

****

**EasyCompare：图神经网络模型格式自动对齐和性能对比分析系统**

**“冯如杯”竞赛组委会**

**2022年3月**

摘要

图神经网络领域正处于蓬勃发展时期，越来越多的模型和框架涌现在市场和学术研究等不同领域。众多框架在很多方面具有不同的运行模式，大多数研究者只关注应用而忽略了其在不同网络模型和数据集搭配模式下表现的优劣差异。本文提出了一款自动化对齐分析工具，在技术层面能够将由抽象接口（本文会提出接口规范）所表示的抽象网络模型自动编译生成为能够训练执行的DGL和PyG框架下的图神经网络模型的Python目标代码。并能够在相同配置（数据集、epoches等）的基础之上执行对齐的框架代码获取训练信息，同时分析信息得到算子、GPU核等方面的细节对比结果。在面向用户层面，抽象接口和结果分析被搭载到Web应用上，能够方便用户随时随地修改训练接口、配置提交训练并查看对比结果，同时本文还会个性化地对用户分析历史提交数据信息，以帮助用户找到更佳的解决方案。借助本项目的平台开发者可以在此公平的对比自己与他人的研究成果，另一方面本项目具有良好的可扩展性，系统多样化分析的结果可以用来设计不同的网络框架架构，得出各运行模式的优势和劣势，从而指导未来GNN框架的进一步优化方向。

**关键词：图神经网络，代码转换，性能分析**

Abstract

The field of graph neural network is in a period of vigorous development, and more and more models and frameworks are emerging in different fields such as market and academic research. Many frameworks have different operation modes in many aspects. Most researchers only focus on the application and ignore the advantages and disadvantages of different network models and data set matching modes. This paper proposes an automatic alignment analysis tool, which can automatically compile the abstract network model represented by the abstract interface (the interface specification will be proposed in this paper) into the python target code of the graph neural network model under the framework of DGL and pyg. It can execute the aligned framework code on the basis of the same configuration (dataset, epoches, etc.) to obtain the training information, and analyze the information to obtain the detailed comparison results of operators, GPU cores, etc. At the user oriented level, the abstract interface and result analysis are carried on the web application, which can facilitate users to modify the training interface, configure and submit training and view the comparison results anytime and anywhere. At the same time, this paper will also analyze the historical submission data information of users to help users find a better solution. With the help of the platform of this project, developers can fairly compare their own research results with those of others. On the other hand, this project has good scalability. The results of system diversification analysis can be used to design different network framework architectures and obtain the advantages and disadvantages of each operation mode, so as to guide the further optimization direction of GNN framework in the future.

**Keywords: GNN, Code conversion, Performance analysis**

目录

[一、绪论 1](#_Toc100409485)

[二、背景 2](#_Toc100409486)

[2.1 图神经网络 2](#_Toc100409487)

[2.2 图神经网络框架 2](#_Toc100409488)

[2.2.1 Deep Graph Library（DGL） 2](#_Toc100409489)

[2.2.2 PyTorch Geometric（PyG） 2](#_Toc100409490)

[2.3 Clang 3](#_Toc100409491)

[三、代码接口和转换 3](#_Toc100409492)

[3.1 代码接口 3](#_Toc100409493)

[3.1.1 Neural Networks（神经网络） 3](#_Toc100409494)

[3.1.2 Activations（激活函数） 3](#_Toc100409495)

[3.1.3 基本代码框架 4](#_Toc100409496)

[3.2 代码转换与网络对齐 5](#_Toc100409497)

[四、系统设计 6](#_Toc100409498)

[4.1 系统需求分析 6](#_Toc100409499)

[4.2 系统前端设计 8](#_Toc100409500)

[4.2.1 登陆主界面设计 8](#_Toc100409501)

[4.2.2 系统主界面设计 9](#_Toc100409502)

[4.2.3 训练信息查询及对比界面设计 9](#_Toc100409503)

[4.2.4 用户主界面设计 10](#_Toc100409504)

[4.2.5 前端Vue框架 11](#_Toc100409505)

[4.3 系统后端设计 11](#_Toc100409506)

[4.3.1 Mysql基本表的定义 11](#_Toc100409507)

[4.3.2 后端Django框架 12](#_Toc100409508)

[五、系统实现 12](#_Toc100409509)

[5.1 后端接口实现 12](#_Toc100409510)

[5.2 ORM 12](#_Toc100409511)

[5.3 抽象代码向框架代码的转换 13](#_Toc100409512)

[5.4 训练过程实现方案 13](#_Toc100409513)

[5.5 记录客户端状态：Session 机制 14](#_Toc100409514)

[5.6 密码密文存储机制 14](#_Toc100409515)

[结论 15](#_Toc100409516)

[参考文献 16](#_Toc100409517)

# 一、绪论

近年来，图神经网络在很多方面的应用已经达到了最先进的水平，如节点分类、连接预测等。不同于传统深度学习模型处理的欧几里得数据，如1D的文本和2D的图像，图神经网络通常运行在非欧几里得数据上，如社交网络、知识图谱、生物基因蛋白结构等数据。随着图神经网络的应用的越发广泛，对于图神经网络的性能分析收到越来越多的关注。

实现图神经网络的主流框架包括DGL[2]和PyG[3]。这两个框架都是基于PyTorch[4]实现的，但在很多方面具有不同的运行模式，对于图神经网络的网络层也各有特异的算子组合和优化方案。对于具有相同网络结构的图神经网络模型，实现框架的不同会造成模型性能表现上的差异。

我们在对比分析不同框架下相同网络架构的性能时，发现了两大主要挑战。

**多个框架下语言风格差异大。**不同的框架对于神经网络的实现有着不同的接口定义、代码组织结构，相同功能的接口可能有着多种不同的选项和功能。为了对比分析不同框架的性能，往往需要在不同框架上实现相同的网络结构单元，这对用户提出了较高的要求。

**缺少高效的性能对比分析工具。**在实现了对齐结构的神经网络后并运行后，用户需要手动实现运行数据的记录和关键信息的提取和整理，从而进一步进行对比，这种做法较为繁杂且低效。

为了提高对比分析不同框架的下图神经网络性能的效率，我们提出了图神经网络模型格式自动转换和性能对比分析系统。在本系统中：

* 我们定义了一套在C++语言上的图神经网络模型的常用实现接口。本套接口简洁易懂，能够高效地定义神经网络模型，相比在图神经网络框架（如DGL和PyG）下实现网络更加简单、容易上手。
* 我们基于Clang的LibTooling工具实现了代码自动转换工具。代码自动转换工具能高效地将我们定义的接口代码自动转换到不同框架（如DGL和PyG）下，生成可执行文件。该工具在转换过程中，能够自动对齐生成到不同框架下的代码中的网络结构，以确保后续性能分析的公正性。
* 我们设计了高效简洁的前端界面，方面用户使用。用户只需要要进行简单的操作，便能一键生成代码并运行和收集运行数据。

