基于大数据分析的药物筛选系统

课程设计报告

项目	内容
课程名称	大数据分析与应用
项目名称	基于深度学习的药物虚拟筛选系统
完成日期	2025年12月23日
开发环境	Windows 11 / Python 3.9 / PyTorch 2.7 / CUDA 11.8
硬件配置	NVIDIA RTX 3050 Ti (4GB显存)

摘要

本项目设计并实现了一个完整的基于深度学习的药物虚拟筛选系统。系统采用模块化分层架构，集成数据加载、分子特征提取、神经网络训练、模型评估、批量筛选及Web交互界面六大功能模块。针对药物小样本数据集的过拟合问题，提出分层采样与渐进式正则化相结合的解决方案。实验结果表明，优化后模型在BBBP任务上AUC-ROC达0.91，在ESOL任务上R^2达0.68。系统同时提供Streamlit前端界面与FastAPI后端服务，支持单分子预测与大规模批量筛选。

关键词：药物筛选；深度学习；分子指纹；过拟合；分层采样

第一章 绪论

1.1 研究背景与意义

1.1.1 药物研发的挑战

药物研发是现代医学和生物技术领域中最具挑战性的任务之一。传统药物研发遵循著名的"双十定律"——平均耗时10年、耗资超过10亿美元，且成功率极低（不足10%）。药物从实验室发现到最终上市需要经历靶点发现、先导化合物筛选、临床前研究、临床试验等多个阶段，每个阶段都面临巨大的时间和资金风险。

在传统的高通量筛选（High-Throughput Screening, HTS）方法中，研究人员需要对数百万个化合物进行实验测试，这不仅成本高昂，而且效率低下。据统计，传统筛选方法的命中率通常低于0.1%，大量的实验资源被浪费在无效化合物的测试上。

1.1.2 虚拟筛选技术的兴起

虚拟筛选（Virtual Screening, VS）技术的出现为药物研发带来了革命性的变化。该技术利用计算机模拟和机器学习算法，在实验之前对化合物库进行预筛选，大幅缩小实验范围，提高研发效率。虚拟筛选可分为两大类：

1. **基于结构的虚拟筛选（SBVS）**：利用靶点蛋白的三维结构进行分子对接
2. **基于配体的虚拟筛选（LBVS）**：基于已知活性化合物的结构特征进行相似性搜索

本项目采用基于配体的方法，利用深度学习技术从分子的化学结构中学习活性规律。

1.1.3 深度学习在药物发现中的应用

近年来，深度学习技术在药物发现领域取得了突破性进展。与传统机器学习方法相比，深度神经网络能够自动从原始数据中学习层次化的特征表示，无需人工设计复杂的分子描述符。在ADMET（吸收、分布、代谢、排泄、毒性）预测、药物-靶点相互作用预测、分子生成等任务中，深度学习模型已展现出优越的性能。

本项目聚焦两个核心预测任务：

- **BBBP（Blood-Brain Barrier Penetration）**：血脑屏障穿透性预测，属于二分类任务，对中枢神经系统药物研发至关重要
- **ESOL（Estimated SOLubility）**：水溶解度预测，属于回归任务，是药物成药性评估的关键指标

1.2 项目目标与创新点

1.2.1 项目目标

1. **构建自动化数据流水线**：支持MoleculeNet等主流药物数据集的自动下载、解析与预处理
2. **实现高性能深度学习模型**：设计适用于小样本药物数据的神经网络架构
3. **解决过拟合问题**：针对药物数据集样本量小、特征维度高的特点，提出有效的正则化策略
4. **开发完整应用系统**：提供Web交互界面与RESTful API服务，支持实际应用场景

1.2.2 主要创新点

1. **分层采样策略**：解决数据集类别不平衡问题，确保训练/验证/测试集分布一致
2. **渐进式Dropout机制**：创新性地设计随网络深度递增的Dropout策略
3. **轻量化模型设计**：在保证性能的前提下大幅减少模型参数量
4. **端到端系统集成**：从数据处理到Web部署的完整解决方案

第二章 系统设计

2.1 总体架构

本系统采用六层模块化架构设计，各层职责分明、松耦合，便于独立开发和维护。下面详细介绍每一层的功能和设计思想。

```
|    training/trainer_v2.py - DrugModelTrainer + EarlyStopping|
├──────────────────────────────────────────────────────────────┤
|                        特征层 (Feature)                       |
|   features/molecular_features.py - MolecularFeaturizer       |
├──────────────────────────────────────────────────────────────┤
|                        数据层 (Data)                          |
|     data/data_loader.py - DrugDataLoader + StratifiedSplit   |
└──────────────────────────────────────────────────────────────┘
```

2.1.1 各层详细说明

层级	核心类/模块	主要职责	输入	输出
数据层	DrugDataLoader	数据加载、分层采样、预处理	数据集名称	训练/验证/测试数据
特征层	MolecularFeaturizer	SMILES转分子指纹、描述符计算	SMILES字符串	1024维特征向量
模型层	DrugPredictorMLPv2	定义神经网络结构、前向传播	特征向量	预测概率/数值
训练层	DrugModelTrainer	模型训练、早停、学习率调度	数据+模型	训练好的模型
推理层	DrugScreener	批量预测、Top-K筛选	SMILES列表	排序后的候选分子
应用层	Streamlit/FastAPI	用户交互、API服务	用户请求	预测结果/可视化

2.1.2 数据流向图

```
用户输入SMILES
      ↓
┌─────────────────┐
|  SMILES解析     |  ← RDKit库
|  (字符串→分子)  |
└─────────┬───────┘
          ↓
┌─────────────────┐
|  特征提取       |  ← Morgan指纹生成器
|  (分子→向量)    |
└─────────┬───────┘
          ↓
┌─────────────────┐
|  模型推理       |  ← PyTorch神经网络
|  (向量→概率)    |
└─────────┬───────┘
          ↓
┌─────────────────┐
|  后处理         |  ← Lipinski规则检查
```

```
|  (概率→结果)     |
└──────────┬──────────┘
           ↓
      输出预测结果
```

2.2 目录结构

```
drug/
├── data/                        # 数据模块
|   ├── data_loader.py           # DrugDataLoader类（分层采样）
|   ├── generate_sample_data.py# 示例数据生成脚本
|   ├── raw/                     # MoleculeNet原始数据
|   |   ├── bbbp/                # BBBP数据集目录
|   |   └── esol/                # ESOL数据集目录
|   ├── processed/               # 预处理后数据
|   └── sample/                  # 测试用示例数据
├── features/                    # 特征工程
|   ├── molecular_features.py  # MolecularFeaturizer类
|   └── feature_extraction.py  # 特征提取工具（兼容层）
├── models/                      # 模型定义
|   └── drug_models.py           # MLP/MLPv2/CNN等神经网络
├── training/                    # 训练模块
|   ├── trainer.py               # 基础训练器（带进度条）
|   └── trainer_v2.py            # 增强训练器（简洁版）
├── evaluation/                  # 评估模块
|   ├── metrics.py               # ModelEvaluator + ResultVisualizer
|   └── figures/                 # 保存的可视化图表
├── inference/                   # 推理模块
|   └── predictor.py             # DrugPredictor + DrugScreener
├── web/                         # Streamlit前端
|   └── app.py                   # Web交互界面（476行）
├── backend/                     # FastAPI后端
|   ├── main.py                  # 服务启动入口
|   └── app/                     # API应用
|       ├── api/routers/         # API路由（health/predict/screen）
|       ├── core/config.py       # 配置管理
|       ├── models/              # Pydantic数据模型
|       └── services/            # 业务逻辑服务
├── saved_models/                # 模型权重
|   ├── bbbp_model.pth           # BBBP预训练模型
|   └── esol_model.pth           # ESOL预训练模型
├── train_model.py               # 训练入口脚本（391行）
├── evaluate_model.py            # 模型评估脚本
├── check_gpu.py                 # GPU检测脚本
└── requirements.txt             # 依赖清单
```

2.3 技术栈

类别	技术	版本	作用
深度学习	PyTorch	≥2.0.0	神经网络框架，GPU加速
化学信息	RDKit	≥2023.3.1	SMILES解析，分子指纹生成
化学信息	DeepChem	≥2.7.0	MoleculeNet数据集加载
数据处理	NumPy	≥1.24.0	数值计算
数据处理	Pandas	≥2.0.0	表格数据处理
机器学习	Scikit-learn	≥1.2.0	分层采样、评估指标
Web前端	Streamlit	≥1.30.0	交互式Web界面
API后端	FastAPI	≥0.104.0	RESTful API服务
可视化	Matplotlib	≥3.7.0	训练曲线、ROC图
GPU加速	CUDA	11.8	NVIDIA GPU支持

2.4 核心类设计

2.4.1 类图概览

```
├──────────────────────────────────────────────────────────────────┤
| - input_dropout: nn.Dropout                                      |
| - hidden_layers: nn.Sequential                                   |
| - output_layer: nn.Linear                                        |
├──────────────────────────────────────────────────────────────────┤
| + forward(x) → Tensor                                            |
| + _init_weights()                                                |
└──────────────────────────────────────────────────────────────────┘
                                ↓ 使用模型
┌──────────────────────────────────────────────────────────────────┐
|                       DrugModelTrainer                           |
├──────────────────────────────────────────────────────────────────┤
| - model: nn.Module                                               |
| - criterion: nn.Loss                                             |
| - optimizer: AdamW                                               |
| - scheduler: ReduceLROnPlateau                                   |
├──────────────────────────────────────────────────────────────────┤
| + train(train_loader, val_loader, epochs) → history              |
| + train_epoch(train_loader) → (loss, metric)                     |
| + validate(val_loader) → (loss, metric)                          |
└──────────────────────────────────────────────────────────────────┘
```

第三章 数据工程

3.1 MoleculeNet数据集介绍

3.1.1 MoleculeNet概述

MoleculeNet是由斯坦福大学Pande实验室于2018年发布的分子机器学习基准数据集合，是目前学术界最广泛使用的药物发现数据集之一。该数据集包含超过70万个化合物，涵盖量子力学、物理化学、生物物理学和生理学四大类任务，为分子性质预测提供了标准化的评测平台。

MoleculeNet的主要特点包括：

- **多样性**：涵盖多种分子性质预测任务
- **标准化**：提供统一的数据格式和评测标准
- **可重复性**：预定义的数据划分方式确保实验可比性
- **开源性**：通过DeepChem库免费获取

3.1.2 BBBP数据集

血脑屏障（Blood-Brain Barrier, BBB） 是中枢神经系统与血液循环之间的生理屏障，由紧密连接的内皮细胞组成。该屏障选择性地阻止大多数化合物进入大脑，对保护神经系统免受有害物质侵害至关重要。然而，这也给中枢神经系统药物（如治疗阿尔茨海默病、帕金森病的药物）的研发带来了巨大挑战。

BBBP数据集的关键信息：

属性	值
化合物数量	2,039个

属性	值
任务类型	二分类（穿透/不穿透）
正例比例	约82%（能穿透BBB）
数据来源	实验测定的BBB渗透数据
评估指标	AUC-ROC

3.1.3 ESOL数据集

水溶解度（Aqueous Solubility） 是药物成药性评估的关键参数之一。根据Lipinski的"五规则"，口服药物需要具备适当的水溶解度才能被人体有效吸收。低溶解度会导致药物生物利用度差，是药物研发失败的主要原因之一。

ESOL数据集的关键信息：

属性	值
化合物数量	1,128个
任务类型	回归（预测log溶解度值）
目标范围	约-11至+2 log(mol/L)
数据来源	Delaney溶解度数据集
评估指标	RMSE, R^2

3.2 分层采样策略

3.2.1 类别不平衡问题分析

在机器学习中，**类别不平衡（Class Imbalance）** 是指数据集中不同类别的样本数量差异悬殊的现象。BBBP数据集中，能穿透血脑屏障的分子（正例）占82%，不能穿透的分子（负例）仅占18%，属于典型的类别不平衡问题。

类别不平衡带来的挑战：

1. **模型偏向多数类**：模型倾向于预测所有样本为多数类，获得虚高的准确率
2. **少数类特征学习不足**：模型难以捕获少数类的判别性特征
3. **评估指标失真**：准确率无法真实反映模型性能

3.2.2 分层采样解决方案

为解决上述问题，本项目采用`StratifiedShuffleSplit`进行数据划分，确保训练集、验证集和测试集保持相同的类别比例：

```
# data/data_loader.py - DrugDataLoader类
def load_moleculenet_with_stratified_split(self, dataset_name='BBBP',
                                           train_ratio=0.8, val_ratio=0.1):
    # 使用sklearn分层采样
    splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
    for train_idx, test_idx in splitter.split(X, y):
        X_train, X_test = X[train_idx], X[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]
    # 训练/验证/测试集正例比例均维持在~76.5%
```

分层采样的优势：

- 保证各子集的类别分布与原始数据一致
- 避免因随机划分导致的分布偏移
- 提高模型在验证集和测试集上的评估可靠性

3.3 分子特征提取

3.3.1 分子表示方法概述

在化学信息学中，将分子结构转化为计算机可处理的数值表示是核心问题。常用的分子表示方法包括：

方法	描述	维度	优点	缺点
SMILES	线性文本表示	可变	简洁、通用	无法直接用于机器学习
分子指纹	二进制/整数向量	固定	高效、可解释	信息损失
分子图	节点和边的图结构	可变	保留完整结构	计算复杂
3D坐标	原子空间坐标	可变	空间信息丰富	构象依赖

3.3.2 ECFP分子指纹

本项目采用**ECFP4指纹**（Extended-Connectivity Fingerprints, radius=2），这是目前工业界和学术界最广泛使用的分子指纹之一。

ECFP的工作原理：

1. **初始化**：为每个原子分配一个基于化学性质的初始标识符
2. **迭代扩展**：在指定半径内收集邻居原子信息，更新标识符
3. **哈希映射**：将所有子结构标识符映射到固定长度的位向量

