeds)$ **do**
24 $bound \leftarrow bound + Pro[i]$
25 **if** $r \leq bound$ **then**
26 $s \leftarrow Seeds[i]$
27 break
28 $k_a \leftarrow position(MU_a)$
29 **while** $f \geq (1-p)^{k_b - k_a}$ **do**
30 $MR_b \leftarrow random(Rules)$
31 $k_b \leftarrow position(MR_b)$
32 $f \leftarrow random(0, 1)$
33 $m \leftarrow Mutate(s, MU_b)$
34 $Models \leftarrow Models \cup \{m\}$
35 **if** $ACC(m) \geq ACC(s)$ **then**
36 $Seeds \leftarrow Seeds \cup \{m\}$
37 $updateScore(s)$
38 $updateScore(MU_b)$
39 $Rules \leftarrow sort(Rules)$
40 $MU_a \leftarrow MU_b$
41 return $Models$

根据选定的种子模型，系统选择一个变异规则执行变异任务。然而，对于给定的种子模型，不同的变异规则在放大不一致性方面可能有不同的效果。具体而言，如果一个变异规则更容易生成不一致性程度大的模型，那么它更有可能被选择用于下一次迭代。因此，每个变异规则拥有一个优先级分数，并对其进行降序排列，分数高的变异规则将更有可能被选中。

算法5.1中描述了本文的模型变异的流程，主要包括种子模型选择、变异规则选择、模型变异等多个步骤。

在创建变异任务时，需要设置变异的参数，包括模型名称、变异规则等控制参数。通过参数配置，可以控制模型的整个变异过程，确保变异任务能够安全和高效执行，提高变异的有效性，从而更大程度触发深度学习框架中存在的缺陷。

初始状态下，设变异规则集合 Rules 中的所有变异规则的优先级分数相等，并随机选择一个 MU_A。接下来，算法迭代模型变异的过程。算法中 4-12 行为选择种子模型的过程，计算每个种子模型的分数，同时累和得到最终选择的种子模型。13-17 行为选择变异规则 MU_b 的过程，计算 MU_a 和 MU_b 之间是否发生转移，如果没有则反复该过程。17-18 行为生成新的深度神经网络模型，并将该模型添加至集合 *Models* 中。19-20 行将对比新生成的模型与变异前的种子模型的不一致性是否被放大。21-22 行将更新种子模型和变异规则的评分。23 行将对所有评分进行重新排序。24 行将 MU_b 赋值给 MU_a。接下来，继续下一次迭代，直到生成 N 个模型。

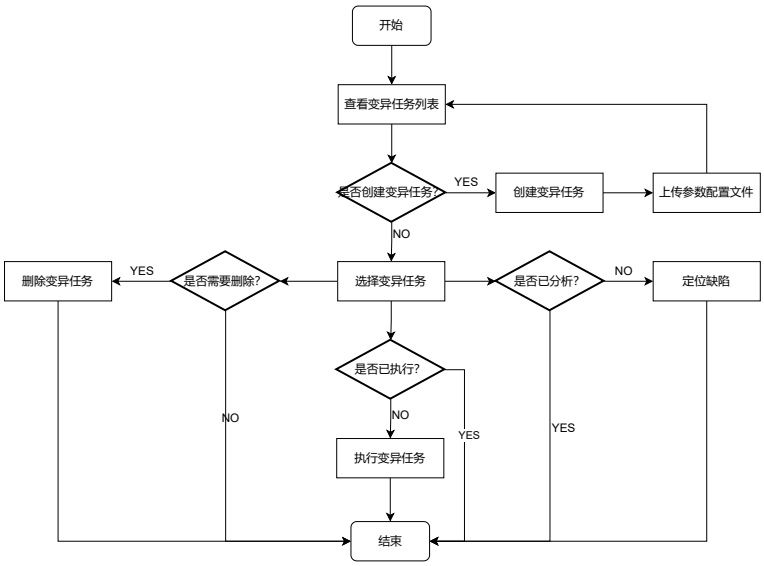

图 5–9: 变异任务管理模块流程图

在变异任务管理模块，系统主要负责完成模型变异和定位缺陷的功能。如图5-9所示，是本模块的实现流程，用户需要查看已经创建的变异任务及其对应参数配置，选择要执行变异的任务。完成变异任务后，用户可以选择该任务进行分析，从而定位深度学习框架下存在的潜在缺陷。同时，用户可以选择删除任一变异任务，系统将删除数据库中存储的相关信息和对应目录下生成的文件。

变异任务管理模块主要实现控制每一个创建的变异任务的执行、分析和删除，其模块的类图如5-10所示。在本模块中，系统负责调用模型变异和定位缺陷的接口，实现变异模型的迭代生成，并分析模型在不同深度学习框架下的不一致性，从而定位框架中存在的潜在缺陷。最终，将分析得到的缺陷信息写入文件保存在当前变异任务的目录下。

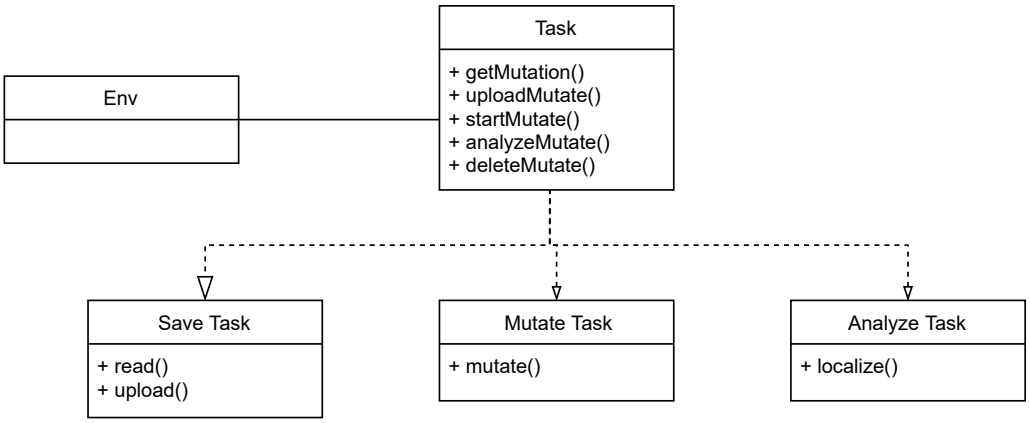

图 5-10: 变异任务管理模块类图

5.3.2 模型变异的实现

本文采取了两种方式对随机生成的模型进行变异，分别是完整层变异和神经元变异。完整层变异涉及整个层的变异，将会对模型引入相对较大的变化。本文使用了 7 个变异规则，分别是删除层、交换层、复制层、添加层、添加多层、激活函数移除和激活函数替换。下面对其进行简单介绍：

- 删除层（LR），其示意图如5-11所示。删除模型中输入形状和输出形状一致的层。

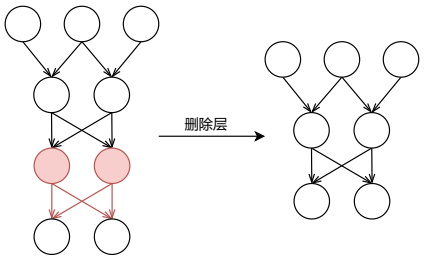

图 5-11: 删除层变异规则示意图

- 交换层（LS），其示意图如5-12所示。交换两个输入形状和输出形状一致的层。

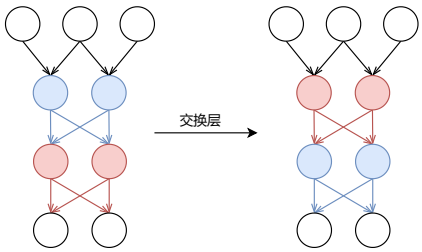

图 5-12: 交换层变异规则示意图

- 复制层（LC），其示意图如5-13所示。复制一个输入形状和输出形状一致的层，然后将被复制的层添加在原层之后。

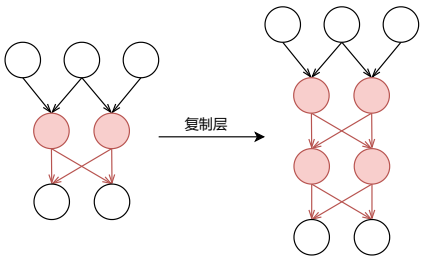

图 5-13: 复制层变异规则示意图

- 添加层（LA），其示意图如5-14所示。将一个输入形状和输出形状一致的层添加至合适的位置。

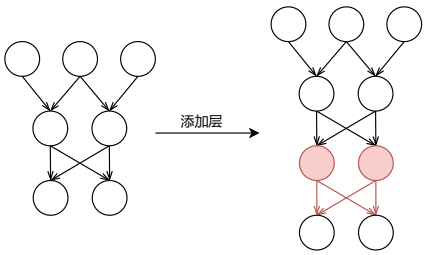

图 5-14: 添加层变异规则示意图

- 添加多层（MLA），其示意图如5-15所示。将多个神经元层连接而成的层添加到模型中，要求第一层的输入形状与添加位置的最后一层的输出形状一致，同时最后一层的输出形状与添加位置的第一层的输入形状一致。