# 二、背景

## 2.1 图神经网络

图神经网络（Graph Neural Network，GNN）是一种直接作用于图结构数据的神经网络，最初被设计用来解决节点分类任务[1]。GNN的核心思想是收集和聚合图的结构信息来提取图内在的特征，从而预测特定节点属性、节点间的连接属性或生成新的图等。

与深度学习（Deep Learning）模型类似，GNN可以由多层组成，每层能完成聚合和更新两个功能。不同的GNN模型都可以统一为统一的消息传递范式[2]。正式地，令为一个图结构，其中是顶点集合，是边集合。令是顶点的特征向量，是边的特征向量。在第步：

|  |  |  |
| --- | --- | --- |
|  |  | (1) |
|  |  | (2) |

其中，是定义在边上的，通过聚合相连顶点的特征和边的特征来生成消息的消息功能（message function）；是定义在顶点上的，通过聚合到达顶点的特征并使用Reduce函数来更新顶点特征的更新函数（update function）。

常用的GNN模型包括GCN（Graph Convolutional Network）、GraphSAGE（Graph Sampling and Aggregation）、GIN（Graph Isomorphism Network）、GAT（Graph Attention Network）等。

## 2.2 图神经网络框架

### 2.2.1 Deep Graph Library（DGL）

DGL[2]是基于多个深度学习框架（DL frameworks，如PyTorch[4]、TensorFlow[5]、MXNet[6]等）建立的图神经网络框架。DGL将GNN的计算模式归纳为一些适合并行化的广义稀疏张量运算，采用抽象DGLGraph接口来支持面向对象编程。由于DGL支持多个深度学习框架，DGL支持用户轻松地调用现存的深度学习框架组件。

本文中主要使用基于PyTorch的DGL框架。

### 2.2.2 PyTorch Geometric（PyG）

PyG[3]是基于PyTorch建立的图神经网络模型框架。采用简单的消息传递（message passing）API，PyG将大量新提出的卷积和池化层封装成简单和统一的框架接口。所有实现的方法遵循图神经网络的消息传递范式(1)(2)，并同时支持CPU和GPU计算。

## 2.3 Clang

Clang是一个基于LLVM的前端编译器，为C语言家族（C，C++，Objective C/C++等）提供了编译前端和基础工具[7]。

LibTooling是一个基于Clang的支持编写独立工具的库。

# 三、代码接口和转换

这部分将介绍本项目的神经网络模型格式对齐系统。首先，我们定义了一套基于C++的代码接口，用户可基于这套代码接口定义不同类型的神经网络模型。然后，对于这套代码接口，我们分别设计了将上述统一代码转换到不同框架下的代码自动转换工具。结合代码接口和自动转换工具，可以达到“写一次代码，得多套代码”的目的。

## 3.1 代码接口

本系统的抽象代码接口是基于C++语言所定义的。通过定义不同的C++类，在形式上贴近图神经网络框架（如DGL和PyG）在Python语言下的格式。由于后续接口代码要分别转换为不同框架下的可执行代码，抽象接口的具体形式参照各个框架下功能相同的接口形式，保证接口定义的功能可以在各框架上实现。同时，本套接口设计简化了各个框架下具体实现代码的细节，相比目标可执行代码更加简洁，方便用户理解和使用。

### 3.1.1 Neural Networks（神经网络）

**表1 Neural Network接口表**

|  |  |  |
| --- | --- | --- |
| 接口名 | 初始化参数 | 描述 |
| GCNConv | int in\_feat, int out\_feat | 对应于DGL::GraphConv，PyG::GCNConv |
| SAGEConv | int in\_feat, int out\_feat | 对应于DGL:SAGEConv，PyG::SAGEConv |
| GATConv | int in\_feat, int out\_feat,  int heads, double attn\_drop,  double negative\_slope | 对应于DGL::GATConv，PyG::GATConv |
| GINConv | int in\_feat, int out\_feat | 对应于DGL::GINConv，PyG::GINConv |
| Linear | int in\_feat, int out\_feat | 对应于Pytorch::Linear |

注1：Neural Networks共同继承自一个父类Conv，其中类Conv中含有方法tensor in(tensor)，表示网络层处理输入。

### 3.1.2 Activations（激活函数）

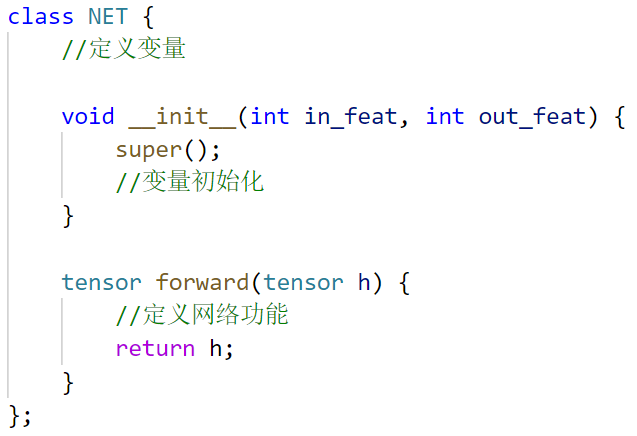
**表2 Activations接口表**

|  |  |  |
| --- | --- | --- |
| 接口名 | 初始化参数 | 描述 |
| Relu | / | 对应于Pytorch::Relu |
| LeakyReLU | double negative\_slope | 对应于Pytorch::LeakyReLU |
| Tanh | / | 对应于Pytorch::Tanh |
| ELU | / | 对应于Pytorch::ELU |

注2：Activations共同继承自一个父类Activation，其中类Activation中含有方法tensor in(tensor)，表示激活函数处理输入。

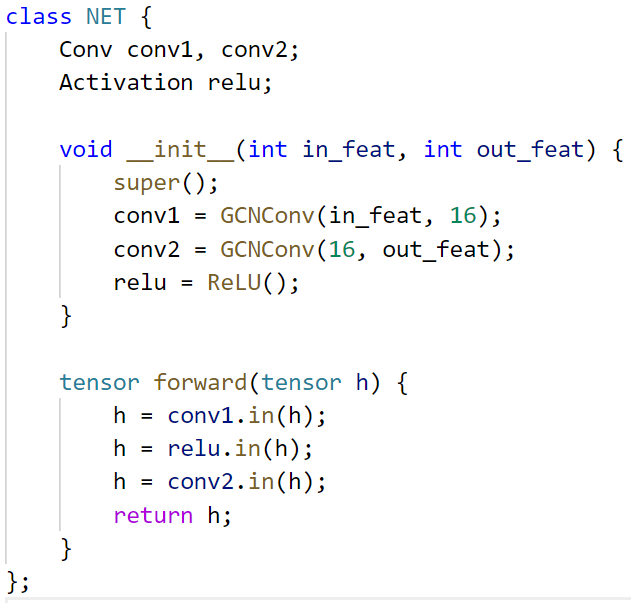
### 3.1.3 基本代码框架

基本代码框架是使用抽象代码接口定义网络结构和训练过程的格式框架。基本代码框架仿照DGL和PyG框架下定义GNN的格式，分为初始化（\_\_init\_\_函数）和训练（forward函数）两部分，如图1 **基本代码框架**所示。其中，\_\_init\_\_函数的参数in\_feat和out\_feat分别表示数据集的输入维度和输出维度。