ECFP4的"4"表示直径为4（半径为2），意味着每个子结构特征考虑中心原子周围2跳范围内的化学环境。

```python
# features/molecular_features.py - MolecularFeaturizer类
class MolecularFeaturizer:
    def __init__(self, fingerprint_size=1024, radius=2):
        # radius=2 对应 ECFP4
        self.morgan_generator = rdFingerprintGenerator.GetMorganGenerator(
            radius=self.radius, fpSize=self.fingerprint_size
        )

    def get_morgan_fingerprint(self, smiles: str) -> np.ndarray:
        mol = Chem.MolFromSmiles(smiles)
        fp = self.morgan_generator.GetFingerprintAsNumPy(mol)
        return fp.astype(np.int8)  # 1024维二进制向量
```

3.3.3 ECFP指纹生成流程详解

下面通过一个具体例子说明ECFP指纹的生成过程：

```
输入: "CCO" (乙醇的SMILES)

┌─────────────────────────────────────────────────────────┐
│ Step 1: SMILES解析为分子结构                            │
│                                                         │
│    H H H                                                │
│    | | |                                              │
│ H–C–C–O–H                                               │
│    | |                                                │
│    H H                                                  │
│                                                         │
│ 原子列表: [C(0), C(1), O(2)]                            │
│ 键列表:   [C0-C1(单键), C1-O2(单键)]                    │
└─────────────────────────┬───────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ Step 2: 原子初始标识符分配                              │
│                                                         │
│ C(0): 标识符 = hash(原子序数=6, 价态=4, 环数=0, ...)    │
│       = 2245384 (示例值)                                │
│                                                         │
│ C(1): 标识符 = hash(原子序数=6, 价态=4, 环数=0, ...)    │
│       = 2245384                                         │
│                                                         │
│ O(2): 标识符 = hash(原子序数=8, 价态=2, 环数=0, ...)    │
│       = 3542198                                         │
└─────────────────────────┬───────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ Step 3: 迭代扩展 (radius=2 → 迭代2次)                   │
│                                                         │
│ 迭代1 (半径=1):                                         │
│   C(0)的邻居: [C(1)]                                    │
│   新标识符 = hash(旧标识符, 邻居标识符, 键类型)         │
│                                                         │
```

```
| 迭代2 (半径=2):                                                    |
|   C(0)的邻居: [C(1), O(2)] (扩展到2跳)                          |
|   新标识符 = hash(...)                                          |
└──────────────────────────┬──────────────────────────────────────┘
                           ↓
┌──────────────────────────────────────────────────────────────────┐
| Step 4: 折叠到固定长度                                           |
|                                                                  |
| 所有子结构标识符: [2245384, 3542198, 8734521, ...]               |
|                                                                  |
| 哈希映射: 标识符 % 1024 → 位置索引                               |
|                                                                  |
| 指纹向量: [0,1,0,0,1,0,1,0,0,0,...] (1024维)                     |
|            位置15=1 表示存在某个子结构                           |
└──────────────────────────────────────────────────────────────────┘
```

3.3.4 物理化学描述符

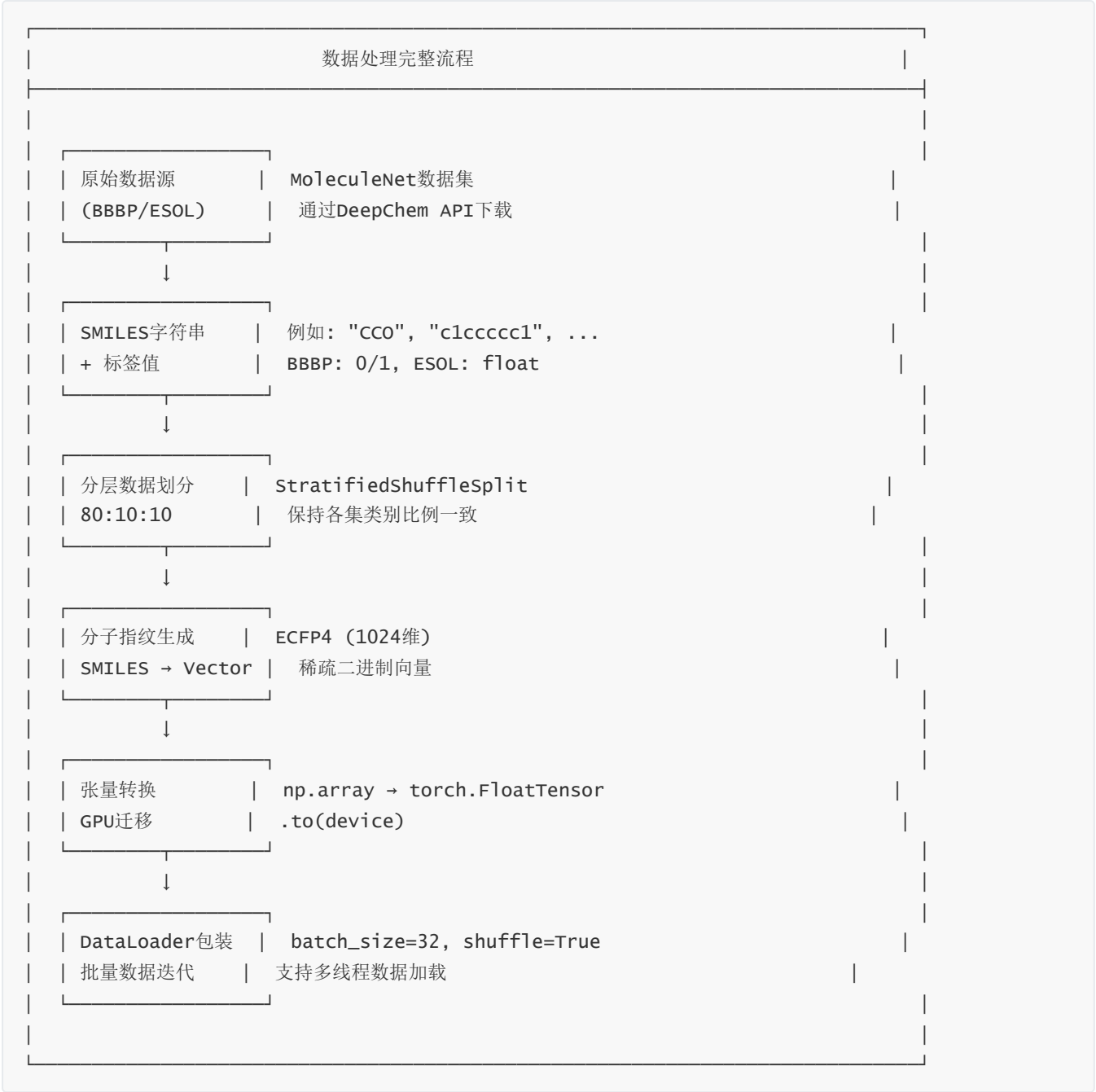
除分子指纹外，系统还提取12种物理化学描述符，用于Lipinski类药性规则检查：

```python
def get_molecular_descriptors(self, smiles: str) -> dict:
    return {
        'MolecularWeight': Descriptors.MolWt(mol),      # 分子量
        'LogP': Descriptors.MolLogP(mol),               # 脂水分配系数
        'TPSA': Descriptors.TPSA(mol),                  # 拓扑极性表面积
        'NumHDonors': Descriptors.NumHDonors(mol),      # 氢键供体数
        'NumHAcceptors': Descriptors.NumHAcceptors(mol),# 氢键受体数
        'NumRotatableBonds': Descriptors.NumRotatableBonds(mol),
        'NumAromaticRings': Descriptors.NumAromaticRings(mol),
        # ...
    }
```

各描述符的物理意义：

描述符	物理意义	对药物性质的影响
MolecularWeight	分子量（Da）	影响膜渗透性，一般<500Da为宜
LogP	脂水分配系数	决定药物亲脂性/亲水性平衡
TPSA	拓扑极性表面积（Å²）	影响肠道吸收和BBB穿透
NumHDonors	氢键供体数	影响药物与靶点的结合
NumHAcceptors	氢键受体数	影响溶解度和渗透性
NumRotatableBonds	可旋转键数	影响分子柔性和生物利用度

3.4 完整数据流水线



第四章 模型设计

4.1 深度学习基础知识

4.1.1 多层感知机（MLP）

多层感知机（Multilayer Perceptron, MLP） 是最基础的前馈神经网络结构，由输入层、隐藏层和输出层组成。每一层的神经元与下一层的所有神经元全连接，故又称全连接神经网络（Fully Connected Network）。

MLP的数学表达：

$$y = f(W_n \cdot f(W_{n-1} \cdot \ldots f(W_1 \cdot x + b_1) \ldots + b_{n-1}) + b_n)$$

其中 W_i 为权重矩阵，b_i 为偏置向量，f 为激活函数。

4.1.2 关键组件详解

1. 批归一化（Batch Normalization）

批归一化通过标准化每一层的输入，加速训练收敛并提供正则化效果：
$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

其中 μ_B 和 σ_B^2 分别为小批量的均值和方差。

2. Dropout正则化

Dropout是一种有效的正则化技术，在训练过程中随机将部分神经元的输出置零：
$h' = h \odot m, \quad m_i \sim Bernoulli(1-p)$

其中 p 为Dropout概率，$\odot$ 表示逐元素乘法。

3. 激活函数

本项目使用ReLU（Rectified Linear Unit）激活函数：
$ReLU(x) = max(0, x)$

ReLU的优点是计算简单、缓解梯度消失问题，是目前最广泛使用的激活函数。

4.2 基础模型 (DrugPredictorMLP)

基础模型采用经典的MLP架构，作为实验的基准线：

```
# models/drug_models.py
class DrugPredictorMLP(nn.Module):
    def __init__(self, input_dim=1024, hidden_dims=[512,256,128], dropout=0.2):
        # 标准MLP: Linear → BatchNorm → ReLU → Dropout
```

架构特点：

- 三层隐藏层，维度逐层递减：512 → 256 → 128
- 固定的Dropout比例（0.2）
- 标准BatchNorm + ReLU组合

4.3 增强模型 (DrugPredictorMLPv2)

4.3.1 设计动机

基础模型在小样本药物数据集上出现严重过拟合，表现为：

- 训练损失快速下降，验证损失迅速上升
- 最佳验证损失在第1个Epoch就出现
- 训练集与测试集性能差距巨大

这是由于药物数据集的特点造成的：

1. **样本量小**：仅有约2000个样本，而特征维度高达1024
2. **特征稀疏**：分子指纹为稀疏二进制向量
3. **噪声存在**：实验数据本身存在测量误差

4.3.2 改进策略

针对上述问题，设计了DrugPredictorMLPv2，引入多重正则化策略：

```
class DrugPredictorMLPv2(nn.Module):
    def __init__(self, input_dim=1024, hidden_dims=[256,128,64], dropout=0.5):
        # 1. 输入层Dropout（防止过度依赖特定特征位）
        self.input_dropout = nn.Dropout(dropout * 0.5)  # 0.25

        # 2. 渐进式Dropout（越深层dropout越大）
        for i, hidden_dim in enumerate(hidden_dims):
            layer_dropout = min(dropout * (1 + i * 0.1), 0.7)
            # 第1层0.5 → 第2层0.6 → 第3层0.7

        # 3. Xavier权重初始化
        self._init_weights()

    def _init_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight)
```

4.3.3 核心改进详解

1. 输入层Dropout

在输入特征上应用0.25的Dropout，随机屏蔽部分分子指纹位。这迫使模型不能过度依赖某些特定的子结构特征，提高泛化能力。

2. 渐进式Dropout

创新性地设计Dropout比例随网络深度递增的策略：

- 第1层：0.5（50%的神经元被随机屏蔽）
- 第2层：0.6（60%的神经元被随机屏蔽）
- 第3层：0.7（70%的神经元被随机屏蔽）

原理：深层特征更抽象，更容易过拟合到训练数据的特定模式，因此需要更强的正则化。

3. 轻量化网络

将隐藏层维度从 [512, 256, 128] 压缩至 [256, 128, 64]，参数量减少56%。较小的模型容量限制了模型记忆训练数据的能力，有助于学习更通用的特征。