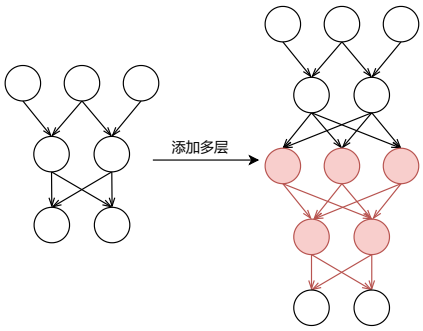

图 5-15: 添加多层变异规则示意图

- 激活函数移除（AFRm），其示意图如5-16所示。移除某一层的激活函数。

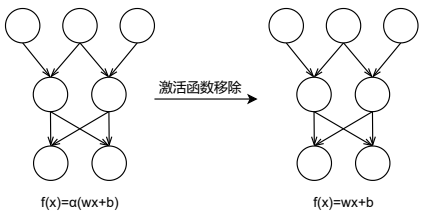

图 5-16: 激活函数替换变异规则示意图

- 激活函数替换（AFRp），其示意图如5-17所示。替换某一层的激活函数。

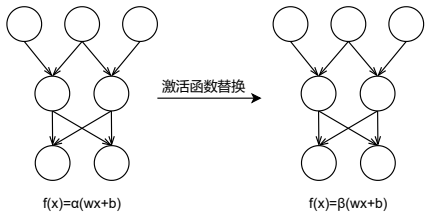

图 5-17: 激活函数替换变异规则示意图

神经元变异作用于一层神经元，比完整层变异粒度更细。系统根据用户设置的变异参数，可以对深度神经网络模型中的单个神经元进行调整，也可以调整多个神经元。神经元作为测试深度学习框架的基本单位，每一层的计算依赖于神经元，因此改变神经元的某些属性 (例如权重和激活状态) 可以更加充分地测试每一层中使用的框架接口，也有助于探索框架接口的不同使用方式，从而提高测试的覆盖率。

本文使用了 5 种变异规则，分别是高斯扰动、权重打乱、神经元激活状态反转、神经元效果阻断和神经元交换。下面对其进行简单介绍：

- 高斯扰动（GF），其示意图如5-18所示。在神经元的权重上添加一个符合高斯分布的噪声。

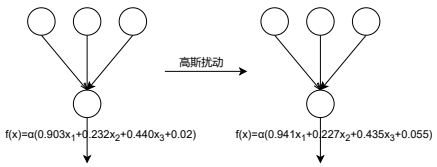

图 5-18: 高斯扰动变异规则示意图

- 权重打乱（WS），其示意图如5-19所示。针对不同输入的不同权重，交换打乱神经元。

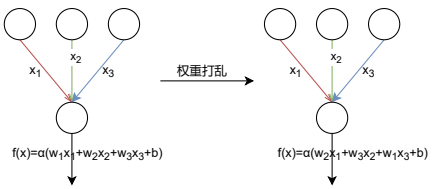

图 5-19: 权重打乱变异规则示意图

- 神经元激活状态反转（NAI），其示意图如5-20所示。在传递给激活函数之前，通过改变神经元输出值的符号来逆转神经元的激活状态。

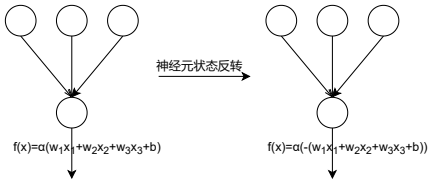

图 5-20: 神经元激活状态反转变异规则示意图

- 神经元效果阻断（NEB），其示意图如5-21所示。将部分神经元失效，使得下一层的输出为0。

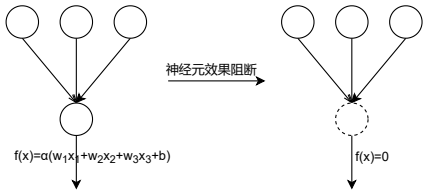

图 5-21: 神经元效果阻断变异规则示意图

- 神经元交换（NS），其示意图如5-22所示。在同一层中的多个神经元之间交换变异算子。

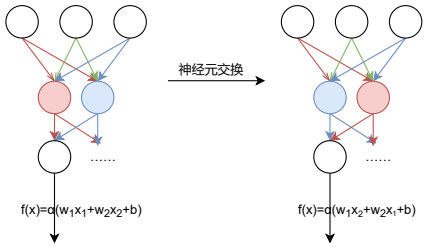

图 5-22: 神经元交换变异规则示意图

如5-23所示，是本系统中实现模型变异的关键代码。在本模块中，变异规则匹配是模型变异的关键组成部分。在每一次迭代变异时，系统将会选择本轮迭代的变异规则和种子模型。根据选定的种子模型，系统选择一个变异规则执行变异任务，调用对应的变异规则接口，从而对模型中的完整层或神经元进行变异。通过反复对比原种子模型和变异模型的不一致性大小，选择不一致程度大的模型作为下一次迭代的种子模型，最后得到指定数目的模型，终止变异任务。

```
def _LA_model_scan(model, new_layers, mutated_layer_indices=None):
    layer_utils = LayerUtils()
    layers = model.layers
    if mutated_layer_indices is None else mutated_layer_indices
    _assert_indices(positions_to_add, len(layers))
    insertion_points = {}
    available_new_layers = [layer for layer in
                            layer_utils.available_model_level_layers.keys()]
    if new_layers is None else new_layers
    for i, layer in enumerate(layers):
        if hasattr(layer, 'activation') and 'softmax' in
            layer.activation.__name__.lower():
            break
        if i in positions_to_add:
            for available_new_layer in available_new_layers:
                if layer_utils.is_input_legal[available_new_layer]\
                        (layer.output.shape):
                    if i not in insertion_points.keys():
                        insertion_points[i] = [available_new_layer]
                    else: insertion_points[i].append(available_new_layer)
    return insertion_points
```

图 5-23: 模型变异关键代码

5.3.3 定位缺陷的实现

如5-24所示，是本系统中实现定位缺陷的关键代码。系统完成变异任务后，用户可以选择该任务进行分析，从而定位深度学习框架下存在的潜在缺陷。通过对比模型在不同深度学习框架下的输出来定位框架中存在的潜在缺陷，并在定位过程中总结缺陷，从而生成分析文件保存在该变异任务的目录下。该模块的实现有利于快速发现和定位缺陷，提升深度学习框架测试的效率，同时方便用户查看分析文件。

```
def localize(mut_model_dir,……,localize_tmp_dir, report_dir,backends):
    identifier_split = select_idntfr.split("_")
    data_index = int(identifier_split[-1][5:])
    model_idntfr = "{}_{}".format(identifier_split[0], identifier_split[1])
    model_path = "{}/{}.h5".format(mut_model_dir, model_idntfr)
    for bk in backends:
        return_stats = os.system(……)
        if return_stats != 0:
            return
    model = keras.models.load_model(model_path,
        custom_objects=ModelUtils.custom_objects())
    for bk1, bk2 in combinations(backends, 2):
        local_res = {}
        local_res = get_outputs_divation_onbackends(model=model,
            backends=[bk1, bk2],model_idntfr=model_idntfr,
            local_res=local_res,data_index=data_index,
            localize_tmp_dir=localize_tmp_dir)
        report_path = os.path.join(report_dir,
            "{}_{}_{}_input{}.csv".format(model_idntfr,bk1,bk2, data_index))
        generate_report(local_res, report_path)
    del model
    K.clear_session()
```

图 5-24: 定位缺陷关键代码

5.4 缺陷分析模块

5.4.1 缺陷分析模块总体实现

在本模块中，系统主要负责对深度神经网络模型执行任务过程中产生的中间数据进行分析，并将分析结果反馈给用户。即本模块需要对任务执行过程中产生的日志文件进行分析，归纳总结深度学习框架中的潜在缺陷及其出现位