**图1 基本代码框架**

框架的最外层为一个名为NET的类。首先需要在类内定义所使用的变量（如神经网络层、激活函数等），并将其在\_\_init\_\_函数内进行初始化。在forward函数内，用户需要定义网络层参数传递的过程。举一个GCN的例子，如图2 **代码接口下的GCN实现**所示。首先在定义了两个神经网络层conv1和conv2，以及一个激活函数relu，随后在\_\_init\_\_函数内分别进行了初始化。然后再forward函数中定义了网络参数传递过程，输入分别经过两个网络层conv1和conv2，并在conv1之后使用激活函数relu进行处理。



**图2 代码接口下的GCN实现**

## 3.2 代码转换与网络对齐

代码转换工具是基于Clang的LibTooling工具设计的可独立运行的代码前端编译器。由于抽象接口代码是基于C++语法规则的，代码自动转换工具的基本思想是基于Clang解析抽象代码生成的抽象语法树来修改原代码，达到补齐细节和转换到Python代码的目标。

在代码转换工具中使用到的所有抽象语法树节点如表3 **抽象语法树节点及其修改作用**所示。修改原代码时，首先需要找到相应的节点，并从节点中导出所在代码的位置（Source Location）。不同节点导出位置的方法不尽相同，需要根据查找Clang官方文档[7]来寻找相应方法。

**表3 抽象语法树节点及其修改作用**

|  |  |  |
| --- | --- | --- |
| 节点名 | 节点描述 | 修改作用 |
| CXXRecordDecl | 类定义 | 增加对nn.Module的继承代码 |
| CXXMethodDecl | 类方法定义 | 修改方法定义格式及其参数格式 |
| ParmVarDecl | 方法参数定义 | 修改方法参数格式 |
| DeclRefExpr | 函数调用 | 修改super()函数格式 |
| CXXThisExpr | this指针 | 修改“this->”为self. |
| MemberExpr | 类内属性指针 | 修改网络层和激活函数等调用格式 |
| CXXTemporaryObjectExpr | 临时对象指针 | 修改网络层和激活函数等初始化格式 |
| CXXFunctionalCastExpr | 函数调用指针 | 修改网络层和激活函数等初始化格式 |
| CXXBoolLiteralExpr | 布尔值表达式 | 修改布尔值表达式为Python语法 |

下面使用的例子来说明代码自动转换工具的转换功能。使用自动转换工具将图2 **代码接口下的GCN实现**的GCN模型分别转换到DGL和PyG框架下的可执行代码，如图3 DGL框架下的GCN实现和图4 **PyG框架下的GCN实现**所示。代码自动转换工具修改了大量的语法，使代码从C++语言的可执行代码转换到了Python的可执行代码。并且代码转换工具还会自动添加细节，使得网络能够分别在不同框架上运行。值得注意的是，代码转换工具会自动对齐不同框架下的网络结构，以便在后续性能对比的公平性。

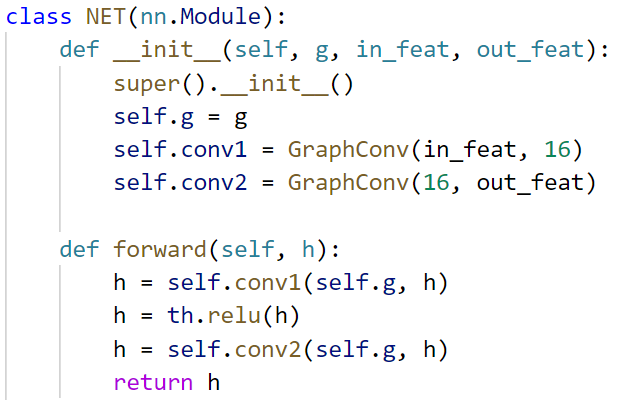
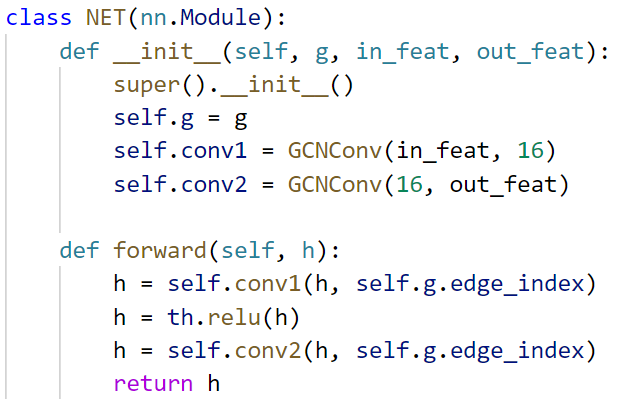