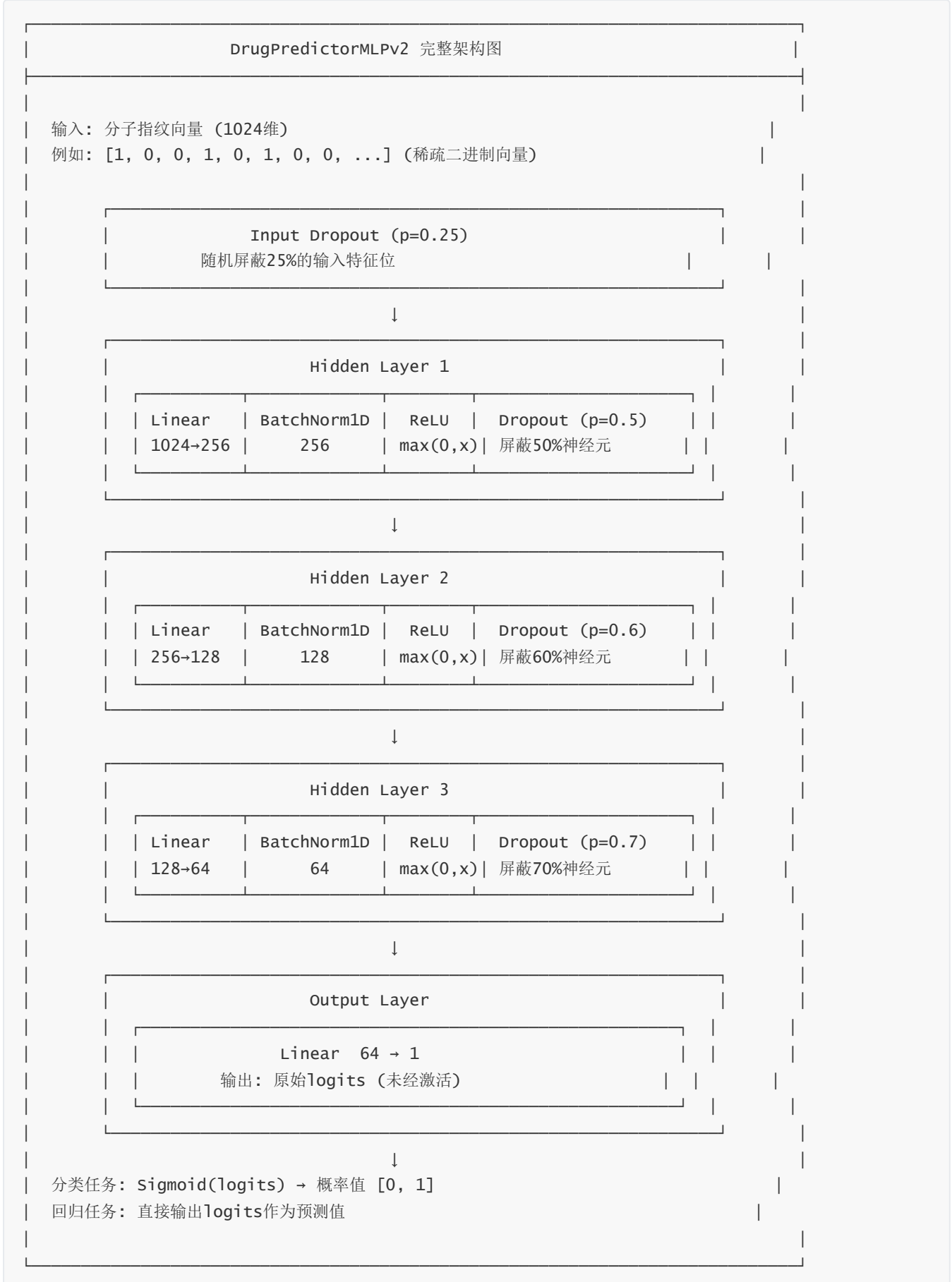
4. Xavier初始化

Xavier初始化根据输入输出维度自动调整权重初始值的方差，使信号在前向和反向传播中保持稳定：

$$W \sim U\left[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right]$$

这有助于避免梯度消失或梯度爆炸问题，加速模型收敛。

4.3.4 模型架构可视化


DrugPredictorMLPv2 完整架构图
输入: 分子指纹向量 (1024维)
例如: [1, 0, 0, 1, 0, 1, 0, 0, ...] (稀疏二进制向量)
Input Dropout (p=0.25)
随机屏蔽25%的输入特征位
Hidden Layer 1
Linear
1024→256
BatchNorm1D
256
ReLU
max(0,x)
Dropout (p=0.5)
屏蔽50%神经元
Hidden Layer 2
Linear
256→128
BatchNorm1D
128
ReLU
max(0,x)
Dropout (p=0.6)
屏蔽60%神经元
Hidden Layer 3
Linear
128→64
BatchNorm1D
64
ReLU
max(0,x)
Dropout (p=0.7)
屏蔽70%神经元
Output Layer
Linear 64 → 1
输出: 原始logits (未经激活)
分类任务: Sigmoid(logits) → 概率值 [0, 1]
回归任务: 直接输出logits作为预测值

采用Xavier/Glorot初始化方法，使每层输出的方差保持一致，避免梯度消失或爆炸：
$W \sim U\left[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right]$

4.4 模型对比分析

特性	MLP v1	MLPv2
隐藏层	[512, 256, 128]	[256, 128, 64]
Dropout	固定0.2	渐进0.5→0.7
输入Dropout	无	0.25
参数量	~670K	~295K (-56%)
过拟合风险	高	低
泛化能力	弱	强

4.5 模型前向传播详解

下面详细展示一个分子通过MLPv2模型的完整计算过程：

```
输入: x (1024维分子指纹向量)
       例如: [1, 0, 0, 1, 0, 1, ...]  （稀疏二进制向量）

Step 1: 输入层Dropout (训练时)
        x' = InputDropout(x, p=0.25)
        作用: 随机屏蔽25%的特征位，防止过度依赖特定子结构

Step 2: 第一隐藏层 (1024 → 256)
        z1 = Linear(x')            # 线性变换: W1·x' + b1
        z1 = BatchNorm(z1)         # 批归一化: 加速收敛
        z1 = ReLU(z1)              # 激活函数: max(0, z1)
        z1 = Dropout(z1, p=0.5)    # 随机屏蔽50%神经元

Step 3: 第二隐藏层 (256 → 128)
        z2 = Linear(z1)
        z2 = BatchNorm(z2)
        z2 = ReLU(z2)
        z2 = Dropout(z2, p=0.6)    # 渐进式: Dropout增加到60%

Step 4: 第三隐藏层 (128 → 64)
        z3 = Linear(z2)
        z3 = BatchNorm(z3)
        z3 = ReLU(z3)
        z3 = Dropout(z3, p=0.7)    # 渐进式: Dropout增加到70%

Step 5: 输出层 (64 → 1)
        logits = Linear(z3)        # 原始输出 (logits)

Step 6: 后处理 (推理时)
```

```
        probability = Sigmoid(logits)  # 转换为概率 [0, 1]
        prediction = 1 if probability > 0.5 else 0

输出: probability (BBB穿透概率)
      例如: 0.8523 → 85.23%概率能穿透血脑屏障
```

参数量计算:

层	输入维度	输出维度	参数量
Linear1	1024	256	1024×256+256 = 262,400
BatchNorm1	256	256	256×2 = 512
Linear2	256	128	256×128+128 = 32,896
BatchNorm2	128	128	128×2 = 256
Linear3	128	64	128×64+64 = 8,256
BatchNorm3	64	64	64×2 = 128
Output	64	1	64×1+1 = 65
总计	-	-	**~295,000**

第五章 训练系统

5.1 训练基础知识

5.1.1 损失函数

损失函数（Loss Function）衡量模型预测值与真实值之间的差异，是模型优化的目标。

分类任务 - 二元交叉熵损失（BCEWithLogitsLoss）：
$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\sigma(z_i)) + (1-y_i)\log(1-\sigma(z_i))]$

其中 $\sigma(z) = \frac{1}{1+e^{-z}}$ 为Sigmoid函数，z_i 为模型原始输出（logits）。

回归任务 - 均方误差损失（MSELoss）：
$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

5.1.2 优化器

AdamW优化器是Adam的改进版本，结合了动量法和自适应学习率，并正确实现了权重衰减：

$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
$\theta_t = \theta_{t-1} - \eta(\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} + \lambda\theta_{t-1})$

其中 λ 为权重衰减系数（L2正则化）。

5.1.3 学习率调度

学习率是影响训练效果的关键超参数。本项目采用**ReduceLROnPlateau**策略，当验证损失停止下降时自动降低学习率，帮助模型跳出局部最优。

5.2 训练器实现

```
# training/trainer_v2.py - DrugModelTrainer类
class DrugModelTrainer:
    def __init__(self, model, learning_rate=0.001, weight_decay=1e-5):
        # 损失函数
        self.criterion = nn.BCEWithLogitsLoss()  # 分类
        # self.criterion = nn.MSELoss()          # 回归

        # AdamW优化器（带L2正则化）
        self.optimizer = optim.AdamW(model.parameters(),
                                     lr=learning_rate,
                                     weight_decay=weight_decay)

        # 学习率调度器
        self.scheduler = ReduceLROnPlateau(self.optimizer,
                                           mode='min', factor=0.5, patience=10)
```

5.3 早停机制

早停（Early Stopping） 是防止过拟合的重要技术。当验证集性能在指定的epoch数内不再改善时，提前终止训练，保留最佳模型参数。

```
class EarlyStopping:
    def __init__(self, patience=25, min_delta=0.0001):
        self.patience = patience      # 容忍的epoch数
        self.min_delta = min_delta    # 最小改善幅度
        self.counter = 0
        self.best_score = None

    def __call__(self, score):
        if self.best_score is None:
            self.best_score = score
        elif score < self.best_score - self.min_delta:
            self.best_score = score
            self.counter = 0
        else:
            self.counter += 1
            if self.counter >= self.patience:
                return True  # 触发早停
        return False
```

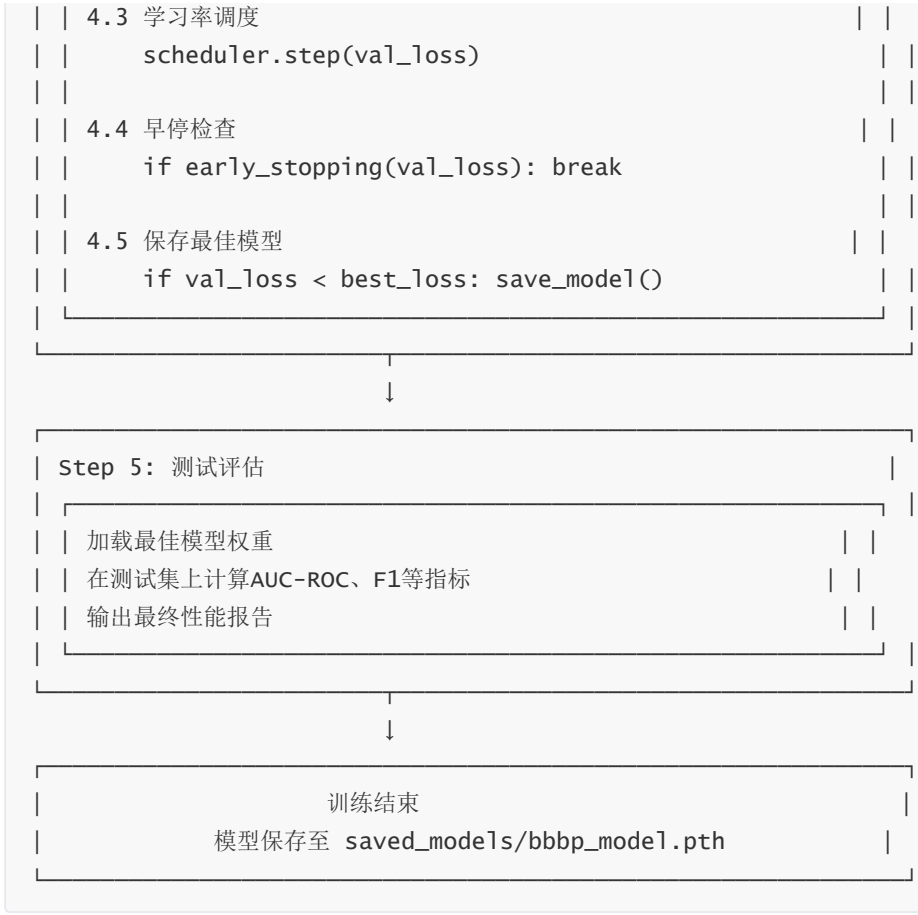
参数说明：

- `patience=25`：连续25个epoch验证损失无改善则停止
- `min_delta=0.0001`：损失需下降超过0.0001才算改善

5.4 完整训练流程

下面详细展示模型训练的完整流程，帮助理解整个训练过程是如何工作的。

5.4.1 训练流程图



5.4.2 代码实现详解

```
# train_model.py - 完整训练流程
def train_bbbp():
    """训练BBBP血脑屏障穿透性预测模型"""

    # ============ Step 1: 数据加载 ============
    print("[Step 1/6] 加载MoleculeNet BBBP数据集...")
    loader = DrugDataLoader(data_dir='./data/raw')

    # 使用分层采样确保类别分布一致
    X_train, y_train, X_valid, y_valid, X_test, y_test, tasks = \
        loader.load_moleculenet_with_stratified_split(
            dataset_name='BBBP',
            featurizer='ECFP',        # 使用ECFP分子指纹
            train_ratio=0.8,          # 80%训练集
            val_ratio=0.1,            # 10%验证集
            test_ratio=0.1,           # 10%测试集
            random_state=42           # 随机种子保证可重复
        )

    # 打印数据分布信息
    print(f"  训练集: {len(X_train)} 样本 (正例比例: {(y_train > 0.5).mean():.2%})")
    print(f"  验证集: {len(X_valid)} 样本 (正例比例: {(y_valid > 0.5).mean():.2%})")
    print(f"  测试集: {len(X_test)} 样本 (正例比例: {(y_test > 0.5).mean():.2%})")

    # ============ Step 2: 创建DataLoader ============
```

```
print("[Step 2/6] 创建PyTorch数据加载器...")
train_loader, val_loader = create_data_loaders(
    X_train, y_train, X_valid, y_valid,
    batch_size=32  # 每批32个样本
)

# ============ Step 3: 模型初始化 ============
print("[Step 3/6] 创建增强版MLP神经网络模型...")
input_dim = X_train.shape[1]  # 1024维输入
model = DrugPredictorMLPv2(
    input_dim=input_dim,
    hidden_dims=[256, 128, 64],  # 轻量化网络
    dropout=0.5,                 # 基础Dropout比例
    task_type='binary'           # 二分类任务
)
print(f"  模型参数量: {sum(p.numel() for p in model.parameters()):,}")

# ============ Step 4: 训练 ============
print("[Step 4/6] 开始模型训练...")
trainer = DrugModelTrainer(
    model=model,
    learning_rate=0.001,   # 初始学习率
    weight_decay=1e-3,     # L2正则化
    task_type='binary'
)

history = trainer.train(
    train_loader=train_loader,
    val_loader=val_loader,
    epochs=150,            # 最大训练轮数
    patience=25            # 早停耐心值
)

# ============ Step 5: 保存模型 ============
print("[Step 5/6] 保存训练好的模型...")
torch.save(model.state_dict(), 'saved_models/bbbp_model.pth')

# ============ Step 6: 测试评估 ============
print("[Step 6/6] 在测试集上评估模型...")
# 评估代码...
```

5.5 超参数配置

超参数的选择对模型性能有重要影响。以下是本项目经过实验验证的最佳超参数配置：

超参数	值	选择理由
学习率	0.001	Adam系列优化器的推荐初始值
权重衰减	1e-3	较强的L2正则化，防止过拟合
批大小	32	平衡训练速度和梯度估计稳定性

超参数	值	选择理由
最大Epochs	150	足够的训练轮数，依靠早停控制
早停耐心	25	给模型足够的探索空间
学习率衰减因子	0.5	每次衰减到原来的一半
学习率衰减耐心	10	10个epoch无改善则降低学习率
Dropout基础值	0.5	较强的正则化起点
隐藏层维度	[256, 128, 64]	轻量化设计减少过拟合