置和次数，最终反馈给用户。当生成的模型不存在时，系统会将相关分析一并删除。

由于人工分析效率低下且存在疏漏，因此本模块对深度神经网络模型生成过程中的中间数据进行自动化分析，通过提取关键信息对模型测试时所触发的框架缺陷进行归纳总结，从而得到较为详细和准确的分析结果。

本文将生成任务执行过程中抛出的异常分为以下五类：模块/属性缺失、参数不匹配、变量未定义、形状错误和 NaN 错误。模块/属性缺失的出现是由于引用的 Python 类字段或函数不存在；参数不匹配是由于调用了实际参数个数与函数定义的参数个数不一致；变量未定义是由于在变量定义之前引用；维度错误通常由于框架复杂的接口定义，致使模型构建时出现操作符参数形状不兼容；NaN 错误通常出现在计算的 loss 中，一种是在 loss 中计算后得到了 NaN 值，另一种是在更新网络权重等数据时出现。表 5-1描述了日志文件出现以上错误的关键信息。

表 5-1: 缺陷分类

缺陷类型	关键信息
模块/属性缺失	No module named <*> <*> has no attribute <*>
参数不匹配	<*> take exactly <*> arguments <*> given <*> got an unexpected keyword argument <*>
变量未定义	name <*> is not defined local variable <*> referenced before assignment
维度错误	Shape must be rank <*> but is rank <*> for <*> op:<*> with input shapes:<*> Cannot feed value of shape <*> for Tensor <*> which has shape <*> Dimensions must be equal,but are <*> and <*>for <*> (op:<*> with input shapes:<*>)
NaN	<*>nan<*>

缺陷分析模块的实现流程如图 5-25所示，主要完成了对产生数据的计算与分析。用户可以对生成任务中产生的数据执行分析任务，归纳总结生成模型的性能和所触发的框架缺陷，其数据主要来源于任务执行过程中产生的日志文件。

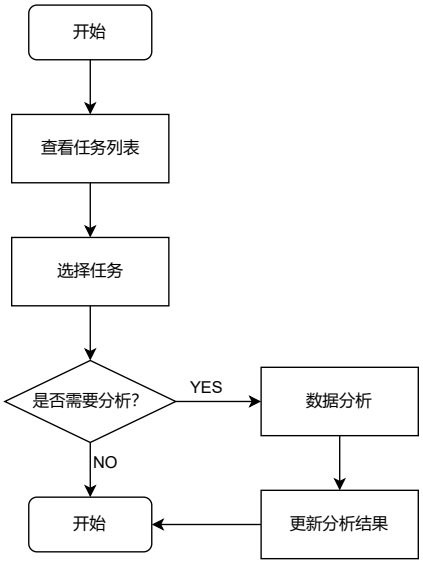

图 5-25: 缺陷分析模块流程图

系统首先会对生成过程中没有出现异常的模型进行分析，提取其中间数据并将其保存；同时分析出现异常的模型的日志文件，提取其中的异常信息进行分析和归纳总结，包括出现的错误及其对应的模型、出现的次数等数据。

缺陷分析模块实现了模型生成结果和缺陷信息的分析，其模块的类图如5-26所示。在本模块中，系统支持自动化地分析生成任务执行时产生的中间数据，提取关键数据并反馈给用户，从而使用户对生成的深度神经网络模型的质量有了量化的衡量；此外，系统支持对异常信息进行分析，将待测框架中的缺陷以更加直观的方式反馈给用户。

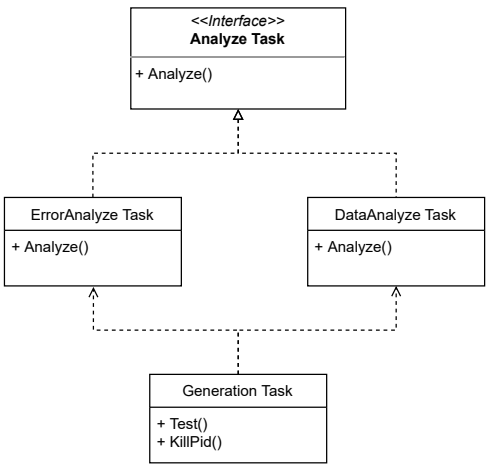

图 5-26: 缺陷分析模块类图

5.4.2 生成过程分析的实现

```
def calculate():
    data_list = ["loss", "acc", "val_loss", "val_acc"]
    loss = r"loss: \d\.\d+ - accuracy: \d\.\d+ - val_loss: \d\.\d+ -
        val_accuracy: \d\.\d+"
    with open("popular/tf2/log/129.log", "r", encoding='UTF-8') as f:
        perdatas =[]
        for line in f.readlines():
            perdata = []
            res = re.findall(loss, line)
            if (res):
                res_list = res[0].split("-")
                for item in res_list:
                    number = item.split(": ")[1]
                    perdata.append(float(number))
                perdatas.append(perdata)
    titles =["loss","acc","val_loss","val_acc"]
    excel_utils.write("popular/tf2/epoch/129.xlsx", perdatas, data_list)
```

图 5-27: 生成过程分析关键代码

如5-27所示，是本系统中分析生成任务中所产生的过程数据的关键代码。由于在对生成的深度神经网络模型进行验证时，产生了大量过程日志文件，其中记录了具体的过程数据，分析这些过程数据将对分析生成模型的效果至关重要。在本模块中，验证结果的数据来源于执行生成任务时产生的日志文件，由于其中详细记录了模型生成时的中间数据，因此需要被完整保存，以便进一步分析。

在本系统中，首先需要读取当前生成任务下产生的日志文件，这些日志文件会保存在不同生成任务所对应的日志目录下；接着，系统会对每一个日志文件进行扫描，读取其中的关键数据，并以统一的数据格式进行处理；最后，提取数据并保存在生成任务所对应的数据目录下，反馈给用户。

5.4.3 错误信息分类的实现

```
def getFile(location):
    path = os.path.join('\HTesting\main\predict', location, 'log')
    files = os.listdir(path)
    nan_error=0
    model=''
    for file in files:
        with open(os.path.join(path, file), "r", encoding='UTF-8') as f:
            for line in f.readlines():
                if 'nan' in line:
                    nan_error+=1
                    model+=file1+" "
                    break
                elif 'get_default_graph' in line:
                    version_error+=1
                    model += file1 +" "
                    break
                ......
    return nan_error, version_error ,dim_error, other_error ,model
```

图 5-28: 错误信息分类关键代码

如5-28所示，是本系统中错误信息分类的关键代码。错误信息分类中需要对生成任务中出现问题的日志文件进行遍历，从而对其中出现的关键错误信息进行归类和累加，计算得出不同框架下模型触发的不同缺陷的次数。在本模块中，不同的生成任务的计算过程均有所不同，由于采集了具体的过程数据，应当将任务执行时产生的中间数据保存在日志文件中，方便进一步分析。

在本系统中，首先需要读取当前生成任务下产生的日志文件，这些日志文件会保存在不同生成任务所对应的日志目录下；接着，系统会对每一个日志文件进行扫描，进行关键词比对，当读取到关键错误信息时，会进行错误信息出现次数的累加，并记录出现错误信息的模型；最后，将分析所得的模型名称、缺陷信息、框架名称及版本号等重要信息反馈给用户。

5.5 编译器测试模块

5.5.1 编译器测试模块总体实现

在编译器测试模块，系统主要负责随机生成用于测试编译器的程序，同时将生成的程序运行于不同版本的编译器下，对比其输出结果来定位缺陷。具体而言，本模块需要完成测试程序的生成和运行，运行过程中产生的日志数据将用于分析待测编译器中存在的潜在缺陷，最终生成分析报告。

如图5-29所示，是本模块的实现流程，主要实现了测试程序的迭代生成和运行。用户完成测试程序生成的配置后，系统将会自动化生成用于测试该编译器的测试程序。用户需要为测试任务的生成配置控制参数，包括任务名称和生成数量等。系统通过解析用户配置的控制参数，迭代生成测试程序，之后会将生成的程序作为测试数据进行测试，以触发深度学习编译器中存在的缺陷。