图3 DGL框架下的GCN实现



**图4 PyG框架下的GCN实现**

# 四、系统设计

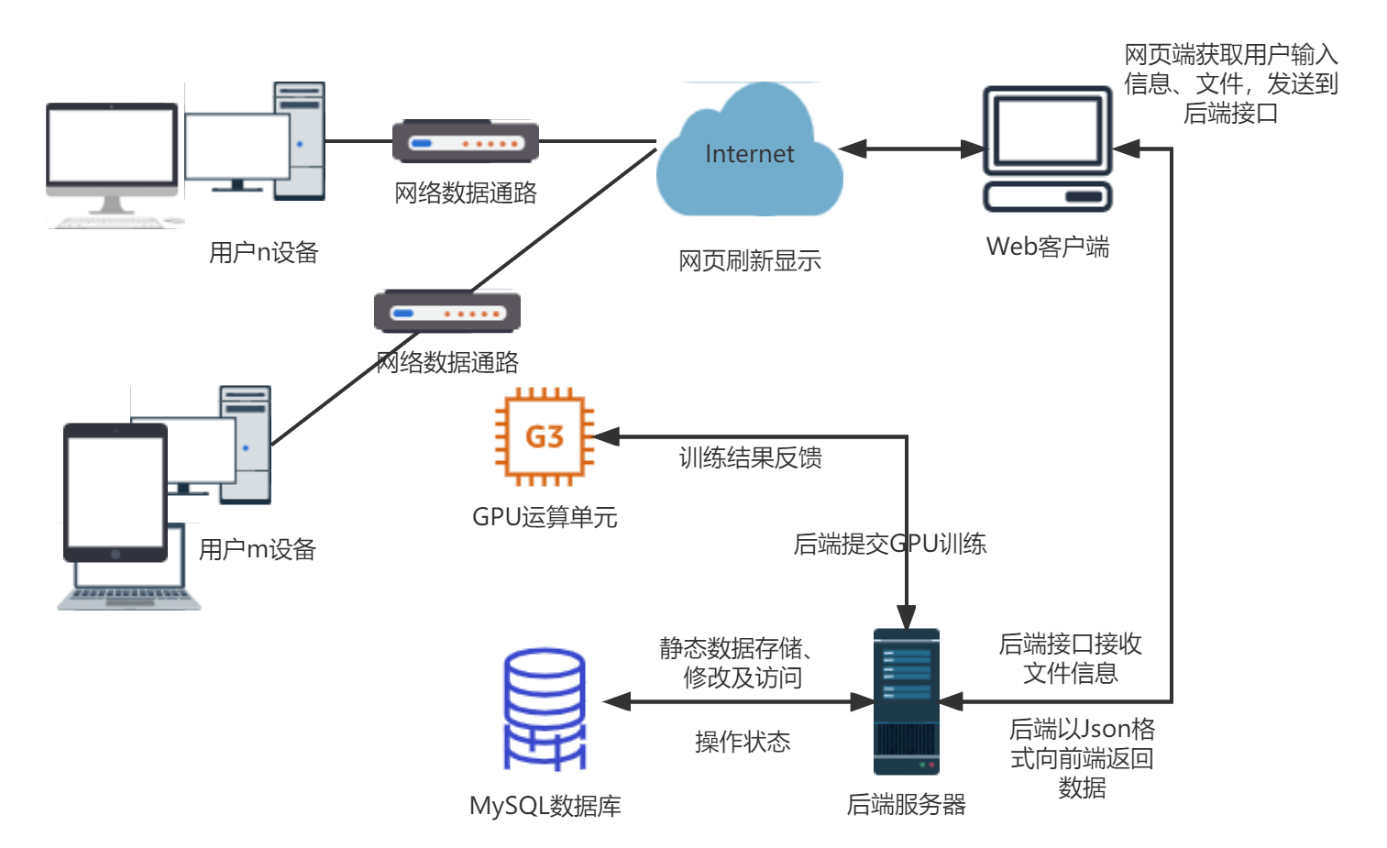
为了能够面向多用户实现便捷高效的功能应用，本项目将采用web前端界面的方式将一系列文档、接口、对比功能呈现给用户。项目前端采用vuetify框架，后端基于django，数据库采用mysql，前后端分离开发有助于系统整体的维护和升级。

## 4.1 系统需求分析

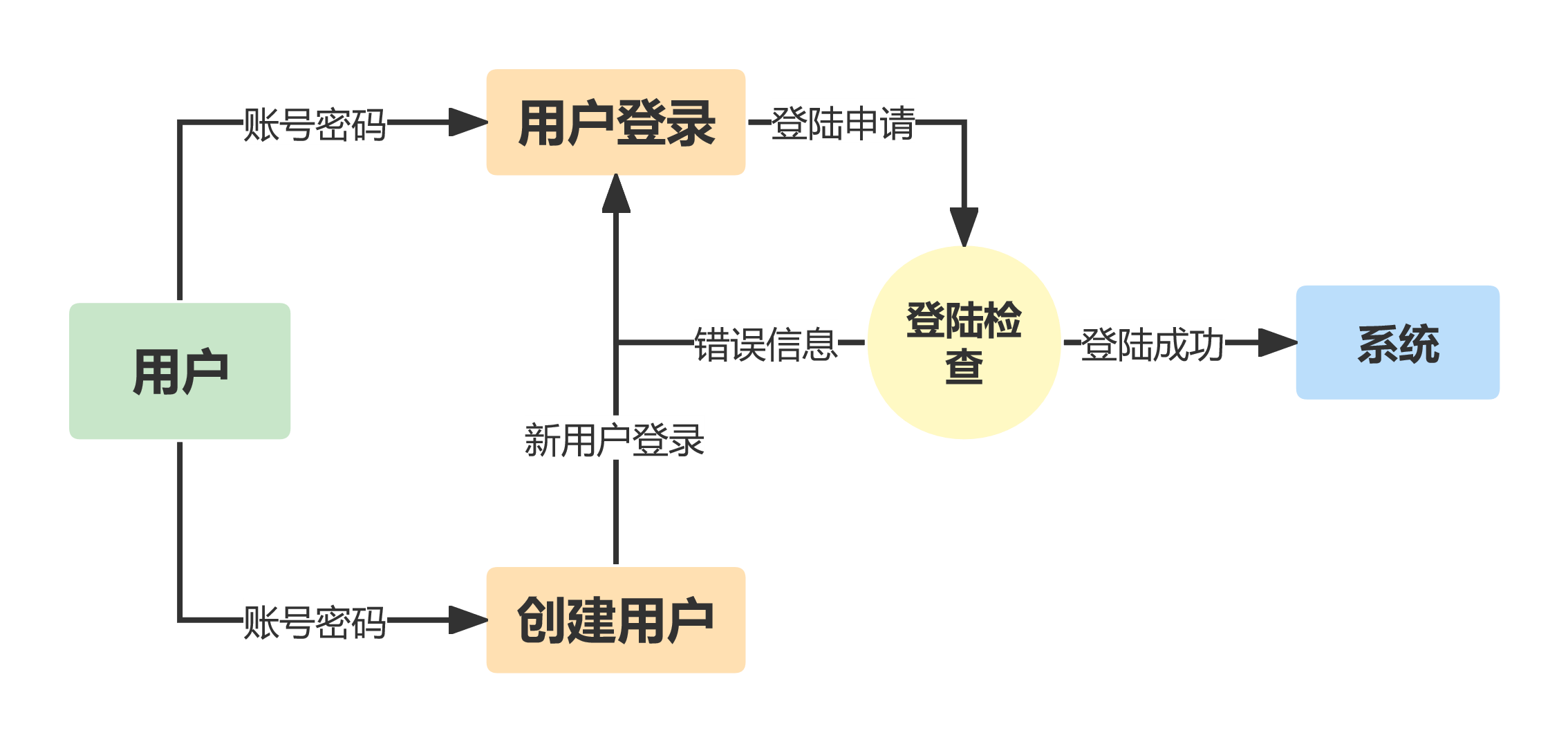
Web界面的前端应用侧重于面向用户，重点在于将系统的代码转换、选择训练、log信息分析对比等主要功能提炼出来以文字、接口等形式呈现给用户进行选择使得用户能够根据自身需求个性化选择应用。