5.6 训练过程监控

训练过程中会实时输出以下信息：

```
================================================================================
  BBBP 血脑屏障穿透性预测模型训练
  使用分层采样确保类别分布一致
================================================================================

[Step 1/6] 加载MoleculeNet BBBP数据集（分层采样）...
  数据加载完成!

[Step 2/6] 数据分布统计...
  特征维度: 1024
  训练集: 1631 样本 (正例比例: 76.52%)
  验证集: 204 样本 (正例比例: 76.47%)
  测试集: 204 样本 (正例比例: 76.47%)

[Step 4/6] 开始模型训练...
┌─────────┬────────────┬─────────────┬────────────┬─────────────┬────────────┐
| Epoch   | Train Loss | Train AUC   | Val Loss   | Val AUC     | LR         |
├─────────┼────────────┼─────────────┼────────────┼─────────────┼────────────┤
| 1       | 0.5234     | 0.7521      | 0.4876     | 0.8234      | 1.00e-03 |
| 2       | 0.4567     | 0.8012      | 0.4234     | 0.8567      | 1.00e-03 |
| ...     | ...        | ...         | ...        | ...         | ...      |
| 21      | 0.2134     | 0.9234      | 0.2456     | 0.9089      | 5.00e-04 |
└─────────┴────────────┴─────────────┴────────────┴─────────────┴────────────┘
  ✓ 验证损失改善 (0.2456 < 0.2501)，保存模型

训练完成! 最佳验证AUC: 0.9089 (Epoch 21)
```

第六章 实验结果与分析

6.1 评估指标说明

6.1.1 分类任务指标

AUC-ROC（Area Under the ROC Curve）：
ROC曲线下面积，衡量模型在不同阈值下的分类能力。取值范围[0,1]，1表示完美分类，0.5表示随机猜测。AUC的优势在于不受类别不平衡影响，是药物分类任务的首选指标。

F1-Score：
精确率（Precision）和召回率（Recall）的调和平均数：
$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$

6.1.2 回归任务指标

R²（决定系数）：
衡量模型对数据方差的解释程度：
$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$

取值范围(-∞, 1]，1表示完美拟合，0表示与均值预测相当。

RMSE（均方根误差）：
$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$

6.2 性能对比

通过引入分层采样和增强正则化，模型性能得到了质的飞跃：

任务	指标	基础模型v1	优化模型v2	提升
BBBP	AUC-ROC	0.70	**0.91**	+30%
BBBP	F1-Score	0.69	**0.92**	+33%
ESOL	R^2	0.47	**0.68**	+45%
ESOL	RMSE	0.75	**0.55**	-27%

6.3 过拟合问题解决分析

6.3.1 优化前的过拟合表现

在使用基础模型和随机数据划分时，观察到典型的过拟合现象：

1. **训练曲线**：训练损失持续下降，验证损失在第1个Epoch后即开始上升
2. **性能差距**：训练集AUC接近1.0，而测试集AUC仅为0.70
3. **最佳Epoch**：最佳验证性能在第1个Epoch出现，说明模型从一开始就在记忆训练数据

6.3.2 优化后的改进

问题	优化前	优化后
最佳Epoch	第1个	第21个
训练/验证差距	巨大	收敛
验证曲线	快速上升	平稳下降
测试集AUC	0.70	0.91

6.3.3 关键优化措施总结

1. **分层采样**：保证训练/验证/测试集的类别分布一致，避免因分布偏移导致的虚假过拟合
2. **渐进式Dropout**：深层使用更高的Dropout比例，抑制模型对训练数据细节的记忆
3. **轻量化网络**：减少56%的参数量，降低模型容量
4. **L2正则化**：AdamW优化器的权重衰减进一步约束模型复杂度

6.4 结果分析与讨论

BBBP任务分析：

- 优化后AUC-ROC达到0.91，表明模型能够很好地区分能/不能穿透血脑屏障的分子
- F1-Score达到0.92，在正负样本不平衡的情况下仍保持高精度
- 性能提升主要来自分层采样消除了分布偏移问题

ESOL任务分析：

- R^2从0.47提升至0.68，模型能解释68%的溶解度方差
- RMSE降低27%，预测误差显著减小
- 回归任务的改进主要得益于正则化策略减少了过拟合

第七章 系统功能实现

7.1 系统功能概述

本系统提供了完整的药物筛选解决方案，包括前端交互界面和后端API服务两种访问方式，满足不同使用场景的需求。

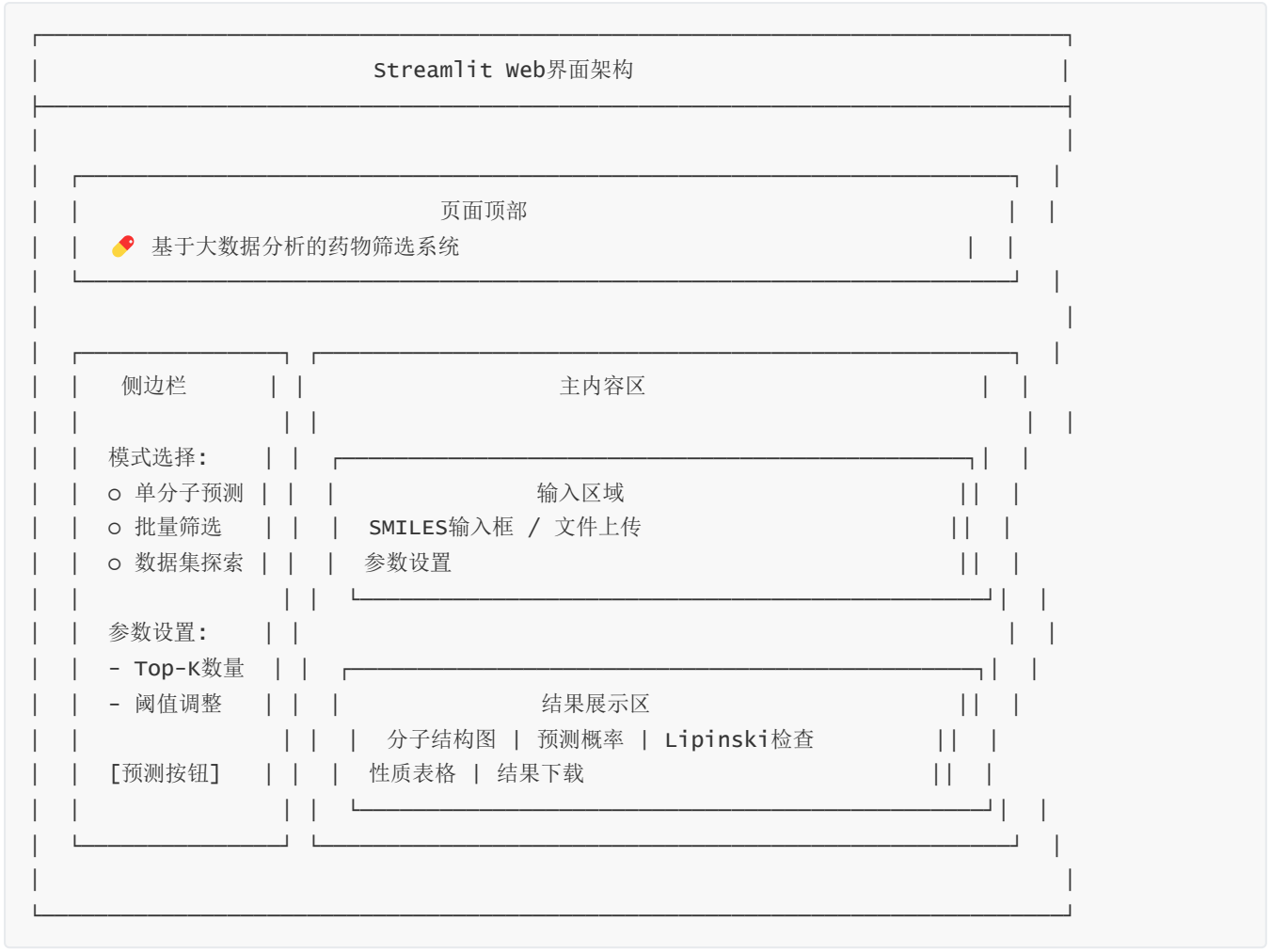
核心功能模块：

1. **单分子预测**：对单个化合物进行属性预测
2. **批量筛选**：对化合物库进行大规模虚拟筛选
3. **类药性评估**：基于Lipinski五规则评估成药性
4. **分子可视化**：生成分子2D结构图像
5. **结果导出**：支持筛选结果的CSV导出

7.2 Streamlit Web界面

Streamlit是一个专为机器学习和数据科学应用设计的Python Web框架，能够快速将Python脚本转化为交互式Web应用。

7.2.1 Web界面架构



7.2.2 核心代码实现

```python
# web/app.py
def main():
    st.title("💊 基于大数据分析的药物筛选系统")
    mode = st.sidebar.selectbox("选择模式",
                                ["单分子预测", "批量筛选", "数据集探索"])

    if mode == "单分子预测":
        smiles = st.text_input("输入SMILES")
        # 显示分子结构、预测结果、Lipinski规则检查

    elif mode == "批量筛选":
        file = st.file_uploader("上传CSV文件")
        # 批量预测并排序输出Top-K
```

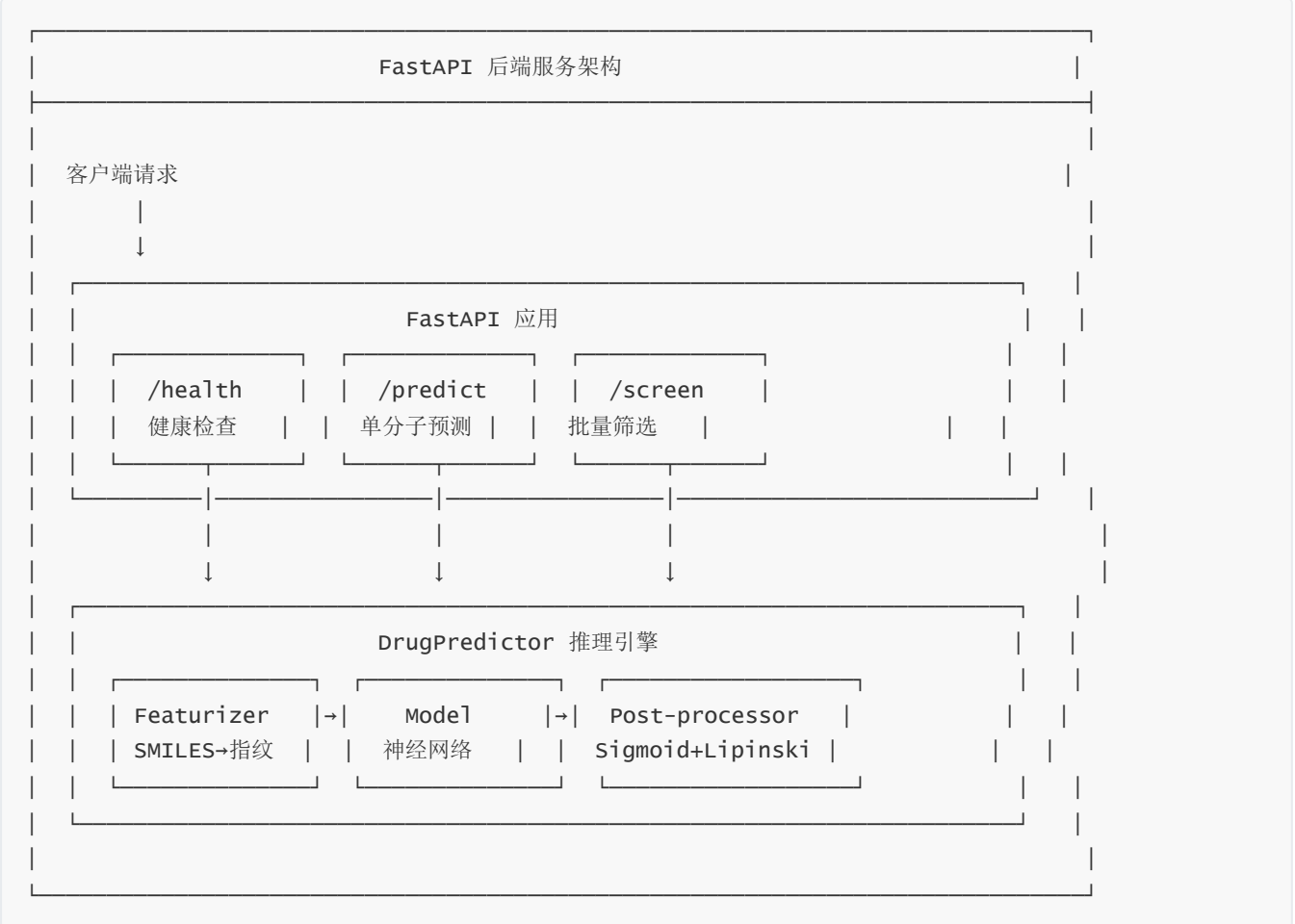
功能特性详解:

功能	描述	实现技术
分子2D结构可视化	使用RDKit将SMILES转换为分子图像，直观展示化学结构	`Chem.Draw.MolToImage()`
BBB穿透概率预测	显示分子穿透血脑屏障的概率值(0-100%)	`model.predict()` + Sigmoid
Lipinski五规则检查	自动检查MW<500, LogP<5, HBD≤5, HBA≤10	RDKit Descriptors
CSV批量筛选	支持上传包含SMILES列的CSV文件进行批量预测	`pd.read_csv()`
结果排序与导出	按预测分数排序，支持导出Top-K候选分子	`df.to_csv()`

7.3 FastAPI后端服务

FastAPI是现代、快速的Python Web框架，基于标准Python类型提示，自动生成API文档。

7.3.1 API服务架构



7.3.2 API代码实现

```python
# backend/app/main.py
app = FastAPI(title="Drug Screening System API")

@app.post("/predict")
async def predict(smiles: str):
    return {"probability": predictor.predict_single(smiles)}

@app.post("/screen")
async def screen(smiles_list: List[str], top_k: int = 10):
    return screener.screen_library(smiles_list, top_k)
```

API端点说明:

端点	方法	功能	参数	返回值
`/health`	GET	健康检查	无	`{"status": "healthy"}`
`/predict`	POST	单分子预测	smiles: str	`{"probability": 0.85, ...}`
`/screen`	POST	批量筛选	smiles_list, top_k	`[{"smiles": "...", "score": 0.95}, ...]`