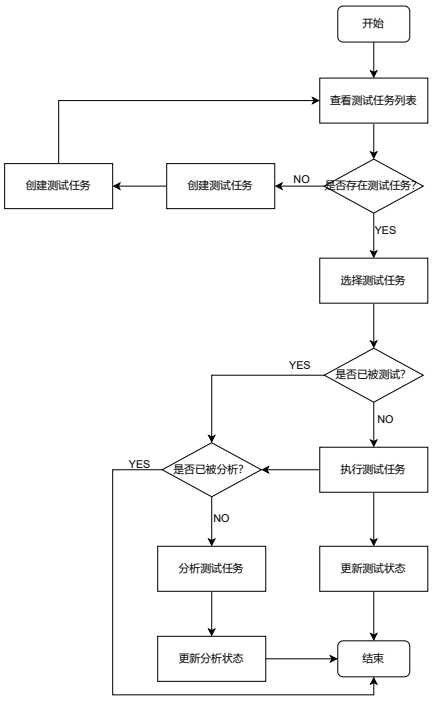

图 5-29: 编译器测试模块流程图

编译器测试模块主要实现测试程序的随机生成和执行，其模块的类图如5-30所示。在本模块中，系统对测试任务的生成和测试进行管理，实行了测试程序的迭代生成和运行，同时系统可以自动化地分析日志数据，最终生成分析报告反馈给用户。

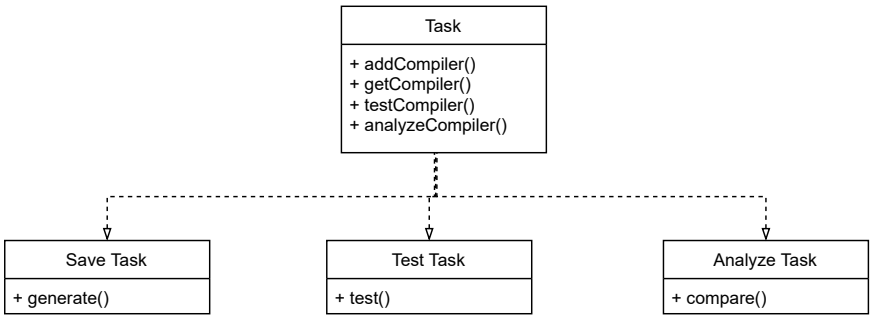

图 5-30: 编译器测试模块类图

5.5.2 生成测试程序的实现

```
def generate(name,num):
    i=0
    while i<num:
        f = open('\main\\ test_compiler \\byproduct\%s\%s.py'%(name,i), 'w')
        f.write('\n')
        print(Magenta('len( ingredient ) = ' + str(len( ingredient ))))
        id = random.randint(0, len( ingredient )-1)
        print(Yellow('id = ' + str(id)))
        if isinstance( ingredient [id], pFunc):
            print(Yellow(' ingredient = ' + str( ingredient [id].funcName)))

        if isinstance( ingredient [id], pFunc):
            generateFunc( ingredient [id], f, True)

        elif isinstance( ingredient [id], pWith):
            generateWith( ingredient [id], f, True)
        else:
            raise Exception('Unexpected element of ingredient ')
        f.close()
        i+=1
```

图 5-31: 生成测试程序关键代码

如5-31所示，是本模块实现测试程序生成的关键代码。在本模块中，系统将根据用户配置的测试任务，迭代生成大量用于测试编译器的程序。通过随机生成由不同接口和参数组成的小程序，同时将小程序运行于不同版本的深度学习编译器下，比较运行后的输出，从而发现和定位编译器中存在的潜在缺陷，以提高深度学习编译器的质量。通过该模块的实现，可以有效提高深度学习编译器的测试质量和效率。

系统首先会生成语法规则，包括生成程序将会调用的接口、参数等关键信息；接下来，系统将根据用户配置的生成参数，同时基于该语法规则构建符合规则的程序，这些程序所调用的接口和参数不一，将尽可能多地覆盖深度学习编译器所提供的接口，从而在不同版本的编译器下运行时能够触发更多缺陷，提高测试质量。

5.5.3 日志分析的实现

如5-32所示，是本模块实现分析测试任务日志信息的关键代码。在本模块中，系统执行测试任务后，会生成不同版本下执行的日志信息，通过对比其输出结果来定位缺陷，如果输出结果不一致，则认为深度学习编译器中存在潜在缺陷。最终系统会根据对比结果生成分析报告反馈给用户，用户可以在系统中查看指定测试任务的缺陷分析报告，以便进行后续定位和修复缺陷工作。该模块的实现有效提高了人工筛查日志的效率，降低分析成本。

```
def compare(name):
    path = os.path.join('log','0.8')
    path_ = os.path.join('log', '0.9')
    files = os. listdir (path)
    f=open('error.txt', 'a+',encoding='UTF-8')
    for file in files :
        if os.path. isfile (os.path.join(path, file)) and
            file .endswith('.log'):
            cmpfile = cmpFile(os.path.join(path, file), os.path.join(path_,
                file))
            difflines = cmpfile.compare()
```

图 5-32: 日志分析关键代码

5.6 本章小结

本章对系统的环境管理模块、生成任务管理模块、变异任务管理模块、缺陷分析模块和编译器测试模块的总体流程实现与模块内的关键技术实现进行详细阐述。基于模块的流程图和核心类图，详细描述了各个模块的详细设计与总体实现，同时通过关键代码对各个模块内关键功能的实现进行了具体说明。

第六章　深度学习框架自动化测试系统的测试与分析

本章主要对基于规则变异的深度学习框架自动化测试系统进行系统测试，包括测试准备、功能性测试、非功能性测试和自动化测试技术分析。

6.1　系统测试

6.1.1　测试目标

在正式部署运行该系统前，将对该系统进行有效的测试。为了对深度学习框架自动化测试系统开展测试，本文针对系统设计了测试用例并执行，在执行过程中检验系统是否实现了核心功能模块，满足第三章的需求分析，主要包括创建环境、查看环境信息列表、生成模型、查看生成任务列表、创建变异任务等功能点，并满足了非功能需求。最后，本文将对提出的深度学习框架自动化测试技术进行有效性测试，从而验证本系统的有效性。

6.1.2　测试环境

表 6-1: 系统测试环境

字段	测试环境
服务器	物理服务器 64GB 内存
操作系统	Ubuntu 16.04
数据库	Mysql 5.5.62、Redis 3.2.1
Python 库	Python 3.6.2、Tensorflow 1.14.0、Keras 2.4.0
测试软件	chrome 浏览器

如表6-1所示为本系统部署运行的环境配置信息，基于规则变异的深度学习框架自动化测试系统在GNU/Linux系统上运行的，该系统使用Linux内核，版本为4.15.0，配备一个6核3.60GHz英特尔酷睿CPUi7-6850K和64GB内存，并配备一个英伟达公司GP102GPU。CUDA版本为10.2，Anaconda版本为4.5.11，Python版本为3.6。

6.1.3 功能性测试

根据需求分析中的用例描述，对本系统进行功能性测试。其中，测试用例主要由给定输入和期望输出构成，通过判断实际结果是否符合预期，来衡量系统功能的正确性，进而不断完善系统，以确保系统功能达到预期效果。本部分主要从创建环境、查看环境信息列表、创建生成任务、生成模型、终止生成、删除生成任务、查看生成任务列表、创建变异任务、模型变异、定位缺陷、删除变异任务、查看变异任务列表、查看缺陷分析列表、缺陷分析、查看编译器测试列表、创建测试任务、测试编译器和分析日志开展功能性测试，针对不同的业务逻辑，开展相应的测试，能充分覆盖用例描述对应的功能点。

表6-2描述了创建环境的测试用例，目的是为了验证创建虚拟环境的流程是否正确，其测试项包括文件上传方式、上传文件类型、输入不符合系统规范等，验证依赖安装文件能否被正确上传，并根据文件自动创建环境和安装依赖包。结果显示测试用例能够全部通过，符合预期需求。

表 6-2: 创建环境测试用例

测试项	操作/输入	预期结果	测试结果
创建环境	1. 用户点击创建虚拟环境按钮 2. 用户点击文件上传 3. 用户将本地 txt 文件拖拽到上传框处 4. 用户点击文件上传框并选择本地 txt 文件 5. 用户选择非 txt 文件进行上传 6. 用户点击上传确认按钮	1. 系统弹出上传文件的提示框 2. 系统显示文件上传框 3. 系统显示文件上传成功 4. 系统显示文件选择框并显示文件上传成功 5. 上传框无法显示不支持类型的文件 6. 系统执行创建环境和安装依赖包任务，进入到环境信息页面	通过