上述需求涉及到前端用户方面以及后端信息的交互，系统后端将采用mysql数据库表的形式来存储相关的信息。

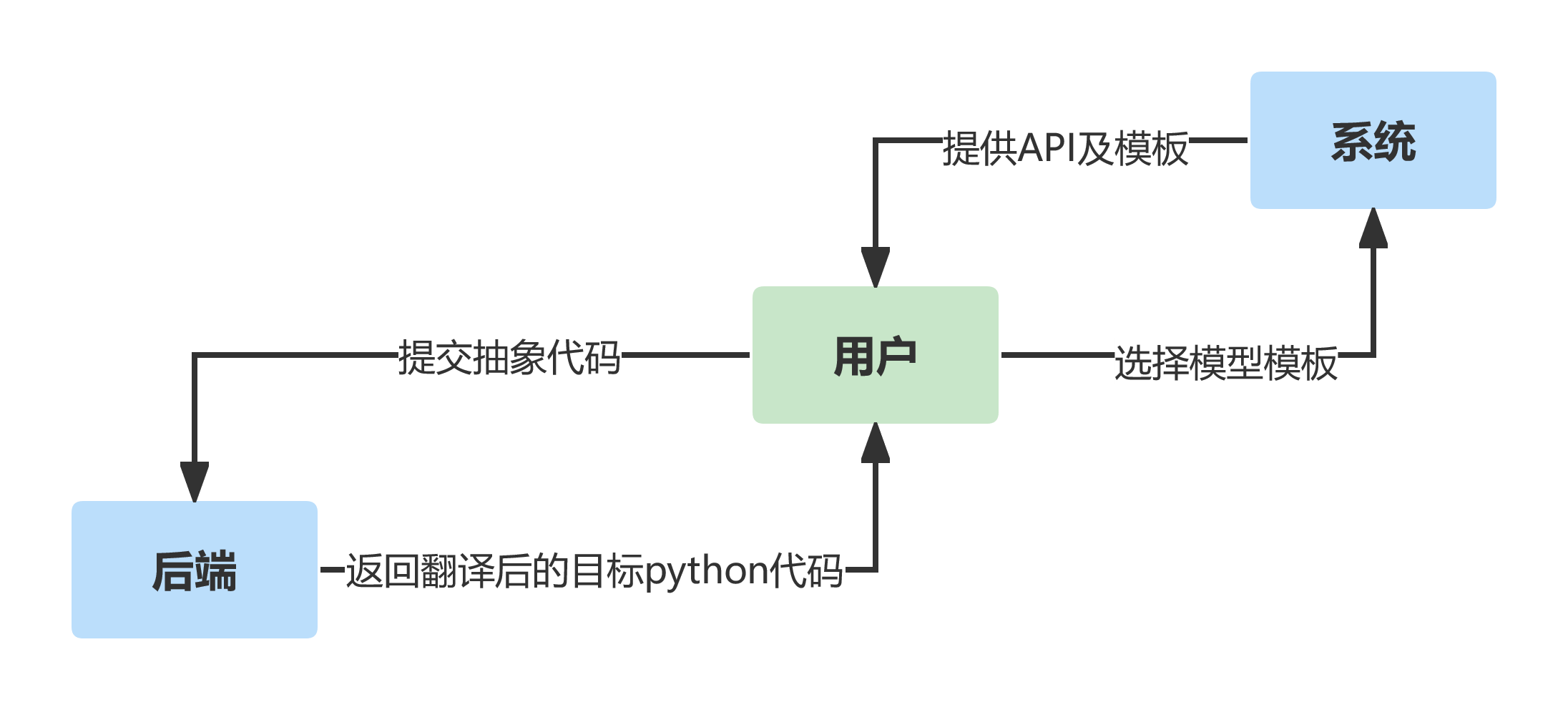
根据需求分析得到本系统的整体网络部署架构图和数据流图如下：



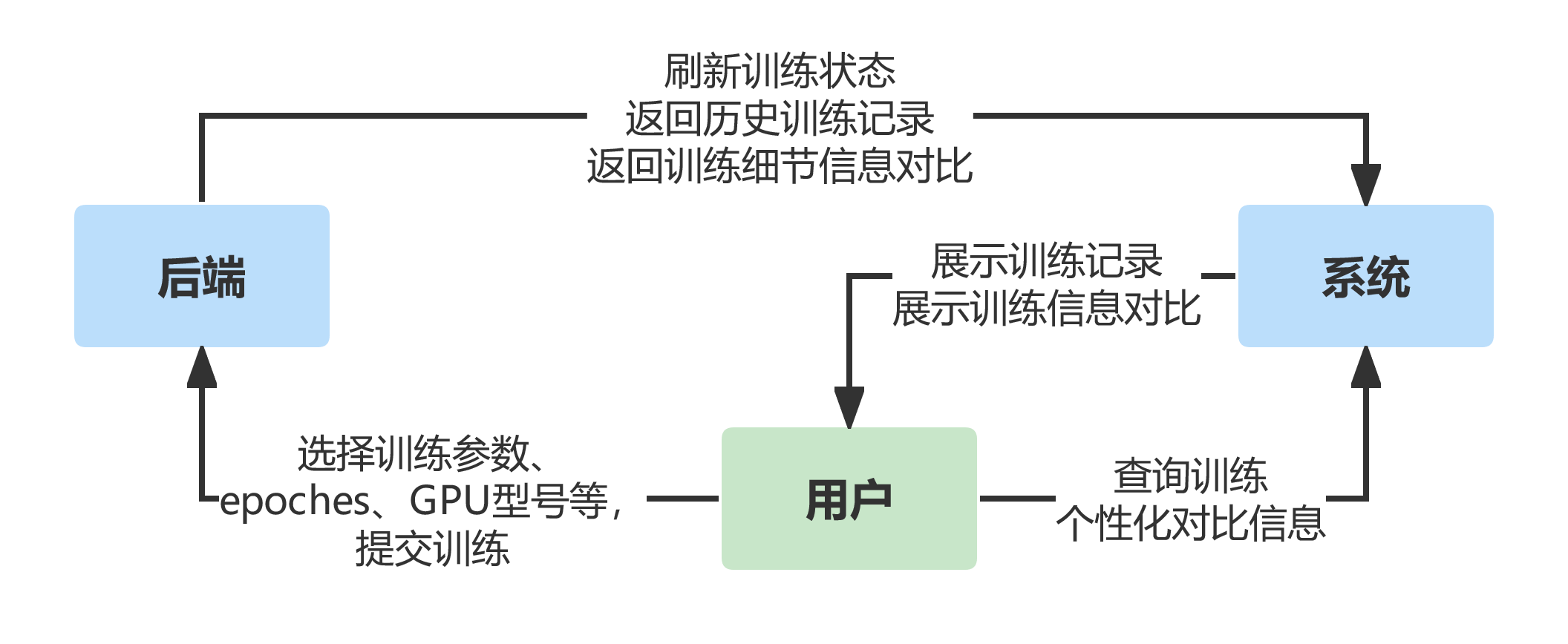
**图5 网络部署架构图**



**图6 用户注册登录数据流**



**图7 目标训练代码生成数据流**



**图8 提交训练以及训练信息对比数据流**

## 4.2 系统前端设计

前端设计的主要原则是面向用户力求简化，应当在保证需求能够完全满足的情况下对前端web界面进行简化和美化以达到实用性和设计性兼得的目标。因此，前端的web界面设计主要分为四大部分：登录主界面设计、系统主界面设计、训练信息查询及对比界面设计以及用户主界面设计。下面将以用户使用本系统的顺序依次介绍各个部分的设计方案。