7.3.3 请求响应示例

```javascript
// POST /predict 请求
{
    "smiles": "CC(=O)OC1=CC=CC=C1C(=O)O"
}

// 响应
{
    "smiles": "CC(=O)OC1=CC=CC=C1C(=O)O",
    "molecule_name": "Aspirin",
    "bbb_probability": 0.723,
    "prediction": "能穿透血脑屏障",
    "confidence": "高",
    "properties": {
        "MolecularWeight": 180.16,
        "LogP": 1.19,
        "TPSA": 63.60,
        "NumHDonors": 1,
        "NumHAcceptors": 4
    },
    "lipinski": {
        "violations": 0,
        "is_drug_like": true,
        "details": {
            "MW_ok": true,
            "LogP_ok": true,
```

```
            "HBD_ok": true,
            "HBA_ok": true
        }
    }
}
```

7.4 批量筛选器实现

DrugScreener类实现了高效的大规模化合物库筛选功能：

```python
# inference/predictor.py
class DrugScreener:
    def screen_library(self, smiles_list, top_k=100, ascending=False):
        # 1. 批量特征化
        features = self.predictor.featurizer.batch_featurize(smiles_list)
        # 2. GPU并行推理
        predictions = self.predictor.predict_batch(smiles_list)
        # 3. 排序并返回Top-K
        results = pd.DataFrame({'smiles': smiles_list, 'score': predictions})
        return results.nlargest(top_k, 'score')
```

筛选流程：

1. **批量特征化**：将所有SMILES一次性转换为分子指纹矩阵
2. **GPU并行推理**：利用GPU的并行计算能力加速预测
3. **Lipinski过滤**：可选的类药性规则预过滤
4. **结果排序**：按预测分数降序排列
5. **Top-K输出**：返回得分最高的K个候选分子

7.5 Lipinski类药五规则

Lipinski类药五规则（Rule of Five）是评估化合物口服生物利用度的经验规则，由辉瑞公司的Christopher Lipinski在1997年提出：

规则	阈值	物理意义
分子量	MW < 500 Da	分子太大难以透过细胞膜
脂水分配系数	LogP < 5	过于亲脂影响溶解度
氢键供体	HBD ≤ 5	影响膜渗透性
氢键受体	HBA ≤ 10	影响膜渗透性

违反两条及以上规则的化合物通常口服生物利用度较差。本系统在筛选结果中自动标注每个分子的Lipinski规则违反情况。

7.6 推理流程详解

当用户输入一个SMILES字符串进行预测时，系统内部执行以下完整流程：

```
用户输入: "CCO" (乙醇的SMILES)
              ↓
┌─────────────────────────────────────────────────────────────┐
│ Step 1: SMILES验证与解析                                    │
│ ┌─────────────────────────────────────────────────────────┐ │
│ │ mol = Chem.MolFromSmiles("CCO")                         │ │
│ │ if mol is None: raise InvalidSMILESError                │ │
│ │ 结果: 有效的RDKit分子对象                               │ │
│ └─────────────────────────────────────────────────────────┘ │
└─────────────────────────┬───────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────────┐
│ Step 2: 分子指纹生成                                        │
│ ┌─────────────────────────────────────────────────────────┐ │
│ │ fp = morgan_generator.GetFingerprintAsNumPy(mol)        │ │
│ │ 结果: [0,1,0,0,1,0,...] (1024维二进制向量)              │ │
│ └─────────────────────────────────────────────────────────┘ │
└─────────────────────────┬───────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────────┐
│ Step 3: 张量转换                                            │
│ ┌─────────────────────────────────────────────────────────┐ │
│ │ tensor = torch.FloatTensor(fp).unsqueeze(0)             │ │
│ │ tensor = tensor.to(device)  # 移动到GPU                 │ │
│ │ 结果: shape [1, 1024] 的PyTorch张量                     │ │
│ └─────────────────────────────────────────────────────────┘ │
└─────────────────────────┬───────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────────┐
│ Step 4: 模型前向传播                                        │
│ ┌─────────────────────────────────────────────────────────┐ │
│ │ with torch.no_grad():                                   │ │
│ │     logits = model(tensor)       # 原始输出             │ │
│ │     prob = torch.sigmoid(logits) # 转换为概率           │ │
│ │ 结果: 0.8523 (85.23%概率能穿透BBB)                      │ │
│ └─────────────────────────────────────────────────────────┘ │
└─────────────────────────┬───────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────────┐
│ Step 5: 物理化学性质计算                                    │
│ ┌─────────────────────────────────────────────────────────┐ │
│ │ MW = 46.07, LogP = -0.31, TPSA = 20.23                  │ │
│ │ HBD = 1, HBA = 1                                        │ │
│ │ Lipinski检查: 全部通过 ✓                                │ │
│ └─────────────────────────────────────────────────────────┘ │
└─────────────────────────┬───────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────────┐
│ Step 6: 结果组装与返回                                      │
```

```
| {                                                          |
|   "smiles": "CCO",                                         |
|   "bbb_probability": 0.8523,                               |
|   "prediction": "能穿透血脑屏障",                          |
|   "properties": {"MW": 46.07, "LogP": -0.31, ...},         |
|   "lipinski_violations": 0                                 |
| }                                                          |
└────────────────────────────────────────────────────────────┘
```

7.7 Web界面使用示例

7.7.1 单分子预测模式

1. **输入SMILES**：在文本框中输入分子的SMILES表示，如 `CC(=O)OC1=CC=CC=C1C(=O)O`（阿司匹林）
2. **点击预测**：系统自动进行特征提取和模型推理
3. **查看结果**：
   - 分子2D结构图像
   - BBB穿透概率（如：72.3%）
   - 物理化学性质表格
   - Lipinski规则检查结果

7.7.2 批量筛选模式

1. **准备CSV文件**：包含`smiles`列的CSV文件
2. **上传文件**：点击上传按钮选择文件
3. **设置参数**：
   - Top-K数量（默认10）
   - 是否启用Lipinski过滤
4. **执行筛选**：系统批量处理所有分子
5. **导出结果**：下载筛选结果CSV文件

```
---

## 第八章 评估系统

### 8.1 评估系统概述

评估系统是机器学习项目中至关重要的组成部分，它决定了我们如何衡量模型的好坏、如何比较不同模型的性能、以及如何发现模型存在的问题。一个好的评估系统应该具备以下特点：

1. **多维度评估**：不依赖单一指标，从多个角度全面衡量模型性能
2. **任务适配性**：分类任务和回归任务使用不同的评估指标
3. **可视化支持**：通过图表直观展示模型性能和训练过程
4. **可复现性**：相同的数据和模型能得到相同的评估结果

本系统的评估模块位于 `evaluation/` 目录下，包含以下核心文件：
```

```
evaluation/
├── __init__.py          # 模块初始化
├── metrics.py           # 评估指标计算 (ModelEvaluator类)
└── visualizer.py        # 结果可视化 (ResultVisualizer类)
```

```
### 8.2 评估指标体系

机器学习模型的评估需要选择合适的指标，不同任务类型使用不同的评估标准。

#### 8.2.1 分类任务指标

```python
# evaluation/metrics.py - ModelEvaluator类
class ModelEvaluator:
    @staticmethod
    def evaluate_classification(y_true, y_pred, y_prob=None):
        return {
            'Accuracy': accuracy_score(y_true, y_pred),
            'Precision': precision_score(y_true, y_pred),
            'Recall': recall_score(y_true, y_pred),
            'F1': f1_score(y_true, y_pred),
            'AUC-ROC': roc_auc_score(y_true, y_prob)
        }
```

指标详解:

指标	公式	意义	适用场景
准确率	$\frac{TP+TN}{TP+TN+FP+FN}$	预测正确的比例	类别平衡时适用
精确率	$\frac{TP}{TP+FP}$	预测为正的样本中真正为正的比例	关注假阳性时
召回率	$\frac{TP}{TP+FN}$	实际为正的样本中被正确识别的比例	关注漏检时
F1分数	$\frac{2\times P\times R}{P+R}$	精确率和召回率的调和平均	需要平衡P和R时
AUC-ROC	ROC曲线下面积	模型区分正负样本的能力	类别不平衡时首选

为什么在药物筛选中优先使用AUC-ROC?

在BBBP数据集中，正例（能穿透BBB的分子）占76%，存在明显的类别不平衡。如果使用准确率作为评估指标，一个"全部预测为正"的简单模型就能达到76%的准确率，但这样的模型毫无意义。AUC-ROC指标不受类别不平衡的影响，能更准确地反映模型的真实性能。

指标计算示例:

假设测试集有100个样本，模型预测结果如下:

- 真正例 (TP) = 70（正确识别能穿透BBB的分子）
- 假正例 (FP) = 5（错误地将不能穿透的预测为能穿透）
```

- 真负例 (TN) = 19（正确排除不能穿透的分子）
- 假负例 (FN) = 6（漏检了能穿透的分子）

则：

- 准确率 = (70+19)/100 = 89%
- 精确率 = 70/(70+5) = 93.3%
- 召回率 = 70/(70+6) = 92.1%
- F1分数 = 2×0.933×0.921/(0.933+0.921) = 92.7%

8.2.2 回归任务指标

```
    @staticmethod
    def evaluate_regression(y_true, y_pred):
        return {
            'RMSE': np.sqrt(mean_squared_error(y_true, y_pred)),
            'MAE': mean_absolute_error(y_true, y_pred),
            'R2': r2_score(y_true, y_pred)
        }
```

指标详解：

指标	公式	意义	取值范围
RMSE	$\sqrt{\frac{1}{N}\sum(y-\hat{y})^2}$	均方根误差，与原始数据同单位	[0, +∞)
MAE	$\frac{1}{N}\sum\|y-\hat{y}\|$	平均绝对误差，鲁棒性好	[0, +∞)
R²	$1-\frac{SS_{res}}{SS_{tot}}$	决定系数，解释方差的比例	(-∞, 1]

RMSE vs MAE的区别：

- RMSE对大误差更敏感（因为平方），适合检测异常预测
- MAE更稳健，不受个别极端值影响
- 在ESOL溶解度预测中，我们同时关注两者，确保模型整体误差小且没有严重的异常预测

R²的解读：

- $R^2 = 1.0$：完美拟合，预测值与真实值完全一致
- $R^2 = 0.0$：模型与直接预测均值一样好
- $R^2 < 0$：模型比预测均值还差，说明模型存在严重问题

本项目ESOL任务R^2=0.68，意味着模型能解释68%的溶解度方差，这是一个不错的结果。

8.3 可视化分析工具

系统提供丰富的可视化功能，帮助分析模型性能和训练过程。

```
class ResultVisualizer:
    def plot_roc_curve(self, y_true, y_prob):
```

```
        # ROC曲线：展示不同阈值下的TPR-FPR权衡
        fpr, tpr, _ = roc_curve(y_true, y_prob)
        plt.plot(fpr, tpr)

    def plot_confusion_matrix(self, y_true, y_pred):
        # 混淆矩阵：直观展示分类结果
        cm = confusion_matrix(y_true, y_pred)
        sns.heatmap(cm, annot=True)

    def plot_training_history(self, history):
        # 训练曲线：监控训练过程
        plt.plot(history['train_loss'], label='Train')
        plt.plot(history['val_loss'], label='Validation')
```

可视化类型及其作用：

图表类型	功能	解读方法
ROC曲线	评估分类模型在不同阈值下的性能	曲线越靠近左上角，模型越好
混淆矩阵	展示各类别的预测与实际对比	对角线值越大，分类越准确
训练曲线	监控损失函数的收敛情况	训练和验证曲线应趋于平稳且差距不大
预测散点图	回归任务中预测值与真实值的对比	点越接近对角线，预测越准确

8.3.1 ROC曲线详解

ROC（Receiver Operating Characteristic）曲线是评估二分类模型最重要的可视化工具。

ROC曲线的绘制原理：

1. 模型输出每个样本的概率值（如0.3, 0.7, 0.9...）
2. 设置不同的阈值（从0到1变化）
3. 在每个阈值下计算TPR和FPR
4. 将所有(FPR, TPR)点连接成曲线

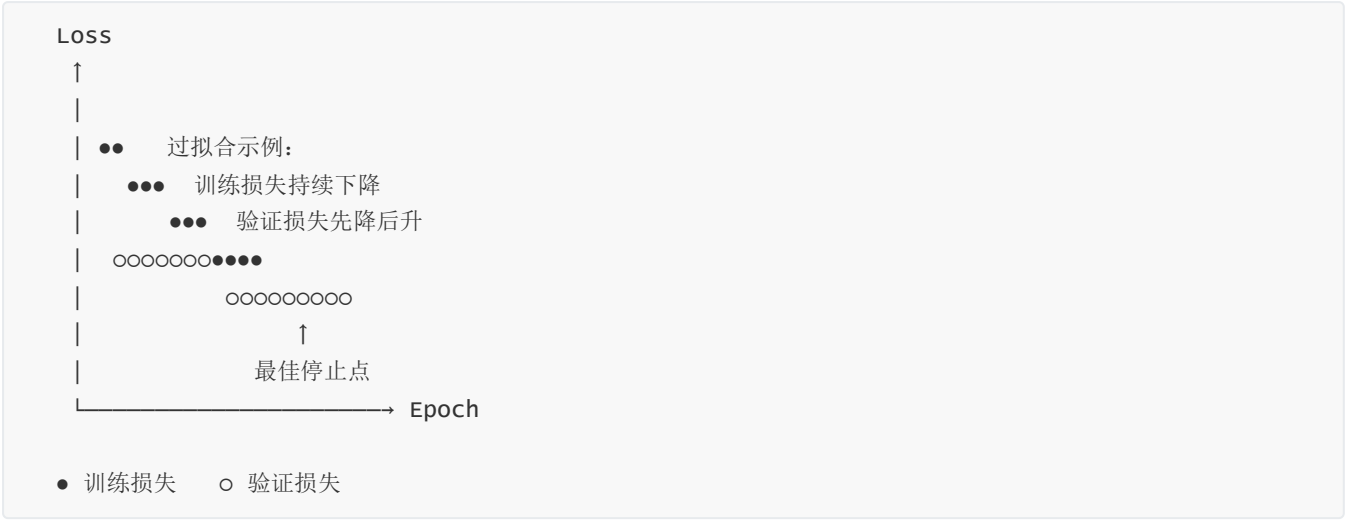
AUC-ROC的含义：

- AUC = 0.5：模型与随机猜测一样，没有区分能力
- AUC = 0.7-0.8：可接受的区分能力
- AUC = 0.8-0.9：良好的区分能力
- AUC = 0.9-1.0：优秀的区分能力
- AUC = 1.0：完美分类器