表6-3描述了查看环境信息列表的测试用例，主要测试环境信息页面的相关操作是否具有正确输出。结果显示测试用例能够全部通过，符合预期需求。

表 6-3: 查看环境信息列表测试用例

测试项	操作/输入	预期结果	测试结果
查看环境信息列表	1. 用户点击菜单栏进入环境配置页面	1. 系统显示已经创建的虚拟环境	通过

表6-4描述了创建生成任务的测试用例，目的是为了验证创建生成任务的流程是否正确，其测试项包括文件上传方式、上传文件类型、保存生成任务等，验证参数配置文件能否被正确上传，并根据文件创建生成任务。结果显示测试用例能够全部通过，符合预期需求。

表 6-4: 创建生成任务测试用例

测试项	操作/输入	预期结果	测试结果
创建生成任务	1. 用户点击创建生成任务按钮 2. 用户点击文件上传 3. 用户将本地 conf 文件拖拽到上传框处 4. 用户点击文件上传框并选择本地 conf 文件 5. 用户选择非 conf 文件进行上传 6. 用户点击上传确认按钮	1. 系统弹出上传文件的提示框 2. 系统显示文件上传框 3. 系统显示文件上传成功 4. 系统显示文件选择框并显示文件上传成功 5. 上传框无法显示不支持类型的文件 6. 系统执行创建生成任务，进入到模型生成页面	通过

表6-5描述了生成模型的测试用例，系统将自动化生成深度神经网络模型。在每次迭代生成深度神经网络时，用户通过配置参数，控制生成任务的名称、数据集、迭代次数等。用户配置成功后，系统会开始根据用户设置的控制参数实现深度神经网络模型的自动生成。生成任务执行完毕后，系统会反馈用户生成的结果。结果显示测试用例能够全部通过，符合预期需求。

表 6-5: 生成模型测试用例

测试项	操作/输入	预期结果	测试结果
生成模型	1. 用户点击生成模型按钮 2. 用户选择需要生成的任务，点击生成按钮	1. 进入到模型生成界面 2. 系统执行生成任务，并返回生成结果	通过

表6-6描述了终止生成的测试用例，涉及到管理深度神经网络模型生成的任务，即终止正在执行的生成任务。用户可以根据需求选择需要被终止的生成任务，系统后台将会终止相应进程。系统终止成功后，会将信息反馈给用户。结果显示测试用例能够全部通过，符合预期需求。

表 6-6: 终止生成测试用例

测试项	操作/输入	预期结果	测试结果
终止生成	1. 用户点击模型生成按钮 2. 用户选择需要终止生成的任务，并点击终止按钮	1. 进入到模型生成界面 2. 系统开始终止生成任务，并提示终止成功	通过

表6-7描述了删除生成任务的测试用例，涉及到删除已经创建完成的生成任务。用户选择需要删除的生成任务后，系统会自动删除该生成任务信息和对应文件。结果显示测试用例能够全部通过，符合预期需求。

表 6-7: 删除生成任务测试用例

测试项	操作/输入	预期结果	测试结果
删除 生成任务	1. 用户点击模型生成按钮 2. 用户点击删除按钮	1. 进入到模型生成界面 2. 系统删除生成任务信息及对应文件	通过

表6-8描述了查看生成任务列表的测试用例，主要测试模型信息页面的相关操作是否具有正确输出。结果显示测试用例能够全部通过，符合预期需求。

表 6-8: 查看生成任务列表测试用例

测试项	操作/输入	预期结果	测试结果
查看生成任务列表	1. 用户点击菜单栏进入模型生成页面	1. 系统显示已经创建的生成任务	通过

表6-9描述了创建变异任务的测试用例，目标是为了验证创建变异任务的流程是否正确，其测试项包括文件上传方式、上传文件类型、保存变异任务等，验证参数配置文件能否被正确上传，并根据文件创建变异任务。结果显示测试用例能够全部通过，符合预期需求。

表 6-9: 创建变异任务测试用例

测试项	操作/输入	预期结果	测试结果
创建变异任务	1. 用户点击创建变异任务按钮 2. 用户点击文件上传 3. 用户将本地 conf 文件拖拽到上传框处 4. 用户点击文件上传框并选择本地 conf 文件 5. 用户选择非 conf 文件进行上传 6. 用户点击上传确认按钮	1. 系统弹出上传文件的提示框 2. 系统显示文件上传框 3. 系统显示文件上传成功 4. 系统显示文件选择框并显示文件上传成功 5. 上传框无法显示不支持类型的文件 6. 系统执行创建变异任务，进入到变异任务页面	通过

表6-10描述了模型变异的测试用例，涉及到对生成的深度神经网络模型进行变异的操作，以最大程度地触发深度学习框架中的缺陷。系统需要用户上传配置文件，用以控制每一次迭代中的变异算法、变异因子、模型地址等。在用户完成配置文件上传后，系统将自动创建变异任务，同时根据用户设置的控制参数变异深度神经网络模型。任务成功执行后，系统会返回给用户任务执行结果。结果显示测试用例能够全部通过，符合预期需求。

表 6-10: 模型变异测试用例

测试项	操作/输入	预期结果	测试结果
模型变异	1. 用户点击模型变异按钮 2. 用户选择需要执行的变异任务，并点击执行开始按钮	1. 进入到模型变异界面 2. 系统执行变异任务	通过

表6-11描述了定位缺陷的测试用例，涉及到对已经完成变异的模型进行检测，定位框架中存在的缺陷。系统会自动定位缺陷，并生成缺陷文件等分析文档。任务执行完成之后，系统会返回给用户深度学习框架中的缺陷信息。结果显示测试用例能够全部通过，符合预期需求。

表 6-11: 定位缺陷测试用例

测试项	操作/输入	预期结果	测试结果
定位缺陷	1. 用户点击模型变异按钮 2. 用户点击分析按钮	1. 进入到模型变异界面 2. 系统执行缺陷定位任务，保存分析结果文件	通过

表6-12描述了删除变异任务的测试用例，涉及到删除已经创建的变异任务。用户选择需要删除的变异任务后，系统会自动删除该变异任务信息和对应参数配置文件。结果显示测试用例能够通过，符合预期需求。

表 6-12: 删除变异任务测试用例

测试项	操作/输入	预期结果	测试结果
删除 变异任务	1. 用户点击模型变异按钮 2. 用户点击删除按钮	1. 进入到模型变异界面 2. 系统删除变异任务信息及对应文件	通过

表6-13描述了查看变异任务列表的测试用例，涉及到查看所有创建好的

变异任务。用户可以查看当前变异任务的参数配置信息，如模型个数、变异方法、变异比例等。结果显示测试用例能够全部通过，符合预期需求。

表 6-13: 查看变异任务列表测试用例

测试项	操作/输入	预期结果	测试结果
查看变异任务列表	1. 用户点击菜单栏进入模型变异页面	1. 系统显示创建完成的变异任务列表	通过

表6-14描述了查看缺陷分析列表的测试用例，涉及到用户可以获取缺陷分析的结果，包括其生成任务的模型信息、框架信息和缺陷定位信息等。结果显示测试用例能够全部通过，符合预期需求。

表 6-14: 查看缺陷分析列表测试用例

测试项	操作/输入	预期结果	测试结果
查看缺陷分析列表	1. 用户点击菜单栏进入数据处理页面	1. 系统显示框架缺陷列表	通过

表6-15描述了缺陷分析的测试用例，涉及到用户可以获取生成深度神经网络模型的过程数据和日志文件。系统会遍历信息，筛选出存在问题的数据和模型，并对每一个问题的出现进行归纳总结，得到缺陷分析结果，最终反馈给用户。设计的测试用例执行后全部通过，表明该功能符合预期需求。

表 6-15: 缺陷分析测试用例

测试项	操作/输入	预期结果	测试结果
缺陷分析	1. 用户点击菜单栏进入数据处理页面 2. 用户选择需要分析的生成任务，并点击分析按钮	1. 系统显示框架缺陷列表 2. 系统执行分析任务，完成后显示分析结果	通过

表6-16描述了查看编译器测试列表的测试用例，涉及到查看所有创建完成的测试任务。用户可以查看当前测试任务的具体信息，如任务名称、生成数量、测试状态等。结果显示测试用例能够全部通过，符合预期需求。