### 4.2.1 登陆主界面设计

前端登陆页面需要包含注册模块和登录模块两大部分，主要完成用户信息的注册填表和登录信息的查表验证功能。注册时，用户名不可以重复，密码需要满足 合法字符的要求，二者均有长度的限制，在成功登录后返回给用户相关信息，后续的登录过程中需要通过该用户名账号进行登录。页面在成功登陆后会 进行全局保存用户的个人信息以供后续页面的访问

登陆页面关联到“用户信息表”其中包含的属性为“用户ID”、“用户真实姓名”和 “密码”等，其中主码为“用户 ID”，唯一且不为空。这里需要指出，为了保证系统有足够高的安全性和稳定性，密码的存储机制采用密文存储的方式，加密实现的介绍将由后续的实现部分介绍。

### 4.2.2 系统主界面设计

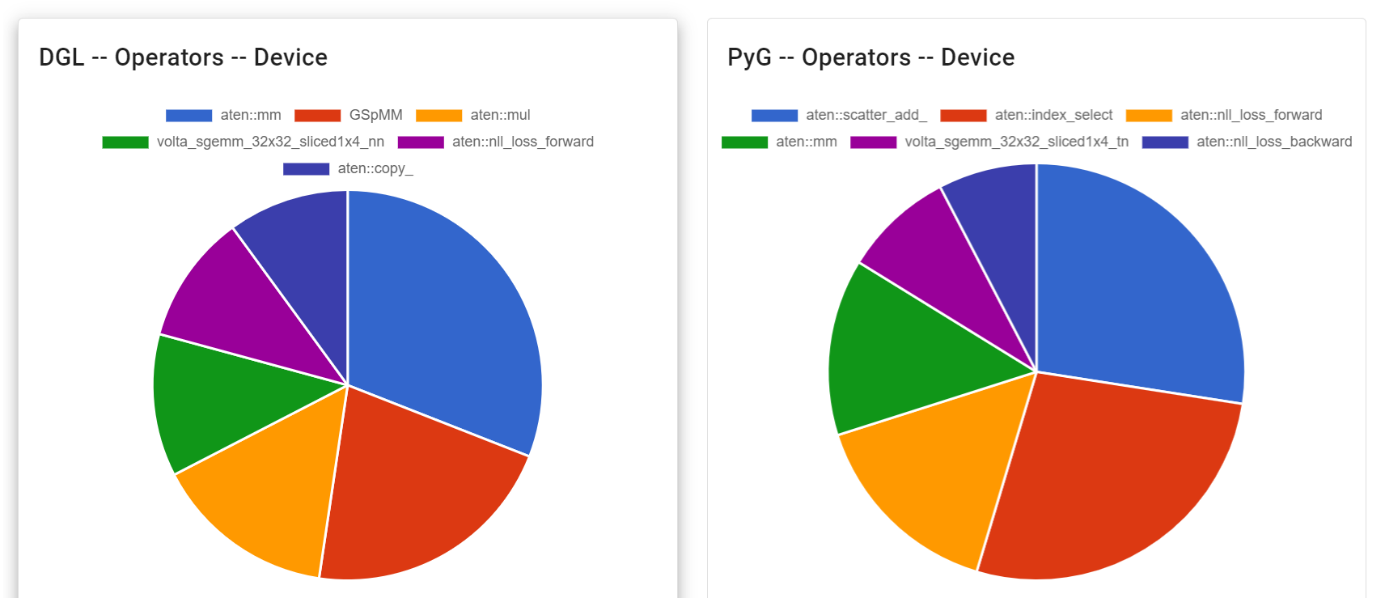
系统主界面用于实现抽象代码向框架代码的转换，用户需要借助系统提供的API和可选择的模板并根据个性化设计需求进行抽象代码段的填写，在确认无误后可以提交抽象代码，后端在进行llvm编译后返回生成的Python训练代码。

在提交训练之前用户可以指定选用的GPU、训练定义标识、数据集以及epoches，之后即可提交训练。

### 4.2.3 训练信息查询及对比界面设计

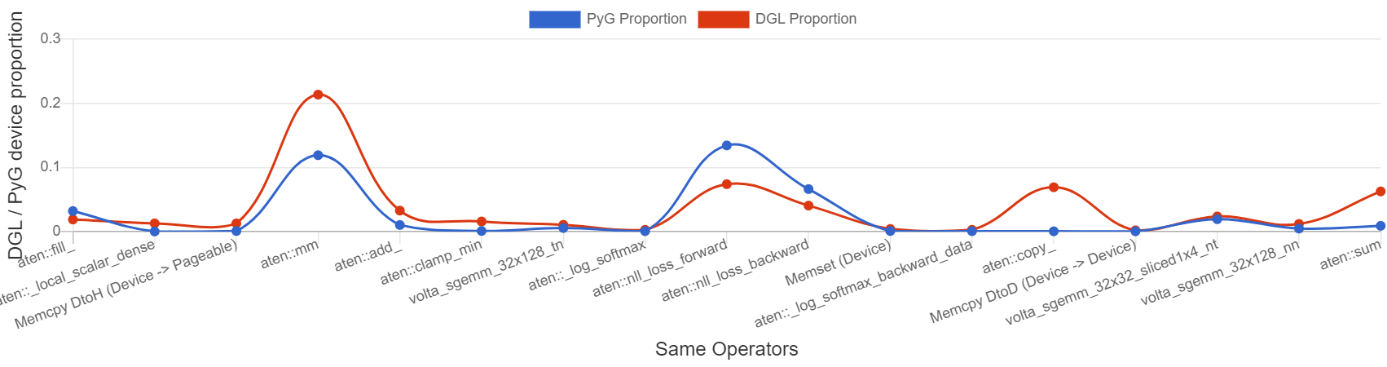
用户可以通过训练信息查询界面随时查询历史训练记录，可以得知当前训练的过程、状态、数据集、GPU利用率等参数。对于历史训练记录，系统提供了多维度的查询功能，根据用户查询的关键字匹配要查询的对象以精准反馈结果。

对于任意一条训练记录，在结果信息对比界面分别有算子维度、核维度的信息，对于算子维度有调用次数、host占用时间以及占比、cuda占用时间以及占比的对比，对于核维度会有调用次数、cuda占用时间以及占比信息的对照，对于上述提到的信息均有相对应的饼状图展示。用户可以根据需求调整饼状图展示的对象数以获得更准确的结果。