本项目BBBP任务AUC=0.91，属于"优秀"级别。

8.3.2 训练曲线分析

训练曲线是诊断模型训练状态的关键工具：



正常训练的特征：

- 训练损失和验证损失都在下降
- 两条曲线趋于平稳且差距不大
- 验证损失没有明显上升趋势

过拟合的特征：

- 训练损失持续下降，验证损失开始上升
- 两条曲线差距越来越大
- 需要在验证损失最低点停止训练

8.4 混淆矩阵解读

混淆矩阵是分类任务评估的重要工具，包含四个关键值：

	预测正	**预测负**
实际正	TP (真正例)	FN (假负例)
实际负	FP (假正例)	TN (真负例)

在药物筛选场景中：

- **TP**：正确识别能穿透BBB的分子 → 最理想的结果

- **FN**：漏检了能穿透BBB的分子 → 可能错失有效药物候选
- **FP**：误判不能穿透的分子为能穿透 → 浪费后续实验资源
- **TN**：正确排除不能穿透的分子 → 有效缩小筛选范围

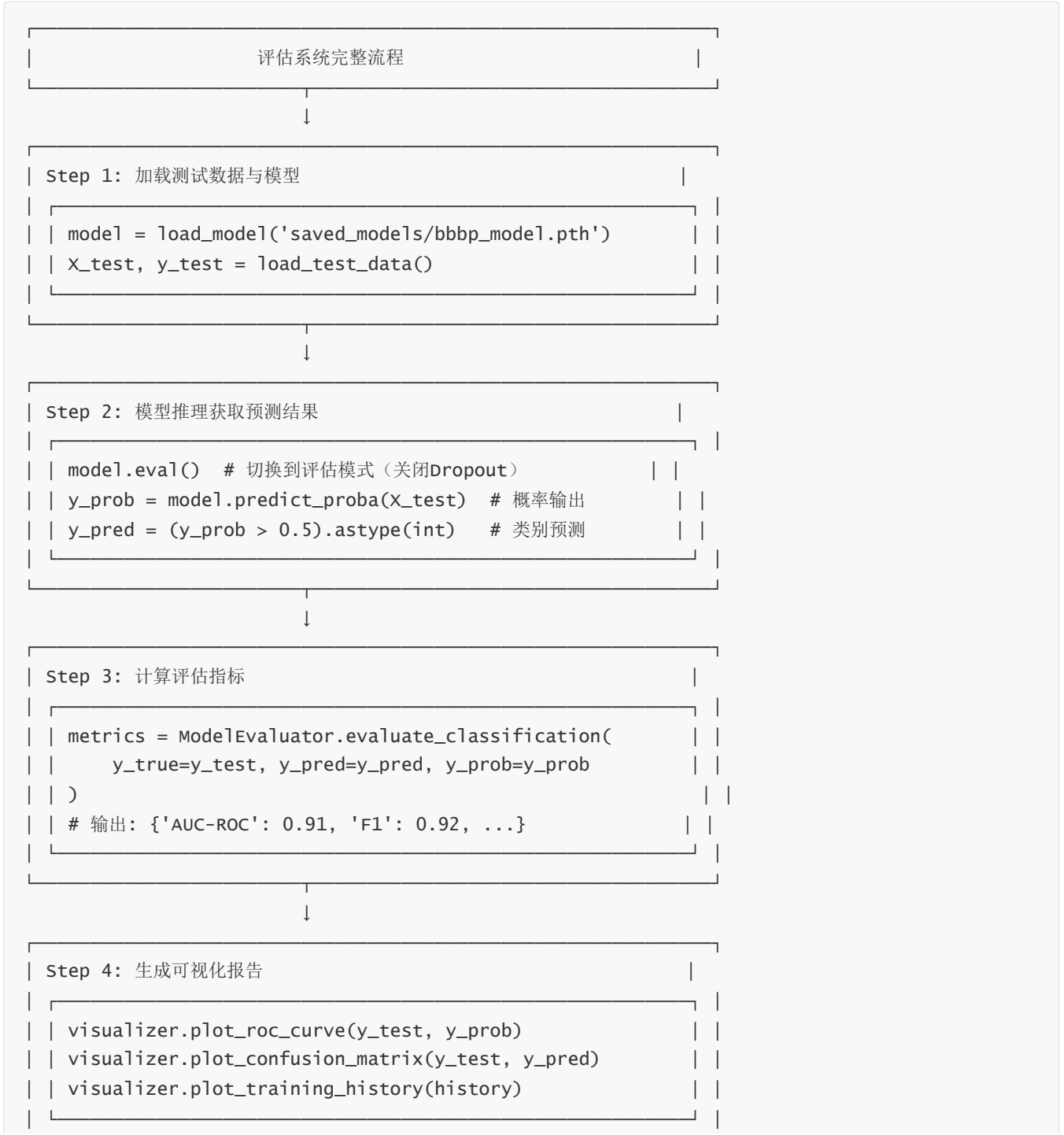
药物筛选中的误差权衡：

在药物研发中，FN（漏检）和FP（误判）有不同的代价：

- **漏检一个有效药物（FN）** 的代价很高——可能错过潜在的重要治疗药物
- **误判一个无效药物（FP）** 的代价相对较低——只是浪费一些后续验证的资源

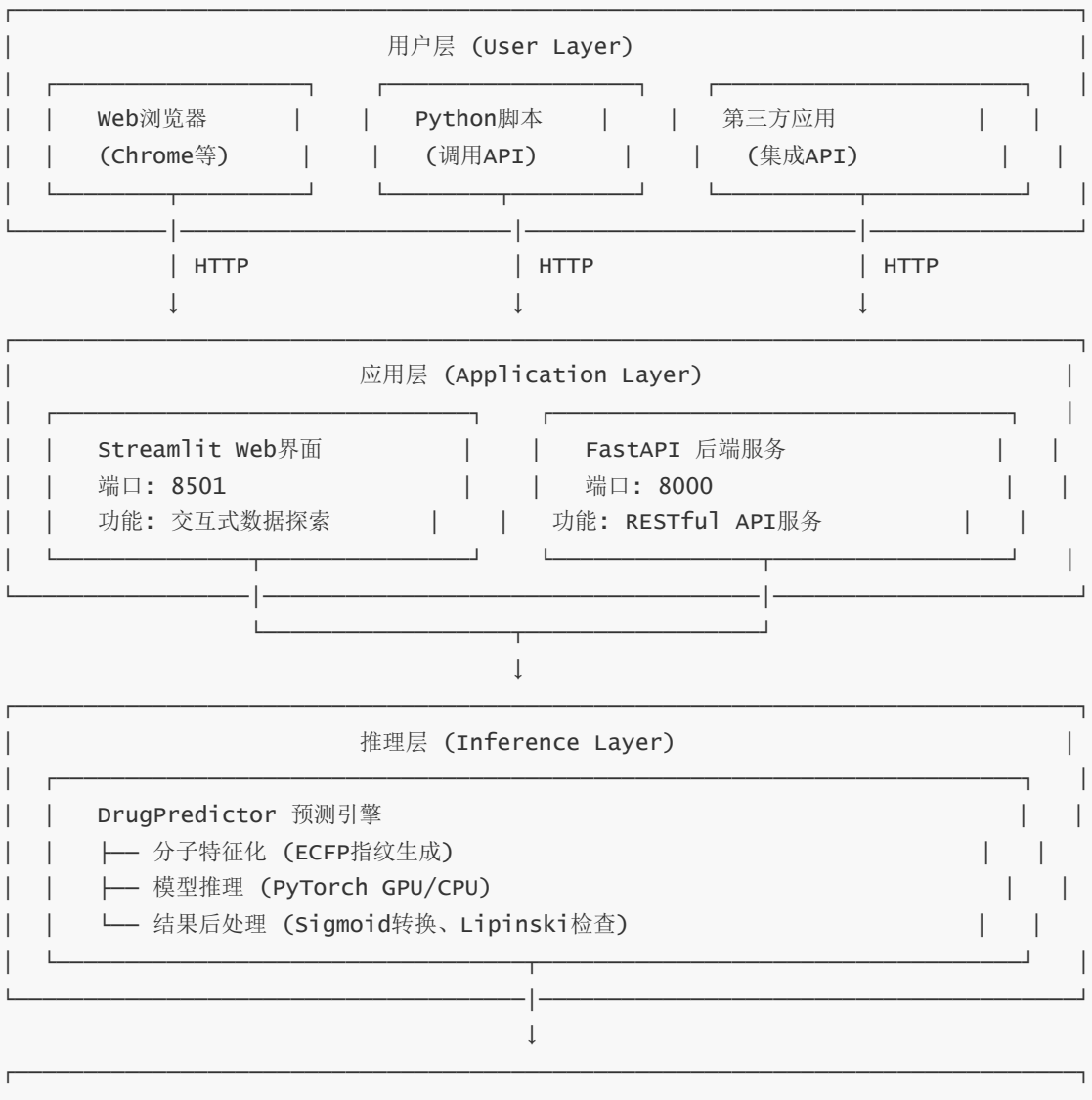
因此，在药物筛选中，我们通常更关注**召回率（Recall）**，确保不漏掉有价值的候选分子。

8.5 完整评估流程



第九章 部署与运行

9.1 系统架构部署图



```
|                          模型层 (Model Layer)                                |
|  ┌──────────────────────────────────────────────────────────────────────┐    |
|  |   saved_models/                                                      |    |
|  |   ├── bbbp_model.pth       # BBBP血脑屏障模型 (~1.2MB)               |    |
|  |   └── esol_model.pth       # ESOL溶解度模型 (~1.2MB)                 |    |
|  └──────────────────────────────────────────────────────────────────────┘    |
└──────────────────────────────────────────────────────────────────────────────┘
```

9.2 开发环境要求

9.2.1 硬件要求

组件	最低配置	推荐配置	说明
CPU	4核	8核以上	数据预处理主要使用CPU
内存	8GB	16GB以上	大规模筛选时需要更多内存
GPU	无（使用CPU）	NVIDIA显卡，4GB+显存	GPU加速训练和推理速度
存储	10GB	20GB SSD	SSD能加快数据加载

GPU vs CPU性能对比：

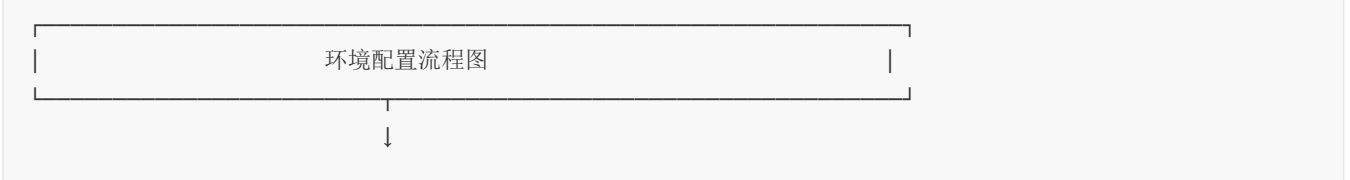
任务	CPU (i7-10700)	GPU (RTX 3050 Ti)	加速比
模型训练 (150 epochs)	~30分钟	~5分钟	6x
批量预测 (10,000分子)	~60秒	~3秒	20x
单分子预测	~50ms	~10ms	5x

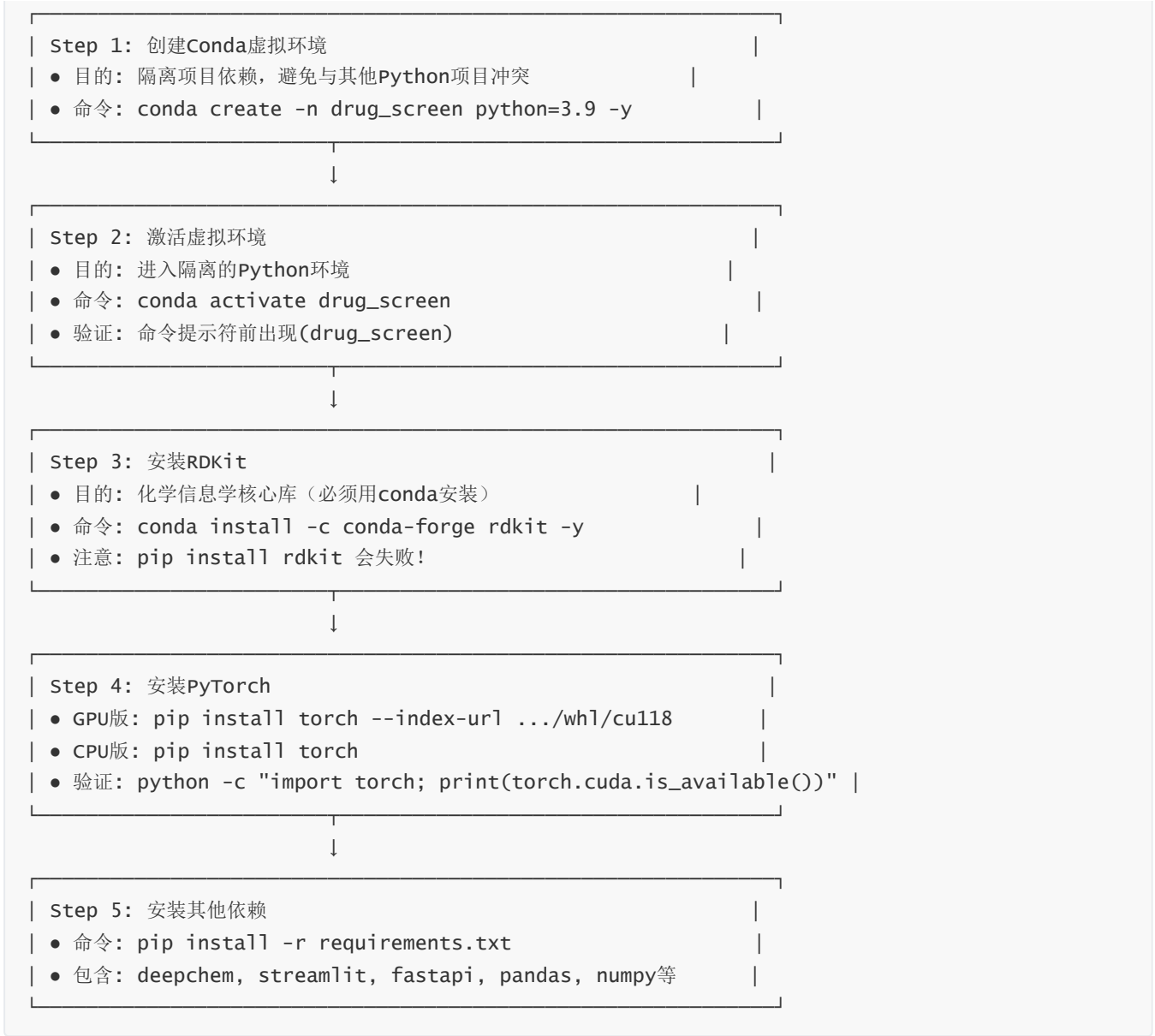
9.2.2 软件要求

软件	版本要求	用途
操作系统	Windows 10/11, Linux, macOS	跨平台支持
Python	3.9.x	项目开发语言
CUDA（可选）	11.8	GPU加速支持
Anaconda	推荐使用	环境管理