表 6-16: 查看编译器测试列表测试用例

测试项	操作/输入	预期结果	测试结果
查看编译器测试列表	1. 用户点击菜单栏进入测试任务页面	1. 系统显示创建完成的测试任务列表	通过

表6-17描述了创建测试任务的测试用例，主要验证创建测试任务的流程是否符合预期，其测试项包括配置测试任务、保存测试任务等。结果显示测试用例能够全部通过，符合预期需求。

表 6-17: 创建测试任务测试用例

测试项	操作/输入	预期结果	测试结果
创建测试任务	1. 用户点击菜单栏进入创建任务页面 2. 用户配置完成，并点击确认按钮	1. 系统显示配置表单 2. 系统执行创建测试任务	通过

表 6-18: 测试编译器测试用例

测试项	操作/输入	预期结果	测试结果
测试编译器	1. 用户点击测试任务按钮 2. 用户选择需要测试的测试任务，并点击测试按钮	1. 进入到测试任务界面 2. 系统开始执行测试任务	通过

表6-18描述了测试编译器的测试用例，涉及到测试已自动生成的程序。本系统已经生成了大量用于测试深度学习编译器的程序，因此，需要运行所生成的程序，用于测试深度学习编译器。结果显示测试用例能够全部通过，符合预期需求。

表6-19描述了分析日志的测试用例，涉及到用户可以获取已经被运行的测试程序的日志文件。系统会遍历不同版本的运行信息，通过对比输出结果来判断编译器中是否存在缺陷。结果显示测试用例能够全部通过，符合预期需求。

表 6-19: 分析日志测试用例

测试项	操作/输入	预期结果	测试结果
分析日志	1. 用户点击菜单栏进入测试任务页面 2. 用户选择需要分析的测试任务，并点击分析按钮	1. 系统显示测试任务列表 2. 系统执行分析任务	通过

6.1.4 非功能性测试

根据深度学习框架自动化测试系统需求分析与概要设计中的非功能需求，本章将对系统开展非功能性测试，分别从时间特性和并发性两个维度进行。非功能性测试与功能性测试同样重要，关系到用户的使用体验感。同时，本文采用 Django 框架保证了系统的可扩展性；环境管理模块支撑了本系统不同任务的顺利执行；缺陷分析模块保证了本系统的可靠性。

系统使用 Fiddle 工具测试页面的响应时间，即一个代理服务器应用程序用于调试 HTTP。测试页面包括创建环境、查看环境信息列表、创建生成任务、终止生成、删除生成任务、查看生成任务列表、创建变异任务、删除变异任务、缺陷分析、查看缺陷分析列表、查看测试任务列表、创建测试任务、日志分析和测试编译器的界面响应时间开展了测试。页面平均、最大和最小响应时间均通过执行 100 次的响应时间获得，如表6-20所示是本次非功能测试的统计结果，测试证明本系统符合时间特性，能够快速响应用户操作，提高了用户的使用体验感。

表 6-20: 系统界面响应时间统计表

页面	平均响应时间	最大响应时间	最小响应时间	测试结果
创建环境	4209ms	7726ms	871ms	通过
查看环境信息列表	1459ms	4964ms	16ms	通过
创建生成任务	1487ms	2108ms	1734ms	通过
终止生成	723ms	1048ms	918ms	通过
删除生成任务	612ms	1239ms	896ms	通过
查看生成任务列表	114ms	260ms	82ms	通过
创建变异任务	2639ms	8725ms	340ms	通过
删除变异任务	2712ms	4750ms	336ms	通过
缺陷分析	2389ms	2843ms	2532ms	通过
查看缺陷分析列表	156ms	243ms	96ms	通过
查看测试任务列表	1649ms	3345ms	30ms	通过
创建测试任务	4948ms	8725ms	340ms	通过
日志分析	2135ms	2368ms	1989ms	通过
测试编译器	1142ms	1821ms	782ms	通过

同时，本文通过 Jmeter[51] 对系统进行压力测试，并发量设置为 60，并且本系统的所有核心接口都被覆盖。最终测试结果反馈均正确，测试证明本系统满足并发性要求。

6.2 实验评估

在对系统业务流程的正确性进行验证后，需要通过多个控制变量实验对系统的实际效果进行验证和评估，主要包括规则变异带来的测试数据生成方式能否保证生成的模型具有动态可用性，本系统中提出的框架测试方法提升触发缺陷的可能性，以及与其它主流框架测试方法的效果相比是否有所提升等问题。

根据以上问题，本节拟通过两个实验来验证该深度学习框架测试技术及系统的有效性和优越性，包括框架测试方法设置实验和缺陷触发效果对比实验。

6.2.1 实验对象

深度学习框架各异，本系列实验中仅以 Tensorflow2.0.0 和 Pytorch1.8.1 作为实验对象。实验中选取 3 种数据集，分别是 minist、fashion_mnist 和 cifar10 数据集，每个数据集对应一个生成任务，epochs 设置为 5，batch_size 设置为 100。

6.2.2 实验设计及结果分析

下面分别对两个实验的具体设计进行介绍。

（1）框架测试方法设置实验

在模型进行学习时，深度学习框架提供了各种运算包括输入输出数据处理、前向传播和反向传播梯度等，具有大量计算量，且计算难度也比较大。因此为了探索深度学习框架影响深度学习模型训练的程度，本文将通过分析生成模型的缺陷触发情况进行研究。

表 6-21: 模型生成结果统计表

任务	对比框架	数据集	运行模式	异常/生成
1	Tensorflow2.0.0 Pytorch1.8.1	mnist	GPU	2/169 1/169

2	Tensorflow2.0.0 Pytorch1.8.1	fashion_mnist	GPU	2/118 0/118
3	Tensorflow2.0.0 Pytorch1.8.1	cifar10	GPU	5/68 3/68

表6-21记录了三个生成任务中测试数据生成情况，三个生成任务分别对应minist、fashion_mnist 和 cifar10 数据集，采用 Tensorflow2.0.0 和 Pytorch1.8.1 两种深度学习框架生成相同模型结构的模型。从表中可以看出，总共生成了 710 个深度神经网络模型结构，能够训练成 694 个深度神经网络模型。

由于随机生成的深度神经网络模型能够尽可能地覆盖到没有使用过的模型架构，因此其触发深度学习框架中存在缺陷的可能性更大。在三个生成任务中，触发到了 13 个由于框架的原因造成训练失败，包括 7 个维度问题和 2 个扩张问题。具体而言，有 9 个 TensorFlow 缺陷和 4 个 Pytorch 缺陷。其生成异常类型总数如图6-1所示。

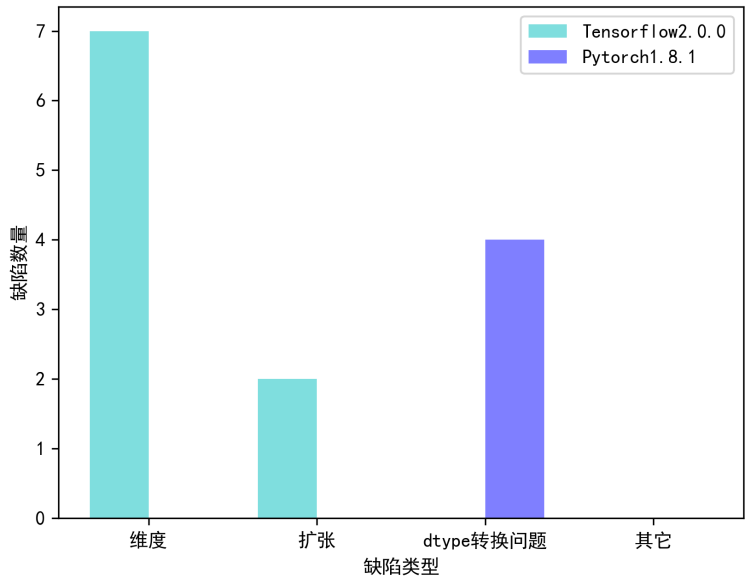

图 6-1: 生成异常类型总数

深度学习框架中的部分操作符由于设计问题，可能会使模型在训练过程中出现崩溃现象。如图6-2所示为生成模型的内部结构，在计算过程中，我们发现 *Conv2dTranspose* 算子的设计存在缺陷。当使用 *Conv2dTranspose* 中 $padding = SAME$ 时，TensorFlow 将会自动设置 $pad = kernel\_size \div 2$，同时 $out\_padding = stride - 1$，从而致使与 $padding = SAME$ 发生冲突，出现崩溃问题。