**图9 算子GPU利用率饼状图**

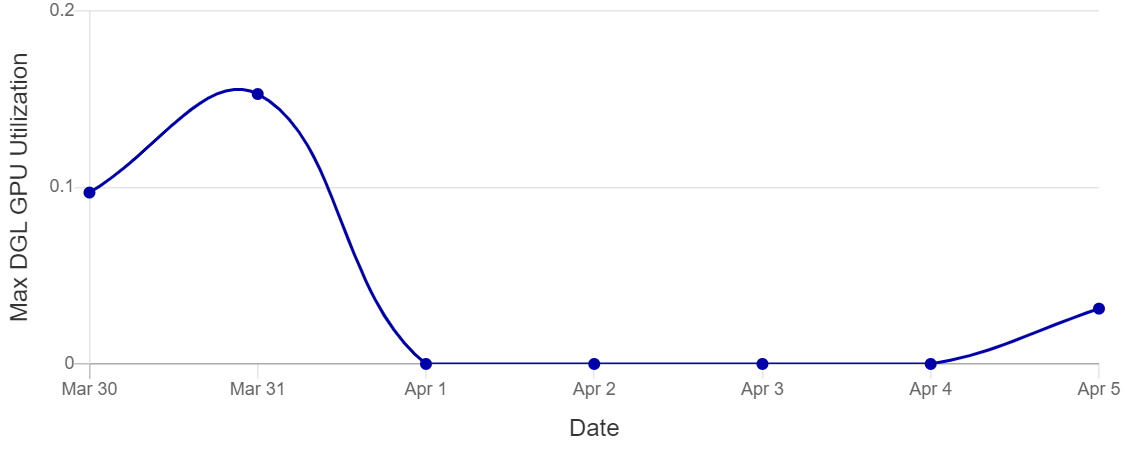
除此之外令设有DGL和PyG两个框架相同算子调用次数的线性对比图，可以清晰直观的看到二者的差异。



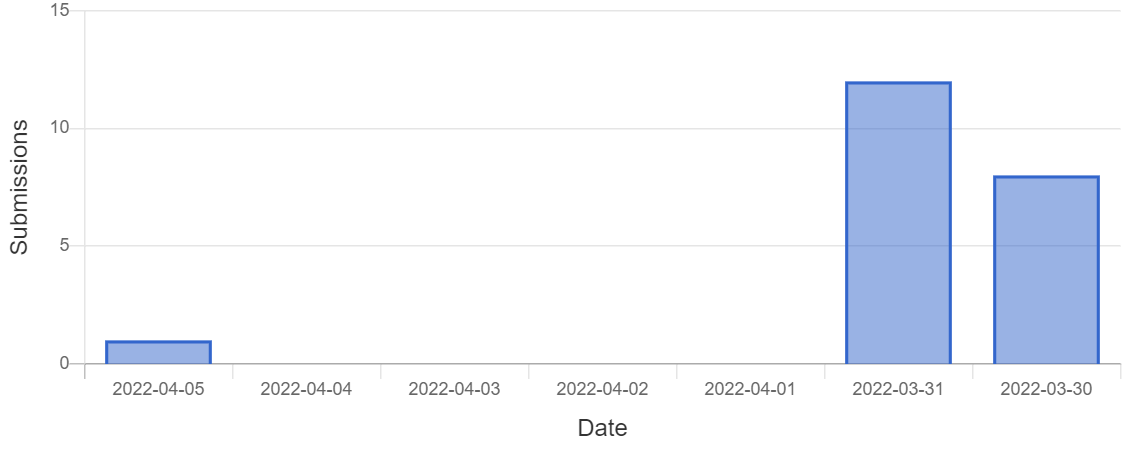
**图10 PyG、DGL相同算子利用率对比折线图**

### 4.2.4 用户主界面设计

用户主界面主要用于统计和该用户相关的一些个人信息以及训练信息，包括：一周内训练提交量、一周内PyG和DGL单日训练最高GPU利用率。同样有用户的个人信息修改以及训练次数的充值等等。



**图11 DGL框架单日提交最高GPU利用率线型图**



**图12 用户周提交次数分布图**

### 4.2.5 前端Vue框架

Vue 是一套用于构架用户界面的渐进式框架。与其他大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

## 4.3 系统后端设计

### 4.3.1 Mysql基本表的定义

系统后端的mysql数据库系统设计遵循3NF原则，数据库中总共包含2个基本表：用户信息表（User）、训练信息表（Training）。关系模型如下，其中下划线部分为主码，斜体部分为外码：

用户信息表User (iduser, name, college, phone, lefttimes, password)

训练表Training (idtrain, *iduser*, status, time, abstractCode, model, dataset, DGLRate, PyGRate, Tag, DGLInfo, PyGInfo)

下表中展示了基本表的定义及名称意义：

**表4 用户表表项及意义**

|  |  |  |
| --- | --- | --- |
| 表项 | 意义 | mysql数据类型 |
| iduser | 用户ID | CharField |
| name | 用户姓名 | CharField |
| college | 所在大学 | CharField |
| phone | 联系方式 | CharField |
| lefttimes | 用户剩余训练次数 | IntegerField |
| password | 用户密码 | CharField |

**表5 训练表表项及意义**

|  |  |  |
| --- | --- | --- |
| 表项 | 意义 | mysql数据类型 |
| idtrain | 训练ID | AutoField |
| iduser | 训练关联用户 | CharField |
| status | 训练状态 | BooleanField |
| time | 训练时间 | DateTimeField |
| abstractCode | 用户给出抽象代码 | TextField |
| model | 模型 | CharField |
| dataset | 数据集 | CharField |
| DGLRate | DGL框架对应GPU利用率 | FloatField |
| PyGRate | PyG框架对应GPU利用率 | FloatField |
| Tag | 用户自定义标识 | CharField |
| DGLInfo | DGL框架训练结果信息 | TextField |
| PyGInfo | PyG框架训练结果信息 | TextField |

### 4.3.2 后端Django框架

Django是一个由Python写成的Web开发框架，采用了模型-视图-模板（MVT） 的框架模式，同时也遵循模型-视图-控制（MVC）设计，遵守BSD版权，能够简便、快速地开发数据库驱动的网站。其主要优点展示如下：

* 采用对象关系映射（ORM），以Python类定义数据模型，拥有非常易于使用的 API，同时也可以直接操纵原生SQL语句
* URL 分派：使用正则表达式匹配 URL，更加灵活
* 模板系统：使用 Django 强大、可扩展的模板语言，可以分隔设计、内容与 Python 代码，且具有可继承性

后端与数据库的操作采用Object-Relationl Mapping的形式对数据库进行访问，它的作用是在关系型数据库和对象之间作一个映射，这样，我们在具体的操作数据库的时候，就不需要再去和复杂的SQL语句打交道，只要像平时操作对象一样操作它就可以了