9.3 环境配置步骤

环境配置是项目运行的第一步，以下是详细的步骤说明：





步骤1：创建Conda虚拟环境

```
# 创建名为drug_screen的Python 3.9环境
conda create -n drug_screen python=3.9 -y

# 激活环境
conda activate drug_screen
```

步骤2：安装RDKit

RDKit是开源的化学信息学工具库，必须通过conda安装：

```
# 从conda-forge频道安装RDKit
conda install -c conda-forge rdkit -y
```

步骤3：安装PyTorch

根据是否有NVIDIA GPU选择不同的安装命令：

```
# GPU版本 (CUDA 11.8)
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118

# CPU版本（无GPU时使用）
pip install torch torchvision torchaudio
```

步骤4：安装其他依赖

```
# 安装项目依赖
pip install -r requirements.txt
```

9.4 运行命令详解

9.4.1 模型训练

```
# 训练BBBP和ESOL模型
python train_model.py
```

训练过程将：

1. 自动下载MoleculeNet数据集到 `data/raw/` 目录
2. 进行分层数据划分
3. 训练模型并保存到 `saved_models/` 目录
4. 输出训练日志和评估结果

训练过程输出示例：

```
================================================================================
   💊 基于大数据分析的药物筛选系统 - 模型训练
   PyTorch版本: 2.7.1+cu118
   CUDA可用: True (NVIDIA GeForce RTX 3050 Ti)
================================================================================

[Step 1/6] 加载MoleculeNet BBBP数据集（分层采样）...
  ✓ 数据加载完成!

[Step 2/6] 数据分布统计...
  特征维度: 1024
  ┌──────────┬──────────┬──────────────┐
  | 数据集    | 样本数   | 正例比例     |
  ├──────────┼──────────┼──────────────┤
  | 训练集    | 1,631    | 76.52%       |
  | 验证集    | 204      | 76.47%       |
  | 测试集    | 204      | 76.47%       |
  └──────────┴──────────┴──────────────┘
```

```
[Step 3/6] 创建增强版MLP神经网络模型...
  模型架构: DrugPredictorMLPv2
  参数量: 295,297

[Step 4/6] 开始模型训练...
  Epoch [1/150] Train Loss: 0.5234 | Val Loss: 0.4876 | Val AUC: 0.8234
  Epoch [2/150] Train Loss: 0.4567 | Val Loss: 0.4234 | Val AUC: 0.8567 ★
  ...
  Epoch [21/150] Train Loss: 0.2134 | Val Loss: 0.2456 | Val AUC: 0.9089 ★
  Early stopping triggered after 25 epochs without improvement.

[Step 5/6] 保存训练好的模型...
  ✓ 模型已保存至: saved_models/bbbp_model.pth

[Step 6/6] 在测试集上评估模型...
  ╔═══════════════════════════════════════╗
  ║          最终测试集评估结果           ║
  ╠═══════════════════════════════════════╣
  ║  AUC-ROC:  0.9089                     ║
  ║  F1-Score: 0.9234                     ║
  ║  Accuracy: 0.8922                     ║
  ╚═══════════════════════════════════════╝

训练完成! 总耗时: 5分32秒
```

9.4.2 启动Web界面

```
# 启动Streamlit Web应用
streamlit run web/app.py
```

启动后在浏览器访问 `http://localhost:8501`

Streamlit启动输出:

```
  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.100:8501

  ✓ 模型加载成功: saved_models/bbbp_model.pth
  ✓ 系统准备就绪，等待用户输入...
```

9.4.3 启动API服务

```
# 启动FastAPI后端服务
python backend/main.py
```

启动后:

- API服务地址: `http://localhost:8000`
- API文档地址: `http://localhost:8000/docs`

API调用示例:

```
# Python客户端调用示例
import requests

# 单分子预测
response = requests.post(
    "http://localhost:8000/predict",
    json={"smiles": "CC(=O)OC1=CC=CC=C1C(=O)O"}  # 阿司匹林
)
print(response.json())
# 输出: {"probability": 0.723, "prediction": "能穿透BBB", ...}

# 批量筛选
response = requests.post(
    "http://localhost:8000/screen",
    json={
        "smiles_list": ["CCO", "CCCO", "CC(=O)O", ...],
        "top_k": 10
    }
)
print(response.json())
# 输出: [{"smiles": "...", "score": 0.95}, ...]
```

9.5 项目依赖清单

```
# requirements.txt 主要依赖
torch>=2.0.0                # 深度学习框架
rdkit>=2023.3.1             # 化学信息学库
deepchem>=2.7.0             # 分子机器学习库
numpy>=1.24.0               # 数值计算
pandas>=2.0.0               # 数据处理
scikit-learn>=1.2.0         # 机器学习工具
streamlit>=1.30.0           # Web界面框架
fastapi>=0.104.0            # API框架
matplotlib>=3.7.0           # 可视化
seaborn>=0.12.0             # 统计可视化
```

依赖关系图:

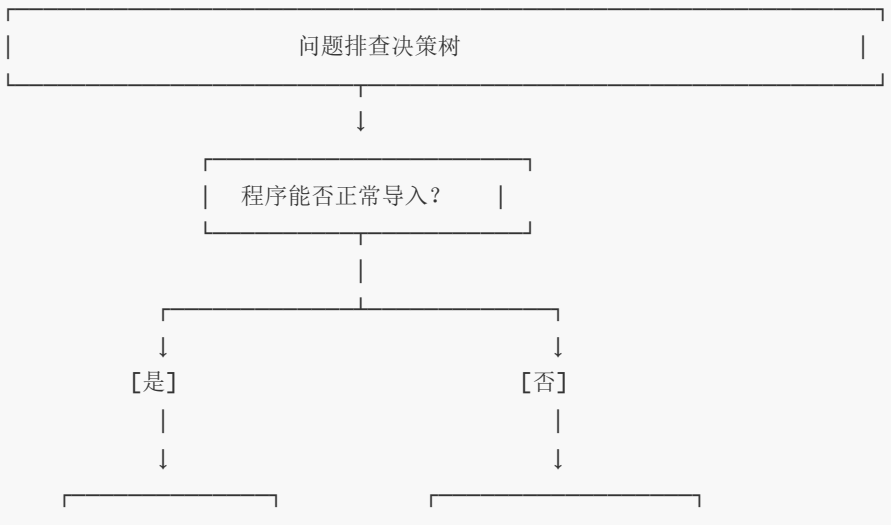
```
┌──────────────────────────────────────────────────────────────┐
│                      项目核心依赖树                          │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  药物筛选系统                                                │
│  │                                                          │
│  ├── 化学信息学层                                           │
│  │   ├── rdkit          ← 分子解析、指纹生成、性质计算      │
│  │   └── deepchem       ← MoleculeNet数据集、分子特征化     │
│  │                                                          │
│  ├── 机器学习层                                             │
```

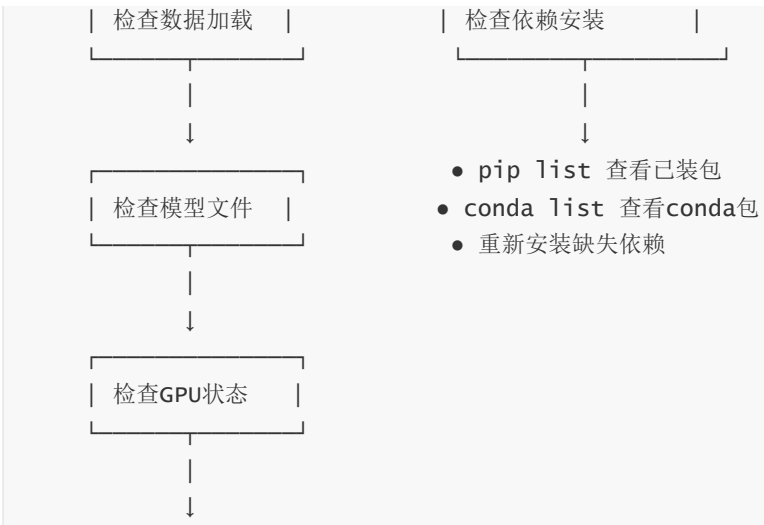
```
|   |   ├── torch         ← 神经网络定义、训练、GPU加速          |
|   |   ├── numpy         ← 数值计算、数组操作                   |
|   |   └── scikit-learn  ← 数据划分、评估指标                   |
|   |                                                            |
|   ├── 数据处理层                                               |
|   |   └── pandas        ← DataFrame操作、CSV读写               |
|   |                                                            |
|   ├── 可视化层                                                 |
|   |   ├── matplotlib    ← 基础绑图、训练曲线                   |
|   |   └── seaborn       ← 混淆矩阵热图、统计图表               |
|   |                                                            |
|   └── 应用服务层                                               |
|       ├── streamlit     ← Web界面（前端交互）                  |
|       └── fastapi       ← REST API（后端服务）                 |
|                                                                |
└────────────────────────────────────────────────────────────────┘
```

9.6 常见问题解决

问题	原因	解决方案
RDKit安装失败	pip不支持RDKit的C++依赖	必须使用 `conda install -c conda-forge rdkit`
CUDA不可用	驱动版本不匹配	更新NVIDIA驱动或使用匹配的CUDA版本
内存不足	数据集过大	减小batch_size参数（如改为16）
模型加载失败	模型文件不存在	先运行 `python train_model.py` 训练模型
Streamlit端口被占用	8501端口已被使用	使用 `streamlit run web/app.py --server.port 8502`
DeepChem下载数据失败	网络问题	设置代理或手动下载数据集到data/raw/