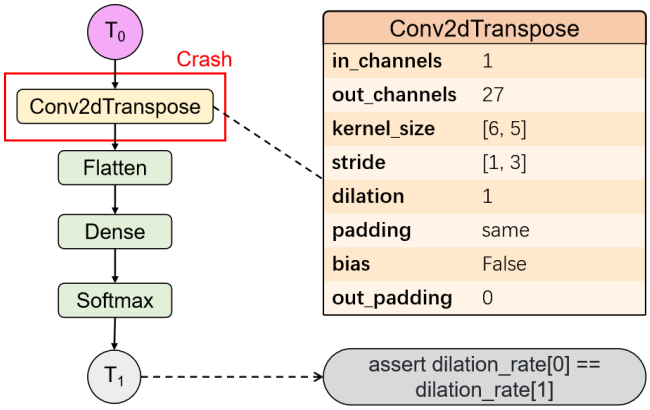

图 6-2: 典型缺陷示意图

（2）缺陷触发效果对比实验

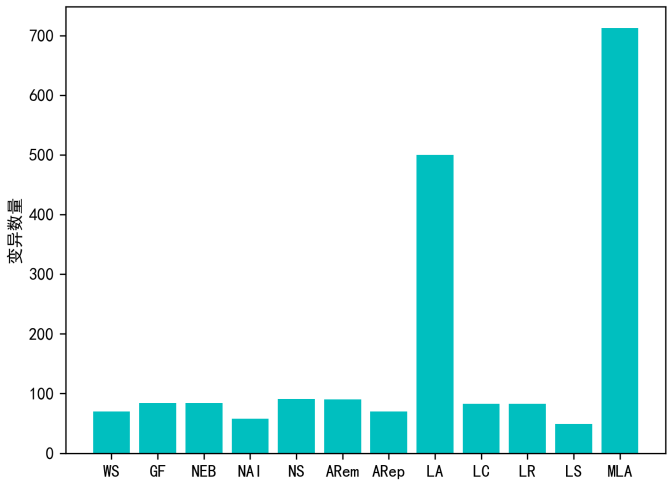

图 6-3: 模型变异情况

该实验选择了目前前沿的框架测试技术 LEMON 作为对比实验，以验证本文提出的深度学习框架系统是否在触发缺陷方面与主流框架测试技术相比具

有优越性。该实验使用 LEMON 将生成的深度神经网络模型为种子模型进行深度学习框架测试，对 340 个 TensorFlow 框架下的模型进行变异，每个模型变异阈值设置为 0.7。通过实验最终发现了发现了 2 个 crash 问题和 1 个 NaN（not a number，非实数）问题。其具体变异情况如图 6-3所示。

表 6-22中记录了该实验得到的结果数据。从表中可以看出，相较于 LEMON，通过规则变异生成的深度神经网络模型可以更大程度地触发框架缺陷，提高测试数据的数量和质量。

表 6-22: 框架测试技术评估实验结果

框架测试工具	维度问题	扩张问题	NaN 问题	总数
LEMON	2	0	1	3
深度学习框架自动化测试系统	7	2	3	12

从以上两个方面的实验结果整体表现来看，深度学习框架自动化测试系统能够在提高测试数据数量的同时，更大程度地保证测试数据的质量，以提高触发框架缺陷的可能性。因此该系统在深度学习框架测试方面的技术是有效且优越的。

6.3 本章小结

本章主要对基于规则变异的深度学习框架自动化测试系统进行功能性测试、非功能性测试和关键技术的测试，并具体阐述了测试目标、测试环境、测试设计和测试情况。根据第三章的需求分析与概要设计，本文进行了功能性测试以及设计测试用例，验证了系统的完整性。同时，为了保证用户使用的体验感，本文开展了非功能性测试，证明系统的可靠性。最后，本文将学术界目前前沿的研究成果与系统核心技术进行对比，验证了技术的有效性。

第七章　总结与展望

7.1　总结

深度学习框架为深度神经网络模型的构建提供算子接口，实现快速、高效构建。同时，深度神经网络模型的质量也与深度学习框架所提供的接口质量息息相关。为了避免深度神经网络模型引入深度学习框架中潜在的缺陷，以及提高深度学习框架的质量，保证模型的安全运行，本文研究了基于规则变异的深度学习框架自动化测试技术，以尽可能多地发现框架中存在的缺陷，实现深度学习框架测试的高效和有效。

在本文中，首先从项目背景和国内外研究现状两方面介绍了基于规则变异的深度学习框架自动化测试系统，接着介绍了本文所涉及的技术要点，对深度学习技术、深度学习框架、深度神经网络模型生成技术、深度学习框架测试技术和 Django 框架开展了较为详尽的调研，为本文的实现起到了技术支撑的作用。

接下来，本文对深度学习框架自动化测试系统进行需求分析与设计，并结合用例图和用例描述做出具体阐述。同时，本文结合“4+1 视图”模型对自动化测试系统进行了总体设计，包含逻辑、进程、开发和部署四个角度。根据需求分析的结果，本文进一步将系统分为环境管理模块、生成任务管理模块、变异任务管理模块、缺陷分析模块和编译器测试模块共五个模块，并针对各个模块的实现进行了详尽的功能设计。

此外，本文对提出的基于规则变异的深度神经网络模型生成技术进行具体阐述。主要介绍了其实现的整体架构，并从该技术的两个重要部分出发，具体介绍了深度神经网络模型迭代生成的算法实现流程，以及安全检查的关键代码。通过以上两个重要部分的实现，最终构建完整的深度神经网络模型。

最终，基于核心类图和必要的关键代码，分别阐述了五个模块的详细设计与具体实现。为了验证系统的可用性，本文对深度学习框架自动化测试系统开展了功能性和非功能性测试，以及对核心技术进行了测试，测试结果进一步证明了本系统的可靠性和安全性。

7.2 展望

在本文中，基于规则变异的深度学习框架自动化测试系统得到了初步的实现，确保了系统能够完成环境创建、模型生成、模型变异和缺陷分析全流程的测试工作。尽管如此，本系统仍然存在一些需要改进的地方，未来希望从以下几个方面加以完善：

首先，本系统实现了在自动化生成深度神经网络模型的基础上，对已生成的模型进行变异，以尽可能多地发现框架中存在的缺陷。但是该变异并不能保证生成的模型架构合规，因此在进一步完善过程中，拟考虑构建更加全面的语法规则和变异方法，确保生成模型的有效性。

其次，由于深度神经网络模型训练时的时间和空间代价较高，容易造成资源的长期占用。在进一步地完善过程中，拟实现训练时的捕获，通过对训练状态的捕获和分析，及时释放资源，提高生成任务的效率。

最后，在本系统的模型变异中，用户无法知道每一次迭代时生成变异模型的内部结构。在进一步地完善过程中，拟考虑实现模型对比功能，将变异时模型产生的变化进行归纳总结，并将其反馈给用户，以便后续实验的分析。

致　　谢

时光荏苒，白驹过隙，研究生的生活悄然走到了尾声。转眼间来到南京大学 iSE 实验室已近两年。在这里我度过了难忘的时光，遇到了很多优秀的人，在他们的帮助关心下，我收获了宝贵的知识、技能和经验，并且顺利完成了毕业设计的系统开发和论文编写。

有太多人需要感谢，感谢这两年短暂时光里的相遇，很荣幸共同度过了一段美好时光。首先我想感谢我的导师陈振宇老师和房春荣老师，很荣幸能在学术生涯的懵懂时期加入 iSE 实验室。正是因为老师们的指导和帮助，我在学业和生活上有了巨大的进步，逐渐成长为更好的自己。其次我要感谢 iSE 实验室的刘佳玮学姐，从进组至今，一步步引导我完成实验任务，在毕业论文编写阶段，学姐为我提供了思路和方向，提出了很多宝贵的意见。最后我还要感谢章许帆博士、虞圣呈博士、杨郁芩、侯韵晗、贺璐等实验室的其他同学，在我刚进组时热情地为我答疑解惑，并在学习和生活上给了我很多照顾，让我度过了惬意快乐的校园生活。

我时常感到自己的幸运，让我来到这里遇到如此多优秀且善良的人，庆幸有他们作为我的师长、榜样和朋友。而今即将奔赴下一场山海，希望我们都有美好的未来。

参考文献

[1] HINTON G E, SALAKHUTDINOV R R. Reducing the dimensionality of data with neural networks[J]. science, 2006, 313(5786): 504–507.

[2] SUN Y, CHEN Y, WANG X, et al. Deep learning face representation by joint identification-verification[J]. Advances in neural information processing systems, 2014, 27.