# 五、系统实现

## 5.1 后端接口实现

后端实现接口采用分包管理的方式，urls中为整体的路由分配方案，即api/后的二级路由索引的对应关系保存。

## 5.2 ORM

在本项目中，我们采用了 Django 框架所提供的 ORM 机制来对数据库进行操 纵，包括由抽象类生成关系表、根据对象对关系表进行增删改查等操作。

Django 框架中，通过在models.py文件中定义继承于django.db.models.Model的类，可以通过执行python manage.py makemigrations、python manage.py migrate两条指令，使数据库模型进行更新并对数据库进行对应的关系表生成与修改。

* AutoField、CharField、DateField 等：均为Django的ORM所提供的接口， 建立模型、迁移数据库后均会在数据库对应表中生成对应的单一属性列。
* ManyToManyField：此处歌手与演出具有多对多关系，因此设置该域。在迁 移完成后，除两个实体对应的表外，会在数据库中额外生成一个关系表，属 性包括自增 ID 以及两个实体对应关系的主键，用于存储多对多关系。
* ImageField：该域在关系表中所对应生成的属性实为 varchar 域，存储内容为 图片的相对存储路径（项目中进行了媒体文件分包管理，媒体文件上传 至./media/… /路径中，后查询时经由此路径访问）。

## 5.3 抽象代码向框架代码的转换

后端在收到抽象代码以及转换的请求后调用本地的编译脚本，使用完成的编译工具来抽象接口进行编译，转换成DGL和PyG框架下的python代码，并将生成的代码以请求应答的方式返回给前端。调用的本地编译脚本如下

python Process/process\_1-2.py -s Users/$1/code1.txt -t Users/$1/code2.cpp

*# DGL*

rm Users/$1/code3\_dgl.py

./Process/dgl-gnn Users/$1/code2.cpp -- >> Users/$1/code3\_dgl.py

python Process/process\_3-4.py -m dgl -s Users/$1/code3\_dgl.py -t Users/$1/code4\_dgl.py

*# PyG*

rm Users/$1/code3\_pyg.py

./Process/pyg-gnn Users/$1/code2.cpp -- >> Users/$1/code3\_pyg.py

python Process/process\_3-4.py -m pyg -s Users/$1/code3\_pyg.py -t Users/$1/code4\_pyg.py

脚本第一行是将code1中保存的抽象代码做一步预处理，给其添加头文件和main函数包裹使其由接口处理成为一个可以被llvm识别的cpp文件。脚本的DGL和PyG框架代码生成在后面列出来，以DGL为例首先要删除掉之前生成过的文件，紧接着将code2中的预处理代码编译成为code3\_dgl中间代码，最后一步生成code4\_dgl DGL框架可执行python代码。PyG的实现与上述相同。

## 5.4 训练过程实现方案

由前端向后端发送训练的基本参数配置比如模型、数据集、epoches等，后端收到配置参数后调用执行脚本来进行训练。执行脚本信息如下。

python Process/run\_dgl.py -u $1 -d $2 -e $3

python Process/run\_pyg.py -u $1 -d $2 -e $3

第一个参数是用户名，第二个参数是所选数据集，第三个参数是训练epoches，其余的参数不参与训练过程。为了能够保证在训练过程中能够实时获取到训练的状态，后端采用先将训练信息部分存入数据库中并置训练状态为未完成，在结束训练后将剩余分析信息补全并置训练状态为完成，这样就可以完成实时的获取训练状态的需求。

## 5.5 记录客户端状态：Session 机制

在进行前端开发的时候，存在页面刷新时全局变量不保存的现象。为了解决这一问题，采用Session机制来记录客户状态。Session是一种记录客户状态的机制，Session保存在服务器上。客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录在服务器上。这就是Session。客户端浏览器再次访问时只需要从该 Session中查找该客户的状态即可。下面展示本次开发中Session实现机制。

在App.vue中的javaScript中created部分添加如下代码，每次刷新页面都会 将全局变量保存到sessionStorage中。

*//在页面刷新时将 vuex 里的信息保存到 sessionStorage 里*

window.addEventListener("beforeunload", () **=>** {

  sessionStorage.setItem("store", JSON.stringify(this.$store.state));

});

此时，刷新页面时就会保存当前页面中需要存储的信息。但在加载页面时并未对 sessionStorage中的内容进行加载。因此，还需要添加如下代码，每次加载页面时都读取sessionStorage中的状态信息。

*//在页面加载时读取 sessionStorage 里的状态信息*

if (sessionStorage.getItem("store")) {

  this.$store.replaceState(

    Object.assign({}, this.$store.state, JSON.parse(sessionStorage.getItem("store")))

  );

}

每次加载页面时都会读取 sessionStorage 中的内容。Session 机制实现 完成。 Session 机制实现之后，每次刷新页面可以将全局变量中的信息进行保存并再 次加载，从而实现刷新页面前后用户无需重复登录。

## 5.6 密码密文存储机制

在数据库保存数据时，采用密文存储的方式。具体的加密方式为 MD5 信息摘要算法。该算法的加密流程可以网络上进行查找。由于普通的MD5加密 过程已经被应用许久，互联网也有相应的破译网站，系统对加密的过程进行了修改，在此不便给出。 通过MD5加密算法，可以有效地防止被直接看到明文，从而做到加密的效果。系统在并不知道用户密码的明码的情况下就可以确定用户登录系统的合法性。

# 结论

本项目主要着力于深度神经网络中的图神经网络框架对比，通过限制相同的网络模型、网络层结构、数据集、epoches以及运算硬件设备（相同的GPU服务器）设计出了一款针对DGL和PyG两个框架的自动化的对比分析工具，实现了将常见神经网络模型在DGL和PyG框架下的网络结构自动对齐，对相同条件的代码进行同等规模的训练，对收集到的数据进行算子、GPU核等多维度的对比分析，同时在这些功能满足的基础之上面向用户提供极其方面的Web端使用接口。能够很大程度上帮助研究者们分析找到不同的模式搭配下最优的方案，更进一步的希望结合分析结果探讨不同的架构设计，可以得出各个运行模式的优势与劣势，从而指导未来GNN框架的进一步优化方向。

# 参考文献

1. Scarselli F, Gori M., Tsoi A. C., Hagenbuchner M., and Monfardini G. The graph neural network model[J]. *IEEE Transactions on Neural Networks*, 2008.
2. Wang M, Zheng D, Ye Z, et al. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks[J]. *arXiv preprint arXiv*, 2019.
3. Fey M, Lenssen J E. Fast Graph Representation Learning with PyTorch Geometric[J]. 2019.
4. Paszke A, Gross S, Massa F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library[J]. 2019.
5. Abadi, Martín, et al. TensorFlow: A System for Large-Scale Machine Learning. *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016.
6. Chen T, Li M, Li Y, et al. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems[J]. *Statistics*, 2015.
7. Clang. Clang: a C language family frontend for LLVM [EB/OL] <https://clang.llvm.org/doxygen/namespaceclang.html>, 2022.