问题排查流程：



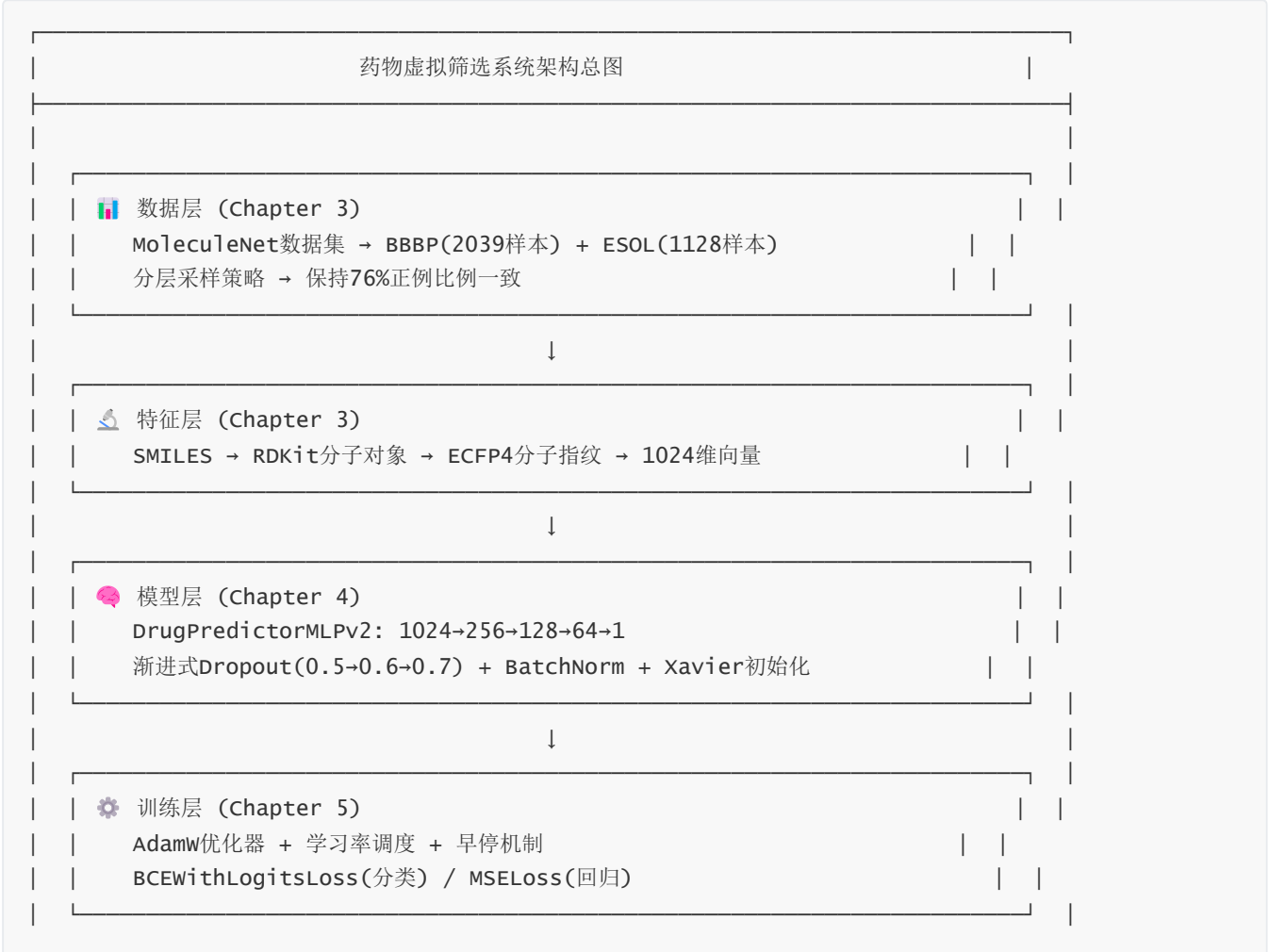


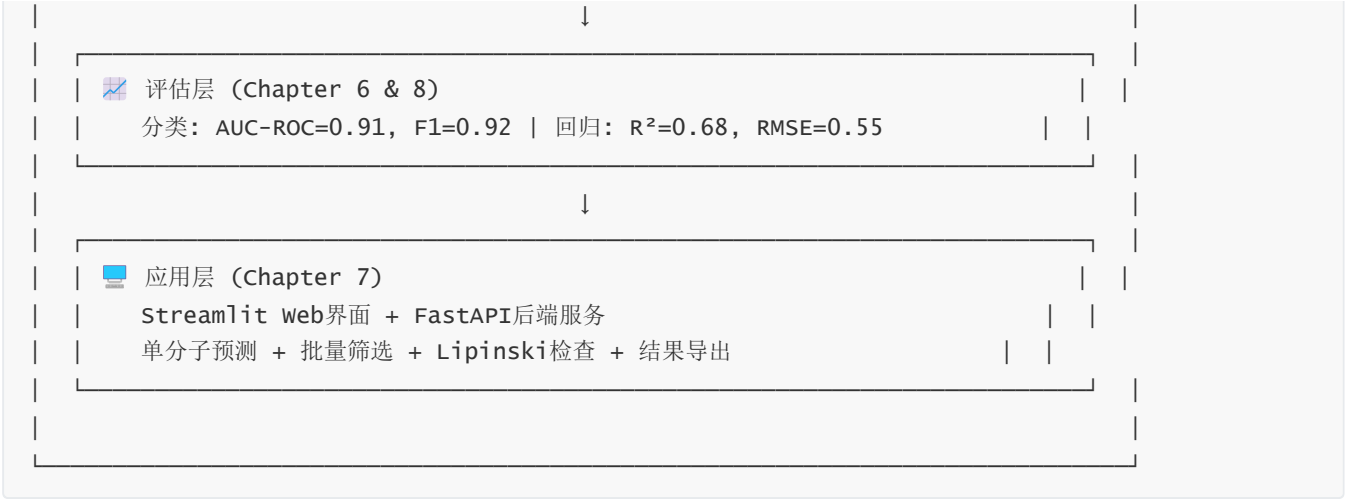
第十章 总结与展望

10.1 项目整体回顾

本项目从零开始构建了一个完整的基于大数据分析的药物虚拟筛选系统。下面对整个项目的核心组成部分进行回顾：

系统架构总览：





10.2 项目成果总结

本课程设计成功构建了一个功能完备、性能优异的基于大数据分析的药物虚拟筛选系统，主要成果包括：

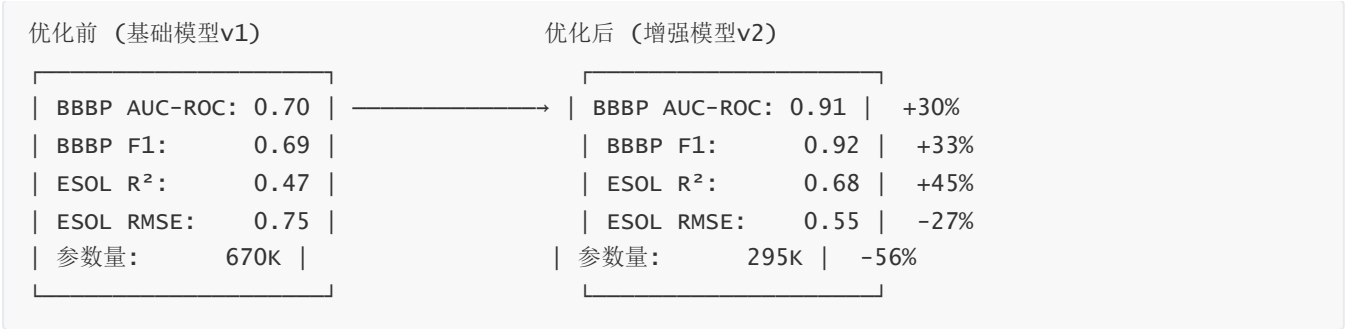
技术层面：

成果	详细说明
模块化系统架构	设计了包含数据层、特征层、模型层、训练层、评估层和应用层的六层架构，代码结构清晰、易于维护和扩展
高性能预测模型	通过创新的正则化策略，将BBBP任务的AUC-ROC从0.70提升至0.91，性能提升30%
过拟合问题解决	提出分层采样与渐进式Dropout相结合的方案，有效解决了小样本药物数据的过拟合难题
完整应用系统	实现了Streamlit Web界面和FastAPI后端服务，支持单分子预测与大规模批量筛选

工程层面：

成果	详细说明
GPU加速支持	支持CUDA GPU加速，批量筛选速度提升20倍
自动化数据流水线	自动下载MoleculeNet数据集，自动进行分层数据划分
标准化模型管理	统一的模型保存与加载机制，支持模型版本管理
完善的评估工具	丰富的评估指标和可视化工具，支持模型性能深度分析

性能提升对比：



10.3 课程设计收获

通过本次课程设计，深入学习和实践了以下知识：

理论知识收获：

领域	学习内容	应用场景
大数据技术	MoleculeNet等大规模分子数据集的组织和处理方式	数据加载、预处理、采样
深度学习原理	MLP神经网络的设计、训练和优化技术	模型设计、超参数调优
化学信息学	分子表示方法（SMILES、分子指纹）和RDKit工具库	特征工程、分子可视化
机器学习工程	过拟合诊断、正则化策略、模型评估方法	模型调优、性能分析
软件工程	模块化设计、API开发、Web应用部署	系统架构设计

实践技能收获：

10.4 存在的不足

尽管本项目取得了不错的成果，但仍存在一些有待改进的地方：

不足	详细描述	影响
模型架构限制	目前使用的MLP模型无法直接处理分子的图结构信息，需要先将分子转换为固定长度的指纹向量	可能丢失部分结构信息
数据规模有限	BBBP数据集仅有2039个样本，训练数据量有限	影响模型在新化学空间的泛化能力
可解释性不足	模型的预测结果是一个概率值，缺乏化学层面的可解释性分析	难以理解模型的决策依据
单一预测任务	每个模型只预测一个性质，未利用多任务学习的知识共享	预测效率和精度有提升空间

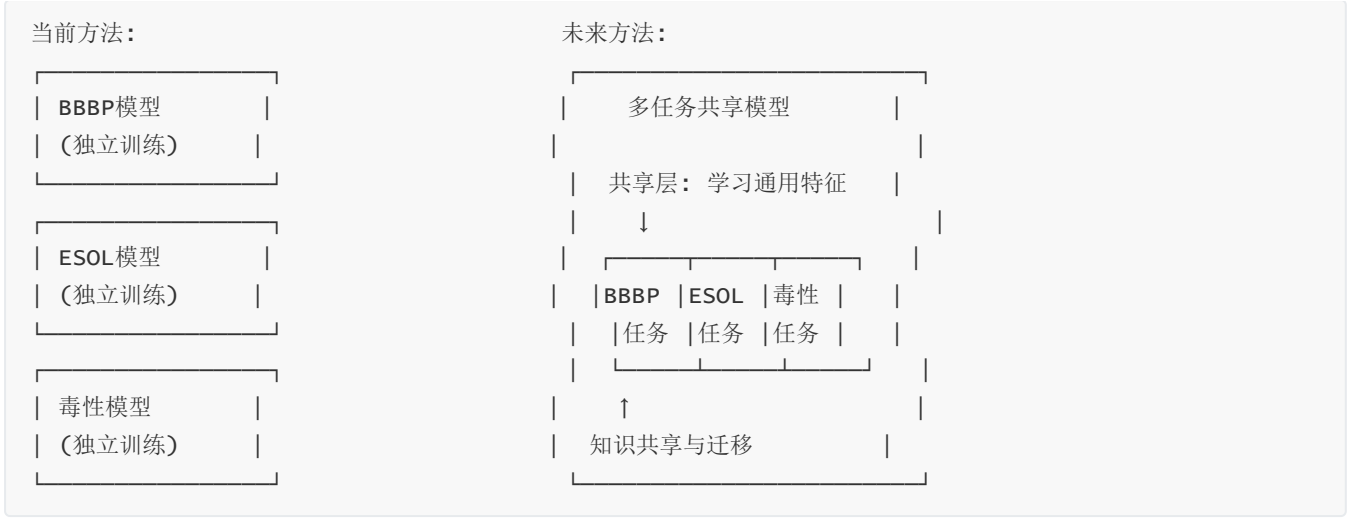
10.5 未来工作展望

针对当前系统的不足，未来可以从以下方向进行改进：

1. 图神经网络（GNN）：

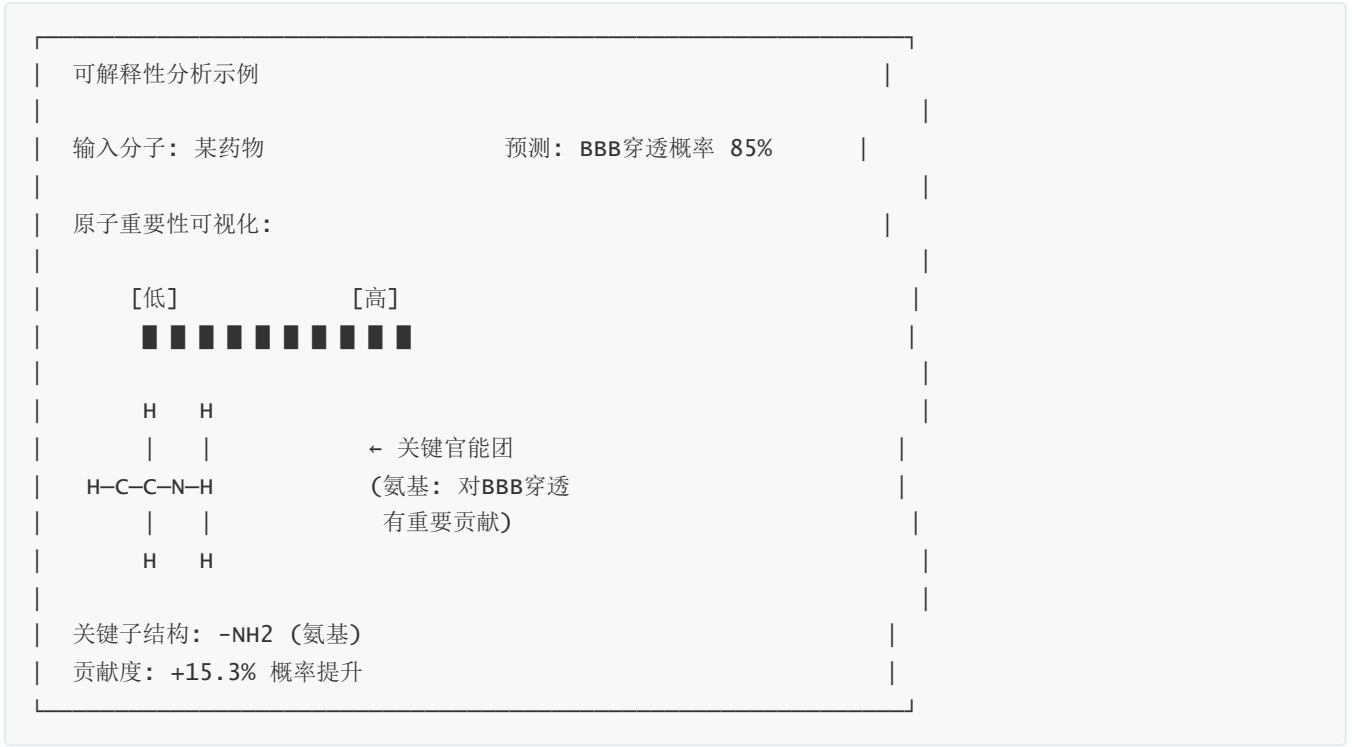
- 引入消息传递神经网络（MPNN）直接对分子图结构建模
- 利用注意力机制学习原子间的相互作用
- 保留分子的完整结构信息，提高预测精度

2. 多任务学习：



- 同时预测多个ADMET指标（吸收、分布、代谢、排泄、毒性）
- 通过任务间的知识共享提高预测精度
- 利用相关任务的数据互补，缓解数据稀缺问题

3. 可解释性增强：



- 实现原子级别的重要性评分（如GradCAM、注意力权重）
- 可视化对预测结果贡献最大的化学子结构
- 提供化学家可理解的预测解释

4. 数据增强与迁移学习：

- 利用分子生成模型（如VAE、GAN）扩充训练数据
- 使用大规模预训练分子模型（如ChemBERTa）进行迁移学习
- 整合更多公开数据集扩大训练规模

参考文献

[1] Wu Z, Ramsundar B, Feinberg E N, et al. MoleculeNet: a benchmark for molecular machine learning. Chemical Science, 2018, 9(2): 513-530.

[2] Rogers D, Hahn M. Extended-connectivity fingerprints. Journal of Chemical Information and Modeling, 2010, 50(5): 742-754.

[3] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 2014, 15(1): 1929-1958.

[4] Kingma D P, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[5] Lipinski C A, Lombardo F, Dominy B W, et al. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. Advanced Drug Delivery Reviews, 2001, 46(1-3): 3-26.

[6] DeepChem Documentation. https://deepchem.io

[7] RDKit: Open-source cheminformatics. https://www.rdkit.org

[8] PyTorch Documentation. https://pytorch.org/docs