[3] CHEN C, SEFF A, KORNHAUSER A, et al. Deepdriving: Learning affordance for direct perception in autonomous driving[C] // Proceedings of the IEEE international conference on computer vision. 2015: 2722–2730.

[4] GRAVES A, MOHAMED A-R, HINTON G. Speech recognition with deep recurrent neural networks[C] // 2013 IEEE international conference on acoustics, speech and signal processing. 2013: 6645–6649.

[5] 张顺, 龚怡宏, 王进军. 深度卷积神经网络的发展及其在计算机视觉领域的应用 [J]. 计算机学报, 2019, 42(3): 453–482.

[6] 付文博, 孙涛, 梁藉, et al. 深度学习原理及应用综述 [J]. 计算机科学, 2018, 45(6A): 11–15.

[7] 徐冰冰, 岑科廷, 黄俊杰, et al. 图卷积神经网络综述 [J]. 计算机学报, 2020, 43(5): 755–780.

[8] 白琮, 黄玲, 陈佳楠, et al. 面向大规模图像分类的深度卷积神经网络优化 [J]. Journal of Software, 2018, 29(4).

[9] PHAM H V, LUTELLIER T, QI W, et al. CRADLE: cross-backend validation to detect and localize bugs in deep learning libraries[C] // 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). 2019: 1027–1038.

[10] GUO Q, CHEN S, XIE X, et al. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms[C] // 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2019 : 810 – 822.

[11] ZHANG Y, CHEN Y, CHEUNG S-C, et al. An empirical study on TensorFlow program bugs[C] // Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2018 : 129 – 140.

[12] HUMBATOVA N, JAHANGIROVA G, BAVOTA G, et al. Taxonomy of real faults in deep learning systems[C] // Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020 : 1110 – 1121.

[13] MA L, ZHANG F, SUN J, et al. Deepmutation: Mutation testing of deep learning systems[C] // 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE). 2018 : 100 – 111.

[14] WANG J, DONG G, SUN J, et al. Adversarial sample detection for deep neural network through model mutation testing[C] // 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). 2019 : 1245 – 1256.

[15] GUO Q, XIE X, LI Y, et al. Audee: Automated testing for deep learning frameworks[C] // 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2020 : 486 – 498.

[16] WANG Z, YAN M, CHEN J, et al. Deep learning library testing via effective model generation[C] // Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020 : 788 – 799.

[17] LIU C, LU J, LI G, et al. Detecting TensorFlow Program Bugs in Real-World Industrial Environment[C] // 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2021 : 55 – 66.

[18] ABADI M, BARHAM P, CHEN J, et al. Tensorflow: A system for large-scale machine learning[C] // 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16). 2016 : 265 – 283.

[19] SEIDE F, AGARWAL A. CNTK: Microsoft's Open-Source Deep-Learning Toolkit[J]. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[20] COLLOBERT R, KAVUKCUOGLU K, FARABET C. Torch7: A matlab-like environment for machine learning[C] // BigLearn, NIPS workshop. 2011.

[21] AL-RFOU R, ALAIN G, ALMAHAIRI A, et al. Theano: A Python framework for fast computation of mathematical expressions[J]. arXiv e-prints, 2016: arXiv – 1605.

[22] JIA Y, SHELHAMER E, DONAHUE J, et al. Caffe: Convolutional architecture for fast feature embedding[C] // Proceedings of the 22nd ACM international conference on Multimedia. 2014: 675 – 678.

[23] ALBAWI S, MOHAMMED T A, AL-ZAWI S. Understanding of a convolutional neural network[C] // 2017 international conference on engineering and technology (ICET). 2017: 1 – 6.

[24] YASAKA K, AKAI H, KUNIMATSU A, et al. Deep learning with convolutional neural network in radiology[J]. Japanese journal of radiology, 2018, 36(4): 257 – 272.

[25] INDOLIA S, GOSWAMI A K, MISHRA S P, et al. Conceptual understanding of convolutional neural network-a deep learning approach[J]. Procedia computer science, 2018, 132: 679 – 688.

[26] CHEN T, MOREAU T, JIANG Z, et al. {TVM}: An Automated {End-to-End} Optimizing Compiler for Deep Learning[C] // 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018: 578 – 594.

[27] RAMPASEK L, GOLDENBERG A. Tensorflow: Biology's gateway to deep learning?[J]. Cell systems, 2016, 2(1): 12 – 14.

[28] GULLI A, PAL S. Deep learning with Keras[M]. [S.l.]: Packt Publishing Ltd, 2017.

[29] 杨丽, 吴雨茜, 王俊丽, et al. 循环神经网络研究综述 [J]. 计算机应用, 2018, 38(A02): 1 – 6.

[30] 王坤峰, 苟超, 段艳杰, et al. 生成式对抗网络 GAN 的研究进展与展望 [J]. 自动化学报, 2017, 43(3): 321 – 332.

[31] 周飞燕, 金林鹏, 董军, et al. 卷积神经网络研究综述 [J]. 计算机学报, 2017, 40(6): 1229 – 1251.

[32] 刘全, 翟建伟, 章宗长, et al. 深度强化学习综述 [J]. 计算机学报, 2018, 41(1): 1 – 27.

[33] JIA Y, HARMAN M. An analysis and survey of the development of mutation testing[J]. IEEE transactions on software engineering, 2010, 37(5): 649 – 678.

[34] SCHULER D, ZELLER A. Javalanche: Efficient mutation testing for Java[C] // Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. 2009: 297 – 298.

[35] NAEEM M R, LIN T, NAEEM H, et al. Scalable mutation testing using predictive analysis of deep learning model[J]. IEEE Access, 2019, 7: 158264 – 158283.

[36] 孟子尧, 谷雪, 梁艳春, et al. 深度神经架构搜索综述 [J]. 计算机研究与发展, 2021, 58(1): 22.

[37] ELSKEN T, METZEN J H, HUTTER F. Neural architecture search: A survey[J]. The Journal of Machine Learning Research, 2019, 20(1): 1997 – 2017.

[38] CHEN T, LI M, LI Y, et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems[J]. arXiv preprint arXiv:1512.01274, 2015.

[39] MCKEEMAN W M. Differential testing for software[J]. Digital Technical Journal, 1998, 10(1): 100 – 107.

[40] CHEN J, HU W, HAO D, et al. An empirical comparison of compiler testing techniques[C] // Proceedings of the 38th International Conference on Software Engineering. 2016: 180 – 190.

[41] LE V, AFSHARI M, SU Z. Compiler validation via equivalence modulo inputs[J]. ACM Sigplan Notices, 2014, 49(6): 216–226.

[42] CHEN J, PATRA J, PRADEL M, et al. A survey of compiler testing[J]. ACM Computing Surveys (CSUR), 2020, 53(1): 1–36.

[43] 吴化尧. 组合测试方法及其有效性研究 [D]. [S.l.]: 南京大学, 2018.

[44] LUO W, CHAI D, RUN X, et al. Graph-based Fuzz Testing for Deep Learning Inference Engines[C] // 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). 2021: 288–299.

[45] ANDREO-MARTÍNEZ P, ORTIZ-MARTÍNEZ V M, MUÑOZ A, et al. A web application to estimate the carbon footprint of constructed wetlands[J]. Environmental Modelling & Software, 2021, 135: 104898.

[46] 黄永祥. 精通 Django 3 Web 开发 [M]. [S.l.]: BEIJING BOOK CO. INC., 2020.

[47] 唐斌斌, 叶奕. Vue. js 在前端开发应用中的性能影响研究 [J]. 电子制作, 2020.

[48] CHRISTIE M, MARRU S, ABEYSINGHE E, et al. An extensible django-based web portal for apache airavata[G] // Practice and Experience in Advanced Research Computing. 2020: 160–167.

[49] 齐金刚, 李滔, 李晋军. Django 框架 Web 数据查询分页技术研究 [J]. 电子设计工程, 2014, 22(5): 33–37.

[50] KRUCHTEN P B. The 4+ 1 view model of architecture[J]. IEEE software, 1995, 12(6): 42–50.

[51] HALILI E H. Apache JMeter[M]. [S.l.]: Packt Publishing Birmingham, 2008.

简历与科研成果

基本信息

何云，女，汉族，1998 年 8 月出生，江苏省南京人。

教育背景

2020 年 9 月 — 2022 年 6 月 南京大学软件学院 硕士

2016 年 9 月 — 2020 年 6 月 南京晓庄学院信息工程学院 本科